van Emde Boas Trees

Thomas Ammer and Peter Lammich

November 9, 2026

Abstract

The *van Emde Boas tree* or *van Emde Boas priority queue* [1, 2] is a data structure supporting membership test, insertion, predecessor and successor search, minimum and maximum determination and deletion in $\mathcal{O}(\log\log|\mathcal{U}|)$ time, where $\mathcal{U} = \{0, ..., 2^n - 1\}$ is the overall range to be considered. The presented formalization follows Chapter 20 of the popular *Introduction to Algorithms (3rd ed.)* [3] by Cormen, Leiserson, Rivest and Stein (CLRS), extending the list of formally verified CLRS algorithms [4]. Our current formalization is based on the first author's bachelor's thesis.

First, we prove correct a *functional* implementation, w.r.t. an abstract data type for sets. Apart from functional correctness, we show a resource bound, and runtime bounds w.r.t. manually defined timing functions [5] for the operations.

Next, we refine the operations to Imperative HOL [6, 7] with time [8], and show correctness and complexity. This yields a practically more efficient implementation, and eliminates the manually defined timing functions from the trusted base of the proof.

Contents

1 Preliminaries and Preparations **5**
- 1.1 Data Type Definition . . . 5
- 1.2 Functions for obtaining high and low bits of an input number. . . . 5
- 1.3 Some auxiliary lemmata . . . 5
- 1.4 Auxiliary functions for defining valid Van Emde Boas Trees . . . 6
- 1.5 Inductive Definition of semantically valid Vam Emde Boas Trees . . . 7
- 1.6 Function for generating an empty tree of arbitrary degree respectively order . . . 9

2 Member Function **19**

3 Insert Function **32**

4 Correctness of the Insert Operation **51**
- 4.1 Validness Preservation . . . 51
- 4.2 Correctness with Respect to Set Interpretation . . . 75

5 The Minimum and Maximum Operation **77**

6 The Successor Operation **86**
- 6.1 Auxiliary Lemmas on Sets and Successorship . . . 86
- 6.2 The actual Function . . . 87
- 6.3 Lemmas for Term Decomposition . . . 87
- 6.4 Correctness Proof . . . 88

7 The Predecessor Operation **107**
- 7.1 Lemmas on Sets and Predecessorship . . . 108
- 7.2 The actual Function for Predecessor Search . . . 109
- 7.3 Auxiliary Lemmas . . . 109
- 7.4 Correctness Proof . . . 110

8 Deletion **128**
- 8.1 Function Definition . . . 128
- 8.2 Auxiliary Lemmas . . . 129
- 8.3 Validness Preservation . . . 172
- 8.4 Correctness with Respect to Set Interpretation . . . 225

9 Uniqueness Property of valid Trees **226**

10 Heights of van Emde Boas Trees **241**

11 Upper Bounds for canonical Functions: Relationships between Run Time and Tree Heights **245**
- 11.1 Membership test . . . 245
- 11.2 Minimum, Maximum, Emptiness Test . . . 255

11.3 Insertion . . . 256
11.4 Successor Function . . . 269
11.5 Predecessor Function . . . 277
11.6 Running Time Bounds for Deletion . . . 284

12 Space Complexity and *buildup* Time Consumption **313**
12.1 Space Comlexity of valid van Emde Boas Trees . . . 313
12.2 Auxiliary Lemmas for List Summation . . . 313
12.3 Actual Space Reasoning . . . 315
12.4 Complexity of Generation Time . . . 317

13 Functional Interface **323**
13.1 Code Generation Setup . . . 323
13.1.1 Code Equations . . . 323
13.2 Correctness Lemmas . . . 326
13.2.1 Space Bound . . . 327
13.2.2 Buildup . . . 327
13.2.3 Equality . . . 327
13.2.4 Member . . . 327
13.2.5 Insert . . . 327
13.2.6 Maximum . . . 327
13.2.7 Minimum . . . 328
13.3 Emptiness determination . . . 328
13.3.1 Successor . . . 328
13.3.2 Predecessor . . . 329
13.3.3 Delete . . . 329
13.4 Interface Usage Example . . . 329
13.5 Lists . . . 331

14 Imperative van Emde Boas Trees **337**
14.1 Assertions on van Emde Boas Trees . . . 338
14.2 High and low Bitsequences Definition . . . 340

15 Imperative Implementation of *vebt − buildup* **341**

16 Minimum and Maximum Determination **352**

17 Membership Test on imperative van Emde Boas Trees **353**
17.1 *minNulli*: empty tree? . . . 356

18 Imperative *vebt − insert* to van Emde Boas Tree **357**

19 Imperative Successor **361**

20 Imperative Predecessor **367**

21 Imperative Delete **372**

22 Imperative Interface **386**
22.1 Code Export . . . 386
22.2 Interface . . . 387
22.2.1 Buildup . . . 387
22.2.2 Member . . . 387
22.2.3 Insert . . . 388
22.2.4 Maximum . . . 388
22.2.5 Minimum . . . 388
22.2.6 Successor . . . 388
22.2.7 Predecessor . . . 389
22.2.8 Delete . . . 389
22.3 Setup of VCG . . . 389

23 Interface Usage Example **390**
23.1 Test Program . . . 390
23.2 Correctness without Time . . . 390
23.3 Time Bound Reasoning . . . 391

24 Conclusion **393**

theory *VEBT-Definitions* **imports**
Main
HOL−Library.Extended-Nat
HOL−Library.Code-Target-Numeral
HOL−Library.Code-Target-Nat

begin

1 Preliminaries and Preparations

1.1 Data Type Definition

datatype *VEBT* = *is-Node*: *Node* (*info*:(*nat*∗*nat*) *option*)(*deg*: *nat*)(*treeList*: *VEBT list*) (*summary*:*VEBT*)
|
is-Leaf: *Leaf* *bool* *bool*

hide-const (**open**) *info deg treeList summary*

locale *VEBT-internal* **begin**

1.2 Functions for obtaining high and low bits of an input number.

definition *high* :: *nat* ⇒ *nat* ⇒ *nat* **where**
high x n = (*x div* (2^*n*))

definition *low* :: *nat* ⇒ *nat* ⇒ *nat* **where**
low x n = (*x mod* (2^*n*))

1.3 Some auxiliary lemmata

lemma *inthall*[*termination-simp*]: (⋀ *x*. *x* ∈ *set xs* ⟹ *P x*) ⟹ *n* < *length xs* ⟹ *P* (*xs* ! *n*)
by *simp*

lemma *intind*: *i* < *n* ⟹ *P x* ⟹ *P* (*replicate n x* ! *i*)
by (*metis in-set-replicate inthall length-replicate*)

lemma *concat-inth*:(*xs* @[*x*]@*ys*)! (*length xs*) = *x*
by *simp*

lemma *pos-n-replace*: *n*<*length xs* ⟹ *length xs* = *length* (*take n xs* @ [*y*] @*drop* (*Suc n*) *xs*)
by *simp*

lemma *inthrepl*: *i* < *n* ⟹ (*replicate n x*) ! *i* = *x* **by** *simp*

lemma *nth-repl*: *m*<*length xs* ⟹ *n* <*length xs* ⟹ *m*≠ *n* ⟹(*take n xs* @ [*x*] @ *drop* (*n*+*1*) *xs*) ! *m* = *xs* ! *m*
by (*metis Suc-eq-plus1 append-Cons append-Nil nth-list-update-neq upd-conv-take-nth-drop*)

lemma [*termination-simp*]:**assumes** *high x deg* < *length treeList*
shows*size* (*treeList* ! *high x deg*) < *Suc* (*size-list size treeList* + *size s*)
proof−
have *treeList* ! *high x deg* ∈ *set treeList*
using *assms* **by** *auto*
then show *?thesis*
using *not-less-eq size-list-estimation* **by** *fastforce*
qed

1.4 Auxiliary functions for defining valid Van Emde Boas Trees

This function checks whether an element occurs in a Leaf

fun *naive-member* :: *VEBT* ⇒ *nat* ⇒ *bool* **where**
naive-member (*Leaf a b*) *x* = (*if x* = *0 then a else if x* = *1 then b else False*)|
naive-member (*Node - 0 - -*) - = *False*|
naive-member (*Node - deg treeList s*) *x* = (*let pos* = *high x* (*deg div 2*) *in*
(*if pos* < *length treeList then naive-member* (*treeList* ! *pos*) (*low x* (*deg div 2*)) *else False*))

Test for elements stored by using the provide min-max-fields

fun *membermima* :: *VEBT* ⇒ *nat* ⇒ *bool* **where**
membermima (*Leaf - -*) - = *False*|
membermima (*Node None 0 - -*)- =*False*|
membermima (*Node* (*Some* (*mi*,*ma*)) *0 - -*) *x* = (*x* = *mi* ∨ *x* = *ma*)|
membermima (*Node* (*Some* (*mi*, *ma*)) *deg treeList -*) *x* = (*x* = *mi* ∨ *x* = *ma* ∨ (
let pos = *high x* (*deg div 2*) *in* (*if pos* < *length treeList*
then membermima (*treeList* ! *pos*) (*low x* (*deg div 2*)) *else False*)))|
membermima (*Node None* (*deg*) *treeList -*) *x* = (*let pos* = *high x* (*deg div 2*) *in*
(*if pos* < *length treeList then membermima* (*treeList* ! *pos*) (*low x* (*deg div 2*)) *else False*))

lemma *length-mul-elem*:(∀ *x* ∈ *set xs*. *length x* = *n*) ⟹ *length* (*concat xs*) = (*length xs*) ∗ *n*
apply(*induction xs*)
apply *auto*
done

We combine both auxiliary functions: The following test returns true if and only if an element occurs in the tree with respect to our interpretation no matter where it is stored.

definition *both-member-options* :: *VEBT* ⇒ *nat* ⇒ *bool* **where**
both-member-options t x = (*naive-member t x* ∨ *membermima t x*)

end
context begin
interpretation *VEBT-internal* **.**

definition *set-vebt* :: *VEBT* ⇒ *nat set* **where**
set-vebt t = {*x*. *both-member-options t x*}
end

1.5 Inductive Definition of semantically valid Vam Emde Boas Trees

Invariant for verification proofs

context begin
interpretation *VEBT-internal* **.**

inductive *invar-vebt*::*VEBT* ⇒ *nat* ⇒ *bool* **where**
invar-vebt (*Leaf a b*) (*Suc 0*) |
(∀ *t* ∈ *set treeList*. *invar-vebt t n*) ⟹ *invar-vebt summary m* ⟹ *length treeList* = 2^*m*
⟹ *m* = *n* ⟹ *deg* = *n* + *m* ⟹ (∄ *i*. *both-member-options summary i*)
⟹(∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*)
⟹ *invar-vebt* (*Node None deg treeList summary*) *deg*|
(∀ *t* ∈ *set treeList*. *invar-vebt t n*) ⟹ *invar-vebt summary m*
⟹ *length treeList* = 2^*m* ⟹ *m* = *Suc n* ⟹ *deg* = *n* + *m* ⟹ (∄ *i*. *both-member-options summary i*)
⟹ (∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*)
⟹ *invar-vebt* (*Node None deg treeList summary*) *deg*|
(∀ *t* ∈ *set treeList*. *invar-vebt t n*) ⟹ *invar-vebt summary m* ⟹ *length treeList* = 2^*m* ⟹ *m* = *n*
⟹*deg* = *n* + *m*⟹ (∀ *i* < 2^*m*. (∃ *x*. *both-member-options* (*treeList* ! *i*) *x*) ⟷ (*both-member-options summary i*)) ⟹
(*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*)) ⟹
mi ≤ *ma* ⟹ *ma* < 2^*deg* ⟹
(*mi* ≠ *ma* ⟶
(∀ *i* < 2^*m*.
(*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *x*. (*high x n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low x n*)
) ⟶ *mi* < *x* ∧ *x* ≤ *ma*)))
⟹ *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*|
(∀ *t* ∈ *set treeList*. *invar-vebt t n*) ⟹*invar-vebt summary m* ⟹ *length treeList* = 2^*m*
⟹ *m* = *Suc n* ⟹*deg* = *n* + *m*⟹(∀ *i* < 2^*m*. (∃ *x*. *both-member-options* (*treeList* ! *i*) *x*) ⟷ (*both-member-options summary i*)) ⟹
(*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*)) ⟹
mi ≤ *ma* ⟹ *ma* < 2^*deg* ⟹
(*mi* ≠ *ma* ⟶
(∀ *i* < 2^*m*.
(*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *x*. (*high x n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low x n*)
) ⟶ *mi* < *x* ∧ *x* ≤ *ma*)))
⟹ *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*

end

context *VEBT-internal* **begin**

definition *in-children n treeList x* ≡ *both-member-options* (*treeList* ! *high x n*) (*low x n*)

functional validness definition

fun *valid'* :: *VEBT* ⇒ *nat* ⇒ *bool* **where**

```
  valid' (Leaf - -) d ⟷ d=1
| valid' (Node mima deg treeList summary) deg' ⟷
  (
    deg=deg' ∧ (
    let n = deg div 2; m = deg − n in
      ( ∀ t ∈ set treeList. valid' t n )
    ∧ valid' summary m
    ∧ length treeList = 2^m
    ∧ (
      case mima of
       None ⇒ (∄ i. both-member-options summary i) ∧ (∀ t ∈ set treeList. ∄ x. both-member-options
t x)
     | Some (mi,ma) ⇒
         mi ≤ ma ∧ ma<2^deg
       ∧ (∀ i < 2^m. (∃ x. both-member-options (treeList ! i) x) ⟷ ( both-member-options summary
i))
       ∧ (if mi=ma then (∀ t ∈ set treeList. ∄ x. both-member-options t x)
          else
            in-children n treeList ma
          ∧ (∀x < 2^deg. in-children n treeList x ⟶ mi<x ∧ x≤ma)
         )
    )
    )
  )
```

equivalence proofs

```
lemma high-bound-aux: ma < 2^(n+m) ⟹ high ma n < 2^m
  unfolding high-def
  by (simp add: add.commute less-mult-imp-div-less power-add)

lemma valid-eq1:
  assumes invar-vebt t d
  shows valid' t d
  using assms apply induction
  apply simp-all
  apply (auto simp: in-children-def dest: high-bound-aux) []
  subgoal for treeList n summary m deg mi ma
    apply (intro allI impI conjI)
    apply (auto simp: in-children-def dest: high-bound-aux) []
    apply (metis add-Suc-right high-bound-aux power-Suc)
    apply (auto simp: in-children-def dest: high-bound-aux) []
    apply (metis add-Suc-right high-bound-aux power-Suc)
    apply (auto simp: in-children-def dest: high-bound-aux) []
    apply (metis add-Suc-right high-bound-aux power-Suc)
    done
  done

lemma even-odd-cases:
```

fixes x :: *nat*
obtains n **where** $x{=}n{+}n$ | n **where** $x = n + Suc\ n$
apply (*cases even x*; *simp*)
apply (*metis add-self-div-2 div-add*)
by (*metis add.commute mult-2 oddE plus-1-eq-Suc*)

lemma *valid-eq2*: *valid' t d* $\Longrightarrow$ *invar-vebt t d*
apply (*induction t d rule*: *valid'.induct*)
apply (*auto intro*: *invar-vebt.intros simp*: *Let-def split*: *option.splits*)
subgoal for *deg treeList summary*
apply (*rule even-odd-cases*[*of deg*]; *simp*)
apply (*rule invar-vebt.intros*(*2*); *simp*)
apply (*rule invar-vebt.intros*(*3*); *simp add*: *algebra-simps*) **by** *presburger*
subgoal for *deg treeList summary mi ma*
apply (*rule even-odd-cases*[*of deg*]; *simp*)
subgoal
apply (*rule invar-vebt.intros*(*4*); *simp?*)
apply (*auto simp*: *in-children-def*) []
apply (*meson le-less-linear le-less-trans*)
apply (*metis div-eq-0-iff div-exp-eq gr-implies-not0 high-def*)
done
subgoal
apply (*rule invar-vebt.intros*(*5*); *simp?*)
apply (*auto*) []
apply (*auto*) []
apply (*auto simp*: *in-children-def*) []
apply (*meson le-less-linear le-less-trans*)
apply (*metis div-eq-0-iff add-Suc-right div-exp-eq high-def power-Suc power-eq-0-iff zero-neq-numeral*)
done
done
done

lemma *valid-eq*: *valid' t d* $\longleftrightarrow$ *invar-vebt t d*
using *valid-eq1 valid-eq2* **by** *auto*

lemma [*termination-simp*]: **assumes** *odd* (*v*::*nat*) **shows** *v div 2* $<$ *v*
by (*simp add*: *assms odd-pos*)

lemma [*termination-simp*]:**assumes** $n > 1$ **and** *odd n* **shows** *Suc* (*n div 2*) $<$ *n*
by (*metis Suc-lessI add-diff-cancel-left' assms*(*1*) *assms*(*2*) *div-eq-dividend-iff div-less-dividend even-Suc even-Suc-div-two odd-pos one-less-numeral-iff plus-1-eq-Suc semiring-norm*(*76*) *zero-less-diff*)

end

1.6 Function for generating an empty tree of arbitrary degree respectively order

context begin
interpretation *VEBT-internal* **.**

```
fun vebt-buildup :: nat ⇒ VEBT where
  vebt-buildup 0 = Leaf False False|
  vebt-buildup (Suc 0) = Leaf False False|
  vebt-buildup n = (if even n then (let half = n div 2 in
                    Node None n (replicate (2^half) (vebt-buildup half)) (vebt-buildup half))
                  else (let half = n div 2 in
                    Node None n ( replicate (2^(Suc half)) (vebt-buildup half)) (vebt-buildup (Suc half))))

end

context VEBT-internal begin

lemma buildup-nothing-in-leaf: ¬ naive-member (vebt-buildup n) x
proof(induction arbitrary: x rule: vebt-buildup.induct)
  case 1
  then show ?case by simp
next
  case (2 v)
  then show ?case
    by simp
next
  case (3 n)
  let ?n = Suc(Suc n)
  show ?case proof(cases even ?n)
    case True
    let ?half = ?n div 2
    have ¬ naive-member (vebt-buildup ?half) y for y
      using 3.IH(1) True by blast
    hence 0:∀ t ∈ set (replicate (2^?half) (vebt-buildup ?half)) . ¬ naive-member t x
      by simp
    have naive-member (vebt-buildup ?n) x ⟹ False
    proof−
      assume naive-member (vebt-buildup ?n) x
      hence high x ?half < 2^?half ∧
            naive-member ((replicate (2^?half) (vebt-buildup ?half)) ! (high x ?half)) (low x ?half)
        by (metis True vebt-buildup.simps(3) length-replicate naive-member.simps(3))
      hence ∃ t ∈ set (replicate (2^?half) (vebt-buildup ?half)) . naive-member t x
        by (metis ‹⋀y. ¬ naive-member (vebt-buildup (Suc (Suc n) div 2)) y› nth-replicate)
      then show False using 0 by simp
    qed
    then show ?thesis
      by blast
  next
    case False
    let ?half = ?n div 2
    have ¬ naive-member (vebt-buildup ?half) y for y
      using 3.IH False by blast
    hence 0:∀ t ∈ set (replicate (2^(Suc ?half)) (vebt-buildup ?half)) . ¬ naive-member t x
```

```
    by simp
  have naive-member (vebt-buildup ?n) x ⟹ False
  proof−
    assume naive-member (vebt-buildup ?n) x
    hence high x ?half < 2^(Suc ?half) ∧
            naive-member ((replicate (2^(Suc ?half)) (vebt-buildup ?half)) ! (high x ?half)) (low x
?half)
      by (metis False vebt-buildup.simps(3) length-replicate naive-member.simps(3))
    hence ∃ t ∈ set (replicate (2^(Suc ?half)) (vebt-buildup ?half)) . naive-member t x
      by (metis ‹⋀y. ¬ naive-member (vebt-buildup (Suc (Suc n)) div 2)) y› nth-replicate)
    then show False using 0 by simp
  qed
  then show ?thesis by force
 qed
qed


lemma buildup-nothing-in-min-max:¬ membermima (vebt-buildup n) x
proof(induction arbitrary: x rule: vebt-buildup.induct)
 case 1
 then show ?case by simp
next
 case 2
 then show ?case by simp
next
 case (3 va)
 let ?n = Suc (Suc va)
 let ?half = ?n div 2
 show ?case proof(cases even ?n)
  case True
  have ¬ membermima (vebt-buildup ?half) y for y
   using 3.IH(1) True by blast
  hence 0:∀ t ∈ set (replicate (2^?half) (vebt-buildup ?half)) . ¬ membermima t x
   by simp
  then show ?thesis
   by (metis 3.IH(1) True vebt-buildup.simps(3) inthrepl length-replicate membermima.simps(5))
 next
  case False
   have ¬ membermima (vebt-buildup ?half) y for y
    using 3.IH False by blast
  moreover hence 0:∀ t ∈ set (replicate (2^(Suc ?half)) (vebt-buildup ?half)) . ¬ membermima t
x
   by simp
  ultimately show ?thesis
   by (metis vebt-buildup.simps(3) inthrepl length-replicate membermima.simps(5))
 qed
qed
```

The empty tree generated by $vebt_buildup$ is indeed a valid tree.

lemma *buildup-gives-valid*: *n>0* $\Longrightarrow$ *invar-vebt* (*vebt-buildup n*) *n*
proof(*induction n rule*: *vebt-buildup.induct*)
case *1*
then show *?case* **by** *simp*
next
case *2*
then show *?case*
by (*simp add*: *invar-vebt.intros*(*1*))
next
case (*3 va*)
let *?n* = *Suc* (*Suc va*)
let *?half* = *?n div 2*
show *?case* **proof**(*cases even ?n*)
case *True*
hence *a*:*vebt-buildup ?n* = *Node None ?n* (*replicate* (*2^?half*) (*vebt-buildup ?half*)) (*vebt-buildup ?half*) **by** *simp*
moreover hence *invar-vebt* (*vebt-buildup ?half*) *?half*
using *3.IH*(*1*) *True* **by** *auto*
moreover hence ($\forall$ *t* $\in$ *set* (*replicate* (*2^?half*) (*vebt-buildup ?half*)). *invar-vebt t ?half*) **by** *simp*
moreover have *length* (*replicate* (*2^?half*) (*vebt-buildup ?half*)) = *2^?half* **by** *auto*
moreover have *?half*+*?half* = *?n*
using *True* **by** *auto*
moreover have $\forall$ *t* $\in$ *set* (*replicate* (*2^?half*) (*vebt-buildup ?half*)). ($\nexists$ *x*. *both-member-options t x*)
proof
fix *t*
assume *t* $\in$ *set* (*replicate* (*2^?half*) (*vebt-buildup ?half*))
hence *t* = (*vebt-buildup ?half*) **by** *simp*
thus $\nexists$ *x*. *both-member-options t x*
by (*simp add*: *both-member-options-def buildup-nothing-in-leaf buildup-nothing-in-min-max*)
qed
moreover have ($\nexists$ *i*. *both-member-options* (*vebt-buildup ?half*) *i*)
using *both-member-options-def buildup-nothing-in-leaf buildup-nothing-in-min-max* **by** *blast*
ultimately have *invar-vebt* (*Node None ?n* (*replicate* (*2^?half*) (*vebt-buildup ?half*)) (*vebt-buildup ?half*)) *?n*
using *invar-vebt.intros*(*2*)[*of replicate* (*2^?half*) (*vebt-buildup ?half*) *?half vebt-buildup ?half ?half ?n*]
by *simp*
then show *?thesis* **using** *a* **by** *auto*
next
case *False*
hence *a*:*vebt-buildup ?n* = *Node None ?n* (*replicate* (*2^*(*Suc ?half*)) (*vebt-buildup ?half*)) (*vebt-buildup* (*Suc ?half*)) **by** *simp*
moreover hence *invar-vebt* (*vebt-buildup* (*Suc ?half*)) (*Suc ?half*)
using *3.IH False* **by** *auto*
moreover have *invar-vebt* (*vebt-buildup ?half*) *?half*
using *3.IH*(*3*) *False* **by** *auto*
moreover hence ($\forall$ *t* $\in$ *set* (*replicate* (*2^*(*Suc ?half*)) (*vebt-buildup ?half*)). *invar-vebt t ?half*) **by** *simp*

```
    moreover have length (replicate (2^(Suc ?half)) (vebt-buildup ?half)) = 2^(Suc ?half) by auto
    moreover have (Suc ?half)+?half = ?n
      using False by presburger
   moreover have ∀ t ∈ set (replicate (2^(Suc ?half)) (vebt-buildup ?half)). (∄ x. both-member-options
t x)
    proof
      fix t
      assume t ∈ set (replicate (2^(Suc ?half)) (vebt-buildup ?half))
      hence t = (vebt-buildup ?half) by simp
      thus ∄ x. both-member-options t x
        by (simp add: both-member-options-def buildup-nothing-in-leaf buildup-nothing-in-min-max)
    qed
    moreover have  (∄ i. both-member-options (vebt-buildup (Suc ?half)) i)
      using both-member-options-def buildup-nothing-in-leaf buildup-nothing-in-min-max by blast
    moreover have ?half + Suc ?half = ?n
      using calculation(6) by auto
      ultimately have invar-vebt  (Node None ?n (replicate (2^(Suc ?half)) (vebt-buildup ?half))
(vebt-buildup (Suc ?half))) ?n
     using invar-vebt.intros(3)[of replicate (2^(Suc ?half)) (vebt-buildup ?half) ?half vebt-buildup (Suc
?half) Suc ?half ?n ]
      by simp
    then show ?thesis using a by auto
  qed
qed

lemma mi-ma-2-deg: assumes invar-vebt (Node (Some (mi, ma)) deg treeList summary) n shows
mi≤ ma ∧ ma < 2^deg
proof−
  from  assms show ?thesis proof cases qed blast+
qed

lemma deg-not-0: invar-vebt t n ⟹ n > 0
  apply(induction t n rule: invar-vebt.induct)
     apply auto
  done

lemma set-n-deg-not-0:assumes ∀ t∈set treeList. invar-vebt t nand length treeList = 2^m shows  n
≥ 1
proof−
  have length treeList > 0
    by (simp add: assms(2))
  then obtain t ts where treeList = t#ts
    by (metis list.size(3) neq-Nil-conv not-less0)
  hence invar-vebt t n
    by (simp add: assms(1))
  hence n ≥ 1
    using deg-not-0 by force
  thus ?thesis by simp
qed
```

lemma *both-member-options-ding*: **assumes***invar-vebt* (*Node info deg treeList summary*) *n* **and** *x*<*2^deg***and** *both-member-options* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))**shows** *both-member-options* (*Node info deg treeList summary*) *x*
proof −
obtain *m n* :: *nat* **where**
deg = *n* + *m* **and** *m* = *n* ∨ *m* = *Suc n* **and** *length treeList* = *2* ^ *m*
using *assms*(*1*)
by (*induction* (*Node info deg treeList summary*) *n rule*: *invar-vebt.induct*; *metis*)

have *deg* > *1*
using *set-n-deg-not-0 assms*(*1*) *valid-eq valid'.simps*(*2*) **by** *fastforce*

have *high x* (*deg div 2*) < *2^m*
using *assms*(*2*) ‹*deg* = *n* + *m*› ‹*m* = *n* ∨ *m* = *Suc n*› *high-bound-aux* **by** *force*

consider
(*mima*) *membermimа* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) |
(*naive*) *naive-member* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))
using *assms*(*3*) *both-member-options-def* **by** *auto*

then show *?thesis*
proof *cases*
case *mima*
then have *membermima* (*Node info deg treeList summary*) *x*
proof (*cases info*)
case *None*
then show *?thesis*
using *mima* ‹*length treeList* = *2* ^ *m*› ‹*deg* > *1*› ‹*high x* (*deg div 2*) < *2^m*›
using *membermima.simps*(*5*)[*of deg*−*1 treeList summary x*]
by *fastforce*
next
case (*Some p*)
then obtain *a b* **where** *p* = (*a*, *b*)
by (*cases p*)
then show *?thesis*
using *Some mima* ‹*length treeList* = *2* ^ *m*› ‹*deg* > *1*› ‹*high x* (*deg div 2*) < *2^m*›
using *membermima.simps*(*5*)[*of deg*−*1 treeList summary x*]
by (*metis One-nat-def Suc-diff-1 Suc-lessD VEBT-internal.membermima.simps*(*4*))
qed
then show *?thesis*
unfolding *both-member-options-def* **..**
next
case *naive*
then have *naive-member* (*Node info deg treeList summary*) *x*
using ‹*length treeList* = *2* ^ *m*› ‹*deg* > *1*› ‹*high x* (*deg div 2*) < *2^m*›
using *less-natE* **by** *fastforce*
then show *?thesis*
unfolding *both-member-options-def* **..**

qed
qed

lemma *exp-split-high-low*: **assumes** $x < 2\hat{}(n+m)$ **and** $n > 0$ **and** $m > 0$
shows *high x n* $< 2\hat{}m$ **and** *low x n* $< 2\hat{}n$
using *assms* **by** (*simp-all add*: *high-bound-aux low-def*)

lemma *low-inv*: **assumes** $x < 2\hat{}n$ **shows** *low* $(y*2\hat{}n + x)$ $n = x$ **unfolding** *low-def*
by (*simp add*: *assms*)

lemma *high-inv*: **assumes** $x < 2\hat{}n$ **shows** *high* $(y*2\hat{}n + x)$ $n = y$ **unfolding** *high-def*
by (*simp add*: *assms*)

lemma *both-member-options-from-chilf-to-complete-tree*:
assumes *high x* (*deg div 2*) < *length treeList* **and** *deg* ≥ 1 **and** *both-member-options* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))
shows *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
proof−
have *membermima* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) ∨
naive-member (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) **using** *assms*
using *both-member-options-def* **by** *blast*
moreover have *membermima* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) ⟹
membermima (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
using *membermima.simps*(*4*)[*of mi ma deg*−*1 treeList summary x*]
by (*metis Suc-1 Suc-leD assms*(*1*) *assms*(*2*) *le-add-diff-inverse plus-1-eq-Suc*)
moreover have *naive-member* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) ⟹
naive-member (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
using *naive-member.simps*(*3*)[*of Some* (*mi*, *ma*) *deg*−*1 treeList summary x*]
by (*metis Suc-1 Suc-leD assms*(*1*) *assms*(*2*) *le-add-diff-inverse plus-1-eq-Suc*)
ultimately show *?thesis*
using *both-member-options-def* **by** *blast*
qed

lemma *both-member-options-from-complete-tree-to-child*:
assumes *deg* ≥ 1 **and** *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
shows *both-member-options* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*)) ∨ $x = mi$ ∨ $x = ma$
proof−
have *naive-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ∨
membermima (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
using *assms*(*2*) *both-member-options-def* **by** *auto*
moreover have *naive-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
⟹ *naive-member* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))
using *naive-member.simps*(*3*)[*of Some* (*mi*, *ma*) *deg*−*1 treeList summary x*]
by (*metis assms*(*1*) *le-add-diff-inverse plus-1-eq-Suc*)
moreover have *membermima* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
⟹ *membermima* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))∨ $x = mi$ ∨ $x = ma$
by (*metis VEBT-internal.membermima.simps*(*4*)[*of mi ma deg* − *1 treeList summary x*] *assms*(*1*)

```
      le-add-diff-inverse[of 1 deg] plus-1-eq-Suc)
  ultimately show ?thesis
    using both-member-options-def by presburger
qed

lemma pow-sum: (divide::nat ⇒ nat ⇒ nat) ((2::nat) ^((a::nat)+(b::nat))) (2^a) = 2^b
  by (induction a) simp+

fun elim-dead::VEBT ⇒ enat ⇒ VEBT where
elim-dead (Leaf a b) - = Leaf a b |
elim-dead (Node info deg treeList summary) ∞ =
 (Node info deg (map (λ t. elim-dead t (enat (2^(deg div 2)))) treeList)
 (elim-dead summary ∞))|
elim-dead  (Node info deg treeList summary) (enat l) =
 (Node info deg (take (l div (2^(deg div 2))) (map (λ t. elim-dead t (enat (2^(deg div 2))))treeList))
                  (elim-dead summary ((enat (l div (2^(deg div 2)))))))

lemma elimnum: invar-vebt (Node info deg treeList summary) n ⟹
    elim-dead (Node info deg treeList summary) (enat ((2::nat)^n)) = (Node info deg treeList summary)
proof(induction rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
    by simp
next
  case (2 treeList n summary m deg)
  have a:i < 2^m ⟶ (elim-dead (treeList ! i) (enat( 2^n)) = treeList ! i) for i
  proof
    assume  i < 2^m
    hence treeList ! i ∈ set treeList
      by (simp add: 2.hyps(2))
    thus elim-dead (treeList ! i) (enat (2 ^ n)) = treeList ! i
      using 2.IH(1) by blast
  qed
  hence b:map (λ t. elim-dead t (enat (2 ^ n))) treeList = treeList
    by (simp add: 2.IH(1) map-idI)
  have deg div 2 = n
    by (simp add: 2.hyps(3) 2.hyps(4))
  hence (2^m ::nat) = ( (2^deg) div (2^(deg div 2))::nat)
    using 2.hyps(4) pow-sum by metis
  hence take (2^deg div (2^(deg div 2)))(map (λ t. elim-dead t (enat (2 ^ n))) treeList) = treeList
    using b 2(4) by simp
  moreover hence  ( elim-dead summary ((enat ((2^deg) div (2^(deg div 2)))))) = summary
    using 2.IH(2)
    by (metis ‹2 ^ m = 2 ^ deg div 2 ^ (deg div 2)›)
  ultimately show ?case using  elim-dead.simps(3)[of None deg treeList summary 2^deg]
    using ‹deg div 2 = n› by metis
next
  case (3 treeList n summary m deg)
  have a:i < 2^m ⟶ (elim-dead (treeList ! i) (enat( 2^n)) = treeList ! i) for i
```

```
  proof
    assume  i < 2^m
    hence treeList ! i ∈ set treeList
      by (simp add: 3.hyps(2))
    thus elim-dead (treeList ! i) (enat (2 ^ n)) = treeList ! i
      using 3.IH(1) by blast
  qed
  hence b:map (λ t. elim-dead t (enat (2 ^ n))) treeList = treeList
    by (simp add: 3.IH(1) map-idI)
  have deg div 2 = n
    by (simp add: 3.hyps(3) 3.hyps(4))
  hence (2^m ::nat) = ( (2^deg) div (2^(deg div 2))::nat)
    using 3.hyps(4) pow-sum by metis
  hence take (2^deg div (2^(deg div 2)))(map (λ t. elim-dead t (enat (2 ^ n))) treeList) = treeList
    using b 3(4) by simp
  moreover hence  ( elim-dead summary ((enat ((2^deg) div (2^(deg div 2))))))) = summary using
3.IH(2)
    by (metis ‹2 ^ m = 2 ^ deg div 2 ^ (deg div 2)›)
  ultimately show ?case using  elim-dead.simps(3)[of None deg treeList summary 2^deg]
    using ‹deg div 2 = n› by metis
next
  case (4 treeList n summary m deg mi ma)
  have a:i < 2^m ⟶ (elim-dead (treeList ! i) (enat( 2^n)) = treeList ! i) for i
  proof
    assume  i < 2^m
    hence treeList ! i ∈ set treeList
      by (simp add: 4.hyps(2))
    thus elim-dead (treeList ! i) (enat (2 ^ n)) = treeList ! i
      using 4.IH(1) by blast
  qed
  hence b:map (λ t. elim-dead t (enat (2 ^ n))) treeList = treeList
    by (simp add: 4.IH(1) map-idI)
  have deg div 2 = n
    by (simp add: 4.hyps(3) 4.hyps(4))
  hence (2^m ::nat) = ( (2^deg) div (2^(deg div 2))::nat)
    using 4.hyps(4) pow-sum by metis
  hence take (2^deg div (2^(deg div 2)))(map (λ t. elim-dead t (enat (2 ^ n))) treeList) = treeList
    using b 4(4) by simp
  moreover hence  ( elim-dead summary ((enat ((2^deg) div (2^(deg div 2))))))) = summary using
4.IH(2)
    by (metis ‹2 ^ m = 2 ^ deg div 2 ^ (deg div 2)›)
  ultimately show ?case using  elim-dead.simps(3)[of Some (mi, ma) deg treeList summary 2^deg]
    using ‹deg div 2 = n› by metis
next
  case (5 treeList n summary m deg mi ma)
  have a:i < 2^m ⟶ (elim-dead (treeList ! i) (enat( 2^n)) = treeList ! i) for i
  proof
    assume  i < 2^m
    hence treeList ! i ∈ set treeList
```

```
    by (simp add: 5.hyps(2))
  thus elim-dead (treeList ! i) (enat (2 ^ n)) = treeList ! i
    using 5.IH(1) by blast
 qed
 hence b:map (λ t. elim-dead t (enat (2 ^ n))) treeList = treeList
  by (simp add: 5.IH(1) map-idI)
 have deg div 2 = n
  by (simp add: 5.hyps(3) 5.hyps(4))
 hence (2^m ::nat) = ( (2^deg) div (2^(deg div 2))::nat)
  using 5.hyps(4) pow-sum by metis
 hence take (2^deg div (2^(deg div 2)))(map (λ t. elim-dead t (enat (2 ^ n))) treeList) = treeList
  using b 5(4) by simp
 moreover hence ( elim-dead summary ((enat ((2^deg) div (2^(deg div 2)))))) = summary using
5.IH(2)
  by (metis ‹2 ^ m = 2 ^ deg div 2 ^ (deg div 2)›)
 ultimately show ?case using elim-dead.simps(3)[of Some (mi, ma) deg treeList summary 2^deg]
  using ‹deg div 2 = n› by metis
qed

lemma elimcomplete:
 assumes invar-vebt (Node info deg treeList summary) n
 shows elim-dead (Node info deg treeList summary) ∞ = (Node info deg treeList summary)
proof −
 have elim-dead (Node info deg treeList summary) ∞ = Node info deg treeList summary
  if deg = n + m and
   m = n ∨ m = Suc n and
   length treeList = 2 ^ m and
   ball-invar-vebt: ∀ t∈set treeList. invar-vebt t n and
   elim-dead summary ∞ = summary
  for m n :: nat and info and deg and treeList and summary
 proof −
  have elim-dead x (enat (2^n)) = x
   if invar-vebt x n for x n
  proof (cases x rule: VEBT.exhaust)
   case (Node x11 x12 x13 x14)
   then show ?thesis
    using elimnum
    by (metis ‹invar-vebt x n›)
  next
   case (Leaf x21 x22)
   then show ?thesis
    by simp
  qed

  then have b: map (λ t. elim-dead t (enat (2 ^ n))) treeList = treeList
   using ball-invar-vebt
   by (simp add: list.map-ident-strong)

  have deg div 2 = n
```

```
    using ‹deg = n + m› ‹m = n ∨ m = Suc n› by fastforce

  hence 2 ^ m = ((2^deg) div (2^(deg div 2))::nat)
    by (simp add: ‹deg = n + m› pow-sum)

  hence take (2^deg div (2^(deg div 2)))(map (λ t. elim-dead t (enat (2 ^ n))) treeList) = treeList
    by (simp add: ‹length treeList = 2 ^ m› b)

  then show ?thesis
    using ‹deg div 2 = n› b that(5)
    using elim-dead.simps(2)[of None  deg treeList summary]
    by simp
 qed

 with assms show ?thesis
 proof(induction rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
    by simp
 qed metis+
qed

end
end

theory VEBT-Member imports VEBT-Definitions
begin
```

2 Member Function

```
context begin
interpretation VEBT-internal .

fun vebt-member :: VEBT ⇒ nat ⇒ bool where
 vebt-member (Leaf a b) x = (if x = 0 then a else if x = 1 then b else False)|
 vebt-member (Node None - - -) x = False|
 vebt-member (Node - 0 - -) x = False|
 vebt-member (Node - (Suc 0) - -) x = False|
 vebt-member (Node (Some (mi, ma)) deg treeList summary) x = (
  if x = mi then True else
  if x = ma then True else
  if x < mi then False else
  if x > ma then False else (
   let h = high x (deg div 2);
       l = low x (deg div 2) in(
        if h < length treeList
        then vebt-member (treeList ! h) l
        else False)))
```

end

context *VEBT-internal* **begin**

lemma *member-inv*:
assumes *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
shows *deg* ≥ *2* ∧
(*x* = *mi* ∨ *x* = *ma* ∨ (*x* < *ma* ∧ *x* > *mi* ∧ *high x* (*deg div 2*) < *length treeList* ∧
vebt-member (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))))
proof(*cases deg*)
case *0*
then show *?thesis* **using** *vebt-member.simps*(*3*)[*of* (*mi*, *ma*) *treeList summary x*]
using *assms* **by** *blast*
next
case (*Suc nat*)
hence *deg* = *Suc nat* **by** *simp*
then show *?thesis* **proof**(*cases nat*)
case *0*
then show *?thesis*
using *Suc assms* **by** *auto*
next
case (*Suc nata*)
hence *deg* ≥ *2*
by (*simp add*: ‹*deg* = *Suc nat*›)
then show *?thesis*
by (*metis vebt-member.simps*(*5*) *Suc* ‹*deg* = *Suc nat*› *assms linorder-neqE-nat*)
qed
qed

definition *bit-concat*::*nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
bit-concat h l d = *h*∗*2*^*d* + *l*

lemma *bit-split-inv*: *bit-concat* (*high x d*) (*low x d*) *d* = *x*
unfolding *bit-concat-def high-def low-def*
by *presburger*

definition *set-vebt'*::*VEBT* ⇒ *nat set* **where**
set-vebt' t = {*x*. *vebt-member t x*}

lemma *Leaf-0-not*: **assumes** *invar-vebt* (*Leaf a b*) *0* **shows** *False*
proof−
from *assms* **show** *?thesis*
proof(*cases*)
qed
qed

lemma *valid-0-not*: *invar-vebt t 0* ⟹ *False*
proof(*induction t*)

```
  case (Node info deg treeList summary)
  from this(3) have  length treeList > 0
    apply cases
    apply auto
    done
  then obtain t where t ∈ set treeList by fastforce
  from Node(3) obtain n where invar-vebt t n
    apply cases
    using Node.IH(2) apply auto
    done
  from  Node(3) have  n ≤ 0
    apply cases
    using Node.IH(2) apply auto
    done
  hence n = 0 by blast
  then show ?case
    using Node.IH(1) ‹t ∈ set treeList› ‹invar-vebt t n› by blast
next
  case (Leaf x1 x2)
  then show ?case
    using Leaf-0-not by blast
qed

theorem valid-tree-deg-neq-0: (¬ invar-vebt t 0)
  using valid-0-not by blast

lemma deg-1-Leafy:  invar-vebt t n ⟹ n = 1 ⟹ ∃ a b. t = Leaf a b
  apply(induction rule: invar-vebt.induct)
  apply simp
  apply presburger
 apply (metis (full-types) Suc-eq-plus1 add-cancel-right-left in-set-replicate list.map-cong0 map-replicate-const
nat-neq-iff not-add-less2 numeral-1-eq-Suc-0 numeral-2-eq-2 numerals(1) order-less-irrefl power-eq-0-iff
valid-tree-deg-neq-0 zero-less-numeral)
  apply (metis odd-add odd-one)
 by (metis Suc-eq-plus1 add-cancel-right-left in-set-replicate list.map-cong0 map-replicate-const nat-neq-iff
not-add-less2 numeral-2-eq-2 power-eq-0-iff valid-tree-deg-neq-0)

lemma deg-1-Leaf: invar-vebt t 1 ⟹  ∃ a b. t = Leaf a b
  using deg-1-Leafy by blast

corollary deg1Leaf: invar-vebt t 1 ⟷  (∃ a b. t = Leaf a b)
  using deg-1-Leaf invar-vebt.intros(1) by auto

lemma deg-SUcn-Node: assumes invar-vebt tree (Suc (Suc n))  shows
              ∃ info treeList s. tree = Node info (Suc (Suc n)) treeList s
proof−
  from assms show ?thesis apply(cases)
    apply blast+
   done
```

qed

lemma *invar-vebt* (*Node info deg treeList summary*) *deg* $\Longrightarrow$ *deg* > *1*
by (*metis VEBT.simps*(*4*) *deg-1-Leafy less-one linorder-neqE-nat valid-tree-deg-neq-0*)

lemma *deg-deg-n*: **assumes** *invar-vebt* (*Node info deg treeList summary*) *n* **shows** *deg* = *n*
proof−
from *assms* **show** *?thesis* **proof**(*cases*)
qed *blast*+
qed

lemma *member-valid-both-member-options*:
invar-vebt tree n $\Longrightarrow$ *vebt-member tree x* $\Longrightarrow$ (*naive-member tree x* $\vee$ *membermima tree x*)
proof(*induction tree n arbitrary*: *x rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case*
using *vebt-member.simps*(*1*) *naive-member.simps*(*1*) **by** *blast*
next
case (*2 treeList n summary m deg*)
then show *?case* **by** *simp*
next
case (*3 treeList n summary m deg*)
then show *?case*
using *vebt-member.simps*(*2*) **by** *blast*
next
case (*4 treeList n summary m deg mi ma*)
hence *deg* ≥ *2*
using *member-inv* **by** *blast*
then show *?case* **proof**(*cases x* = *mi* $\vee$ *x* = *ma*)
case *True*
then show *?thesis*
by (*metis* (*full-types*) *4*(*12*) *vebt-member.simps*(*3*) *membermima.simps*(*4*) *old.nat.exhaust*)
next
case *False*
hence *1*:*mi* < *x* ∧ *x* < *ma* ∧ (*high x* (*deg div 2*)) < *length treeList* ∧ *vebt-member* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))
using *member-inv*[*of mi ma deg treeList summary x*] *4*(*12*) **by** *blast*
hence (*treeList* ! (*high x* (*deg div 2*))) ∈ *set treeList*
by *simp*
hence *both-member-options* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))
using *1 4*.*IH*(*1*) *both-member-options-def* **by** *blast*
then show *?thesis*
by (*metis 4*.*prems False Suc-1 Suc-leD both-member-options-def*
both-member-options-from-chilf-to-complete-tree member-inv)
qed
next
case (*5 treeList n summary m deg mi ma*)
hence *deg* ≥ *2*
using *member-inv* **by** *presburger*

```
  then show ?case proof(cases x = mi ∨ x = ma)
    case True
    then show ?thesis
      by (metis (full-types) 5(12) vebt-member.simps(3) membermima.simps(4) old.nat.exhaust)
  next
    case False
     hence 1:mi < x ∧ x < ma ∧ (high x (deg div 2 )) < length treeList ∧ vebt-member (treeList !
(high x (deg div 2))) (low x (deg div 2))
      using member-inv[of mi ma deg treeList summary x] 5(12) by blast
    hence  (treeList ! (high x (deg div 2))) ∈ set treeList
      by simp
    hence both-member-options  (treeList ! (high x (deg div 2))) (low x (deg div 2))
      using 1 5.IH(1) both-member-options-def by blast
    then show ?thesis
      by (metis 5.prems False Suc-1 Suc-leD both-member-options-def
          both-member-options-from-chilf-to-complete-tree member-inv)
  qed
qed

lemma  member-bound: vebt-member tree x ⟹ invar-vebt tree n  ⟹ x < 2^n
proof(induction tree x arbitrary: n rule: vebt-member.induct)
  case (1 a b x)
 then show ?case  by (metis vebt-member.simps(1) One-nat-def le-neq-implies-less nat-power-eq-Suc-0-iff

                              numeral-eq-iff numerals(1) one-le-numeral one-le-power semiring-norm(85)
valid-tree-deg-neq-0
                      zero-less-numeral zero-less-power)
next
  case (2 uu uv uw x)
  then show ?case by simp
next
  case (3 v uy uz x)
  then show ?case by simp
next
  case (4 v vb vc x)
  then show ?case by simp
next
  case (5 mi ma va treeList summary x)
  hence 111: n =  Suc (Suc va)
    using deg-deg-n by fastforce
  hence ma < 2^n
    using 5.prems(2) mi-ma-2-deg by blast
  then show ?case
    by (metis 5.prems(1) 5.prems(2) le-less-trans less-imp-le-nat member-inv mi-ma-2-deg)
qed

theorem inrange: assumes invar-vebt t n shows  set-vebt' t ⊆ {0..2^n−1}
proof
  fix x
```

assume $x \in$ *set-vebt'* t
hence *vebt-member* t x
using *set-vebt'-def* **by** *auto*
hence $x < 2\hat{}n$
using *assms member-bound* **by** *blast*
then show $x \in \{0..2\hat{}n-1\}$ **by** *simp*
qed

theorem *buildup-gives-empty*: *set-vebt'* (*vebt-buildup* n) = {}
unfolding *set-vebt'-def*
by (*metis Collect-empty-eq vebt-member.simps*(1) *vebt-member.simps*(2) *vebt-buildup.elims*)

fun *minNull*::*VEBT* ⇒ *bool* **where**
minNull (*Leaf False False*) = *True*|
minNull (*Leaf* - -) = *False*|
minNull (*Node None* - - -) = *True*|
minNull (*Node* (*Some* -) - - -) = *False*

lemma *min-Null-member*: *minNull* $t \Longrightarrow \neg$ *vebt-member* t x
apply(*induction* t)
using *vebt-member.simps*(2) *minNull.elims*(2) **apply** *blast*
apply *auto*
done

lemma *not-min-Null-member*: ¬ *minNull* $t \Longrightarrow \exists$ *y. both-member-options* t *y*
proof(*induction* t)
case (*Node info deg treeList summary*)
obtain *mi ma* **where** *info* = *Some*(*mi* , *ma*)
by (*metis Node.prems minNull.simps*(4) *not-None-eq surj-pair*)
then show *?case*
by (*metis* (*full-types*) *both-member-options-def membermima.simps*(3) *membermima.simps*(4) *not0-implies-Suc*)
next
case (*Leaf x1 x2*)
then show *?case*
by (*metis* (*full-types*) *both-member-options-def minNull.simps*(1) *naive-member.simps*(1) *zero-neq-one*)
qed

lemma *valid-member-both-member-options*: *invar-vebt* t $n \Longrightarrow$ *both-member-options* t $x \Longrightarrow$ *vebt-member* t x
proof(*induction* t n *arbitrary*: x *rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case*
by (*simp add*: *both-member-options-def*)
next
case (*2 treeList n summary m deg*)
hence *0*: (∀ $t \in$ *set treeList. invar-vebt* t n) **and** *1*: *invar-vebt summary n* **and** *2*: *length treeList* = $2\hat{}n$ **and**
3: *deg* = $2*n$ **and** *4*: (∄ *i. both-member-options summary i*) **and** *5*: (∀ $t \in$ *set treeList.* ∄ *y. both-member-options* t *y*) **and** *6*: $n > 0$

apply *blast+*
apply (*auto simp add*: *2.hyps*(*3*) *2.hyps*)
using *2.hyps*(*1*) *2.hyps*(*3*) *neq0-conv valid-tree-deg-neq-0* **by** *blast*
have *both-member-options* (*Node None deg treeList summary*) *x* ⟹ *False*
proof−
assume *both-member-options* (*Node None deg treeList summary*) *x*
hence *naive-member* (*Node None deg treeList summary*) *x* ∨ *membermima* (*Node None deg treeList summary*) *x* **unfolding** *both-member-options-def* **by** *simp*
then show *False*
proof(*cases naive-member* (*Node None deg treeList summary*) *x*)
case *True*
hence *high x n* < *length treeList* ∧ *naive-member* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis 1 2.hyps*(*3*) *2.hyps*(*4*) *add-cancel-right-left add-self-div-2 naive-member.simps*(*3*) *old.nat.exhaust valid-tree-deg-neq-0*)
then show *?thesis*
using *5* **by** (*auto simp add*: *both-member-options-def*)
next
case *False*
hence *membermima* (*Node None deg treeList summary*) *x*
using ‹*naive-member* (*Node None deg treeList summary*) *x* ∨ *membermima* (*Node None deg treeList summary*) *x*› **by** *blast*
moreover have *Suc* (*deg*−*1*) =*deg*
by (*simp add*: *2.hyps*(*4*) *6*)
moreover hence (*let pos* = *high x* (*deg div 2*) *in if pos* < *length treeList then membermima* (*treeList* ! *pos*) (*low x* (*Suc* (*deg* − *1*) *div 2*)) *else False*)
using *calculation*(*1*) *membermima.simps*(*5*) **by** *metis*
moreover hence *if high x* (*deg div 2*) < *length treeList then membermima* (*treeList* ! (*high x* (*deg div 2*))) (*low x*(*deg div 2*)) *else False*
using *calculation*(*2*) **by** *metis*
ultimately
have *high x* (*deg div 2*) < *length treeList* ∧ *membermima* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis 2.hyps*(*3*) *2.hyps*(*4*) *add-self-div-2*)
then show *?thesis* **using** *5*
by (*auto simp add*: *both-member-options-def 2.hyps*)
qed
qed
then show *?case*
by (*simp add*: *2.prems*)
next
case (*3 treeList n summary m deg*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary m* **and** *2*: *length treeList* = *2*^*m* **and**
3: *deg* = *n*+*m* **and** *4*: (∄ *i*. *both-member-options summary i*) **and** *5*: (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*) **and** *6*: *n*> *0* **and** *7*: *m*> *0*
and *8*: *n*+*1* = *m*
using *3.hyps 3.IH*
apply *blast+*
apply (*simp-all add*: *3.hyps*(*3*))
using *3.IH*(*1*) *3.hyps*(*2*) *set-n-deg-not-0 zero-less-iff-neq-zero* **apply** *blast*

done
have *both-member-options* (*Node None deg treeList summary*) *x* ⟹ *False*
proof−
assume *both-member-options* (*Node None deg treeList summary*) *x*
hence *naive-member* (*Node None deg treeList summary*) *x* ∨ *membermima* (*Node None deg treeList summary*) *x* **unfolding** *both-member-options-def* **by** *simp*
then show *False*
proof(*cases naive-member* (*Node None deg treeList summary*) *x*)
case *True*
hence *high x n* < *length treeList* ∧ *naive-member* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis 3 3.hyps*(*3*) *add-Suc-right add-self-div-2 even-Suc-div-two naive-member.simps*(*3*) *odd-add*)
then show *?thesis*
using *5* **by** (*auto simp add*: *both-member-options-def*)
next
case *False*
hence *membermima* (*Node None deg treeList summary*) *x*
using ‹*naive-member* (*Node None deg treeList summary*) *x* ∨ *membermima* (*Node None deg treeList summary*) *x*› **by** *blast*
moreover have *Suc* (*deg*−*1*) =*deg*
by (*simp add*: *3 3.hyps*(*3*))
moreover hence (*let pos* = *high x* (*deg div 2*) *in if pos* < *length treeList then membermima* (*treeList* ! *pos*) (*low x* (*Suc* (*deg* − *1*) *div 2*)) *else False*)
using *calculation*(*1*) *membermima.simps*(*5*) **by** *metis*
moreover hence *11*: *if high x* (*deg div 2*) < *length treeList then membermima* (*treeList* ! (*high x* (*deg div 2*))) (*low x*(*deg div 2*)) *else False*
using *calculation*(*2*) **by** *metis*
ultimately
have *high x* (*deg div 2*) < *length treeList* ∧ *membermima* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis 3 3.hyps*(*3*) *add-Suc-right add-self-div-2 even-Suc-div-two odd-add*)
then show *?thesis* **using** *3.IH 5 both-member-options-def inthall 11* **by** *auto*
qed
qed
then show *?case*
using *3.prems* **by** *blast*
next
case (*4 treeList n summary m deg mi ma*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary n* **and** *2*:*length treeList* = *2*^*n* **and** *3*: *deg* = *n*+*m* **and** *n*=*m* **and**
4: (∀ *i* < *2*^*n*. (∃ *y*. *both-member-options* (*treeList* ! *i*) *y*) ⟷ (*both-member-options summary i*)) **and**
5: (*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)) **and** *6*:*mi* ≤ *ma* ∧ *ma* < *2*^*deg* **and**
7: (*mi* ≠ *ma* ⟶ (∀ *i* < *2*^*m*. (*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *ma*)))
using *4.prems* **by** *auto*
hence *n*>*0*

by (*metis neq0-conv valid-tree-deg-neq-0*)
then show *?case* **proof**(*cases* $x = mi \vee x = ma$)
case *True*
hence *xmimastmt*: $x = mi \vee x{=}ma$ **by** *simp*
then show *?thesis* **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary x*]
by (*metis 3 4.hyps*(*3*) ‹*0* < *n*› *add-diff-inverse-nat add-numeral-left add-self-div-2 div-if nat-neq-iff numerals*(*1*) *plus-1-eq-Suc semiring-norm*(*2*))
next
case *False*
hence *xmimastmt*: $x \neq mi \wedge x{\neq}ma$ **by** *simp*
hence $mi = ma \Longrightarrow False$
proof−
assume $mi = ma$
hence *astmt*: ($\forall$ $t \in$ *set treeList*. $\nexists$ *y*. *both-member-options t y*) **using** *5* **by** *simp*
have *bstmt*: *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
by (*simp add*: *4.prems*)
then show *False*
proof(*cases* *naive-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*)
case *True*
hence *high x n* < *length treeList* $\wedge$ *naive-member* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis* (*no-types*, *opaque-lifting*) *3 4.hyps*(*1*) *4.hyps*(*3*) *add-self-div-2 naive-member.simps*(*3*) *old.nat.exhaust valid-0-not zero-eq-add-iff-both-eq-0*)
then show *?thesis*
using *5* **by** (*auto simp add*: ‹*mi* = *ma*› *both-member-options-def*)
next
case *False*
hence *membermima* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* **using** *bstmt* **unfolding** *both-member-options-def* **by** *blast*
hence *x* = *mi* $\vee$ *x* = *ma* $\vee$ (*if high x n* < *length treeList then* *membermima* (*treeList* ! (*high x n*)) (*low x n*) *else False*)
using *membermima.simps*(*4*)[*of mi ma deg*−*1 treeList summary x*]
by (*metis 3 4.hyps*(*3*) *One-nat-def Suc-diff-Suc* ‹*0* < *n*› *add-gr-0 add-self-div-2 diff-zero*)
hence *high x n* < *length treeList* $\wedge$ *membermima* (*treeList* ! (*high x n*)) (*low x n*) **using** *xmimastmt*
by *presburger*
then show *?thesis* **using** *both-member-options-def inthall membermima.simps*(*4*)[*of mi ma n treeList summary x*] *astmt*
by *auto*
qed
qed
hence $mi \neq ma$ **by** *blast*
hence *followstmt*: ($\forall$ *i* < 2^*m*. (*high ma n* = *i* $\longrightarrow$ *both-member-options* (*treeList* ! *i*) (*low ma n*)) $\wedge$
($\forall$ *y*. (*high y n* = *i* $\wedge$ *both-member-options* (*treeList* ! *i*) (*low y n*)) $\longrightarrow$ *mi* < *y* $\wedge$ *y* $\leq$ *ma*))
using *7* **by** *simp*
have *10*:
high x n < *length treeList* (**is** *?LESS-THAN*)
naive-member (*treeList* ! (*high x n*)) (*low x n*) $\vee$ *membermima* (*treeList* ! (*high x n*)) (*low x n*)

(**is** *?NAIVE-OR-MIMA*)
proof −
show *?LESS-THAN*
by (*metis 3 4.hyps*(*3*) *4.prems False VEBT-internal.membermima.simps*(*4*)
VEBT-internal.naive-member.simps(*3*) ‹*0* < *n*› *add-eq-0-iff-both-eq-0 add-self-div-2*
bot-nat-0.not-eq-extremum both-member-options-def not0-implies-Suc)
show *?NAIVE-OR-MIMA*
by (*metis 3 4.hyps*(*3*) *4.prems False* ‹*0* < *n*› *add-self-div-2 both-member-options-def*
both-member-options-from-complete-tree-to-child less-add-same-cancel1 less-natE nat-le-iff-add
plus-1-eq-Suc)
qed
hence *11*:*both-member-options* (*treeList* ! (*high x n*)) (*low x n*)
by (*simp add*: *both-member-options-def*)
have *12*:*high x n*< *2*^*m*
using *10 4.hyps*(*2*) **by** *auto*
hence *mi* < *x* ∧ *x* < *ma* **proof**−
have (∀ *y*. (*high y n* = (*high x n*) ∧ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)) ⟶
mi < *y* ∧ *y* ≤ *ma*)
using *12 followstmt* **by** *auto*
then show *?thesis*
using *11 False order.not-eq-order-implies-strict* **by** *blast*
qed
obtain *n′* **where** *n* = *Suc n′*
using ‹*0* < *n*›
by (*metis less-natE*)
then have *deg* = *Suc* (*Suc* (*n′* + *n′*))
using ‹*deg* = *n* + *m*› ‹*m* = *n*›
by *presburger*
have *vebt-member* (*treeList* ! (*high x n*)) (*low x n*)
using *10 11* **by** (*simp add*: *4.IH*)
then show *?thesis*
unfolding ‹*deg* = *Suc* (*Suc* (*n′* + *n′*))› *vebt-member.simps*
using *10*(*1*) ‹*mi* < *x* ∧ *x* < *ma*› ‹*n* = *Suc n′*› **by** *auto*
qed
next
case (*5 treeList n summary m deg mi ma*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary m* **and** *2*:*length treeList*
= *2*^*m* **and** *3*: *deg* = *n*+*m* **and** *Suc n*=*m* **and**
4: (∀ *i* < *2*^*m*. (∃ *y*. *both-member-options* (*treeList* ! *i*) *y*) ⟷ (*both-member-options summary*
i)) **and**
5: (*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)) **and** *6*:*mi* ≤ *ma* ∧ *ma* <
2^*deg* **and**
7: (*mi* ≠ *ma* ⟶ (∀ *i* < *2*^*m*. (*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*))
∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low y n*))
⟶ *mi* < *y* ∧ *y* ≤ *ma*)))
using *5.prems* **by** *auto*
from *0 1 2 3 4 5 6 7* ‹*m* = *Suc n*› *valid-tree-deg-neq-0* [*of summary*] *set-n-deg-not-0* [*of treeList*]
have *n*>*0*

```
    apply simp
    apply (metis 2 gr0I not-one-le-zero set-n-deg-not-0)
    done
  then show ?case proof(cases x = mi ∨ x = ma)
    case True
    hence xmimastmt: x = mi ∨ x=ma by simp
    then show ?thesis using vebt-member.simps(5)[of mi ma deg−2 treeList summary x]
      using 3 5.hyps(3) ‹0 < n› by auto
  next
    case False
    hence xmimastmt: x ≠ mi ∧ x≠ma by simp
    hence mi = ma ⟹ False
    proof−
      assume mi = ma
      hence astmt: (∀ t ∈ set treeList. ∄ y. both-member-options t y) using 5 by simp
      have bstmt: both-member-options (Node (Some (mi, ma)) deg treeList summary) x
        by (simp add: 5.prems)
      then show False
      proof(cases naive-member (Node (Some (mi, ma)) deg treeList summary) x)
        case True
        hence high x n < length treeList ∧ naive-member (treeList ! (high x n)) (low x n)
          by (metis 3 5.hyps(3) add-Suc-right add-self-div-2 even-Suc-div-two naive-member.simps(3)
odd-add)
        then show ?thesis
          using 5 ‹mi = ma› by (auto simp add: both-member-options-def)
      next
        case False
       hence membermima (Node (Some (mi, ma)) deg treeList summary) x using bstmt unfolding
both-member-options-def by blast
        hence x = mi ∨ x = ma ∨ (if high x n < length treeList then membermima (treeList ! (high
x n)) (low x n) else False)
          using membermima.simps(4)[of mi ma deg−1 treeList summary x]
          using 3 5.hyps(3) by auto
          hence high x n < length treeList ∧ membermima (treeList ! (high x n)) (low x n) using
xmimastmt
          by presburger
        then show ?thesis using both-member-options-def inthall membermima.simps(4)[of mi ma n
treeList summary x] astmt
          by auto
      qed
    qed
    hence mi ≠ ma by blast
   hence followstmt: (∀ i < 2^m. (high ma n = i ⟶ both-member-options (treeList ! i) (low ma n))
∧
                                    (∀ y. (high y n = i ∧ both-member-options (treeList ! i) (low y n) )
⟶ mi < y ∧ y ≤ ma))
      using 7 by simp
    have 10:
      high x n < length treeList (is ?LESS-THAN)
```

naive-member (*treeList* ! (*high x n*)) (*low x n*) ∨ *membermima* (*treeList* ! (*high x n*)) (*low x n*)
(**is** *?NAIVE-OR-MIMA*)
proof −
show *?LESS-THAN*
using *3 5.hyps*(*3*) *5.prems False both-member-options-def* **by** *force*
show *?NAIVE-OR-MIMA*
by (*metis 3 5.hyps*(*3*) *5.prems False add-Suc-right add-leD1 add-self-div-2*
both-member-options-def both-member-options-from-complete-tree-to-child even-Suc-div-two
even-add
le-add2 plus-1-eq-Suc)
qed
hence *11*:*both-member-options* (*treeList* ! (*high x n*)) (*low x n*)
by (*simp add*: *both-member-options-def*)
have *12*:*high x n*< *2*^*m*
using *10 5.hyps*(*2*) **by** *auto*
hence *mi* < *x* ∧ *x* < *ma* **proof**−
have (∀ *y*. (*high y n* = (*high x n*) ∧ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)) ⟶
mi < *y* ∧ *y* ≤ *ma*)
using *12 followstmt* **by** *auto*
then show *?thesis*
using *11 False order.not-eq-order-implies-strict* **by** *blast*
qed
obtain *n′* **where** *n* = *Suc n′*
using ‹*0* < *n*›
by (*metis less-natE*)
then have *deg* = *Suc* (*Suc* (*Suc* (*n′* + *n′*)))
using ‹*deg* = *n* + *m*› ‹*m* = *Suc n*›
by *presburger*
have *vebt-member* (*treeList* ! (*high x n*)) (*low x n*)
using *10 11 5.IH* **by** *simp*
then show *?thesis*
unfolding ‹*deg* = *Suc* (*Suc* (*Suc* (*n′* + *n′*)))› *vebt-member.simps*
using *10*(*1*) ‹*mi* < *x* ∧ *x* < *ma*› ‹*n* = *Suc n′*› **by** *auto*
qed
qed

corollary *both-member-options-equiv-member*: **assumes** *invar-vebt t n*
shows *both-member-options t x* ⟷ *vebt-member t x*
using *assms both-member-options-def member-valid-both-member-options valid-member-both-member-options*
by *blast*

lemma *member-correct*: *invar-vebt t n* ⟹ *vebt-member t x* ⟷ *x* ∈ *set-vebt t*
using *both-member-options-equiv-member set-vebt-def* **by** *auto*

corollary *set-vebt-set-vebt′-valid*: **assumes** *invar-vebt t n* **shows** *set-vebt t* =*set-vebt′ t*
unfolding *set-vebt-def set-vebt′-def*
apply *auto*
using *assms valid-member-both-member-options* **apply** *auto*[*1*]
using *assms both-member-options-equiv-member* **by** *auto*

lemma *set-vebt-finite*: *invar-vebt t n* $\Longrightarrow$ *finite* (*set-vebt' t*)
using *finite-subset inrange* **by** *blast*

lemma *mi-eq-ma-no-ch*:**assumes** *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg* **and** *mi* = *ma*
shows ($\forall$ *t* $\in$ *set treeList*. $\nexists$ *x*. *both-member-options t x*) $\wedge$ ($\nexists$ *x*. *both-member-options summary x*)
proof−
from *assms*(*1*) **show** *?thesis*
proof(*cases*)
case (*4 n m*)
have *0*:$\forall$ *t*$\in$*set treeList*. $\neg$ *Ex* (*both-member-options t*)
by (*simp add*: *4*(*7*) *assms*(*2*))
moreover have *both-member-options summary x* $\Longrightarrow$ *False* **for** *x*
proof−
assume *both-member-options summary x*
hence *vebt-member summary x*
using *4*(*2*) *valid-member-both-member-options* **by** *auto*
moreover hence *x* < *2*^*m*
using *4*(*2*) *member-bound* **by** *auto*
ultimately have $\exists$ *y*. *both-member-options* (*treeList* ! (*high x n*)) *y*
using *0 4*(*3*) *4*(*4*) *4*(*6*) ‹*both-member-options summary x*› *inthall*
by (*metis nth-mem*)
then show *?thesis*
by (*metis 0 4*(*3*) *4*(*4*) *div-eq-0-iff* ‹*x* < *2* ^ *m*› *high-def nth-mem zero-less-numeral zero-less-power*)
qed
then show *?thesis*
using *calculation* **by** *blast*
next
case (*5 n m*)
have *0*:$\forall$ *t*$\in$*set treeList*. $\neg$ *Ex* (*both-member-options t*)
using *5*(*7*) *assms*(*2*) **by** *blast*
moreover have *both-member-options summary x* $\Longrightarrow$ *False* **for** *x*
proof−
assume *both-member-options summary x*
hence *vebt-member summary x*
using *5*(*2*) *valid-member-both-member-options* **by** *auto*
moreover hence *x* < *2*^*m*
using *5*(*2*) *member-bound* **by** *auto*
ultimately have $\exists$ *y*. *both-member-options* (*treeList* ! (*high x n*)) *y*
using *0 5*(*3*) *5*(*4*) *5*(*6*) ‹*both-member-options summary x*› *inthall*
by (*metis nth-mem*)
then show *?thesis*
by (*metis 0 5*(*3*) *5*(*6*) ‹*both-member-options summary x*› ‹*x* < *2* ^ *m*› *nth-mem*)
qed
then show *?thesis*
using *calculation* **by** *blast*
qed

qed

end
end

theory *VEBT-Insert* **imports** *VEBT-Member*
begin

3 Insert Function

context begin
interpretation *VEBT-internal* **.**

fun *vebt-insert* :: *VEBT* ⇒ *nat* ⇒*VEBT* **where**
vebt-insert (*Leaf a b*) *x* = (*if x=0 then Leaf True b else if x = 1 then Leaf a True else Leaf a b*)|
vebt-insert (*Node info 0 ts s*) *x* = (*Node info 0 ts s*)|
vebt-insert (*Node info* (*Suc 0*) *ts s*) *x* = (*Node info* (*Suc 0*) *ts s*)|
vebt-insert (*Node None* (*Suc deg*) *treeList summary*) *x* =
(*Node* (*Some* (*x*,*x*)) (*Suc deg*) *treeList summary*)|
vebt-insert (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = (
let xn = (*if x* < *mi then mi else x*);
minn = (*if x* < *mi then x else mi*);
l = *low xn* (*deg div 2*);
h = *high xn* (*deg div 2*) *in* (
if h < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*) *then*
Node (*Some* (*minn*, *max xn ma*)) *deg* (*treeList*[*h*:= *vebt-insert* (*treeList* ! *h*) *l*])
(*if minNull* (*treeList* ! *h*) *then vebt-insert summary h else summary*)
else (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)))

end

context *VEBT-internal* **begin**

lemma *insert-simp-norm*:
assumes *high x* (*deg div 2*) < *length treeList* **and** (*mi*::*nat*)< *x* **and** *deg*≥ *2* **and** *x* ≠ *ma*
shows *vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
Node (*Some* (*mi*, *max x ma*)) *deg* (*treeList* [(*high x* (*deg div 2*)):= *vebt-insert* (*treeList* ! (*high x* (*deg div 2*))) (*low x* (*deg div 2*))])
(*if minNull* (*treeList* ! (*high x* (*deg div 2*))) *then vebt-insert summary* (*high x* (*deg div 2*)) *else summary*)
proof−
obtain *deg*′ **where** *deg* = *Suc* (*Suc deg*′)
using ‹*2* ≤ *deg*› *add-2-eq-Suc le-Suc-ex* **by** *blast*
have *11*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
(*let xn* = (*if x* < *mi then mi else x*); *minn* = (*if x*< *mi then x else mi*);
l= *low xn* (*deg div 2*); *h* = *high xn* (*deg div 2*)
in
(*if h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
Node (*Some* (*minn*, *max xn ma*)) *deg* (*treeList* [*h*:= *vebt-insert* (*treeList* ! *h*) *l*])

```
        (if minNull (treeList ! h) then  vebt-insert summary h else summary)
      else  (Node (Some (mi, ma)) deg treeList summary)))
  unfolding ‹deg = -›
  by simp
 also have ... = Node (Some (mi, max x ma)) deg (treeList[(high x (deg div 2)) := vebt-insert
(treeList ! (high x (deg div 2))) (low x (deg div 2))])
                        (if minNull (treeList ! (high x (deg div 2))) then  vebt-insert summary (high
x (deg div 2)) else summary)
   by (smt (verit, best) assms(1,2,4) linorder-not-less nat-less-le)
 finally show ?thesis .
qed

lemma insert-simp-excp:
 assumes high mi (deg div 2) < length treeList  and  (x::nat) < mi and deg≥ 2 and x ≠ ma
 shows vebt-insert (Node (Some (mi,ma)) deg treeList summary) x =
          Node (Some (x, max mi ma)) deg (treeList[(high mi  (deg div 2)) := vebt-insert (treeList
! (high mi  (deg div 2))) (low mi  (deg div 2))])
           (if minNull (treeList ! (high mi  (deg div 2))) then  vebt-insert summary (high mi  (deg
div 2)) else summary)
proof−
 obtain deg′ :: nat where deg = Suc (Suc deg′)
   using ‹2 ≤ deg› add-2-eq-Suc le-Suc-ex by blast
 then show ?thesis
   using assms by (simp add: Let-def)
qed


lemma insert-simp-mima: assumes x = mi ∨ x = ma and deg ≥ 2
 shows vebt-insert (Node (Some (mi,ma)) deg treeList summary) x =  (Node (Some (mi,ma)) deg
treeList summary)
proof −
 obtain deg′ :: nat where deg = Suc (Suc deg′)
   using ‹2 ≤ deg›
   by (metis add-2-eq-Suc less-eqE)
 then show ?thesis
   using assms(1) by simp
qed


lemma valid-insert-both-member-options-add: invar-vebt t n ⟹ x< 2^n ⟹ both-member-options
(vebt-insert t x) x
proof(induction t n arbitrary: x rule: invar-vebt.induct)
 case (1 a b)
 then show ?case apply(cases x)
   by (auto simp add: both-member-options-def)
next
 case (2 treeList n summary m deg)
 then obtain n′ where n = Suc n′
   by (metis One-nat-def Suc-le-D set-n-deg-not-0)
```

```
  then have deg = Suc (Suc (n' + n'))
    using 2 by linarith
  then show ?case
    by (auto simp: both-member-options-def)
next
  case (3 treeList n summary m deg)
  then obtain n' where n = Suc n'
    by (metis One-nat-def Suc-le-D set-n-deg-not-0)
  then have deg = Suc (Suc (Suc (n' + n')))
    using 3 by linarith
  then show ?case
    by (auto simp: both-member-options-def)
next
  case (4 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis deg-not-0 less-natE)
  then have deg = Suc (Suc (n' + n'))
    using 4 by linarith

  have length treeList =2^n using 4 by blast
  hence high x n < length treeList
    using 4.hyps(1) 4.hyps(3) 4.hyps(4) 4.prems deg-not-0 exp-split-high-low(1) by auto
  hence mi < 2^deg
    using 4.hyps(7) 4.hyps(8) le-less-trans by blast
  then show ?case
  proof(cases x = mi ∨ x = ma)
    case True
    then show ?thesis
      by (auto simp add: ‹deg = Suc (Suc -)› both-member-options-def)
  next
    case False
    hence ¬ (x = mi ∨ x = ma) by simp
    then show ?thesis
    proof(cases x < mi)
      case True
      hence high mi n < length treeList
        using 4.hyps(1) 4.hyps(2) 4.hyps(3) 4.hyps(4) ‹mi < 2 ^ deg› deg-not-0 exp-split-high-low(1)
by auto
      hence vebt-insert ( Node (Some (mi, ma)) deg treeList summary) x =
                    Node (Some (x, max mi ma)) deg ( treeList[(high mi n):= vebt-insert (treeList !
(high mi n)) (low mi n)] )
                      (if minNull (treeList ! high mi n) then  vebt-insert summary (high mi n) else
summary)
          by (metis 4.hyps(1) 4.hyps(3) 4.hyps(4) False True add-self-div-2 div-if insert-simp-excp
not-less valid-tree-deg-neq-0)
      then show ?thesis
      by (smt (verit, ccfv-threshold) 4.hyps(1,3,4) VEBT-internal.membermima.simps(4) add-2-eq-Suc
            add-self-div-2 both-member-options-def deg-not-0 div-greater-zero-iff le-add-diff-inverse)
    next
```

```
    case False
    hence vebt-insert ( Node (Some (mi, ma)) deg treeList summary) x =
                  Node (Some (mi, max x ma)) deg (treeList[ (high x n):=vebt-insert (treeList ! (high
x n)) (low x n)])
                 (if minNull (treeList ! high x n) then  vebt-insert summary (high x n) else summary)
      by (metis 4.hyps(1) 4.hyps(3) 4.hyps(4) ‹¬ (x = mi ∨ x = ma)› ‹high x n < length treeList›
add-self-div-2 div-if insert-simp-norm linorder-neqE-nat not-less valid-tree-deg-neq-0)
    have low x n < 2^n ∧ high x n < 2^n
      using 4.hyps(2) 4.hyps(3) ‹high x n < length treeList› low-def by auto
    have invar-vebt (treeList ! (high x n)) n
      using 4.IH(1) ‹high x n < length treeList› by simp
    hence both-member-options (vebt-insert (treeList ! (high x n)) (low x n)) (low x n)
      by (simp add: 4.IH(1) ‹high x n < length treeList› low-def)
   have (treeList[ (high x n):=vebt-insert (treeList ! (high x n)) (low x n)]) ! (high x n) = vebt-insert
(treeList ! (high x n)) (low x n)
      by (simp add: ‹high x n < length treeList›)
    then show ?thesis
      using both-member-options-ding[of Some (mi, max x ma) deg
      (take (high x n) treeList @ [vebt-insert (treeList ! (high x n)) (low x n)] @ drop (high x n +1)
treeList)
      if minNull (treeList ! high x n) then  vebt-insert summary (high x n) else summary n x]
      by (metis 4.hyps(2) 4.hyps(3) 4.hyps(4) Suc-1 Suc-leD
          ‹vebt-insert (Node (Some (mi, ma)) deg treeList summary) x = Node (Some (mi, max x
ma)) deg (treeList[high x n := vebt-insert (treeList ! high x n) (low x n)]) (if minNull (treeList ! high x
n) then vebt-insert summary (high x n) else summary)› ‹both-member-options (vebt-insert (treeList !
high x n) (low x n)) (low x n)› ‹low x n < 2 ^ n ∧ high x n < 2 ^ n› ‹invar-vebt (treeList ! high x n) n›
add-self-div-2 both-member-options-from-chilf-to-complete-tree deg-not-0 div-greater-zero-iff length-list-update)
   qed
  qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' where n = Suc n'
    by (metis One-nat-def Suc-le-D set-n-deg-not-0)
  then have deg = Suc (Suc (Suc (n' + n')))
    using 5 by linarith

  have length treeList =2^m using 5 by blast
  hence high x n < length treeList
     by (metis 5.hyps(4) 5.prems div-eq-0-iff div-exp-eq high-def length-0-conv length-greater-0-conv
zero-less-numeral zero-less-power)
  hence mi<2^deg
    using 5.hyps(7) 5.hyps(8) le-less-trans by blast
  then show ?case
  proof(cases x = mi ∨ x = ma)
    case True
    then show ?thesis
      by (auto simp add: ‹deg = Suc (Suc -)› both-member-options-def)
  next
    case False
```

hence ¬ (*x* = *mi* ∨ *x* = *ma*) **by** *simp*
then show *?thesis*
proof(*cases x* < *mi*)
case *True*
hence *high mi n* < *length treeList*
by (*metis 5.hyps*(*2*) *5.hyps*(*4*) *div-eq-0-iff* ‹*mi* < *2* ^ *deg*› *div-exp-eq high-def length-0-conv length-greater-0-conv zero-less-numeral zero-less-power*)
hence *vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
Node (*Some* (*x*, *max mi ma*)) *deg* (*treeList*[(*high mi n*):=*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)])
(*if minNull* (*treeList* ! *high mi n*) *then vebt-insert summary* (*high mi n*) *else summary*)
using *insert-simp-excp*[*of mi deg treeList x ma summary*]
5(*1*) *5.hyps*(*3*) *5.hyps*(*4*) *False True add-Suc-right add-self-div-2*
append-Cons div-less even-Suc-div-two in-set-conv-decomp not-less odd-add valid-tree-deg-neq-0
by (*smt* (*verit*) *nth-mem*)
then show *?thesis*
by (*simp add*: *5.hyps*(*3*) *5.hyps*(*4*) *both-member-options-def*)
next
case *False*
hence *vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
Node (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[(*high x n*):= *vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
(*if minNull* (*treeList* ! *high x n*) *then vebt-insert summary* (*high x n*) *else summary*)
by (*smt* (*verit*) *5.IH*(*1*) *5.hyps*(*3*) *5.hyps*(*4*) ‹¬ (*x* = *mi* ∨ *x* = *ma*)› ‹*high x n* < *length treeList*› *add-Suc-right add-self-div-2 deg-not-0 div-greater-zero-iff even-Suc-div-two insert-simp-norm linorder-neqE-nat nth-mem odd-add*)
have *low x n* < *2*^*n* ∧ *high x n* < *2*^*m*
using *5.hyps*(*2*) *5.hyps*(*3*) ‹*high x n* < *length treeList*› *low-def* **by** *auto*
have *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *5.IH*(*1*) ‹*high x n* < *length treeList*› **by** *simp*
hence *both-member-options* (*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)) (*low x n*)
using *5.IH*(*1*) ‹*high x n* < *length treeList*› ‹*low x n* < *2* ^ *n* ∧ *high x n* < *2* ^ *m*› **by** *simp*
have (*treeList*[(*high x n*):=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)]) ! (*high x n*) = *vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)
by (*meson* ‹*high x n* < *length treeList*› *nth-list-update-eq*)
then show *?thesis*
using *both-member-options-ding*[*of Some* (*mi*, *max x ma*) *deg*
(*treeList*[(*high x n*):=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
if minNull (*treeList* ! *high x n*) *then vebt-insert summary* (*high x n*) *else summary n x*]
using *5.hyps*(*2*) *5.hyps*(*3*) *5.hyps*(*4*) ‹*vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Node* (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)]) (*if minNull* (*treeList* ! *high x n*) *then vebt-insert summary* (*high x n*) *else summary*)› ‹*both-member-options* (*vebt-insert* (*treeList* ! *high x n*) (*low x n*)) (*low x n*)› ‹*low x n* < *2* ^ *n* ∧ *high x n* < *2* ^ *m*› *both-member-options-from-chilf-to-complete-tree* **by** *auto*
qed
qed
qed

lemma *valid-insert-both-member-options-pres*: *invar-vebt t n* $\Longrightarrow$ *x*<*2^n* $\Longrightarrow$ *y* < *2^n* $\Longrightarrow$ *both-member-options t x*
$\Longrightarrow$ *both-member-options* (*vebt-insert t y*) *x*
proof(*induction t n arbitrary*: *x y rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case* **by** (*simp add*: *both-member-options-def*)
next
case (*2 treeList n summary m deg*)
then show *?case*
using *vebt-member.simps*(*2*) *invar-vebt.intros*(*2*) *valid-member-both-member-options* **by** *blast*
next
case (*3 treeList n summary m deg*)
then show *?case*
using *vebt-member.simps*(*2*) *invar-vebt.intros*(*3*) *valid-member-both-member-options* **by** *blast*
next
case (*4 treeList n summary m deg mi ma*)
hence *00*:*deg* = *n* + *m* ∧ *length treeList* = *2^n* ∧ *n* = *m* ∧ *n* ≥ *1* ∧ *deg* ≥ *2*
by (*metis One-nat-def Suc-leI add-mono-thms-linordered-semiring*(*1*) *deg-not-0 one-add-one*)
hence *xyprop*: *high x n* < *2^m* ∧ *high y n* < *2^m*
by (*metis 4.prems*(*1*) *4.prems*(*2*) *high-def less-mult-imp-div-less mult-2 power2-eq-square power-even-eq*)
have *low x n* <*2^n* ∧ *low y n*< *2^n*
by (*simp add*: *low-def*)
hence *x* = *mi* ∨ *x* = *ma* ∨ *both-member-options* (*treeList* ! (*high x n*)) (*low x n*)
by (*metis 00 4.prems*(*3*) *add-self-div-2 both-member-options-from-complete-tree-to-child trans-le-add1*)
have *001*:*invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg* **using** *invar-vebt.intros*(*4*)[*of treeList n summary m deg mi ma*] *4* **by** *simp*
then show *?case*
proof(*cases x* = *y*)
case *True*
hence *both-member-options* (*vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*) *y*
using *001 valid-insert-both-member-options-add*[*of* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg y*]
using *4.prems*(*2*) **by** *blast*
then show *?thesis*
by (*simp add*: *True*)
next
case *False*
then show *?thesis*
proof(*cases y* = *mi* ∨ *y* = *ma*)
case *True*
have *Suc* (*Suc* (*deg* −*2*)) = *deg*
using *00* **by** *linarith*
hence *vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y* = *Node* (*Some* (*mi*, *ma*)) *deg treeList summary*
using *vebt-insert.simps*(*5*)[*of mi ma deg*−*2 treeList summary x*] *00* *True insert-simp-mima* **by** *blast*
then show *?thesis*

by (*simp add*: *4.prems*(*3*))
next
case *False*
hence *0*:$y \neq mi \wedge y \neq ma$ **by** *simp*
then show *?thesis*
proof(*cases* $x = mi$)
case *True*
hence *1*:$x = mi$ **by** *simp*
then show *?thesis*
proof(*cases* $x < y$)
case *True*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*x*, *max y ma*)) *deg* (*treeList* [(*high y n*):=*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)])
(*if minNull* (*treeList* ! (*high y n*)) *then vebt-insert summary* (*high y n*) *else summary*)
using *00 1 False True insert-simp-norm xyprop* **by** *auto*
then show *?thesis*
by (*metis 001 Suc-pred both-member-options-def deg-not-0 membermima.simps*(*4*))
next
case *False*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*y*, *max x ma*)) *deg* (*treeList* [(*high x n*) :=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
(*if minNull* (*treeList* ! (*high x n*)) *then vebt-insert summary* (*high x n*) *else summary*)
by (*metis 0 00 False True add-self-div-2 insert-simp-excp linorder-neqE-nat xyprop*)
have *15*: *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *4*(*1*) *4.hyps*(*2*) *xyprop* **by** *simp*
hence *16*: *both-member-options* (*vebt-insert* (*treeList* ! *high x n*) (*low x n*)) (*low x n*)
using ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *valid-insert-both-member-options-add* **by** *blast*
then show *?thesis*
by (*metis 00 14 Suc-1 add-leD1 add-self-div-2 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq plus-1-eq-Suc xyprop*)

qed
next
case *False*
hence $mi \neq ma$
using *001 4.prems*(*3*) *less-irrefl member-inv valid-member-both-member-options* **by** *fastforce*
hence *both-member-options* (*treeList* !(*high x n*)) (*low x n*) ∨ $x = ma$
using *False* ‹$x = mi \vee x = ma \vee$ *both-member-options* (*treeList* ! *high x n*) (*low x n*)› **by** *blast*
have *high ma n* < *2*^*n*
by (*metis 4.hyps*(*3*) *4.hyps*(*4*) *4.hyps*(*8*) *high-def less-mult-imp-div-less mult-2 power2-eq-square power-even-eq*)
hence *both-member-options* (*treeList* !(*high ma n*)) (*low ma n*)
using *4.hyps*(*3*) *4.hyps*(*9*) ‹*mi* ≠ *ma*› **by** *blast*
hence *both-member-options* (*treeList* !(*high x n*)) (*low x n*)
using ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*) ∨ $x = ma$› **by** *blast*

then show *?thesis*
proof(*cases mi* < *y*)
case *True*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*mi*, *max y ma*)) *deg* (*treeList*[(*high y n*):=*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)])
(*if minNull* (*treeList* ! (*high y n*)) *then vebt-insert summary* (*high y n*) *else summary*)
using *0 00 True insert-simp-norm xyprop* **by** *auto*
have *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *4.IH*(*1*) *4.hyps*(*2*) *xyprop* **by** *simp*
then show *?thesis*
proof(*cases high x n* = *high y n*)
case *True*
have *both-member-options* (*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)) (*low x n*)
using *4.IH*(*1*) *4.hyps*(*2*) *True* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *xyprop* **by** *auto*
then show *?thesis*
by (*metis 00 14 Suc-1 True add-leD1 add-self-div-2 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq plus-1-eq-Suc xyprop*)
next
case *False*
have (*treeList*[(*high y n*):=*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)]) ! (*high x n*) = *treeList* ! (*high x n*)
using *False* **by** *auto*
then show *?thesis*
by (*metis 00 14 One-nat-def Suc-leD* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› *add-self-div-2 both-member-options-from-chilf-to-complete-tree length-list-update numeral-2-eq-2 xyprop*)
qed
next
case *False*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*y*, *max mi ma*)) *deg* (*treeList*[(*high mi n*):=*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)])
(*if minNull* (*treeList* ! (*high mi n*)) *then vebt-insert summary* (*high mi n*) *else summary*)
using *insert-simp-excp*[*of mi deg treeList y ma summary*]
using *0 00 4.hyps*(*7*,*8*) *False high-bound-aux* **by** *auto*
have *mimaprop*: *high mi n* < *2*^*n* ∧ *low mi n* < *2*^*n*
by (*metis 00 4.hyps*(*7*) *4.hyps*(*8*) *div-eq-0-iff div-exp-eq high-def le-less-trans low-def mod-less-divisor zero-less-numeral zero-less-power*)
have *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *4.IH*(*1*) *4.hyps*(*2*) *xyprop* **by** *simp*
then show *?thesis*
proof(*cases high x n* = *high mi n*)
case *True*
have *both-member-options* (*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)) (*low x n*)
by (*metis 4.IH*(*1*) *4.hyps*(*2*) *True* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)›

‹low x n < 2 ^ n ∧ low y n < 2 ^ n› mimaprop nth-mem xyprop)
then show ?thesis
by (metis 00 14 Suc-1 Suc-leD True add-self-div-2 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq xyprop)
next
case False
have (treeList[(high mi n):=vebt-insert (treeList ! (high mi n)) (low mi n)]) ! (high x n) = treeList ! (high x n)
using False by force
then show ?thesis
by (metis 00 14 One-nat-def Suc-leD ‹both-member-options (treeList ! high x n) (low x n)› add-self-div-2 both-member-options-from-chilf-to-complete-tree length-list-update numeral-2-eq-2 xyprop)
qed
qed
qed
qed
qed
next
case (5 treeList n summary m deg mi ma)
hence 00:deg = n + m ∧ length treeList = 2^m ∧ Suc n = m ∧ n ≥ 1 ∧ deg ≥ 2 ∧ n = deg div 2
by (metis Suc-1 add-Suc-right add-mono-thms-linordered-semiring(1) add-self-div-2 even-Suc-div-two le-add1 odd-add plus-1-eq-Suc set-n-deg-not-0)
hence xyprop: high x n < 2^m ∧ high y n < 2^m
by (metis 5.prems(1) 5.prems(2) Suc-1 div-exp-eq div-if high-def nat.discI power-not-zero)
have low x n <2^n ∧ low y n< 2^n
by (simp add: low-def)
hence x = mi ∨ x = ma ∨ both-member-options (treeList ! (high x n)) (low x n)
by (metis 00 5.prems(3) add-leD1 both-member-options-from-complete-tree-to-child one-add-one)
have 001:invar-vebt (Node (Some (mi, ma)) deg treeList summary) deg
using invar-vebt.intros(5)[of treeList n summary m deg mi ma] 5 by simp
then show ?case
proof(cases x = y)
case True
hence both-member-options (vebt-insert (Node (Some (mi, ma)) deg treeList summary) y) y
using 001 valid-insert-both-member-options-add[of (Node (Some (mi, ma)) deg treeList summary) deg y]
using 5.prems(2) by blast
then show ?thesis
by (simp add: True)
next
case False
then show ?thesis
proof(cases y = mi ∨ y = ma)
case True
have Suc (Suc (deg −2)) = deg
using 00 by linarith
hence vebt-insert (Node (Some (mi, ma)) deg treeList summary) y = Node (Some (mi, ma)) deg treeList summary

using *vebt-insert.simps(5)[of mi ma deg−2 treeList summary x] 00 True insert-simp-mima* **by** *blast*
then show *?thesis*
by (*simp add*: *5.prems(3)*)
next
case *False*
hence *0*:$y \neq mi \wedge y \neq ma$ **by** *simp*
then show *?thesis*
proof(*cases* $x = mi$)
case *True*
hence *1*:$x = mi$ **by** *simp*
then show *?thesis*
proof(*cases* $x < y$)
case *True*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*x*, *max y ma*)) *deg* (*treeList*[(*high y n*):=*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)])
(*if minNull* (*treeList* ! (*high y n*)) *then vebt-insert summary* (*high y n*) *else summary*)
using *00 1 False True insert-simp-norm xyprop* **by** *metis*
then show *?thesis*
by (*metis 001 Suc-pred both-member-options-def deg-not-0 membermima.simps(4)*)
next
case *False*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*y*, *max x ma*)) *deg* (*treeList*[(*high x n*):=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
(*if minNull* (*treeList* ! (*high x n*)) *then vebt-insert summary* (*high x n*) *else summary*)
by (*metis 0 00 False True add-self-div-2 insert-simp-excp linorder-neqE-nat xyprop*)
have *15*: *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *5(1) 5.hyps(2) xyprop* **by** *simp*
hence *16*: *both-member-options* (*vebt-insert* (*treeList* ! *high x n*) (*low x n*)) (*low x n*)
using ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *valid-insert-both-member-options-add* **by** *blast*
then show *?thesis*
by (*metis 00 14 Suc-1 add-leD1 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq plus-1-eq-Suc xyprop*)
qed
next
case *False*
hence $mi \neq ma$
using *001 5.prems(3) less-irrefl member-inv valid-member-both-member-options* **by** *fastforce*
hence *both-member-options* (*treeList* !(*high x n*)) (*low x n*) ∨ $x = ma$
using *False* ‹$x = mi \vee x = ma \vee$ *both-member-options* (*treeList* ! *high x n*) (*low x n*)› **by** *blast*
have *high ma n* < *2*^*m*
by (*metis 00 5.hyps(8) div-eq-0-iff div-exp-eq high-def nat-zero-less-power-iff power-not-zero zero-power2*)
hence *both-member-options* (*treeList* !(*high ma n*)) (*low ma n*)
using *5.hyps(3) 5.hyps(9)* ‹*mi* ≠ *ma*› **by** *blast*

hence *both-member-options* (*treeList* !(*high x n*)) (*low x n*)
using ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*) ∨ *x* = *ma*› **by** *blast*
then show *?thesis*
proof(*cases mi* < *y*)
case *True*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*mi*, *max y ma*)) *deg* (*treeList*[(*high y n*):= *vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)])
(*if minNull* (*treeList* ! (*high y n*)) *then vebt-insert summary* (*high y n*) *else summary*)
by (*metis 0 00 True insert-simp-norm xyprop*)
have *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *5.IH*(*1*) *5.hyps*(*2*) *xyprop* **by** *simp*
then show *?thesis*
proof(*cases high x n* = *high y n*)
case *True*
have *both-member-options* (*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)) (*low x n*)
by (*metis 5.IH*(*1*) *5.hyps*(*2*) *True* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *nth-mem xyprop*)
then show *?thesis*
by (*metis 00 14 Suc-1 True add-leD1 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq plus-1-eq-Suc xyprop*)
next
case *False*
have (*treeList*[(*high y n*):=*vebt-insert* (*treeList* ! (*high y n*)) (*low y n*)]) ! (*high x n*) = *treeList* ! (*high x n*)
using *False* **by** *force*
then show *?thesis*
by (*metis 00 14 One-nat-def Suc-leD* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› *both-member-options-from-chilf-to-complete-tree length-list-update numeral-2-eq-2 xyprop*)
qed
next
case *False*
have *14*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *y* =
Node (*Some* (*y*, *max mi ma*)) *deg* (*treeList*[(*high mi n*):= *vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)])
(*if minNull* (*treeList* ! (*high mi n*)) *then vebt-insert summary* (*high mi n*) *else summary*)
using *insert-simp-excp*[*of mi deg treeList y ma summary*]
by (*metis 0 00 5.hyps*(*7*) *5.hyps*(*8*) *div-eq-0-iff False antisym-conv3 div-exp-eq high-def le-less-trans power-not-zero zero-neq-numeral*)
have *mimaprop*: *high mi n* < *2*^*m* ∧ *low mi n* < *2*^*n* **using** *exp-split-high-low*[*of mi n m*] *00 5*(*9*,*10*) **by** *force*
have *invar-vebt* (*treeList* ! (*high x n*)) *n*
using *5.IH*(*1*) *5.hyps*(*2*) *xyprop* **by** *simp*
then show *?thesis*
proof(*cases high x n* = *high mi n*)
case *True*
have *both-member-options* (*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)) (*low x n*)

by (*metis 5.IH(1) 5.hyps(2) True* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *mimaprop nth-mem*)
then show *?thesis*
by (*metis 00 14 Suc-1 True add-leD1 both-member-options-from-chilf-to-complete-tree length-list-update nth-list-update-eq plus-1-eq-Suc xyprop*)
next
case *False*
have (*treeList*[(*high mi n*):=*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)]) ! (*high x n*) = *treeList* ! (*high x n*)
by (*metis False nth-list-update-neq*)
then show *?thesis*
by (*metis 00 14 One-nat-def Suc-leD* ‹*both-member-options* (*treeList* ! *high x n*) (*low x n*)› *both-member-options-from-chilf-to-complete-tree length-list-update numeral-2-eq-2 xyprop*)
qed
qed
qed
qed
qed
qed

lemma *post-member-pre-member*:*invar-vebt t n* ⟹ *x*< *2*^*n* ⟹ *y* <*2*^*n* ⟹ *vebt-member* (*vebt-insert t x*) *y* ⟹ *vebt-member t y* ∨ *x* = *y*
proof(*induction t n arbitrary*: *x y rule*: *invar-vebt.induct*)
case (*1 a b*) **then show** *?case* **by** *auto*
next
case (*2 treeList n summary m deg*)
then obtain *n'* :: *nat* **where** *n* = *Suc n'*
by (*metis deg-not-0 less-iff-Suc-add*)
then have *deg* = *Suc* (*Suc* (*n'* + *n'*))
using *2* **by** *linarith*
then have *x* = *y*
by (*metis 2.prems*(*3*) *member-inv vebt-insert.simps*(*4*) *vebt-member.simps*(*5*))
then show *?case* **..**
next
case (*3 treeList n summary m deg*)
then obtain *n'* :: *nat* **where** *n* = *Suc n'*
by (*metis One-nat-def lessI not-less set-n-deg-not-0 vebt-buildup.cases*)
then have *deg* = *Suc* (*Suc* (*Suc* (*n'* + *n'*)))
using *3* **by** *linarith*
then have *x* = *y*
by (*metis 3.prems*(*3*) *member-inv vebt-insert.simps*(*4*) *vebt-member.simps*(*5*))
then show *?case* **..**
next
case (*4 treeList n summary m deg mi ma*)
hence *00*:*deg* = *n*+*m* ∧ *n* ≥ *0* ∧ *n* = *m* ∧ *deg* ≥ *2* ∧ *length treeList* =*2*^*n*
by (*metis div-eq-0-iff add-self-div-2 deg-not-0 not-less zero-le*)
hence *xyprop*: *high x n* < *2*^*n* ∧ *high y n* < *2*^*n*
using *4.hyps*(*1*) *4.prems*(*1*) *4.prems*(*2*) *deg-not-0 exp-split-high-low*(*1*) **by** *blast*
have *low x n* <*2*^*n* ∧ *low y n*< *2*^*n*

by (*simp add*: *low-def*)
then show *?case*
proof(*cases* $x = mi \vee x = ma$)
case *True*
then show *?thesis*
using *00 4.prems*(*3*) *insert-simp-mima* **by** *auto*
next
case *False*
hence *mimaxyprop*: $\neg (x = mi \vee x = ma) \wedge high\ x\ n < 2\hat{}n \wedge high\ mi\ n < 2\hat{}n \wedge low\ x\ n < 2\hat{}n \wedge low\ mi\ n < 2\hat{}n \wedge length\ treeList = 2\hat{}n$
using *00 4.hyps*(*1*) *4.hyps*(*7*) *4.hyps*(*8*) ‹*low x n < 2 ^ n ∧ low y n < 2 ^ n*› *deg-not-0 exp-split-high-low*(*1*) *exp-split-high-low*(*2*) *le-less-trans xyprop* **by** *blast*
then show *?thesis*
proof(*cases mi < x*)
case *True*
hence *vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
Node (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[(*high x n*) :=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
(*if minNull* (*treeList* ! (*high x n*)) *then vebt-insert summary* (*high x n*) *else summary*)
using *insert-simp-norm*[*of x n treeList mi ma summary*] *mimaxyprop 00 add-self-div-2 insert-simp-norm* **by** *metis*
then show *?thesis*
proof(*cases* $y = mi \vee y = max\ x\ ma$)
case *True*
then show *?thesis*
proof(*cases y = mi*)
case *True*
then show *?thesis*
using *00 le-Suc-ex* **by** *fastforce*
next
case *False*
hence $y = max\ x\ ma$
using *True* **by** *blast*
then show *?thesis*
proof(*cases x < ma*)
case *True*
then show *?thesis*
using *00* ‹$y = max\ x\ ma$› *le-Suc-ex* **by** *fastforce*
next
case *False*
then show *?thesis*
using ‹$y = max\ x\ ma$› **by** *linarith*
qed
qed
next
case *False*
hence *vebt-member* ((*treeList*[(*high x n*):= *vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)]) ! (*high y n*)) (*low y n*)

by (*metis 4.hyps(3) 4.hyps(4) 4.prems(3)* ‹*vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Node* (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)]) (*if minNull* (*treeList* ! *high x n*) *then vebt-insert summary* (*high x n*) *else summary*)› *add-self-div-2 member-inv*)
then show *?thesis*
proof(*cases high x n* = *high y n*)
case *True*
hence *000*:*vebt-member* (*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)) (*low y n*)
using ‹*vebt-member* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *high y n*) (*low y n*)› *mimaxyprop* **by** *auto*
have *001*:*invar-vebt* (*treeList* ! (*high x n*)) *n* ∧ *treeList* ! (*high x n*) ∈ *set treeList*
by (*simp add*: *4.IH*(*1*) *mimaxyprop*)
hence *002*:*vebt-member* (*treeList* ! (*high x n*)) (*low y n*) ∨ *low y n* = *low x n*
using *000 4.IH*(*1*) ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› **by** *fastforce*
hence *003*:*both-member-options* (*treeList* ! (*high x n*)) (*low y n*) ∨ *low y n* = *low x n*
using ‹*invar-vebt* (*treeList* ! *high x n*) *n* ∧ *treeList* ! *high x n* ∈ *set treeList*› *both-member-options-equiv-member* **by** *blast*
have *004*:*naive-member* (*treeList* ! (*high x n*)) (*low y n*) ⟹
naive-member (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
by (*metis 00 Suc-le-D True add-self-div-2 mimaxyprop naive-member.simps*(*3*) *one-add-one plus-1-eq-Suc*)
hence *005*:*both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y* ∨ *x* = *y*
by (*metis 00 001 002 Suc-le-D True add-self-div-2 bit-split-inv both-member-options-def member-valid-both-member-options membermima.simps*(*4*) *mimaxyprop one-add-one plus-1-eq-Suc*)
then show *?thesis*
by (*smt* (*verit*) *00 001 002 003 4*(*11*) *4*(*8*) *vebt-member.simps*(*5*) *True add-numeral-left add-self-div-2 bit-split-inv le-add-diff-inverse mimaxyprop not-less not-less-iff-gr-or-eq numerals*(*1*) *plus-1-eq-Suc semiring-norm*(*2*))
next
case *False*
hence *000*:*vebt-member* (*treeList* ! (*high y n*)) (*low y n*)
using ‹*vebt-member* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *high y n*) (*low y n*)› **by** *auto*
moreover have *004*:*naive-member* (*treeList* ! (*high y n*)) (*low y n*) ⟹
naive-member (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
by (*metis 00 Suc-le-D add-self-div-2 naive-member.simps*(*3*) *one-add-one plus-1-eq-Suc xyprop*)
moreover have *001*:*invar-vebt* (*treeList* ! (*high y n*)) *n* ∧ *treeList* ! (*high y n*) ∈ *set treeList*
using *4.IH*(*1*) *4.hyps*(*2*) *4.hyps*(*3*) *xyprop* **by** *simp*
moreover have *both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
by (*metis 00 000 001 004 Suc-le-D add-self-div-2 both-member-options-def member-valid-both-member-options membermima.simps*(*4*) *one-add-one plus-1-eq-Suc xyprop*)
moreover have *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
using *both-member-options-equiv-member*[*of* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg y*]
invar-vebt.intros(*4*)[*of treeList n summary m deg mi ma*]
using *4 calculation*(*4*) **by** *blast*
then show *?thesis* **by** *simp*
qed

```
    qed
  next
    case False
    hence x < mi
      using mimaxyprop nat-neq-iff by blast
    hence vebt-insert (Node (Some (mi,ma)) deg  treeList summary) x =
            Node (Some (x, max mi ma)) deg (treeList[ (high mi n):=vebt-insert (treeList ! (high mi
n)) (low mi n)])
                  (if minNull (treeList ! (high mi n)) then  vebt-insert summary (high mi n) else
summary)
       using insert-simp-excp[of mi n treeList x ma summary] mimaxyprop 00 add-self-div-2 in-
sert-simp-excp by metis

    consider y = x | y = max mi ma | y ≠ x y ≠ max mi ma
      by satx
    then show ?thesis
    proof cases
      case 1
      then show ?thesis
        by (simp add: 00)
    next
      case 2
      then show ?thesis
        using 00 le-Suc-ex by fastforce
    next
      case 3
      hence vebt-member ((treeList[(high mi n) :=vebt-insert (treeList ! (high mi n)) (low mi n)]) !
(high y n)) (low y n)
        by (metis 4.hyps(3) 4.hyps(4) 4.prems(3) ‹vebt-insert (Node (Some (mi, ma)) deg treeList
summary) x = Node (Some (x, max mi ma)) deg (treeList[high mi n := vebt-insert (treeList ! high mi
n) (low mi n)]) (if minNull (treeList ! high mi n) then vebt-insert summary (high mi n) else summary)›
add-self-div-2 member-inv)
      then show ?thesis
      proof(cases high mi n  = high y n)
        case True
        hence 000:vebt-member (vebt-insert (treeList ! (high mi n)) (low mi n)) (low y n)
          by (metis ‹vebt-member (treeList[high mi n := vebt-insert (treeList ! high mi n) (low mi n)]
! high y n) (low y n)› mimaxyprop nth-list-update-eq)
        have 001:invar-vebt (treeList ! (high mi n)) n ∧ treeList ! (high mi n) ∈ set treeList
          by (simp add: 4.IH(1) mimaxyprop)
        hence 002:vebt-member (treeList ! (high mi n)) (low y n) ∨ low y n = low mi n
          using 000 4.IH(1) ‹low x n < 2 ^ n ∧ low y n < 2 ^ n› mimaxyprop by fastforce
        hence 003:both-member-options (treeList ! (high mi n)) (low y n) ∨ low y n = low mi n
            using ‹invar-vebt (treeList ! high mi n) n ∧ treeList ! high mi n ∈ set treeList›
both-member-options-equiv-member by blast
        have 004:naive-member (treeList ! (high mi n)) (low y n) ⟹
        naive-member (Node (Some (mi , ma)) deg treeList summary) y using naive-member.simps(3)[of
Some (mi, ma) deg−1 treeList summary y]
          using 00 True mimaxyprop by fastforce
```

```
        hence 005:both-member-options (Node (Some (mi , ma)) deg treeList summary) y ∨ x = y
          by (metis 00 001 002 Suc-le-D True add-self-div-2 bit-split-inv both-member-options-def
member-valid-both-member-options membermima.simps(4) mimaxyprop one-add-one plus-1-eq-Suc)
        then show ?thesis
       by (smt (verit) 00 001 002 003 4.hyps(6) 4.hyps(9) vebt-member.simps(5) True add-numeral-left
add-self-div-2 bit-split-inv le-add-diff-inverse mimaxyprop not-less not-less-iff-gr-or-eq numerals(1) plus-1-eq-Suc
semiring-norm(2))
      next
        case False
        hence 000:vebt-member (treeList ! (high y n)) (low y n)
          using ‹vebt-member (treeList[high mi n := vebt-insert (treeList ! high mi n) (low mi n)] !
high y n) (low y n)› by auto
        moreover have 004:naive-member (treeList ! (high y n)) (low y n) ⟹
          naive-member (Node (Some (mi , ma)) deg treeList summary) y
           by (metis 00 Suc-le-D add-self-div-2 naive-member.simps(3) one-add-one plus-1-eq-Suc
xyprop)
       moreover have 001:invar-vebt (treeList ! (high y n)) n ∧ treeList ! (high y n) ∈ set treeList
         using 4.IH(1) 4.hyps(2) 4.hyps(3) xyprop by simp
        moreover have  both-member-options (Node (Some (mi , ma)) deg treeList summary) y
       by (metis 00 000 001 004 Suc-le-D add-self-div-2 both-member-options-def member-valid-both-member-options
membermima.simps(4) one-add-one plus-1-eq-Suc xyprop)
        then show ?thesis using both-member-options-equiv-member[of (Node (Some (mi, ma)) deg
treeList summary) deg y]
          invar-vebt.intros(4)[of treeList n summary m deg mi ma] 4 by blast
      qed
    qed
  qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis One-nat-def lessI not-less set-n-deg-not-0 vebt-buildup.cases)
  then have deg = Suc (Suc (Suc (n' + n')))
    using 5 by linarith

  hence 00:deg = n+m ∧ n ≥ 0 ∧ Suc n = m ∧ deg ≥ 2 ∧ length treeList =2^m ∧ n ≥ 1
    using 5
   by (metis Suc-1 add-mono-thms-linordered-semiring(1) le-add1 plus-1-eq-Suc set-n-deg-not-0 zero-le)
  hence xyprop: high x n < 2^m ∧ high y n < 2^m
    using 5.prems(1) 5.prems(2) exp-split-high-low(1) by auto
  have low x n <2^n ∧ low y n< 2^n
    by (simp add: low-def)
  then show ?case
  proof(cases x = mi ∨ x = ma)
    case True
    then show ?thesis
      using 00 5.prems(3) insert-simp-mima by auto
  next
    case False
```

hence *mimaxyprop*: ¬ (*x* = *mi* ∨ *x* = *ma*) ∧ *high x n* < 2^*m* ∧ *high mi n* < 2^*m* ∧ *low x n* <2^*n* ∧ *low mi n* < 2^*n* ∧ *length treeList* = 2^*m*
using *00 5* ‹*low x n* < 2 ^*n* ∧ *low y n* < 2 ^*n*› *deg-not-0 exp-split-high-low*(*1*) *exp-split-high-low*(*2*) *le-less-trans xyprop*
by (*smt* (*verit*, *best*) *dual-order.strict-trans1 less-numeral-extra*(*1*))
then show *?thesis*
proof(*cases mi* < *x*)
case *True*
hence *vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
Node (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[(*high x n*):=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)])
(*if minNull* (*treeList* ! (*high x n*)) *then vebt-insert summary* (*high x n*) *else summary*)
using *insert-simp-norm*[*of x deg treeList mi ma summary*]
by (*smt* (*verit*, *del-insts*) *00 Suc-mult-cancel1 add-Suc-right diff-Suc-1 even-Suc even-add mimaxyprop*
mult-2 odd-two-times-div-two-nat one-add-one plus-1-eq-Suc)
then show *?thesis*
proof(*cases y* = *mi* ∨ *y* = *max x ma*)
case *True*
then show *?thesis*
proof(*cases y* = *mi*)
case *True*
then show *?thesis*
using *00 le-Suc-ex* **by** *fastforce*
next
case *False*
hence *y* = *max x ma*
using *True* **by** *blast*
then show *?thesis*
proof(*cases x* < *ma*)
case *True*
then show *?thesis*
using *00* ‹*y* = *max x ma*› *le-Suc-ex* **by** *fastforce*
next
case *False*
then show *?thesis*
using ‹*y* = *max x ma*› **by** *linarith*
qed
qed
next
case *False*
hence *vebt-member* ((*treeList*[(*high x n*):=*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)]) ! (*high y n*)) (*low y n*)
using *5.hyps*(*3*) *5.hyps*(*4*) *5.prems*(*3*) ‹*vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Node* (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)]) (*if minNull* (*treeList* ! *high x n*) *then vebt-insert summary* (*high x n*) *else summary*)› *add-Suc-right add-self-div-2 member-inv* **by** *force*
then show *?thesis*

proof(*cases high x n = high y n*)
case *True*
hence *000*:*vebt-member* (*vebt-insert* (*treeList* ! (*high x n*)) (*low x n*)) (*low y n*)
by (*metis 5.hyps*(*2*) ‹*vebt-member* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *high y n*) (*low y n*)› *nth-list-update-eq xyprop*)
have *001*:*invar-vebt* (*treeList* ! (*high x n*)) *n* ∧ *treeList* ! (*high x n*) ∈ *set treeList*
by (*simp add*: *5.IH*(*1*) *5.hyps*(*2*) *xyprop*)
hence *002*:*vebt-member* (*treeList* ! (*high x n*)) (*low y n*) ∨ *low y n* = *low x n*
using *000 5.IH*(*1*) ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› **by** *fastforce*
hence *003*:*both-member-options* (*treeList* ! (*high x n*)) (*low y n*) ∨ *low y n* = *low x n*
using ‹*invar-vebt* (*treeList* ! *high x n*) *n* ∧ *treeList* ! *high x n* ∈ *set treeList*› *both-member-options-equiv-member* **by** *blast*
have *004*:*naive-member* (*treeList* ! (*high x n*)) (*low y n*) ⟹
naive-member (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
using *00 True xyprop* **by** *auto*
hence *005*:*both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y* ∨ *x* = *y*
by (*metis 00 003 True add-Suc-right add-leD1 add-self-div-2 bit-split-inv*
both-member-options-from-chilf-to-complete-tree even-Suc-div-two even-add one-add-one
xyprop)
then show *?thesis*
using *both-member-options-equiv-member*[*of* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg y*]
invar-vebt.intros(*5*)[*of treeList n summary m deg mi ma*] *5* **by** *blast*
next
case *False*
hence *000*:*vebt-member* (*treeList* ! (*high y n*)) (*low y n*)
using ‹*vebt-member* (*treeList*[*high x n* := *vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *high y n*) (*low y n*)› **by** *fastforce*
moreover have *004*:*naive-member* (*treeList* ! (*high y n*)) (*low y n*) ⟹
naive-member (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
using *00 xyprop* **by** *auto*
moreover have *001*:*invar-vebt* (*treeList* ! (*high y n*)) *n* ∧ *treeList* ! (*high y n*) ∈ *set treeList*
using *5 xyprop* **by** *simp*
moreover have *both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
using *00 000 001 both-member-options-def member-valid-both-member-options xyprop* **by** *fastforce*
moreover have *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
using *both-member-options-equiv-member*[*of* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg y*]
invar-vebt.intros(*5*)[*of treeList n summary m deg mi ma*] *5 calculation*(*4*) **by** *blast*
then show *?thesis* **by** *simp*
qed
qed
next
case *False*
hence *x* < *mi*
using *mimaxyprop nat-neq-iff* **by** *blast*
hence *vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
Node (*Some* (*x*, *max mi ma*)) *deg* (*treeList*[(*high mi n*):=*vebt-insert* (*treeList* ! (*high mi*

n)) (*low mi n*)])
(*if minNull* (*treeList* ! (*high mi n*)) *then vebt-insert summary* (*high mi n*) *else summary*)
using *insert-simp-excp*[*of mi n treeList x ma summary*] *mimaxyprop 00 add-self-div-2 insert-simp-excp*
by (*smt* (*verit*) *add-Suc-right even-Suc-div-two odd-add*)

consider $y = x \mid y = max\ mi\ ma \mid y \neq x\ y \neq max\ mi\ ma$
by *satx*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by (*simp add*: *00*)
next
case *2*
then show *?thesis*
using *00* ‹*y* = *max mi ma*› *le-Suc-ex* **by** *fastforce*
next
case *3*
hence *vebtmem*: *vebt-member* ((*treeList*[(*high mi n*):=*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)]) ! (*high y n*)) (*low y n*)
using *5.hyps*(*3*) *5.hyps*(*4*) *5.prems*(*3*) ‹*vebt-insert* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Node* (*Some* (*x*, *max mi ma*)) *deg* (*treeList*[*high mi n* := *vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)]) (*if minNull* (*treeList* ! *high mi n*) *then vebt-insert summary* (*high mi n*) *else summary*)› *member-inv* **by** *force*

have *invar*: *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*
using *00 5.IH*(*1*) *5.hyps*(*1*,*5*,*6*,*7*,*8*,*9*) *invar-vebt.intros*(*5*) **by** *force*

moreover have *both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
proof(*cases high mi n* = *high y n*)
case *True*
hence *000*:*vebt-member* (*vebt-insert* (*treeList* ! (*high mi n*)) (*low mi n*)) (*low y n*)
by (*metis vebtmem mimaxyprop nth-list-update-eq*)
have *001*:*invar-vebt* (*treeList* ! (*high mi n*)) *n* ∧ *treeList* ! (*high mi n*) ∈ *set treeList*
by (*simp add*: *5.IH*(*1*) *mimaxyprop*)
moreover have *vebt-member* (*treeList* ! (*high mi n*)) (*low y n*) ∨ *low y n* = *low mi n*
using *000 5.IH*(*1*) ‹*low x n* < *2* ^ *n* ∧ *low y n* < *2* ^ *n*› *mimaxyprop* **by** *fastforce*
moreover have *naive-member* (*treeList* ! (*high mi n*)) (*low y n*) ⟹
naive-member (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y* **using** *naive-member.simps*(*3*)[*of Some* (*mi*, *ma*) *deg*−*1 treeList summary y*]
using *00 True mimaxyprop* **by** *fastforce*
ultimately show *both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
by (*smt* (*verit*) *00 True add-Suc-right add-self-div-2 bit-split-inv both-member-options-def even-Suc-div-two member-valid-both-member-options membermima.simps*(*4*) *odd-add xyprop*)
next
case *False*
hence *vebt-member* (*treeList* ! (*high y n*)) (*low y n*)

using *vebtmem* **by** *auto*
moreover have *invar-vebt* (*treeList* ! (*high y n*)) *n* ∧ *treeList* ! (*high y n*) ∈ *set treeList*
using *5.IH*(*1*) *5.hyps*(*2*) *5.hyps*(*3*) *xyprop* **by** *simp*
ultimately show *both-member-options* (*Node* (*Some* (*mi* , *ma*)) *deg treeList summary*) *y*
using *00 both-member-options-def member-valid-both-member-options xyprop* **by** *fastforce*
qed

ultimately show *?thesis*
using *invar* **by** (*simp add*: *both-member-options-equiv-member*)
qed
qed
qed
qed

end
end

theory *VEBT-InsertCorrectness* **imports** *VEBT-Member VEBT-Insert*
begin

context *VEBT-internal* **begin**

4 Correctness of the Insert Operation

4.1 Validness Preservation

theorem *valid-pres-insert*: *invar-vebt t n* ⟹ *x*< *2*^*n* ⟹ *invar-vebt* (*vebt-insert t x*) *n*
proof(*induction t n arbitrary*: *x rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case* **using** *vebt-insert.simps*(*1*)[*of a b x*]
by (*simp add*: *invar-vebt.intros*(*1*))
next
case (*2 treeList n summary m deg*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary n* **and** *2*: *length treeList* = *2*^*n* **and**
3: *deg* = *2*∗*n* **and** *4*: (∄ *i*. *both-member-options summary i*) **and** *5*: (∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*) **and** *6*: *n*≥ *1*
using *2.prems* **by** (*auto simp add*: *Suc-leI deg-not-0*)
let *?t* = *Node None deg treeList summary*
let *?tnew* = *vebt-insert ?t x*
have *6*:*?tnew* = (*Node* (*Some* (*x*,*x*)) *deg treeList summary*) **using** *vebt-insert.simps*(*4*)[*of deg*−*2 treeList summary x*]
by (*metis 1 2.hyps*(*3*) *2.hyps*(*4*) *add-2-eq-Suc add-diff-inverse-nat add-self-div-2 deg-not-0 div-less gr-implies-not0*)
have *7*:(*x* = *x* ⟶ (∀ *t* ∈ *set treeList*. ∄ *x*. *both-member-options t x*))
using ‹∀ *t*∈*set treeList*. ∄*x*. *both-member-options t x*› **by** *blast*
have *8*:*x* ≤ *x* **by** *simp*
have *9*: *x* < *2*^*deg*
by (*simp add*: *2.prems*)

have *10*:$(x \neq x \longrightarrow (\forall\ i < 2\hat{}(2\hat{}n).\ (high\ x\ deg = i \longrightarrow both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ x\ deg)) \wedge$
$(\forall\ y.\ (high\ y\ deg = i \wedge both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ y\ deg)\ ) \longrightarrow x < y \wedge y \leq x)\ ))$
by *simp*
then show *?case* **using** *0 1 2 3 4 5 6 7 8 9 10 invar-vebt.intros*(*4*)[*of treeList n summary m deg x x*]
by (*metis 2.hyps*(*3*) *2.hyps*(*4*) *nth-mem*)
next
case (*3 treeList n summary m deg*)
hence *0*: $(\forall\ t \in set\ treeList.\ invar\text{-}vebt\ t\ n)$ **and** *1*: *invar-vebt summary m* **and** *2*: *length treeList* $= 2\hat{}m$ **and**
3: $deg = n+m$ **and** *4*: $(\nexists\ i.\ both\text{-}member\text{-}options\ summary\ i)$ **and** *5*: $(\forall\ t \in set\ treeList.\ \nexists\ x.\ both\text{-}member\text{-}options\ t\ x)$ **and** *6*: $n \geq 1$
and *7*: *Suc n* = *m* **using** *3.prems* **apply** *auto*
by (*metis 3.hyps*(*2*) *One-nat-def set-n-deg-not-0*)
let *?t* = *Node None deg treeList summary*
let *?tnew* = *vebt-insert ?t x*
have *6*:*?tnew* = (*Node* (*Some* (*x*,*x*)) *deg treeList summary*) **using** *vebt-insert.simps*(*4*)[*of deg*−*2 treeList summary x*]
using *3 3.hyps*(*3*) *6* **by** *simp*
have *7*:$(x = x \longrightarrow (\forall\ t \in set\ treeList.\ \nexists\ x.\ both\text{-}member\text{-}options\ t\ x))$
using ‹$\forall\ t{\in}set\ treeList.\ \nexists x.\ both\text{-}member\text{-}options\ t\ x$› **by** *blast*
have *8*:$x \leq x$ **by** *simp*
have *9*: $x < 2\hat{}deg$
by (*simp add*: *3.prems*)
have *10*:$(x \neq x \longrightarrow (\forall\ i < 2\hat{}(2\hat{}n).\ (high\ x\ deg = i \longrightarrow both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ x\ deg)) \wedge$
$(\forall\ y.\ (high\ y\ deg = i \wedge both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ y\ deg)\ ) \longrightarrow x < y \wedge y \leq x)\ ))$
by *simp*
then show *?case* **using** *0 1 2 3 4 5 6 7 8 9 10 invar-vebt.intros*(*5*)[*of treeList n summary m deg x x*] *3.hyps*(*3*) *nth-mem* **by** *force*
next
case (*4 treeList n summary m deg mi ma*)
hence *myIHs*: $x \in set\ treeList \Longrightarrow invar\text{-}vebt\ x\ n \Longrightarrow xa < 2\ \hat{}\ n \Longrightarrow invar\text{-}vebt\ (vebt\text{-}insert\ x\ xa)\ n$ **for** *x xa* **by** *simp*
hence *0*: $(\forall\ t \in set\ treeList.\ invar\text{-}vebt\ t\ n)$ **and** *1*: *invar-vebt summary m* **and** *2*:*length treeList* $= 2\hat{}m$ **and** *3*: $deg = n+m$ **and**
4: $(\forall\ i < 2\hat{}m.\ (\exists\ y.\ both\text{-}member\text{-}options\ (treeList\ !\ i)\ y) \longleftrightarrow (\ both\text{-}member\text{-}options\ summary\ i))$ **and**
5: $(mi = ma \longrightarrow (\forall\ t \in set\ treeList.\ \nexists\ y.\ both\text{-}member\text{-}options\ t\ y))$ **and** *6*:$mi \leq ma\ \wedge\ ma < 2\hat{}deg$ **and**
7: $(mi \neq ma \longrightarrow (\forall\ i < 2\hat{}m.\ (high\ ma\ n = i \longrightarrow both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ ma\ n)) \wedge$
$(\forall\ y.\ (high\ y\ n = i \wedge both\text{-}member\text{-}options\ (treeList\ !\ i)\ (low\ y\ n)\ ) \longrightarrow mi < y \wedge y \leq ma)))$
and *8*: $n = m$ **and** *9*: *deg div 2* = *n* **using** *4 add-self-div-2* **by** *blast*+

have *2 ≤ deg*
by (*metis 1 4.hyps(3) 9 div-less not-le-imp-less valid-0-not*)

show *?case*
proof(*cases x = mi ∨ x = ma*)
case *True*
show *?thesis*
unfolding *insert-simp-mima*[*of x mi ma deg treeList summary*, *OF True* ‹*2 ≤ deg*›]
using *invar-vebt.intros*(*4*)[*of treeList n summary m deg mi ma*]
using *0 1 2 3 4 4.hyps*(*7*,*8*) *5 7 8* **by** *argo*
next
case *False*
hence *mimaxrel*: *x ≠ mi ∧ x ≠ ma* **by** *simp*
then show *?thesis*
proof(*cases mi < x*)
case *True*
hence *abcdef*: *mi < x* **by** *simp*
let *?h = high x n* **and** *?l=low x n*
have *highlowprop*: *high x n < 2^m ∧ low x n < 2^n*
using *1 3 4.hyps*(*3*) *4.prems deg-not-0 exp-split-high-low*(*1*) *exp-split-high-low*(*2*) **by** *blast*
have *10*:*vebt-insert* (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
Node (*Some* (*mi*, *max x ma*)) *deg* (*treeList*[*?h*:=*vebt-insert* (*treeList* ! *?h*) *?l*])
(*if minNull* (*treeList* ! *?h*) *then vebt-insert summary ?h else summary*)
using *2 3 False True* ‹*high x n < 2 ^ m ∧ low x n < 2 ^ n*› *insert-simp-norm* **by** (*metis 1 4.hyps*(*3*) *9 deg-not-0 div-greater-zero-iff*)
let *?maxnew = max x ma* **and** *?nextTreeList* = (*take ?h treeList* @ [*vebt-insert* (*treeList* ! *?h*) *?l*] @ *drop* (*?h+1*) *treeList*) **and**
?nextSummary = (*if minNull* (*treeList* ! *?h*) *then vebt-insert summary ?h else summary*)
have *11*: (∀ *t* ∈ *set ?nextTreeList*. *invar-vebt t n*) **proof**
fix *t*
assume *t* ∈ *set ?nextTreeList*
hence *111*:*t* ∈ *set* (*take ?h treeList*) ∨ *t* ∈ *set* ([*vebt-insert* (*treeList* ! *?h*) *?l*] @ *drop* (*?h+1*) *treeList*) **by** *auto*
show *invar-vebt t n*
proof(*cases t* ∈ *set* (*take ?h treeList*))
case *True*
then show *?thesis*
by (*meson 0 in-set-takeD*)
next
case *False*
hence *1110*: *t = vebt-insert* (*treeList* ! *?h*) *?l* ∨ *t* ∈ *set* (*drop* (*?h+1*) *treeList*)
using *111* **by** *auto*
then show *?thesis*
proof(*cases t = vebt-insert* (*treeList* ! *?h*) *?l*)
case *True*
have *11110*: *invar-vebt* (*treeList* ! *?h*) *n*
by (*simp add*: *2 4.IH*(*1*) *highlowprop*)
have *11111*: *?l < 2^n*
by (*simp add*: *low-def*)

```
        then show ?thesis using myIHs[of treeList ! ?h]
          by (simp add: 11110 2 True highlowprop)
      next
        case False
        then show ?thesis
          by (metis 0 1110 append-assoc append-take-drop-id in-set-conv-decomp)
      qed
    qed
  qed
  have 12: invar-vebt ?nextSummary n
  proof(cases minNull (treeList ! high x n))
    case True
    then show ?thesis
      using 4.IH(2) 8 highlowprop by auto
  next
    case False
    then show ?thesis
      by (simp add: 1 8)
  qed
 have 13: ∀ i < 2^m. (∃ y. both-member-options (?nextTreeList ! i) y) ⟷ ( both-member-options
?nextSummary i)
  proof
    fix i
   show i < 2 ^ m ⟶ (∃ y. both-member-options   ((?nextTreeList) ! i) y) = both-member-options
(?nextSummary) i
    proof
      assume i< 2^m
   show (∃ y. both-member-options ((?nextTreeList) ! i) y) = both-member-options (?nextSummary)
i
      proof(cases minNull (treeList ! high x n))
        case True
        hence tc: minNull (treeList ! high x n) by simp
        hence nsprop: ?nextSummary = vebt-insert summary ?h by simp
        have insprop:?nextTreeList ! ?h = vebt-insert (treeList ! ?h) ?l
            by (metis 2 Suc-eq-plus1 append-Cons highlowprop nth-list-update-eq self-append-conv2
upd-conv-take-nth-drop)
        then show ?thesis
        proof(cases i = ?h)
          case True
          have 161:∄ y. vebt-member (treeList ! ?h) y
            by (simp add: min-Null-member tc)
          hence 162:∄ y. both-member-options (treeList ! ?h) y
            by (metis 2 4.IH(1) highlowprop nth-mem valid-member-both-member-options)
          hence 163:¬ both-member-options summary i
            using 11 2 4 True ‹i < 2 ^ m› by blast
          have 164:?nextTreeList ! i = vebt-insert (treeList ! ?h) ?l
            using True insprop by simp
          have 165:invar-vebt (vebt-insert (treeList ! ?h) ?l) n
            by (simp add: 11)
```

```
          have 166:both-member-options (vebt-insert (treeList ! ?h) ?l) ?l using myIHs[of treeList !
?h ?l]
            by (metis 0 2 highlowprop nth-mem valid-insert-both-member-options-add)
           have 167:∃ y. both-member-options ((?nextTreeList) ! i) y
            using 164 166 by auto
           then show ?thesis
            using 1 11 2 True nsprop valid-insert-both-member-options-add highlowprop by auto
         next
          case False
          have ?nextTreeList ! i = treeList ! i
            by (metis 2 False ‹i < 2 ^ m› highlowprop nth-repl)
              have fstprop:both-member-options ((?nextTreeList) ! i) y ⟹ both-member-options
(?nextSummary) i for y
                using 1 4 ‹(take (high x n) treeList @ [VEBT-Insert.vebt-insert (treeList ! high
x n) (low x n)] @ drop (high x n + 1) treeList) ! i = treeList ! i› ‹i < 2 ^ m› highlowprop
valid-insert-both-member-options-pres by auto
           moreover have both-member-options (?nextSummary) i ⟹ ∃ y . both-member-options
((?nextTreeList) ! i) y
          proof−
            assume both-member-options (?nextSummary) i
            then have both-member-options summary i
              using 12
              unfolding if-P[OF tc]
                  by (metis 1 False ‹i < 2 ^ m› both-member-options-equiv-member highlowprop
post-member-pre-member)
            hence ∃ y. both-member-options (treeList ! i) y
              by (simp add: 4 ‹i < 2 ^ m›)
            then show ?thesis
             using ‹(take (high x n) treeList @ [VEBT-Insert.vebt-insert (treeList ! high x n) (low x
n)] @ drop (high x n + 1) treeList) ! i = treeList ! i› by presburger
          qed
          ultimately show ?thesis by auto
         qed
        next
          case False
          hence ?nextSummary = summary by simp
          hence ∃ y. both-member-options (treeList ! high x n) y
            using not-min-Null-member False by blast
          hence both-member-options summary (high x n)
            using 4 highlowprop by blast
          hence both-member-options (?nextTreeList ! high x n) ?l
           by (metis 0 2 Suc-eq-plus1 append-Cons append-Nil highlowprop nth-list-update-eq nth-mem
upd-conv-take-nth-drop valid-insert-both-member-options-add)
          then show ?thesis
            by (smt (verit, best) 2 4 False ‹both-member-options summary (high x n)› ‹i < 2 ^ m›
highlowprop nth-repl)
        qed
      qed
    qed
```

have *14*: (*mi* = *max x ma* ⟶ (∀ *t* ∈ *set ?nextTreeList*. ∄ *y*. *both-member-options t y*))
using *True max-less-iff-conj* **by** *blast*
have *15*: *mi* ≤ *max x ma* ∧ *max x ma* < *2^deg*
using *4.hyps*(*8*) *4.prems abcdef* **by** *auto*
have *16*: (*high* (*max x ma*) *n* = *i* ⟶
both-member-options (*?nextTreeList* ! *i*) (*low* (*max x ma*) *n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *max x ma*)
if *mi* ≠ *max x ma* **and** *i* < *2^m* **for** *i*
proof (*intro conjI impI allI*)
assume *high* (*max x ma*) *n* = *i*
show *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max x ma*) *n*)
proof (*cases high x n* = *high ma n*)
case *True*
have *invar-vebt* (*treeList* ! *i*) *n*
using *0 2* ‹*i* < *2* ^ *m*› **by** *simp*
have *length ?nextTreeList* = *2^m*
using *2 highlowprop* **by** *auto*
hence *?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *i*) (*low x n*)
using *concat-inth*[*of take* (*high x n*) *treeList vebt-insert* (*treeList* ! *i*) (*low x n*) *drop* (*high x n* + *1*) *treeList*]
2 True ‹*high* (*max x ma*) *n* = *i*› ‹*i* < *2* ^ *m*› *concat-inth length-take max-def*
by (*metis Suc-eq-plus1 append-Cons append-Nil nth-list-update-eq upd-conv-take-nth-drop*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low x n*) **using** *Un-iff* ‹*i* < *2* ^ *m*›
‹*invar-vebt* (*treeList* ! *i*) *n*› *both-member-options-equiv-member highlowprop*
list.set-intros(*1*) *set-append valid-insert-both-member-options-add*
by (*metis 11 True* ‹*high* (*max x ma*) *n* = *i*› *max-def*)
then show *?thesis* **proof**(*cases mi* = *ma*)
case *True*
then show *?thesis*
by (*metis* ‹(*take* (*high x n*) *treeList* @ [*VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] @ *drop* (*high x n* + *1*) *treeList*) ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low x n*)› ‹*mi* ≠ *max x ma*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def valid-insert-both-member-options-add*)
next
case *False*
hence *vebt-member* (*treeList* ! *i*) (*low ma n*)
by (*metis 7 True* ‹*high* (*max x ma*) *n* = *i*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *both-member-options-equiv-member highlowprop linorder-cases max.absorb3 max.absorb4 mimaxrel*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low ma n*) ∨ (*low ma n* = *low x n*)
using *post-member-pre-member*[*of* (*treeList* ! *i*) *n low x n low ma n*]
by (*metis 2 4.IH*(*1*) ‹(*take* (*high x n*) *treeList* @ [*VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] @ *drop* (*high x n* + *1*) *treeList*) ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low x n*)› ‹*i* < *2* ^ *m*› *both-member-options-equiv-member highlowprop member-bound nth-mem valid-insert-both-member-options-pres*)
then show *?thesis*
by (*metis 2 4.IH*(*1*) *True* ‹(*take* (*high x n*) *treeList* @ [*VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] @ *drop* (*high x n* + *1*) *treeList*) ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low x n*)› ‹*high* (*max x ma*) *n* = *i*› *both-member-options-equiv-member highlowprop max-def nth-mem valid-insert-both-member-options-add*)

```
      qed
    next
      case False
      then show ?thesis
      proof(cases x < ma)
        case True
        then show ?thesis
          by (metis 2 7 False ‹high (max x ma) n = i› highlowprop max.absorb4 nth-repl that(1,2))
      next
        case False
        hence x > ma
          using mimaxrel nat-neq-iff by blast
        then show ?thesis
       by (metis 2 4.IH(1) One-nat-def ‹high (max x ma) n = i› add.right-neutral add-Suc-right ap-
pend-Cons highlowprop max.strict-order-iff nth-list-update-eq nth-mem self-append-conv2 upd-conv-take-nth-drop
valid-insert-both-member-options-add)
      qed
    qed
  next
    fix y
    assume bb:high y n = i ∧ both-member-options (?nextTreeList ! i) (low y n)
    show mi < y and y ≤ max x ma
      unfolding atomize-conj
    proof(cases i = high x n)
      case True
      hence cc: i = high x n by simp
      have invar-vebt (treeList ! i ) n
        using 0 2 ‹i < 2 ^ m› by simp
      have length ?nextTreeList = 2^m
        using 2 highlowprop by auto
      hence aa:?nextTreeList ! i = vebt-insert (treeList ! i) (low x n)
        using concat-inth[of take (high x n) treeList vebt-insert (treeList ! i) (low x n) drop (high x
n + 1) treeList]
          by (metis 2 Suc-eq-plus1 append-Cons append-self-conv2 cc highlowprop nth-list-update-eq
upd-conv-take-nth-drop)
      hence invar-vebt (?nextTreeList ! i) n
        by (simp add: 11 True)
      hence vebt-member (treeList ! i) (low y n) ∨ (low y n) = (low x n)
     by (metis ‹invar-vebt (treeList ! i) n› aa bb highlowprop member-bound post-member-pre-member
valid-member-both-member-options)
      then show mi < y ∧ y ≤ max x ma
      proof(cases low y n = low x n)
        case True
        hence high x n = high y n ∧ low y n = low x n
          by (simp add: bb cc)
        hence x = y
          by (metis bit-split-inv)
        then show ?thesis
          using abcdef by auto
```

```
        next
          case False
          hence vebt-member (treeList ! i) (low y n)
            using ‹vebt-member (treeList ! i) (low y n) ∨ low y n = low x n› by blast
          hence mi ≠ ma  using 5 inthall
            by (metis 2 ‹i < 2 ^ m› min-Null-member not-min-Null-member)
          then show ?thesis
            by (metis 2 4.IH(1) 7 False True
           ‹vebt-member (treeList ! i) (low y n) ∨ low y n = low x n› bb both-member-options-equiv-member
highlowprop le-max-iff-disj nth-mem)
        qed
      next
        case False
        have invar-vebt (treeList ! i ) n
          using 0 2 ‹i < 2 ^ m› by simp
        have length ?nextTreeList = 2^m
          using 2 highlowprop by auto
        hence aa:?nextTreeList ! i = (treeList ! i)
          by (metis 2 False ‹i < 2 ^ m› highlowprop nth-repl)
        hence both-member-options (treeList !i) (low y n)
          using bb by auto
        hence mi ≠ ma  using 5 2 ‹i < 2 ^ m› by force
        then show mi < y ∧ y ≤ max x ma using 7
          using ‹both-member-options (treeList ! i) (low y n)› bb le-max-iff-disj that(2) by blast
      qed
    qed

    have 17: ?nextTreeList = treeList[high x n := vebt-insert (treeList ! high x n) (low x n)]
      by (simp add: 2 highlowprop upd-conv-take-nth-drop)

    have invar-vebt (Node (Some (mi, ?maxnew)) deg ?nextTreeList ?nextSummary) deg
      by (rule invar-vebt.intros(4)[where n = n and m = m])
        (metis 2 3 8 11 12 13 14 15 16 17 length-list-update)+

    then show ?thesis
      unfolding 10 17 .
  next
    case False
    hence abcdef: x < mi
      using antisym-conv3 mimaxrel by blast
    let ?h = high mi n and ?l=low mi n
    have highlowprop: high mi n < 2^m ∧ low mi n < 2^n
      using 1 3 4.hyps(3) 4.hyps(7) 4.hyps(8) deg-not-0 exp-split-high-low(1) exp-split-high-low(2)
le-less-trans by blast
    have 10:vebt-insert (Node (Some (mi,ma)) deg  treeList summary) x =
            Node (Some (x, max mi ma)) deg  (treeList[?h:=vebt-insert (treeList ! ?h) ?l])
                  (if minNull (treeList ! ?h) then  vebt-insert summary ?h else summary)
      by (metis 1 2 4.hyps(3) 9 abcdef deg-not-0 div-greater-zero-iff highlowprop insert-simp-excp
mimaxrel)
```

```
    let ?maxnew = max mi ma and ?nextTreeList = (treeList[ ?h :=vebt-insert (treeList ! ?h) ?l])
and
      ?nextSummary = (if minNull (treeList ! ?h) then  vebt-insert summary ?h else summary)
    have 11: ( ∀  t ∈ set ?nextTreeList. invar-vebt t n) proof
     fix t
     assume  t ∈ set ?nextTreeList
     then obtain i where ?nextTreeList ! i = t ∧ i < 2^m
       by (metis 2 in-set-conv-nth length-list-update)
     show invar-vebt t n
       by (metis 2 4.IH(1) ‹treeList[high mi n := VEBT-Insert.vebt-insert (treeList ! high mi n)
(low mi n)] ! i = t ∧ i < 2 ^ m› highlowprop nth-list-update-eq nth-list-update-neq nth-mem)
    qed
    have 12: invar-vebt ?nextSummary n
     using 1 4.IH(2) 8 highlowprop by presburger
   have 13: ∀  i < 2^m. (∃  y. both-member-options (?nextTreeList ! i) y) ⟷ ( both-member-options
?nextSummary i)
    proof
     fix i
    show i < 2 ^ m ⟶ (∃ y. both-member-options   ((?nextTreeList) ! i) y) = both-member-options
(?nextSummary) i
     proof
       assume i< 2^m
     show (∃ y. both-member-options ((?nextTreeList) ! i) y) = both-member-options (?nextSummary)
i
       proof(cases minNull (treeList ! high mi n))
         case True
         hence tc: minNull (treeList ! high mi n) by simp
         hence nsprop: ?nextSummary = vebt-insert summary ?h by simp
         have insprop:?nextTreeList ! ?h = vebt-insert (treeList ! ?h) ?l
           by (simp add: 2 highlowprop)
         then show ?thesis
         proof(cases i = ?h)
           case True
           have 161:∄  y. vebt-member (treeList ! ?h) y
             by (simp add: min-Null-member tc)
           hence 162:∄  y. both-member-options (treeList ! ?h) y
             by (metis 2 4.IH(1) highlowprop nth-mem valid-member-both-member-options)
           hence 163:¬ both-member-options summary i
             using 11 2 4 True ‹i < 2 ^ m› by blast
           have 164:?nextTreeList ! i = vebt-insert (treeList ! ?h) ?l
             using True insprop by simp
           have 165:invar-vebt (vebt-insert (treeList ! ?h) ?l) n
             by (simp add: 2 4.IH(1) highlowprop)
           have 166:both-member-options (vebt-insert (treeList ! ?h) ?l) ?l using myIHs[of treeList !
?h ?l]
             using 0 2 highlowprop apply auto
             apply (subst valid-insert-both-member-options-add)
               apply auto
             done
```

```
          have 167:∃ y. both-member-options ((?nextTreeList) ! i) y
            using 164 166 by auto
          then show ?thesis
            using 1 11 2 True nsprop valid-insert-both-member-options-add highlowprop by auto
        next
          case False
          have ?nextTreeList ! i = treeList ! i
            using False by fastforce
              have fstprop:both-member-options ((?nextTreeList) ! i) y ⟹ both-member-options
(?nextSummary) i  for y
               using 1 4 ‹i < 2 ^ m› ‹treeList[high mi n := VEBT-Insert.vebt-insert (treeList ! high
mi n) (low mi n)] ! i = treeList ! i› highlowprop valid-insert-both-member-options-pres by auto
          moreover  have both-member-options (?nextSummary) i ⟹ ∃ y . both-member-options
((?nextTreeList) ! i) y
          proof−
            assume  both-member-options (?nextSummary) i
            moreover have i ≠ high mi n
              by (simp add: False)
            ultimately have both-member-options summary i
              by (smt (verit, best) 1 12 ‹i < 2 ^ m› both-member-options-equiv-member highlowprop
post-member-pre-member)
            hence ∃ y. both-member-options (treeList ! i) y
              by (simp add: 4 ‹i < 2 ^ m›)
            then show ?thesis
              by (simp add: ‹treeList[high mi n := VEBT-Insert.vebt-insert (treeList ! high mi n)
(low mi n)] ! i = treeList ! i›)
          qed
          ultimately show ?thesis by auto
        qed
      next
        case False
        hence ?nextSummary = summary by simp
        hence ∃ y. both-member-options (treeList ! high mi n) y
          using not-min-Null-member False by blast
        hence both-member-options summary (high mi n)
          using 4 highlowprop by blast
        hence  both-member-options (?nextTreeList ! high mi n) ?l
        by (metis 0 2 highlowprop nth-list-update-eq nth-mem valid-insert-both-member-options-add)
        then show ?thesis
          by (metis (full-types, opaque-lifting) 4 False ‹both-member-options summary (high mi n)›
‹i < 2 ^ m› nth-list-update-neq)
      qed
    qed
  qed
  have 14: (x = max ma mi ⟶ (∀ t ∈ set ?nextTreeList. ∄ y. both-member-options t y))
    using mimaxrel by linarith
  have 15: x ≤ max ma mi  ∧ max ma mi < 2^deg
    using 6 abcdef by linarith
  have 16:  (x ≠ max ma mi ⟶ (∀ i < 2^m. (high (max ma mi) n = i ⟶ both-member-options
```

(*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)))
proof
assume *x* ≠ *max ma mi*
show (∀ *i* < 2^*m*. (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*))
proof
fix *i*::*nat*
show *i* < 2 ^ *m*⟶
(*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
assume *i* < 2^*m*
show (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
show (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*))
proof
assume *high* (*max ma mi*) *n* = *i*
show *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)
proof(*cases high mi n* = *high ma n*)
case *True*
have *invar-vebt* (*treeList* ! *i*) *n*
using *0 2* ‹*i* < 2 ^ *m*› **by** *simp*
have *length ?nextTreeList* = 2^*m*
using *2 highlowprop* **by** *auto*
hence *?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *i*) (*low mi n*)
using *concat-inth*[*of take* (*high x n*) *treeList vebt-insert* (*treeList* ! *i*) (*low x n*) *drop* (*high x n* + *1*) *treeList*]
by (*metis 2 True* ‹*high* (*max ma mi*) *n* = *i*› *highlowprop max-def nth-list-update-eq*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low mi n*)
by (*metis 11 2 True* ‹*high* (*max ma mi*) *n* = *i*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def set-update-memI valid-insert-both-member-options-add valid-member-both-member-options*)
then show *?thesis*
proof(*cases mi* = *ma*)
case *True*
then show *?thesis*
using ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low mi n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop valid-insert-both-member-options-add* **by** *force*
next

case *False*
hence *vebt-member* (*treeList* ! *i*) (*low ma n*)
using *6 7* ‹*high* (*max ma mi*) *n* = *i*› ‹*i* < *2* ^ *m*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *both-member-options-equiv-member* **by** *auto*
hence *vebt-member* (*?nextTreeList* ! *i*) (*low ma n*) ∨ (*low ma n* = *low mi n*)
using *post-member-pre-member*[*of* (*treeList* ! *i*) *n low mi n low ma n*]
by (*metis 11 2 7 True* ‹*high* (*max ma mi*) *n* = *i*› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low mi n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def member-bound set-update-memI valid-insert-both-member-options-pres valid-member-both-member-options*)
then show *?thesis*
by (*metis 11 2 4.hyps*(*7*) *7 False True* ‹*high* (*max ma mi*) *n* = *i*› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low mi n*)› *both-member-options-equiv-member highlowprop less-irrefl max.commute max-def set-update-memI*)
qed
next
case *False*
hence *?nextTreeList* ! *i* = *treeList* ! *i*
by (*metis 4.hyps*(*7*) ‹*high* (*max ma mi*) *n* = *i*› *max.commute max-def nth-list-update-neq*)
then show *?thesis*
by (*metis 4.hyps*(*7*) *7 False* ‹*high* (*max ma mi*) *n* = *i*› ‹*i* < *2* ^ *m*› *max.orderE*)
qed
qed
show (∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
fix *y*
show *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*
proof
assume *bb*:*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)
show *x* < *y* ∧ *y* ≤ *max ma mi*
proof(*cases i* = *high mi n*)
case *True*
hence *cc*: *i* = *high mi n* **by** *simp*
have *invar-vebt* (*treeList* ! *i*) *n*
using *0 2* ‹*i* < *2* ^ *m*› **by** *simp*
have *length ?nextTreeList* = *2*^*m*
using *2 highlowprop* **by** *auto*
hence *aa*:*?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *i*) (*low mi n*)
using *concat-inth*[*of take* (*high x n*) *treeList vebt-insert* (*treeList* ! *i*) (*low x n*) *drop* (*high x n* + *1*) *treeList*]
by (*simp add*: *cc highlowprop*)
hence *invar-vebt* (*?nextTreeList* ! *i*) *n*
by (*simp add*: *2 4.IH*(*1*) *cc highlowprop*)
hence *vebt-member* (*treeList* ! *i*) (*low y n*) ∨ (*low y n*) = (*low mi n*)
by (*metis* ‹*invar-vebt* (*treeList* ! *i*) *n*› *aa bb both-member-options-equiv-member highlowprop member-bound post-member-pre-member*)

```
                then show ?thesis
                proof(cases low y n = low mi n)
                  case True
                  hence high mi n = high y n ∧ low y n = low mi n
                    by (simp add: bb cc)
                  hence mi = y
                    by (metis bit-split-inv)
                  then show ?thesis
                    using abcdef by auto
                next
                  case False
                  hence vebt-member (treeList ! i) (low y n)
                    using ‹vebt-member (treeList ! i) (low y n) ∨ low y n = low mi n› by blast
                  hence mi ≠ ma  using 5 inthall
                    by (metis 2 ‹i < 2 ^ m› min-Null-member not-min-Null-member)
                  then show ?thesis
                    using 7
                   by (metis ‹i < 2 ^ m› ‹vebt-member (treeList ! i) (low y n)› ‹invar-vebt (treeList !
i) n› abcdef bb both-member-options-equiv-member max.absorb1 max.strict-order-iff max-less-iff-conj)
                qed
              next
                case False
                have invar-vebt (treeList ! i ) n
                  using 0 2 ‹i < 2 ^ m› by simp
                have length ?nextTreeList = 2^m
                  using 2 highlowprop by auto
                hence aa:?nextTreeList ! i = (treeList ! i)
                  using False by auto
                hence both-member-options (treeList ! i) (low y n)
                  using bb by auto
                hence mi ≠ ma  using 5  2 ‹i < 2 ^ m› by force
                then show ?thesis using 7
                    by (metis ‹both-member-options (treeList ! i) (low y n)› ‹i < 2 ^ m› abcdef bb
max.absorb1 max.strict-order-iff max-less-iff-conj)
              qed
            qed
          qed
        qed
      qed
    qed
    then show ?thesis using invar-vebt.intros(4)[of ?nextTreeList n ?nextSummary m deg x max ma
mi]
      using 10 11 12 13 14 15 2 3 4.hyps(3,7) by force
  qed
 qed
next
 case (5 treeList n summary m deg mi ma)
 hence myIHs: x ∈ set treeList ⟹ invar-vebt x n ⟹ xa < 2 ^ n ⟹ invar-vebt (vebt-insert x xa)
```

```
n for x xa by simp
  hence 0: ( ∀ t ∈ set treeList. invar-vebt t n) and 1: invar-vebt summary m and 2:length treeList
= 2^m and 3: deg = n+m and
    4: (∀ i < 2^m. (∃ y. both-member-options (treeList ! i) y) ⟷ ( both-member-options summary
i)) and
    5: (mi = ma ⟶ (∀ t ∈ set treeList. ∄ y. both-member-options t y)) and 6:mi ≤ ma ∧ ma <
2^deg and
    7: (mi ≠ ma ⟶ (∀ i < 2^m. (high ma n = i ⟶ both-member-options (treeList ! i) (low ma n))
∧
                                        (∀ y. (high y n = i ∧ both-member-options (treeList ! i) (low y n) )
⟶ mi < y ∧ y ≤ ma)))
    and 8: Suc n = m and 9: deg div 2 = n
    using 5 add-self-div-2 apply blast+ by (simp add: 5.hyps(3) 5.hyps(4))

  have 2 ≤ deg
   by (metis 0 2 3 8 9 add-le-mono div-less dual-order.refl linorder-not-less numeral-2-eq-2 plus-1-eq-Suc
set-n-deg-not-0)

  show ?case
  proof(cases x = mi ∨ x = ma)
    case True
    show ?thesis
      unfolding insert-simp-mima[of x mi ma deg treeList summary, OF True ‹2 ≤ deg›]
      using invar-vebt.intros(5)[of treeList n summary m deg mi ma]
      using 0 1 2 3 4 5 5.hyps(3,7,8) 7 by argo
  next
    case False
    hence mimaxrel: x ≠ mi ∧ x ≠ ma by simp
    then show ?thesis
    proof(cases mi < x)
      case True
      hence abcdef: mi < x by simp
      let ?h = high x n and ?l=low x n
      have highlowprop: high x n < 2^m ∧ low x n < 2^n
     by (metis 1 2 3 5.IH(1) 5.prems div-eq-0-iff add-nonneg-eq-0-iff deg-not-0 div-exp-eq exp-split-high-low(2)
high-def not-one-le-zero one-add-one power-not-zero set-n-deg-not-0 zero-le-one zero-neq-one)
      have 10:vebt-insert (Node (Some (mi,ma)) deg treeList summary) x =
              Node (Some (mi, max x ma)) deg (treeList[?h :=vebt-insert (treeList ! ?h) ?l])
                      (if minNull (treeList ! ?h) then vebt-insert summary ?h else summary)
       using 2 9 False ‹2 ≤ deg› abcdef highlowprop insert-simp-norm by presburger
      let ?maxnew = max x ma and ?nextTreeList = (treeList[ ?h :=vebt-insert (treeList ! ?h) ?l])
and
        ?nextSummary = (if minNull (treeList ! ?h) then vebt-insert summary ?h else summary)
      have 11: ( ∀ t ∈ set ?nextTreeList. invar-vebt t n)
      proof
        fix t
        assume t ∈ set ?nextTreeList
        then obtain i where i <2^m ∧ ?nextTreeList ! i = t
          by (metis 2 in-set-conv-nth length-list-update)
```

show *invar-vebt t n*
by (*metis 2 5.IH(1)* ‹$i < 2 \hat{} m \wedge$ *treeList*[*high x n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! $i = t$› *highlowprop nth-list-update-eq nth-list-update-neq nth-mem*)
qed
have *12*: *invar-vebt ?nextSummary m*
by (*simp add*: *1 5.IH(2) highlowprop*)
have *13*: $\forall\ i < 2\hat{}m.$ ($\exists\ y.$ *both-member-options* (*?nextTreeList* ! *i*) *y*) $\longleftrightarrow$ (*both-member-options ?nextSummary i*)
proof
fix *i*
show $i < 2 \hat{} m \longrightarrow (\exists\, y.$ *both-member-options* ((*?nextTreeList*) ! *i*) *y*) = *both-member-options* (*?nextSummary*) *i*
proof
assume $i< 2\hat{}m$
show ($\exists\, y.$ *both-member-options* ((*?nextTreeList*) ! *i*) *y*) = *both-member-options* (*?nextSummary*) *i*
proof(*cases minNull* (*treeList* ! *high x n*))
case *True*
hence *tc*: *minNull* (*treeList* ! *high x n*) **by** *simp*
hence *nsprop*: *?nextSummary* = *vebt-insert summary ?h* **by** *simp*
have *insprop*:*?nextTreeList* ! *?h* = *vebt-insert* (*treeList* ! *?h*) *?l*
by (*simp add*: *2 highlowprop*)
then show *?thesis*
proof(*cases i* = *?h*)
case *True*
have *161*:$\nexists$ *y. vebt-member* (*treeList* ! *?h*) *y*
by (*simp add*: *min-Null-member tc*)
hence *162*:$\nexists$ *y. both-member-options* (*treeList* ! *?h*) *y*
by (*metis 0 2 highlowprop nth-mem valid-member-both-member-options*)
hence *163*:$\neg$ *both-member-options summary i*
using *11 2 4 True* ‹$i < 2 \hat{} m$› **by** *blast*
have *164*:*?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *?h*) *?l*
using *True insprop* **by** *simp*
have *165*:*invar-vebt* (*vebt-insert* (*treeList* ! *?h*) *?l*) *n*
by (*simp add*: *11 2 highlowprop set-update-memI*)
have *166*:*both-member-options* (*vebt-insert* (*treeList* ! *?h*) *?l*) *?l* **using** *myIHs*[*of treeList* ! *?h ?l*]
using *0 2 highlowprop valid-insert-both-member-options-add* [*of* ‹*treeList* ! *high x n*› ‹$2 \hat{} n$› ‹*low x n*›]
by *auto* (*metis nth-mem valid-insert-both-member-options-add*)
have *167*:$\exists$ *y. both-member-options* ((*?nextTreeList*) ! *i*) *y*
using *164 166* **by** *auto*
then show *?thesis*
using *1 11 2 True nsprop valid-insert-both-member-options-add highlowprop* **by** *auto*
next
case *False*
have *?nextTreeList* ! *i* = *treeList* ! *i*
using *False* **by** *auto*
have *fstprop*:*both-member-options* ((*?nextTreeList*) ! *i*) *y* $\Longrightarrow$ *both-member-options*

```
(?nextSummary) i  for y
            using 1 4 ‹i < 2 ^ m› ‹treeList[high x n := VEBT-Insert.vebt-insert (treeList ! high x
n) (low x n)] ! i = treeList ! i› highlowprop valid-insert-both-member-options-pres by auto
          moreover  have both-member-options (?nextSummary) i ⟹ ∃ y . both-member-options
((?nextTreeList) ! i) y
          proof−
            assume  both-member-options (?nextSummary) i
            moreover have i ≠ high x n
              by (simp add: False)
            ultimately have both-member-options summary i
              by (smt (verit, best) 1 12 ‹i < 2 ^ m› both-member-options-equiv-member highlowprop
post-member-pre-member)
            hence ∃ y. both-member-options (treeList ! i) y
              by (simp add: 4 ‹i < 2 ^ m›)
            then show ?thesis
              by (simp add: ‹treeList[high x n := VEBT-Insert.vebt-insert (treeList ! high x n) (low
x n)] ! i = treeList ! i›)
          qed
          ultimately show ?thesis by auto
        qed
      next
        case False
        hence ?nextSummary = summary by simp
        hence ∃ y. both-member-options (treeList ! high x n) y
          using not-min-Null-member False by blast
        hence both-member-options summary (high x n)
          using 4 highlowprop by blast
        hence  both-member-options (?nextTreeList ! high x n) ?l
        by (metis 0 2 highlowprop nth-list-update-eq nth-mem valid-insert-both-member-options-add)
        then show ?thesis
          by (metis (full-types) 4 False ‹both-member-options summary (high x n)› ‹i < 2 ^ m›
nth-list-update-neq)
      qed
    qed
  qed
  have 14: (mi = max ma x ⟶ (∀ t ∈ set ?nextTreeList. ∄ y. both-member-options t y))
    using True max-less-iff-conj by blast
  have 15: mi ≤ max ma x  ∧ max ma x < 2^deg
    using 5.hyps(8) 5.prems abcdef by auto
  have 16:  (mi ≠ max ma x ⟶ (∀ i < 2^m. (high (max ma x) n = i ⟶ both-member-options
(?nextTreeList ! i) (low (max ma x) n)) ∧
                                (∀ y. (high y n = i ∧ both-member-options (?nextTreeList ! i) (low y
n)  ) ⟶ mi < y ∧ y ≤ max ma x)))
  proof
    assume mi ≠ max ma x
    show (∀ i < 2^m. (high (max ma x) n = i ⟶ both-member-options (?nextTreeList ! i) (low
(max ma x) n)) ∧
                                (∀ y. (high y n = i ∧ both-member-options (?nextTreeList ! i) (low y
n)  ) ⟶ mi < y ∧ y ≤ max ma x))
```

proof
fix *i*::*nat*
show $i < 2 \hat{} m \longrightarrow$
(*high* (*max ma x*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma x*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *mi* < *y* ∧ *y* ≤ *max ma x*)
proof
assume $i < 2\hat{}m$
show (*high* (*max ma x*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma x*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *mi* < *y* ∧ *y* ≤ *max ma x*)
proof
show (*high* (*max ma x*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma x*) *n*))
proof
assume *high* (*max ma x*) *n* = *i*
show *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma x*) *n*)
proof(*cases high x n* = *high ma n*)
case *True*
have *invar-vebt* (*treeList* ! *i*) *n*
using *0 2* ‹*i* < *2* ^ *m*› **by** *simp*
have *length ?nextTreeList* = $2\hat{}m$
using *2 highlowprop* **by** *auto*
hence *?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *i*) (*low x n*)
using *concat-inth*[*of take* (*high x n*) *treeList vebt-insert* (*treeList* ! *i*) (*low x n*) *drop* (*high x n* + *1*) *treeList*]
by (*metis 2 False True* ‹*high* (*max ma x*) *n* = *i*› *highlowprop linorder-neqE-nat max.commute max.strict-order-iff nth-list-update-eq*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low x n*)
by (*metis 11 2 True* ‹*high* (*max ma x*) *n* = *i*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def set-update-memI valid-insert-both-member-options-add valid-member-both-member-options*)
then show *?thesis*
proof(*cases mi* = *ma*)
case *True*
then show *?thesis*
by (*metis* ‹*treeList*[*high x n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low x n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *abcdef highlowprop max.commute max.strict-order-iff valid-insert-both-member-options-add*)
next
case *False*
hence *vebt-member* (*treeList* ! *i*) (*low ma n*)
by (*metis 7 True* ‹*high* (*max ma x*) *n* = *i*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def valid-member-both-member-options*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low ma n*) ∨ (*low ma n* = *low x n*)
using *post-member-pre-member*[*of* (*treeList* ! *i*) *n low x n low ma n*]
by (*metis 1 11 2 3 5.hyps*(*8*) *7 False True* ‹*treeList*[*high x n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high x n*) (*low x n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low x n*)› ‹*invar-vebt*

(treeList ! i) n› deg-not-0 exp-split-high-low(2) highlowprop nth-list-update-neq set-update-memI valid-insert-both-member valid-member-both-member-options)

then show *?thesis*

by (*metis 11 2 True ‹high (max ma x) n = i› ‹treeList[high x n := VEBT-Insert.vebt-insert (treeList ! high x n) (low x n)] ! i = VEBT-Insert.vebt-insert (treeList ! i) (low x n)› ‹invar-vebt (treeList ! i) n› both-member-options-equiv-member highlowprop max-def set-update-memI valid-insert-both-member-options-add*)

qed

next

case *False*

then show *?thesis*

by (*metis 0 2 7 ‹high (max ma x) n = i› ‹i < 2 ^ m› ‹mi ≠ max ma x› highlowprop max-def nth-list-update-eq nth-list-update-neq nth-mem valid-insert-both-member-options-add*)

qed

qed

show (∀ *y*. *high y n = i ∧ both-member-options (?nextTreeList ! i) (low y n) ⟶ mi < y ∧ y ≤ max ma x*)

proof

fix *y*

show *high y n = i ∧ both-member-options (?nextTreeList ! i) (low y n) ⟶ mi < y ∧ y ≤ max ma x*

proof

assume *bb*:*high y n = i ∧ both-member-options (?nextTreeList ! i) (low y n)*

show *mi < y ∧ y ≤ max ma x*

proof(*cases i = high x n*)

case *True*

hence *cc*: *i = high x n* **by** *simp*

have *invar-vebt (treeList ! i) n*

using *0 2 ‹i < 2 ^ m›* **by** *simp*

have *length ?nextTreeList = 2^m*

using *2 highlowprop* **by** *auto*

hence *aa*:*?nextTreeList ! i = vebt-insert (treeList ! i) (low x n)*

using *concat-inth*[*of take (high x n) treeList vebt-insert (treeList ! i) (low x n) drop (high x n + 1) treeList*]

by (*simp add*: *cc highlowprop*)

hence *invar-vebt (?nextTreeList ! i) n*

by (*simp add*: *2 5.IH(1) cc highlowprop*)

hence *vebt-member (treeList ! i) (low y n) ∨ (low y n) = (low x n)*

by (*metis ‹high y n = i ∧ both-member-options ((treeList*[*?h*:=*vebt-insert (treeList ! high x n) (low x n)*]) *! i) (low y n)› ‹invar-vebt (treeList ! i) n› aa highlowprop member-bound post-member-pre-member valid-member-both-member-options*)

then show *?thesis*

proof(*cases low y n = low x n*)

case *True*

hence *high x n = high y n ∧ low y n = low x n*

by (*simp add*: *bb cc*)

hence *x = y*

by (*metis bit-split-inv*)

```
                then show ?thesis
                  using abcdef by auto
              next
                case False
                hence vebt-member (treeList ! i) (low y n)
                  using ‹vebt-member (treeList ! i) (low y n) ∨ low y n = low x n› by blast
                hence mi ≠ ma  using 5 inthall
                  by (metis 2 ‹i < 2 ^ m› min-Null-member not-min-Null-member)
                then show ?thesis
                  using 7 ‹i < 2 ^ m› ‹vebt-member (treeList ! i) (low y n)› ‹invar-vebt (treeList !
i) n› bb both-member-options-equiv-member max.coboundedI1 by blast
              qed
            next
              case False
              have invar-vebt (treeList ! i ) n
                using 0 2 ‹i < 2 ^ m› by simp
              have length ?nextTreeList = 2^m
                using 2 highlowprop by auto
              hence aa:?nextTreeList ! i = (treeList ! i)
                using False by auto
              hence both-member-options (treeList ! i) (low y n)
                using bb by auto
              hence mi ≠ ma  using 5
                using 2 ‹i < 2 ^ m› by fastforce
              then show ?thesis using 7
              using ‹both-member-options (treeList ! i) (low y n)› ‹i < 2 ^ m› bb max.coboundedI1
by blast
            qed
          qed
        qed
      qed
    qed
  qed
  show ?thesis
    unfolding 10 max.commute[of x ma]
    using invar-vebt.intros(5)[of ?nextTreeList  n ?nextSummary m deg mi max ma x]
    using 11 12 13 14 15 16 2 3 5.hyps(3) by fastforce
next
  case False
  hence abcdef: x < mi
    using antisym-conv3 mimaxrel by blast
  let ?h = high mi n and ?l=low mi n
  have highlowprop: high mi n < 2^m ∧ low mi n < 2^n
      by (metis (full-types) 1 2 3 5.IH(1) 5.hyps(7) 5.hyps(8) deg-not-0 exp-split-high-low(1)
exp-split-high-low(2) le-less-trans not-one-le-zero set-n-deg-not-0)
  have 10:vebt-insert (Node (Some (mi,ma)) deg  treeList summary) x =
          Node (Some (x, max mi ma)) deg  (treeList[ ?h :=vebt-insert (treeList ! ?h) ?l])
                  (if minNull (treeList ! ?h) then  vebt-insert summary ?h else summary)
```

by (*metis 0 2 9 abcdef div-less highlowprop insert-simp-excp mimaxrel not-less not-one-le-zero set-n-deg-not-0*)
let *?maxnew* = *max mi ma* **and** *?nextTreeList* = (*treeList*[*?h*:=*vebt-insert* (*treeList* ! *?h*) *?l*]) **and**
?nextSummary = (*if minNull* (*treeList* ! *?h*) *then vebt-insert summary ?h else summary*)
have *11*: (∀ *t* ∈ *set ?nextTreeList*. *invar-vebt t n*)
proof
fix *t*
assume *t* ∈ *set ?nextTreeList*
then obtain *i* **where** *i* < 2^*m* ∧ *?nextTreeList* ! *i* = *t*
by (*metis 2 in-set-conv-nth length-list-update*)
thus *invar-vebt t n*
by (*metis 2 5.IH*(*1*) *highlowprop nth-list-update-eq nth-list-update-neq nth-mem*)
qed
have *12*: *invar-vebt ?nextSummary m*
by (*simp add*: *1 5.IH*(*2*) *highlowprop*)
have *13*: ∀ *i* < 2^*m*. (∃ *y*. *both-member-options* (*?nextTreeList* ! *i*) *y*) ⟷ (*both-member-options ?nextSummary i*)
proof
fix *i*
show *i* < 2 ^ *m* ⟶ (∃ *y*. *both-member-options* ((*?nextTreeList*) ! *i*) *y*) = *both-member-options* (*?nextSummary*) *i*
proof
assume *i*< 2^*m*
show (∃ *y*. *both-member-options* ((*?nextTreeList*) ! *i*) *y*) = *both-member-options* (*?nextSummary*) *i*
proof(*cases minNull* (*treeList* ! *high mi n*))
case *True*
hence *tc*: *minNull* (*treeList* ! *high mi n*) **by** *simp*
hence *nsprop*: *?nextSummary* = *vebt-insert summary ?h* **by** *simp*
have *insprop*:*?nextTreeList* ! *?h* = *vebt-insert* (*treeList* ! *?h*) *?l*
by (*simp add*: *2 highlowprop*)
then show *?thesis*
proof(*cases i* = *?h*)
case *True*
have *161*:∄ *y*. *vebt-member* (*treeList* ! *?h*) *y*
by (*simp add*: *min-Null-member tc*)
hence *162*:∄ *y*. *both-member-options* (*treeList* ! *?h*) *y*
by (*metis 0 2 highlowprop nth-mem valid-member-both-member-options*)
hence *163*:¬ *both-member-options summary i*
using *11 2 4 True* ‹*i* < 2 ^ *m*› **by** *blast*
have *164*:*?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *?h*) *?l*
using *True insprop* **by** *simp*
have *165*:*invar-vebt* (*vebt-insert* (*treeList* ! *?h*) *?l*) *n*
by (*simp add*: *11 2 highlowprop set-update-memI*)
have *166*:*both-member-options* (*vebt-insert* (*treeList* ! *?h*) *?l*) *?l* **using** *myIHs*[*of treeList* ! *?h ?l*]
using *0 2 highlowprop*
apply (*subst valid-insert-both-member-options-add*) **apply** *auto*

```
        done
      have 167:∃ y. both-member-options ((?nextTreeList) ! i) y
        using 164 166 by auto
      then show ?thesis
        using 1 11 2 True nsprop valid-insert-both-member-options-add highlowprop by auto
    next
      case False
      have ?nextTreeList ! i = treeList ! i
        by (metis False nth-list-update-neq)
          have fstprop:both-member-options ((?nextTreeList) ! i) y ⟹ both-member-options
(?nextSummary) i for y
           using 1 4 ‹i < 2 ^ m› ‹treeList[high mi n := VEBT-Insert.vebt-insert (treeList ! high
mi n) (low mi n)] ! i = treeList ! i› highlowprop valid-insert-both-member-options-pres by auto
      moreover have both-member-options (?nextSummary) i ⟹ ∃ y . both-member-options
((?nextTreeList) ! i) y
      proof−
        assume both-member-options (?nextSummary) i
        moreover have i ≠ high mi n
          by (simp add: False)
        ultimately have both-member-options summary i
          by (smt (verit, best) 1 12 ‹i < 2 ^ m› both-member-options-equiv-member highlowprop
post-member-pre-member)
        hence ∃ y. both-member-options (treeList ! i) y
          by (simp add: 4 ‹i < 2 ^ m›)
        then show ?thesis
          by (simp add: ‹treeList[high mi n := VEBT-Insert.vebt-insert (treeList ! high mi n)
(low mi n)] ! i = treeList ! i›)
      qed
      ultimately show ?thesis by auto
    qed
  next
    case False
    hence ?nextSummary = summary by simp
    hence ∃ y. both-member-options (treeList ! high mi n) y
      using not-min-Null-member False by blast
    hence both-member-options summary (high mi n)
      using 4 highlowprop by blast
    hence both-member-options (?nextTreeList ! high mi n) ?l
    by (metis 0 2 highlowprop nth-list-update-eq nth-mem valid-insert-both-member-options-add)
    then show ?thesis
      by (metis (full-types) 4 False ‹both-member-options summary (high mi n)› ‹i < 2 ^ m›
nth-list-update-neq)
  qed
 qed
qed
have 14: (x = max ma mi ⟶ (∀ t ∈ set ?nextTreeList. ∄ y. both-member-options t y))
  using mimaxrel by linarith
have 15: x ≤ max ma mi ∧ max ma mi < 2^deg
  using 6 abcdef by linarith
```

have *16*: (*x* ≠ *max ma mi* ⟶ (∀ *i* < 2^*m*. (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)))
proof
assume *x* ≠ *max ma mi*
show (∀ *i* < 2^*m*. (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*)) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*))
proof
fix *i*::*nat*
show *i* < *2* ^ *m*⟶
(*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
assume *i* < 2^*m*
show (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)) ∧
(∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
show (*high* (*max ma mi*) *n* = *i* ⟶ *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*))
proof
assume *high* (*max ma mi*) *n* = *i*
show *both-member-options* (*?nextTreeList* ! *i*) (*low* (*max ma mi*) *n*)
proof(*cases high mi n* = *high ma n*)
case *True*
have *invar-vebt* (*treeList* ! *i*) *n*
using *0 2* ‹*i* < *2* ^ *m*› **by** *simp*
have *length ?nextTreeList* = *2*^*m*
using *2 highlowprop* **by** *auto*
hence *?nextTreeList* ! *i* = *vebt-insert* (*treeList* ! *i*) (*low mi n*)
using *concat-inth*[*of take* (*high x n*) *treeList vebt-insert* (*treeList* ! *i*) (*low x n*) *drop* (*high x n* + *1*) *treeList*]
by (*metis 2 5*.*hyps*(*7*) *True* ‹*high* (*max ma mi*) *n* = *i*› *highlowprop max*.*orderE nth-list-update-eq*)
hence *vebt-member* (*?nextTreeList* ! *i*) (*low mi n*)
by (*metis 11 2 True* ‹*high* (*max ma mi*) *n* = *i*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max-def set-update-memI valid-insert-both-member-options-add valid-member-both-member-options*)
then show *?thesis*
proof(*cases mi* = *ma*)
case *True*
then show *?thesis*
using ‹*treeList*[*high mi n* := *VEBT-Insert*.*vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert*.*vebt-insert* (*treeList* ! *i*) (*low mi n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop*

valid-insert-both-member-options-add **by** *auto*
next
case *False*
hence *vebt-member* (*treeList* ! *i*) (*low ma n*)
using *6 7* ‹*high* (*max ma mi*) *n* = *i*› ‹*i* < *2* ^ *m*› ‹*invar-vebt* (*treeList* ! *i*) *n*› *both-member-options-equiv-member* **by** *auto*
hence *vebt-member* (*?nextTreeList* ! *i*) (*low ma n*) ∨ (*low ma n* = *low mi n*)
using *post-member-pre-member*[*of* (*treeList* ! *i*) *n low mi n low ma n*]
by (*metis 1 11 2 3 5.hyps*(*8*) *7 True* ‹*high* (*max ma mi*) *n* = *i*› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low mi n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *deg-not-0 exp-split-high-low*(*2*) *highlowprop max-def set-update-memI valid-insert-both-member-options-pres valid-member-both-member-options*)
then show *?thesis*
by (*metis 5.hyps*(*7*) *7 False* ‹*high* (*max ma mi*) *n* = *i*› ‹*i* < *2* ^ *m*› ‹*vebt-member* (*treeList* ! *i*) (*low ma n*)› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *VEBT-Insert.vebt-insert* (*treeList* ! *i*) (*low mi n*)› ‹*invar-vebt* (*treeList* ! *i*) *n*› *highlowprop max.absorb1 member-bound valid-insert-both-member-options-pres*)
qed
next
case *False*
hence *?nextTreeList* ! *i* = *treeList* ! *i*
by (*metis 5.hyps*(*7*) ‹*high* (*max ma mi*) *n* = *i*› *max.commute max-def nth-list-update-neq*)
then show *?thesis*
proof(*cases mi* < *ma*)
case *True*
then show *?thesis*
by (*metis 5.hyps*(*7*) *7 False* ‹*high* (*max ma mi*) *n* = *i*› ‹*i* < *2* ^ *m*› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *treeList* ! *i*› *max.commute max-def*)
next
case *False*
hence *mi* ≥ *ma* **by** *simp*
hence *mi* = *ma*
by (*simp add*: *6 eq-iff*)
hence ¬*both-member-options* (*treeList* ! *i*) (*low* (*max ma mi*) *n*) **using** *5 2* ‹*i* < *2* ^ *m*› **by** *auto*
then show *?thesis*
by (*metis 11 2* ‹*high* (*max ma mi*) *n* = *i*› ‹*mi* = *ma*› ‹*treeList*[*high mi n* := *VEBT-Insert.vebt-insert* (*treeList* ! *high mi n*) (*low mi n*)] ! *i* = *treeList* ! *i*› *highlowprop max.idem nth-list-update-eq set-update-memI valid-insert-both-member-options-add*)
qed
qed
qed
show (∀ *y*. *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*)
proof
fix *y*
show *high y n* = *i* ∧ *both-member-options* (*?nextTreeList* ! *i*) (*low y n*) ⟶ *x* < *y* ∧ *y* ≤ *max ma mi*

```
proof
  assume bb:high y n = i ∧ both-member-options (?nextTreeList ! i) (low y n)
  show  x < y ∧ y ≤ max ma mi
  proof(cases i = high mi n)
    case True
    hence cc: i = high mi n by simp
    have invar-vebt (treeList ! i ) n
      using 0 2 ‹i < 2 ^ m› by simp
    have length ?nextTreeList = 2^m
      using 2 highlowprop by auto
    hence aa:?nextTreeList ! i = vebt-insert (treeList ! i) (low mi n)
      using concat-inth[of take (high x n) treeList vebt-insert (treeList ! i) (low x n) drop
(high x n + 1) treeList]
      by (simp add: cc highlowprop)
    hence invar-vebt (?nextTreeList ! i) n
      by (simp add: 2 5.IH(1) ‹i < 2 ^ m› highlowprop)
    hence vebt-member (treeList ! i) (low y n) ∨ (low y n) = (low mi n)
        by (metis ‹invar-vebt (treeList ! i) n› aa bb both-member-options-equiv-member
highlowprop member-bound post-member-pre-member)
    then show ?thesis
    proof(cases low y n = low mi n)
      case True
      hence high mi n = high y n ∧ low y n = low mi n
        by (simp add: bb cc)
      hence mi = y
        by (metis bit-split-inv)
      then show ?thesis
        using abcdef by auto
    next
      case False
      hence vebt-member (treeList ! i) (low y n)
        using ‹vebt-member (treeList ! i) (low y n) ∨ low y n = low mi n› by blast
      hence mi ≠ ma  using 5 inthall
        by (metis 2 ‹i < 2 ^ m› min-Null-member not-min-Null-member)
      then show ?thesis
        using 7
       by (metis ‹i < 2 ^ m› ‹vebt-member (treeList ! i) (low y n)› ‹invar-vebt (treeList !
i) n› abcdef bb both-member-options-equiv-member max.absorb1 max.strict-order-iff max-less-iff-conj)
    qed
  next
    case False
    have invar-vebt (treeList ! i ) n
      using 0 2 ‹i < 2 ^ m› by simp
    have length ?nextTreeList = 2^m
      using 2 highlowprop by auto
    hence aa:?nextTreeList ! i = (treeList ! i)
      using False by auto
    hence both-member-options (treeList ! i) (low y n)
      using bb by auto
```

```
            hence mi ≠ ma  using 5 2 ‹i < 2 ^ m› by fastforce
            then show ?thesis using 7
              by (metis ‹both-member-options (treeList ! i) (low y n)› ‹i < 2 ^ m› abcdef bb
max.absorb1 max.strict-order-iff max-less-iff-conj)
          qed
        qed
      qed
    qed
  qed
 qed
 then show ?thesis
  unfolding 10 max.commute[of mi ma]
  using invar-vebt.intros(5)[of ?nextTreeList n ?nextSummary m deg x max ma mi]
  using 11 12 13 14 15 2 3 5.hyps(3) by simp
 qed
qed
qed
```

4.2 Correctness with Respect to Set Interpretation

```
theorem insert-corr:
 assumes invar-vebt t n  and x < 2^n
 shows  set-vebt' t ∪ {x} = set-vebt' (vebt-insert t x)
proof
 show set-vebt' t ∪ {x} ⊆ set-vebt' (vebt-insert t x)
 proof
  fix y
  assume y ∈ set-vebt' t ∪ {x}
  show y ∈set-vebt' (vebt-insert t x)
  proof(cases x=y)
   case True
   then show ?thesis
     by (metis (full-types) assms(1) assms(2) both-member-options-equiv-member mem-Collect-eq
set-vebt'-def valid-insert-both-member-options-add valid-pres-insert)
  next
   case False
   have vebt-member t y
    using False ‹y ∈ set-vebt' t ∪ {x}› set-vebt'-def by auto
   hence vebt-member (vebt-insert t x) y
   by (meson assms(1) assms(2) both-member-options-equiv-member member-bound valid-insert-both-member-options-
valid-pres-insert)
   then show ?thesis
    by (simp add: set-vebt'-def)
  qed
 qed
 show  set-vebt' (vebt-insert t x) ⊆ set-vebt' t ∪ {x}
 proof
  fix y
```

```
    assume y ∈ set-vebt' (vebt-insert t x)
    show y ∈set-vebt' t ∪ {x}
    proof(cases x=y)
      case True
      then show ?thesis by simp
    next
      case False
      hence vebt-member t y ∨ x=y using post-member-pre-member
      using ‹y ∈ set-vebt' (vebt-insert t x)› assms(1) assms(2) set-vebt'-def member-bound valid-pres-insert
by fastforce
      hence vebt-member t y
        by (simp add: False)
      hence y ∈ set-vebt' t
        by (simp add: set-vebt'-def)
      then show ?thesis by simp
    qed
  qed
qed

corollary insert-correct: assumes invar-vebt t n and x < 2^n shows
   set-vebt t ∪ {x} = set-vebt (vebt-insert t x)
  using assms(1) assms(2) insert-corr set-vebt-set-vebt'-valid valid-pres-insert by blast

fun insert'::VEBT ⇒ nat ⇒ VEBT where
  insert' (Leaf a b) x = vebt-insert (Leaf a b) x|
  insert' (Node info deg treeList summary) x =
   (if x ≥ 2^deg then (Node info deg treeList summary )
                 else vebt-insert (Node info deg treeList summary) x)

theorem insert'-pres-valid: assumes invar-vebt t n shows invar-vebt (insert' t x) n
  using assms
  apply cases
  apply (metis One-nat-def deg1Leaf insert'.simps(1) vebt-insert.simps(1))
  apply (metis assms insert'.simps(2) leI valid-pres-insert)+
  done

theorem insert'-correct: assumes invar-vebt t n
  shows set-vebt (insert' t x) = (set-vebt t ∪ {x})∩{0..2^n−1}
proof(cases t)
  case (Node x11 x12 x13 x14)



  show ?thesis
  proof(cases x < 2^n)
    case True
    hence set-vebt (insert'  t x) = set-vebt(vebt-insert t x)
      by (metis Node assms deg-deg-n insert'.simps(2) leD)
    also have ... = set-vebt t ∪ {x}
```

```
    using True assms insert-correct by auto
  also have ... = (set-vebt t ∪ {x})∩{0..2^n−1}
    by (metis True assms inf.orderE inrange insert-corr set-vebt-set-vebt'-valid valid-pres-insert)
  finally show ?thesis .
 next
  case False
  hence set-vebt (insert' t x) = set-vebt t
    by (metis Node assms deg-deg-n insert'.simps(2) leI)
  also have set-vebt t = {0..2^n−1} ∩ set-vebt t
    by (metis assms inf-commute inrange le-iff-inf set-vebt-set-vebt'-valid)
  also have ... = {0..2^n−1} ∩ insert x (set-vebt t)
    by (rule Int-insert-right-if0[symmetric]) (use False not-less-eq-eq in auto)
  finally show ?thesis
    by blast
 qed
next
 case (Leaf x21 x22)
 then show ?thesis
  apply(auto simp add: insert'.simps vebt-insert.simps set-vebt-def both-member-options-def)
  using assms
    apply cases
    apply simp+
  using assms
    apply cases
    apply simp+
  using assms
    apply cases
    apply simp+
  using assms
    apply cases
    apply simp+
  done
qed

end
end

theory VEBT-MinMax imports VEBT-Member
begin
```

5 The Minimum and Maximum Operation

```
fun vebt-mint :: VEBT ⇒ nat option where
 vebt-mint (Leaf a b) = (if a then Some 0 else if b then Some 1 else None)|
 vebt-mint (Node None - - -) = None|
 vebt-mint (Node (Some (mi,ma)) - - - ) = Some mi


fun vebt-maxt :: VEBT ⇒ nat option where
```

vebt-maxt (*Leaf a b*) = (*if b then Some 1 else if a then Some 0 else None*)|
vebt-maxt (*Node None - - -*) = *None*|
vebt-maxt (*Node* (*Some* (*mi*,*ma*)) *- - -*) = *Some ma*

context *VEBT-internal* **begin**

fun *option-shift*::('*a*⇒'*a*⇒'*a*) ⇒'*a option* ⇒'*a option*⇒ '*a option* **where**
option-shift - None - = None|
option-shift - - None = None|
option-shift f (*Some a*) (*Some b*) = *Some* (*f a b*)

definition *power*::*nat option* ⇒ *nat option* ⇒ *nat option* (**infixl**‹$\widehat{}_o$› *81*) **where**
power= *option-shift* (^)

definition *add*::*nat option* ⇒ *nat option* ⇒ *nat option* (**infixl**‹$+_o$› *79*) **where**
add= *option-shift* (+)

definition *mul*::*nat option* ⇒ *nat option* ⇒ *nat option* (**infixl**‹$*_o$› *80*) **where**
mul = *option-shift* (∗)

fun *option-comp-shift*::('*a* ⇒ '*a* ⇒ *bool*) ⇒ '*a option* ⇒ '*a option* ⇒ *bool* **where**
option-comp-shift - None - = False|
option-comp-shift - - None = False|
option-comp-shift f (*Some x*) (*Some y*) = *f x y*

fun *less*::*nat option* ⇒ *nat option* ⇒ *bool* (**infixl**‹$<_o$› *80*) **where**
less x y= *option-comp-shift* (<) *x y*

fun *lesseq*::*nat option* ⇒ *nat option* ⇒ *bool* (**infixl**‹$\leq_o$› *80*) **where**
lesseq x y = *option-comp-shift* (≤) *x y*

fun *greater*::*nat option* ⇒ *nat option* ⇒ *bool* (**infixl**‹$>_o$› *80*) **where**
greater x y = *option-comp-shift* (>) *x y*

lemma *add-shift*:*x*+*y* = *z* ⟷ *Some x* +$_o$ *Some y* = *Some z*
by (*simp add*: *add-def*)

lemma *mul-shift*:*x*∗*y* = *z* ⟷ *Some x* ∗$_o$ *Some y* = *Some z* **by** (*simp add*: *mul-def*)

lemma *power-shift*:*x*^*y* = *z* ⟷ *Some x* ^$_o$ *Some y* = *Some z* **by** (*simp add*: *power-def*)

lemma *less-shift*: *x* < *y* ⟷ *Some x* <$_o$ *Some y* **by** *simp*

lemma *lesseq-shift*: *x* ≤ *y* ⟷ *Some x* ≤$_o$ *Some y* **by** *simp*

lemma *greater-shift*: *x* > *y* ⟷ *Some x* >$_o$ *Some y* **by** *simp*

definition *max-in-set* :: *nat set* ⇒ *nat* ⇒ *bool* **where**

max-in-set xs x $\longleftrightarrow$ $(x \in xs \wedge (\forall\ y \in xs.\ y \leq x))$

lemma *maxt-member*: *invar-vebt t n* $\Longrightarrow$ *vebt-maxt t = Some maxi* $\Longrightarrow$ *vebt-member t maxi*
proof(*induction t n arbitrary*: *maxi rule*: *invar-vebt.induct*)
case (*1 a b*)
  then show *?case*
   by (*metis VEBT-Member.vebt-member.simps*(*1*) *vebt-maxt.simps*(*1*) *option.distinct*(*1*) *option.inject zero-neq-one*)
next
case (*2 treeList n summary m deg*)
  then show *?case*
    by *simp*
next
case (*3 treeList n summary m deg*)
  then show *?case*
    by *simp*
next
  case (*4 treeList n summary m deg mi ma*)
  hence *deg* $\geq$ *2*
    by (*metis One-nat-def Suc-le-eq add-mono deg-not-0 numeral-2-eq-2 plus-1-eq-Suc*)
  then show *?case*
    by (*metis 4.prems VEBT-Member.vebt-member.simps*(*5*) *Suc-diff-Suc Suc-pred lessI less-le-trans vebt-maxt.simps*(*3*) *numeral-2-eq-2 option.inject zero-less-Suc*)
next
  case (*5 treeList n summary m deg mi ma*)
  hence *deg* $\geq$ *2*
     by (*metis Suc-leI le-add2 less-add-same-cancel2 less-le-trans not-less-iff-gr-or-eq not-one-le-zero numeral-2-eq-2 plus-1-eq-Suc set-n-deg-not-0*)
  then show *?case*
   by (*metis 5.prems VEBT-Member.vebt-member.simps*(*5*) *add-2-eq-Suc le-add-diff-inverse vebt-maxt.simps*(*3*) *option.inject*)
qed

lemma *maxt-corr-help*:
  assumes *invar-vebt t n vebt-maxt t = Some maxi vebt-member t x*
  shows *maxi* $\geq$ *x*
  using *assms*(*2*,*1*,*3*)
proof (*cases t Some maxi rule*: *vebt-maxt.elims*)
  case (*1 a b*)
  then show *?thesis*
    by (*metis assms*(*3*) *le0 le-numeral-extra*(*4*) *option.inject vebt-member.simps*(*1*))
next
  case (*2 uu uv uw*)
  then have *False*
    by *simp*
  then show *?thesis* **..**
next
  case (*3 mi ma ux uy uz*)

then show *?thesis*
by (*metis member-inv assms*(*1*,*3*) *le-eq-less-or-eq mi-ma-2-deg option.inject*)
qed

lemma *maxt-corr-help-empty*: *invar-vebt t n* $\Longrightarrow$ *vebt-maxt t* = *None* $\Longrightarrow$ *set-vebt' t* = {}
by (*metis* (*full-types*) *VEBT-Member.vebt-member.simps*(*1*) *empty-Collect-eq vebt-maxt.elims min-Null.simps*(*4*) *min-Null-member option.distinct*(*1*) *set-vebt'-def*)

theorem *maxt-corr*:**assumes** *invar-vebt t n* **and** *vebt-maxt t* = *Some x* **shows** *max-in-set* (*set-vebt' t*) *x*
unfolding *set-vebt'-def Max-def max-in-set-def*
using *assms*(*1*) *assms*(*2*) *maxt-corr-help maxt-member* **by** *blast*

theorem *maxt-sound*:**assumes** *invar-vebt t n* **and** *max-in-set* (*set-vebt' t*) *x* **shows** *vebt-maxt t* = *Some x*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *empty-Collect-eq le-less max-in-set-def maxt-corr-help maxt-corr-help-empty maxt-member mem-Collect-eq not-le option.exhaust set-vebt'-def*)

definition *min-in-set* :: *nat set* $\Rightarrow$ *nat* $\Rightarrow$ *bool* **where**
min-in-set xs x $\longleftrightarrow$ ($x \in xs \wedge (\forall\ y \in xs.\ y \geq x)$)

lemma *mint-member*: *invar-vebt t n* $\Longrightarrow$ *vebt-mint t* = *Some maxi* $\Longrightarrow$ *vebt-member t maxi*
proof(*induction t n arbitrary*: *maxi rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case*
by (*metis VEBT-Member.vebt-member.simps*(*1*) *vebt-mint.simps*(*1*) *option.distinct*(*1*) *option.inject zero-neq-one*)
next
case (*2 treeList n summary m deg*)
then show *?case*
by *simp*
next
case (*3 treeList n summary m deg*)
then show *?case*
by *simp*
next
case (*4 treeList n summary m deg mi ma*)
hence *deg* $\geq$ *2*
by (*metis One-nat-def Suc-le-eq add-mono deg-not-0 numeral-2-eq-2 plus-1-eq-Suc*)
then show *?case*
by (*metis 4*.*prems VEBT-Member.vebt-member.simps*(*5*) *One-nat-def Suc-diff-Suc Suc-pred dual-order.strict-trans1 le-imp-less-Suc le-numeral-extra*(*4*) *vebt-mint.simps*(*3*) *numeral-2-eq-2 option.inject zero-le-one*)
next
case (*5 treeList n summary m deg mi ma*)
hence *deg* $\geq$ *2*
by (*metis Suc-leI le-add2 less-add-same-cancel2 less-le-trans not-less-iff-gr-or-eq not-one-le-zero numeral-2-eq-2 plus-1-eq-Suc set-n-deg-not-0*)

then show *?case* **using** *5.prems VEBT-Member.vebt-member.simps(5) add-2-eq-Suc le-add-diff-inverse vebt-mint.simps(3)*
by (*metis option.inject*)
qed

lemma *mint-corr-help*:
assumes *invar-vebt t n vebt-mint t = Some mini vebt-member t x*
shows *mini* ≤ *x*
using *assms(2,1,3)*
proof (*cases t Some mini rule*: *vebt-mint.elims*)
case (*1 a b*)
then show *?thesis*
by (*metis assms(3) le-numeral-extra(3,4) option.inject vebt-member.simps(1) zero-le-one*)
next
case (*2 uu uv uw*)
then have *False*
by *simp*
then show *?thesis* **..**
next
case (*3 mi ma ux uy uz*)
then show *?thesis*
by (*metis antisym-conv1 assms(1,3) member-inv mi-ma-2-deg nless-le option.inject*)
qed

lemma *mint-corr-help-empty*: *invar-vebt t n* ⟹ *vebt-mint t = None* ⟹ *set-vebt' t* = {}
by (*metis VEBT-internal.maxt-corr-help-empty option.distinct(1) vebt-maxt.simps(1) vebt-maxt.simps(2) vebt-mint.elims*)

theorem *mint-corr*:**assumes** *invar-vebt t n* **and** *vebt-mint t = Some x* **shows** *min-in-set* (*set-vebt' t*) *x*
using *assms(1) assms(2) min-in-set-def mint-corr-help mint-member set-vebt'-def* **by** *auto*

theorem *mint-sound*:**assumes** *invar-vebt t n* **and** *min-in-set* (*set-vebt' t*) *x* **shows** *vebt-mint t = Some x*
by (*metis assms(1) assms(2) empty-Collect-eq eq-iff mem-Collect-eq min-in-set-def mint-corr-help mint-corr-help-empty mint-member option.exhaust set-vebt'-def*)

lemma *summaxma*:**assumes** *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg* **and** *mi* ≠ *ma*
shows *the* (*vebt-maxt summary*) = *high ma* (*deg div 2*)
proof−
from *assms(1)* **show** *?thesis*
proof(*cases*)
case (*4 n m*)
have *both-member-options summary* (*high ma n*)
using *4(10) 4(2) 4(4) 4(5) 4(6) 4(9) assms(2) deg-not-0 exp-split-high-low(1)* **by** *blast*
have *high ma n* ≤ *the* (*vebt-maxt summary*) **using** *4(2)* ‹*both-member-options summary* (*high ma n*)› *empty-Collect-eq option.inject maxt-corr-help maxt-corr-help-empty*

not-None-eq set-vebt'-def valid-member-both-member-options
by (*metis option.exhaust-sel*)
have *high ma n < the* (*vebt-maxt summary*) ⟹ *False*
proof−
assume *high ma n < the* (*vebt-maxt summary*)
obtain *maxs* **where** *Some maxs = vebt-maxt summary*
by (*metis 4*(*2*) ‹*both-member-options summary* (*high ma n*)› *empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
hence ∃ *x*. *both-member-options* (*treeList* ! *maxs*) *x*
by (*metis 4*(*2*) *4*(*6*) *both-member-options-equiv-member maxt-member member-bound*)
then obtain *x* **where** *both-member-options* (*treeList* ! *maxs*) *x*
by *auto*
hence *vebt-member* (*treeList* ! *maxs*) *x*
by (*metis 4*(*1*) *4*(*2*) *4*(*3*) ‹*Some maxs = vebt-maxt summary*› *maxt-member member-bound nth-mem valid-member-both-member-options*)
have *maxs* < *2*^*m*
by (*metis 4*(*2*) ‹*Some maxs = vebt-maxt summary*› *maxt-member member-bound*)
have *invar-vebt* (*treeList* ! *maxs*) *n*
using *4*(*1*) *4*(*3*) ‹*maxs* < *2* ^ *m*› **by** *simp*
hence *x* < *2*^*n*
using ‹*vebt-member* (*treeList* ! *maxs*) *x*› *member-bound* **by** *auto*
let *?X* = *2*^*n*∗*maxs* + *x*
have *high ?X n* = *maxs*
by (*simp add*: ‹*x* < *2* ^ *n*› *high-inv mult.commute*)
hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2*^*n*∗*maxs* + *x*)
by (*metis 4*(*3*) *4*(*4*) *4*(*5*) *One-nat-def Suc-leI* ‹*both-member-options* (*treeList* ! *maxs*) *x*› ‹*maxs* < *2* ^ *m*› ‹*x* < *2* ^ *n*› *add-self-div-2 assms*(*1*) *both-member-options-from-chilf-to-complete-tree deg-not-0 low-inv mult.commute*)
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?X*
using *assms*(*1*) *both-member-options-equiv-member* **by** *auto*
have *high ?X n*> *high ma n*
by (*metis* ‹*Some maxs = vebt-maxt summary*› ‹*high* (*2* ^ *n* ∗ *maxs* + *x*) *n* = *maxs*› ‹*high ma n* < *the* (*vebt-maxt summary*)› *option.exhaust-sel option.inject option.simps*(*3*))
hence *?X* > *ma*
by (*metis div-le-mono high-def not-le*)
then show *?thesis*
by (*metis 4*(*8*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2* ^ *n* ∗ *maxs* + *x*)› *leD member-inv not-less-iff-gr-or-eq*)
qed
then show *?thesis*
using *4*(*4*) *4*(*5*) ‹*high ma n* ≤ *the* (*vebt-maxt summary*)› **by** *fastforce*
next
case (*5 n m*)
have *both-member-options summary* (*high ma n*)
by (*metis 5*(*10*) *5*(*5*) *5*(*6*) *5*(*9*) *div-eq-0-iff assms*(*2*) *div-exp-eq high-def nat.simps*(*3*) *numerals*(*2*) *power-not-zero*)
have *high ma n* ≤ *the* (*vebt-maxt summary*)
by (*metis 5*(*2*) *VEBT-Member.vebt-member.simps*(*2*) ‹*both-member-options summary* (*high ma n*)› *vebt-maxt.elims maxt-corr-help minNull.simps*(*1*) *min-Null-member option.exhaust-sel option.simps*(*3*)

valid-member-both-member-options)
have *high ma n* < *the* (*vebt-maxt summary*) ⟹ *False*
proof−
assume *high ma n* < *the* (*vebt-maxt summary*)
obtain *maxs* **where** *Some maxs* = *vebt-maxt summary*
by (*metis 5*(*2*) ‹*both-member-options summary* (*high ma n*)› *empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
hence ∃ *x*. *both-member-options* (*treeList* ! *maxs*) *x*
by (*metis 5*(*2*) *5*(*6*) *both-member-options-equiv-member maxt-member member-bound*)
then obtain *x* **where** *both-member-options* (*treeList* ! *maxs*) *x*
by *auto*
hence *vebt-member* (*treeList* ! *maxs*) *x*
by (*metis 5*(*1*) *5*(*2*) *5*(*3*) ‹*Some maxs* = *vebt-maxt summary*› *both-member-options-equiv-member maxt-member member-bound nth-mem*)
have *maxs* < *2*^*m*
by (*metis 5*(*2*) ‹*Some maxs* = *vebt-maxt summary*› *maxt-member member-bound*)
have *invar-vebt* (*treeList* ! *maxs*) *n*
using *5*(*1*) *5*(*3*) ‹*maxs* < *2* ^ *m*› **by** *simp*
hence *x* < *2*^*n*
using ‹*vebt-member* (*treeList* ! *maxs*) *x*› *member-bound* **by** *auto*
let *?X* = *2*^*n*∗*maxs* + *x*
have *high ?X n* = *maxs*
by (*simp add*: ‹*x* < *2* ^ *n*› *high-inv mult.commute*)
have *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2*^*n*∗*maxs* + *x*)
proof (*rule both-member-options-from-chilf-to-complete-tree*)
show *high ?X* (*deg div 2*) < *length treeList*
using *5*(*3*,*4*,*5*) ‹*high ?X n* = *maxs*› ‹*maxs* < *2* ^ *m*› **by** *auto*
show *1* ≤ *deg*
by (*simp add*: *5*(*4*,*5*))
show *both-member-options* (*treeList* ! *high ?X* (*deg div 2*)) (*low ?X* (*deg div 2*))
by (*metis 5*(*4*,*5*) ‹*both-member-options* (*treeList* ! *maxs*) *x*› ‹*high ?X n* = *maxs*› ‹*x* < *2* ^ *n*› *add-Suc-right add-self-div-2 even-Suc-div-two even-add low-inv mult.commute*)
qed
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?X*
using *assms*(*1*) *both-member-options-equiv-member* **by** *auto*
have *high ?X n*> *high ma n*
by (*metis* ‹*Some maxs* = *vebt-maxt summary*› ‹*high* (*2* ^ *n* ∗ *maxs* + *x*) *n* = *maxs*› ‹*high ma n* < *the* (*vebt-maxt summary*)› *option.sel*)
hence *?X* > *ma*
by (*metis div-le-mono high-def not-le*)
then show *?thesis*
by (*metis 5*(*8*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2* ^ *n* ∗ *maxs* + *x*)› *leD member-inv not-less-iff-gr-or-eq*)
qed
then show *?thesis*
using *5*(*4*) *5*(*5*) ‹*high ma n* ≤ *the*(*vebt-maxt summary*)› **by** *fastforce*
qed
qed

lemma *maxbmo*: *vebt-maxt t = Some x ⟹ both-member-options t x*
apply(*induction t rule*: *vebt-maxt.induct*)
apply *auto*
apply (*metis both-member-options-def naive-member.simps*(*1*) *option.distinct*(*1*) *option.sel zero-neq-one*)
by (*metis One-nat-def Suc-le-D both-member-options-def div-by-1 div-greater-zero-iff membermima.simps*(*3*) *membermima.simps*(*4*) *not-gr0*)

lemma *misiz*:*invar-vebt t n ⟹ Some m = vebt-mint t ⟹ m < 2^n*
by (*metis member-bound mint-member*)

lemma *mintlistlength*: **assumes** *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *n*
mi ≠ ma **shows** *ma > mi ∧* (*∃ m. Some m = vebt-mint summary ∧ m < 2^*(*n − n div 2*))
using *assms*(*1*)
proof *cases*
case (*4 n m*)
hence *both-member-options* (*treeList* ! *high ma n*) (*low ma n*)
by (*metis assms*(*2*) *high-bound-aux*)
moreover hence *both-member-options summary* (*high ma n*)
using *4*(*10*) *4*(*6*) *4*(*7*) *high-bound-aux* **by** *blast*
moreover then obtain *mini* **where** *Some mini = vebt-mint summary*
by (*metis 4*(*3*) *empty-Collect-eq mint-corr-help-empty option.exhaust-sel set-vebt'-def valid-member-both-member-optio*
moreover hence *mini < 2^m*
by (*metis 4*(*3*) *mint-member member-bound*)
moreover have *m = (deg − deg div 2)* **using** *4*(*6*) *4*(*5*)
by *auto*
ultimately show *?thesis* **using** *4*(*1*) *assms 4*(*9*) **by** *auto*
next
case (*5 n m*)
hence *both-member-options* (*treeList* ! *high ma n*) (*low ma n*)
by (*metis assms*(*2*) *high-bound-aux*)
moreover hence *both-member-options summary* (*high ma n*)
using *5*(*10*) *5*(*6*) *5*(*7*) *high-bound-aux* **by** *blast*
moreover then obtain *mini* **where** *Some mini = vebt-mint summary*
by (*metis 5*(*3*) *empty-Collect-eq mint-corr-help-empty option.exhaust-sel set-vebt'-def valid-member-both-member-optio*
moreover hence *mini < 2^m*
by (*metis 5*(*3*) *mint-member member-bound*)
moreover have *m = (deg − deg div 2)* **using** *5*(*6*) *5*(*5*)
by *auto*
ultimately show *?thesis* **using** *5*(*1*) *assms 5*(*9*) **by** *auto*
qed

lemma *power-minus-is-div*:
b ≤ a ⟹ (*2* :: *nat*) *^* (*a − b*) = *2 ^ a div 2 ^ b*
apply (*induct a arbitrary*: *b*)
apply *simp*
apply (*erule le-SucE*)
apply (*clarsimp simp*:*Suc-diff-le le-iff-add power-add*)
apply *simp*

done

lemma *nested-mint*:**assumes** *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *n* *n* = *Suc* (*Suc va*)
¬ *ma* < *mi* *ma* ≠ *mi* **shows**
high (*the* (*vebt-mint summary*) ∗ (2 ∗ 2 ^ (*va div* 2)) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div* 2))
< *length treeList*
proof−
have *setprop*: *t* ∈ *set treeList* ⟹ *invar-vebt t* (*n div* 2) **for** *t* **using** *assms*(1)
by (*cases*) *simp*+
have *listlength*: *length treeList* = 2^(*n* − *n div* 2) **using** *assms*(1)
by (*cases*) *simp*+
have *sumprop*: *invar-vebt summary* (*n* − *n div* 2) **using** *assms*(1)
by (*cases*) *simp*+
have *mimaxprop*: *mi* ≤ *ma* ∧ *ma* ≤ 2^*n* **using** *assms*(1)
by *cases* *simp*+
hence *xbound*: *mi* ≤ *x* ⟹ *x* ≤ *ma* ⟹ *high x* (*n div* 2) ≤ *length treeList* **for** *x*
using *div-le-dividend div-le-mono high-def listlength power-minus-is-div* **by** *auto*
have *contcong*:*i* < *length treeList* ⟹ ∃ *x*. *both-member-options* (*treeList* ! *i*) *x* ⟷ *both-member-options summary i* **for** *i*
using *assms*(1)**by** *cases* *auto*+
obtain *m* **where** *Some m* = *vebt-mint summary* ∧ *m* < 2^(*n* − *n div* 2)
using *assms*(1) *assms*(4) *mintlistlength* **by** *blast*
then obtain *miny* **where** (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*))) =*Some miny*
by (*metis both-member-options-equiv-member contcong empty-Collect-eq listlength mint-corr-help-empty mint-member nth-mem option.exhaust-sel option.sel setprop sumprop set-vebt'-def*)
hence *miny* < 2^(*n div* 2)
by (*metis* ‹⋀*thesis*. (⋀*m*. *Some m* = *vebt-mint summary* ∧ *m* < 2 ^ (*n* − *n div* 2) ⟹ *thesis*) ⟹ *thesis*› *listlength misiz nth-mem option.sel setprop*)
then show *?thesis*
by (*metis* ‹⋀*thesis*. (⋀*m*. *Some m* = *vebt-mint summary* ∧ *m* < 2 ^ (*n* − *n div* 2) ⟹ *thesis*) ⟹ *thesis*› ‹*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)) = *Some miny*› *assms*(2) *div2-Suc-Suc high-inv listlength option.sel power-Suc*)
qed

lemma *minminNull*: *vebt-mint t* = *None* ⟹ *minNull t*
by (*metis minNull.simps*(1) *minNull.simps*(4) *vebt-mint.elims option.distinct*(1))

lemma *minNullmin*: *minNull t* ⟹ *vebt-mint t* = *None*
by (*metis minNull.elims*(2) *vebt-mint.simps*(1) *vebt-mint.simps*(2))

end
end

theory *VEBT-Succ* **imports** *VEBT-Insert VEBT-MinMax*
begin

6 The Successor Operation

definition *is-succ-in-set* :: *nat set* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
is-succ-in-set xs x y = (*y* ∈ *xs* ∧ *y* > *x* ∧ (∀ *z* ∈ *xs*. (*z* > *x* ⟶ *z* ≥ *y*)))

context *VEBT-internal* **begin**

corollary *succ-member*: *is-succ-in-set* (*set-vebt'* *t*) *x y* = (*vebt-member t y* ∧ *y* > *x* ∧ (∀ *z*. *vebt-member t z* ∧ *z* > *x* ⟶ *z* ≥ *y*))
using *is-succ-in-set-def set-vebt'-def* **by** *auto*

6.1 Auxiliary Lemmas on Sets and Successorship

lemma *finite* (*A*:: *nat set*) ⟹ *A* ≠ {}⟹ *Min A* ∈ *A*
by(*induction A rule*: *finite.induct*)(*blast* | *meson Min-in finite-insert*)+

lemma *obtain-set-succ*: **assumes** (*x*::*nat*) < *z* **and** *max-in-set A z* **and** *finite B* **and** *A*=*B* **shows** ∃ *y*. *is-succ-in-set A x y*
proof−
have {*y* ∈ *A*. *y* > *x*} ≠ {}
using *assms*(*1*) *assms*(*2*) *max-in-set-def* **by** *auto*
have *Min* {*y* ∈ *A*. *y* > *x*} ∈ {*y* ∈ *A*. *y* > *x*}
by (*metis* (*full-types*) *Collect-mem-eq* ‹{*y* ∈ *A*. *x* < *y*} ≠ {}› *assms*(*3*) *assms*(*4*) *eq-Min-iff finite-Collect-conjI*)
have *i* ∈ *A*⟹ *i* > *x* ⟹ *i* ≥ *Min* {*y* ∈ *A*. *y* > *x*} **for** *i*
by (*simp add*: *assms*(*3*) *assms*(*4*))
have *is-succ-in-set A x* (*Min* {*y* ∈ *A*. *y* > *x*})
using *is-succ-in-set-def* ‹*Min* {*y* ∈ *A*. *x* < *y*} ∈ {*y* ∈ *A*. *x* < *y*}› ‹⋀*i*. ⟦*i* ∈ *A*; *x* < *i*⟧ ⟹ *Min* {*y* ∈ *A*. *x* < *y*} ≤ *i*› **by** *blast*
then show*?thesis* **by** *auto*
qed

lemma *succ-none-empty*: **assumes** (∄ *x*. *is-succ-in-set* (*xs*) *a x*) **and** *finite xs***shows** ¬ (∃ *x* ∈ *xs*. *ord-class.greater x a*)
proof−
have ∃ *x* ∈ *xs*. *ord-class.greater x a* ⟹ *False*
proof−
assume ∃ *x* ∈ *xs*. *ord-class.greater x a*
hence {*x* ∈ *xs*. *ord-class.greater x a*} ≠ {} **by** *auto*
have *Min* {*y* ∈ *xs*. *y* > *a*} ∈ {*y* ∈ *xs*. *y* > *a*}
by (*metis* (*full-types*) *Collect-mem-eq Min-in* ‹{*x* ∈ *xs*. *a* < *x*} ≠ {}› *assms*(*2*) *finite-Collect-conjI*)
have *i* ∈ *xs* ⟹ *ord-class.greater i a*⟹
ord-class.greater-eq i (*Min* {*y* ∈ *xs*. *ord-class.greater y a*}) **for** *i*
by (*simp add*: *assms*(*2*))
have *is-succ-in-set xs a* (*Min* {*y* ∈ *xs*. *y* > *a*})
using *is-succ-in-set-def* ‹*Min* {*y* ∈ *xs*. *a* < *y*} ∈ {*y* ∈ *xs*. *a* < *y*}› ‹⋀*i*. ⟦*i* ∈ *xs*; *a* < *i*⟧ ⟹ *Min* {*y* ∈ *xs*. *a* < *y*} ≤ *i*› **by** *blast*
then show *False*
using *assms*(*1*) **by** *blast*
qed

then show *?thesis* **by** *blast*
qed

end

6.2 The actual Function

context begin
interpretation *VEBT-internal* **.**

fun *vebt-succ* :: *VEBT* ⇒ *nat* ⇒ *nat option* **where**
vebt-succ (*Leaf* - *b*) *0* = (*if b then Some 1 else None*)|
vebt-succ (*Leaf* - -) (*Suc n*) = *None*|
vebt-succ (*Node None* - - -) - = *None*|
vebt-succ (*Node* - *0* - -) - = *None*|
vebt-succ (*Node* - (*Suc 0*) - -) - = *None*|
vebt-succ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (
if x < *mi then* (*Some mi*)
else (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in*
if h < *length treeList then*
let maxlow = *vebt-maxt* (*treeList* ! *h*) *in* (
if maxlow ≠ *None* ∧ (*Some l* <*o* *maxlow*) *then*
Some (*2*^(*deg div 2*)) ∗*o* *Some h* +*o* *vebt-succ* (*treeList* ! *h*) *l*
else let sc = *vebt-succ summary h in*
if sc = *None then None*
else Some (*2*^(*deg div 2*)) ∗*o* *sc* +*o* *vebt-mint* (*treeList* ! *the sc*))
else None))

end

6.3 Lemmas for Term Decomposition

context *VEBT-internal* **begin**

lemma *succ-min*: **assumes** *deg* ≥ *2* **and** (*x*::*nat*) < *mi* **shows**
vebt-succ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*
by (*metis add-2-eq-Suc assms*(*1*) *assms*(*2*) *le-add-diff-inverse vebt-succ.simps*(*6*))

lemma *succ-greatereq-min*: **assumes** *deg* ≥ *2* **and** (*x*::*nat*) ≥ *mi* **shows**
vebt-succ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in*
if h < *length treeList then*

let maxlow = *vebt-maxt* (*treeList* ! *h*) *in*
(*if maxlow* ≠ *None* ∧ (*Some l* <*o* *maxlow*) *then*
Some (*2*^(*deg div 2*)) ∗*o* *Some h* +*o* *vebt-succ* (*treeList* ! *h*) *l*
else let sc = *vebt-succ summary h in*
if sc = *None then None*
else Some (*2*^(*deg div 2*)) ∗*o* *sc* +*o* *vebt-mint* (*treeList* ! *the sc*))

else None)
by (*smt* (*verit*) *add-numeral-left arith-simps*(*1*) *assms*(*1*) *assms*(*2*) *le-add-diff-inverse not-less numerals*(*1*) *plus-1-eq-Suc vebt-succ.simps*(*6*))

lemma *succ-list-to-short*: **assumes** *deg* ≥ *2* **and** *x* ≥ *mi* **and** *high x* (*deg div 2*) ≥ *length treeList*
shows
vebt-succ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *succ-greatereq-min* **by** *auto*

lemma *succ-less-length-list*: **assumes** *deg* ≥ *2* **and** *x* ≥ *mi* **and** *high x* (*deg div 2*) < *length treeList*
shows
vebt-succ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
(*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) ; *maxlow* = *vebt-maxt* (*treeList* ! *h*) *in*
(*if maxlow* ≠ *None* ∧ (*Some l* <$_o$ *maxlow*) *then*
Some (*2*^(*deg div 2*)) *∗*$_o$ *Some h* +$_o$ *vebt-succ* (*treeList* ! *h*) *l*
else let sc = *vebt-succ summary h in*
if sc = *None then None*
else Some (*2*^(*deg div 2*)) *∗*$_o$ *sc* +$_o$ *vebt-mint* (*treeList* !*the sc*)))
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *succ-greatereq-min*)

6.4 Correctness Proof

theorem *succ-corr*: *invar-vebt t n* ⟹ *vebt-succ t x* = *Some sx* == *is-succ-in-set* (*set-vebt*′ *t*) *x sx*
proof(*induction t n arbitrary*: *x sx rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case* **proof**(*cases x*)
case *0*
then show *?thesis*
by (*simp add*: *succ-member*)
next
case (*Suc nat*)
then show *?thesis* **proof**(*cases nat*)
case *0*
then show *?thesis*
by (*simp add*: *Suc succ-member*)
next
case (*Suc nat*)
then show *?thesis* **by** (*metis* (*no-types*) *VEBT-Member.vebt-member.simps*(*1*) *Suc-eq-plus1 add-cancel-right-left le-add2 le-imp-less-Suc not-add-less2 not-less0 old.nat.exhaust option.distinct*(*1*) *option.simps*(*1*) *vebt-succ.simps*(*1*) *vebt-succ.simps*(*2*) *succ-member*)
qed
qed
next
case (*2 treeList n summary m deg*)
then show *?case*
by (*simp add*: *succ-member*)
next
case (*3 treeList n summary m deg*)
then show *?case*

by (*simp add*: *succ-member*)
next
case (*4 treeList n summary m deg mi ma*)
hence $n = m$ **and** $n \geq 1$ **and** $deg \geq 2$ **and** $deg = n + m$
apply *blast+*
using *4.hyps*(*2*) *4.hyps*(*5*) *Suc-le-eq deg-not-0* **apply** *auto*[*1*]
using *4.hyps*(*2*) *4.hyps*(*5*) *4.hyps*(*6*) *deg-not-0* **apply** *fastforce*
by (*simp add*: *4.hyps*(*6*))
hence *deg div 2* =*n* **and** *length treeList* = *2^n*
using *add-self-div-2* **apply** *blast* **by** (*simp add*: *4.hyps*(*4*) *4.hyps*(*5*))
then show *?case* **proof**(*cases* $x < mi$)
case *True*
hence *0*: *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x = *Some mi*
by (*simp add*: ‹*2* ≤ *deg*› *succ-min*)
have *1*:*mi* = *the* (*vebt-mint* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) **by** *simp*
hence *mi* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*metis VEBT-Member.vebt-member.simps*(*5*) ‹*2* ≤ *deg*› *add-numeral-left arith-simps*(*1*) *le-add-diff-inverse mem-Collect-eq numerals*(*1*) *plus-1-eq-Suc set-vebt'-def*)
hence *2*:*y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ $y \geq x$ **for** *y*
using *4.hyps*(*9*) *True member-inv set-vebt'-def* **by** *fastforce*
hence *3*: *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ ($y > mi$ ⟹ $y \geq x$) **for** *y* **by** *blast*
hence *4*: ∀ *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*). $y > mi$ ⟶ $y \geq x$ **by** *blast*
hence *is-succ-in-set* (*set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x mi*
by (*metis* (*mono-tags*, *lifting*) *4.hyps*(*9*) *True* ‹*mi* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)› *eq-iff less-imp-le-nat mem-Collect-eq member-inv succ-member set-vebt'-def*)
then show *?thesis* **using** *0*
by (*metis is-succ-in-set-def antisym-conv option.inject*)
next
case *False*
hence $x \geq mi$**by** *simp*
then show *?thesis*
proof(*cases high x* (*deg div 2*)< *length treeList*)
case *True*
hence *high x n* < *2^n* ∧ *low x n* < *2^n*
by (*simp add*: ‹*deg div 2* = *n*› ‹*length treeList* = *2* ^ *n*› *low-def*)
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
let *?maxlow* = *vebt-maxt* (*treeList* ! *?h*)
let *?sc* = *vebt-succ summary ?h*
have *1*:*vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
(*if ?maxlow* ≠ *None* ∧ (*Some ?l* <*o* *?maxlow*) *then*
Some (*2^*(*deg div 2*)) ∗*o* *Some ?h* +*o* *vebt-succ* (*treeList* ! *?h*) *?l*
else if ?sc = *None then None*
else Some (*2^*(*deg div 2*)) ∗*o* *?sc* +*o* *vebt-mint* (*treeList* ! *the ?sc*))
by (*smt* (*verit*) *True* ‹*2* ≤ *deg*› ‹*mi* ≤ *x*› *succ-less-length-list*)
then show *?thesis*

proof(*cases ?maxlow ≠ None ∧ (Some ?l <$_o$?maxlow)*)
case *True*
then obtain *maxl* **where** *00*:*Some maxl = ?maxlow ∧ ?l < maxl* **by** *auto*
have *01*:*invar-vebt* ((*treeList* ! *?h*)) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*
by (*simp add*: *4.hyps*(*1*) ‹*deg div 2 = n*› ‹*high x n < 2 ^ n ∧ low x n < 2 ^ n*› ‹*length treeList = 2 ^ n*›)
have *02*:*vebt-member* ((*treeList* ! *?h*)) *maxl*
using *00 01 maxt-member* **by** *auto*
hence *03*: ∃ *y*. *y > ?l ∧ vebt-member* ((*treeList* ! *?h*)) *y*
using *00* **by** *blast*
hence *afinite*: *finite* (*set-vebt'* (*treeList* ! *?h*))
using *01 set-vebt-finite* **by** *blast*
then obtain *succy* **where** *04*:*is-succ-in-set* (*set-vebt'* (*treeList* ! *?h*)) *?l succy*
using *00 01 maxt-corr obtain-set-succ* **by** *fastforce*
hence *05*:*Some succy = vebt-succ* (*treeList* ! *?h*) *?l* **using** *4*(*1*) *01* **by** *force*
hence *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some* (*2^(deg div 2)∗ ?h + succy*)
by (*metis 1 True add-def mul-def option-shift.simps*(*3*))
hence *06*: *succy* ∈ *set-vebt'* (*treeList* ! *?h*)
using *04 is-succ-in-set-def* **by** *blast*
hence *07*: *succy < 2^(deg div 2) ∧ ?h < 2^(deg div 2) ∧ deg div 2 + deg div 2 = deg*
using *01 04 4.hyps*(*5*) *4.hyps*(*6*) ‹*high x n < 2 ^ n ∧ low x n < 2 ^ n*› *member-bound succ-member* **by** *auto*
let *?y = 2^(deg div 2)∗ ?h + succy*
have *08*: *vebt-member* (*treeList* ! *?h*) *succy*
using *06 set-vebt'-def* **by** *auto*
hence *09*: *both-member-options* (*treeList* ! *?h*) *succy*
using *01 both-member-options-equiv-member* **by** *blast*
have *10*: *high ?y* (*deg div 2*) = *?h ∧ low ?y* (*deg div 2*) = *succy*
by (*simp add*: *07 high-inv low-inv mult.commute*)
hence *11*: *naive-member* (*treeList* ! *?h*) *succy*
⟹ *naive-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *naive-member.simps*(*3*)[*of Some* (*mi*, *ma*) *deg−1 treeList summary ?y*]
by (*metis 07 4.hyps*(*4*) *4.hyps*(*5*) *One-nat-def Suc-pred* ‹*2 ≤ deg*› ‹*deg div 2 = n*› *add-gr-0 div-greater-zero-iff zero-less-numeral*)
have *12*: *?y ≥ mi ∧ ?y ≤ ma*
by (*metis 01 07 09 10 4.hyps*(*11*) *4.hyps*(*5*) *4.hyps*(*8*) ‹*deg div 2 = n*› *less-imp-le-nat*)
hence *13*: *membermima* (*treeList* ! *?h*) *succy*
⟹ *membermima* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *membermima.simps*(*4*)[*of mi ma deg −1 treeList summary ?y*]
apply(*cases ?y = mi ∨ ?y = ma*)
apply (*metis 07 One-nat-def Suc-pred* ‹*2 ≤ deg*› *add-gr-0 div-greater-zero-iff zero-less-numeral*)
by (*metis 07 10 4.hyps*(*4*) *4.hyps*(*5*) *One-nat-def Suc-pred* ‹*2 ≤ deg*› ‹*deg div 2 = n*› *add-gr-0 div-greater-zero-iff zero-less-numeral*)
hence *14*:*both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *09 11 both-member-options-def* **by** *blast*
have *15*: *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
proof (*rule valid-member-both-member-options*[*OF - 14*])
show *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*

using *4.hyps* ‹*length treeList = 2 ^ n*› *invar-vebt.intros*(*4*) **by** *auto*
qed
have *16*: *Some ?y = vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
by (*simp add*: ‹*vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x = Some* (*2 ^* (*deg div 2*) ** high x* (*deg div 2*) *+ succy*)›)
have *17*: *x = ?h * 2^*(*deg div 2*) *+ ?l*
using *bit-concat-def bit-split-inv* **by** *auto*
have *18*: *?y − x = ?h * 2^*(*deg div 2*) *+ succy − ?h * 2^*(*deg div 2*) *− ?l*
by (*metis 17 diff-diff-add mult.commute*)
hence *?y −x > 0*
using *04 is-succ-in-set-def* **by** *auto*
hence *19*: *?y > x*
using *zero-less-diff* **by** *blast*
have *20*: *z > x* ⟹ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ⟹ *z*≥ *?y* **for** *z*
proof−
assume *z > x* **and** *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z*
hence *high z* (*deg div 2*) ≥ *high x* (*deg div 2*)
by (*simp add*: *div-le-mono high-def*)
then show *?thesis* **proof**(*cases high z* (*deg div 2*) = *high x* (*deg div 2*))
case *True*
hence *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *vebt-member.simps*(*5*)[*of mi ma deg−2 treeList summary z*]
by (*metis 01 07 4.hyps*(*11*) *4.hyps*(*5*) *False* ‹*deg div 2 = n*› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z*› ‹*x < z*› *both-member-options-equiv-member member-inv*)
hence *succy* ≤ *low z* (*deg div 2*) **using** *04* **unfolding** *is-succ-in-set-def*
by (*metis True* ‹*x < z*› *add-diff-cancel-left′ bit-concat-def bit-split-inv diff-diff-left mem-Collect-eq set-vebt′-def zero-less-diff*)
hence *?y* ≤ *z*
by (*metis True add-le-cancel-left bit-concat-def bit-split-inv mult.commute*)
then show *?thesis* **by** *blast*
next
case *False*
hence *high z* (*deg div 2*) > *high ?y* (*deg div 2*)
using *10* ‹*high x* (*deg div 2*) ≤ *high z* (*deg div 2*)› **by** *linarith*
then show *?thesis*
by (*metis div-le-mono high-def nat-le-linear not-le*)
qed
qed
hence *is-succ-in-set* (*set-vebt′* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x ?y*
by (*simp add*: *15 19 succ-member*)
then show *?thesis* **using** *16*
by (*metis eq-iff option.inject succ-member*)
next
case *False*
hence *i1*:*?maxlow* = *None* ∨ ¬ (*Some ?l* <$_o$ *?maxlow*) **by** *simp*
hence *2*: *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if ?sc* = *None then None*
else Some (*2^*(*deg div 2*)) **$_o$ *?sc* +$_o$ *vebt-mint* (*treeList* ! *the ?sc*))

using *1* **by** *auto*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *4*(*1*) *True* **by** *simp*
hence *33*:∄ *u*. *vebt-member* (*treeList* ! *?h*) *u* ∧ *u* > *?l*
proof(*cases ?maxlow* = *None*)
case *True*
then show *?thesis* **using** *maxt-corr-help-empty*[*of treeList* ! *?h n*]
by (*simp add*: ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *set-vebt'-def*)
next
case *False*
obtain *maxilow* **where** *?maxlow* =*Some maxilow*
using *False* **by** *blast*
hence *maxilow* ≤ *?l*
using *i1* **by** *auto*
then show *?thesis*
by (*meson* ‹*vebt-maxt* (*treeList* ! *high x* (*deg div 2*)) = *Some maxilow*› ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *le-imp-less-Suc le-less-trans maxt-corr-help not-less-eq*)
qed
then show *?thesis*
proof(*cases ?sc* = *None*)
case *True*
hence *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*
by (*simp add*: *2*)
hence ∄ *i*. *is-succ-in-set* (*set-vebt'* *summary*) *?h i*
using *4.hyps*(*3*) *True* **by** *force*
hence ∄ *i*. *i* > *?h* ∧ *vebt-member summary i* **using** *succ-none-empty*[*of set-vebt' summary ?h*]
proof −
{ fix *nn* :: *nat*
have ∀ *n*. ((*is-succ-in-set* (*Collect* (*vebt-member summary*)) (*high x* (*deg div 2*)) *esk1-0* ∨ *infinite* (*Collect* (*vebt-member summary*))) ∨ *n* ∉ *Collect* (*vebt-member summary*)) ∨ ¬ *high x* (*deg div 2*) < *n*
using ‹∄ *i*. *is-succ-in-set* (*set-vebt'* *summary*) (*high x* (*deg div 2*)) *i*› *succ-none-empty set-vebt'-def* **by** *auto*
then have ¬ *high x* (*deg div 2*) < *nn* ∨ ¬ *vebt-member summary nn*
using *4.hyps*(*2*) ‹∄ *i*. *is-succ-in-set* (*set-vebt'* *summary*) (*high x* (*deg div 2*)) *i*› *set-vebt'-def set-vebt-finite* **by** *auto* **}**
then show *?thesis*
by *blast*
qed
hence (*i* > *x* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*) ⟹ *False* **for** *i*
proof−
fix *i*
assume *i* > *x* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*
hence *20*: *i* = *mi* ∨ *i* = *ma* ∨ (*high i* (*deg div 2*) < *length treeList*
∧ *vebt-member* (*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary i*]
using *member-inv* **by** *blast*

have $i \neq mi$
using ‹$mi \leq x$› ‹$x < i \wedge$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *not-le* **by** *blast*
hence $mi \neq ma$
using ‹$x < i \wedge$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *member-inv not-less-iff-gr-or-eq* **by** *blast*
hence $i < 2\hat{}deg$
using *4.hyps*(*10*) ‹$i \neq mi$› ‹$x < i \wedge$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *member-inv* **by** *fastforce*
hence *aa*:$i = ma \implies$ *both-member-options*(*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))
using *4.hyps*(*11*) *4.hyps*(*2*) *4.hyps*(*5*) *4.hyps*(*6*) ‹$mi \neq ma$› *deg-not-0 exp-split-high-low*(*1*) **by** *auto*
hence *abc*:*invar-vebt* (*treeList* ! (*high i* (*deg div 2*))) *n*
using *4.hyps*(*1*) *4.hyps*(*2*) *4.hyps*(*5*) *4.hyps*(*6*) ‹*deg div 2* = *n*› ‹$i < 2\hat{}deg$›
‹*length treeList* = $2\hat{}n$› *deg-not-0 exp-split-high-low*(*1*) **by** *simp*
hence *abd*:$i = ma \implies$ *vebt-member*(*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))
using *aa valid-member-both-member-options* **by** *blast*
hence *abe*:*vebt-member*(*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))
using *20* ‹$i \neq mi$› **by** *blast*
hence *abf*:*both-member-options*(*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))
using ‹*invar-vebt* (*treeList* ! *high i* (*deg div 2*)) *n*› *both-member-options-equiv-member* **by** *blast*
hence *abg*:*both-member-options summary* (*high i* (*deg div 2*))
by (*metis 20 4.hyps*(*10*) *4.hyps*(*2*) *4.hyps*(*4*) *4.hyps*(*6*) *4.hyps*(*7*) ‹$2 \leq deg$› ‹*deg div 2* = *n*› ‹$i \neq mi$› *deg-not-0 div-greater-zero-iff exp-split-high-low*(*1*) *zero-less-numeral*)
hence *abh*:*vebt-member summary* (*high i* (*deg div 2*))
using *4.hyps*(*2*) *valid-member-both-member-options* **by** *blast*
have *aaa*:(*high i* (*deg div 2*)) = (*high x* (*deg div 2*)) $\implies$ *vebt-member* (*treeList* ! *?h*) (*low i* (*deg div 2*))
using ‹*vebt-member* (*treeList* ! *high i* (*deg div 2*)) (*low i* (*deg div 2*))› **by** *auto*
have *abi*:(*high i* (*deg div 2*)) = (*high x* (*deg div 2*)) $\implies$ *low i* (*deg div 2*) > *?l*
by (*metis* ‹$x < i \wedge$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *add-le-cancel-left bit-concat-def bit-split-inv le-neq-implies-less less-imp-le-nat nat-neq-iff*)
hence *abj*:(*high i* (*deg div 2*)) = (*high x* (*deg div 2*)) $\implies$ *False* **using** *33 aaa* **by** *blast*
hence *abk*: (*high i* (*deg div 2*)) $\in$ (*set-vebt'* *summary*) $\wedge$ (*high i* (*deg div 2*)) > (*high x* (*deg div 2*))
by (*metis* (*full-types*) ‹*vebt-member summary* (*high i* (*deg div 2*))› ‹$x < i \wedge$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *div-le-mono high-def le-less mem-Collect-eq set-vebt'-def*)

then show *?thesis*
using ‹$\neg$ ($\exists$ *i*>*high x* (*deg div 2*). *vebt-member summary i*)› *abh* **by** *blast*
qed
then show *?thesis*
using ‹*vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*› *succ-member* **by** *auto*
next
case *False*
hence *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =

Some (2^(*deg div* 2)) $*_o$ *?sc* $+_o$ *vebt-mint* (*treeList* ! *the ?sc*)
by (*simp add*: *False* 2)
obtain *sc* **where** *?sc* = *Some sc*
using *False* **by** *blast*
hence *is-succ-in-set* (*set-vebt'* *summary*) *?h sc*
using *4.hyps*(3) **by** *blast*
hence *vebt-member summary sc*
using *succ-member* **by** *blast*
hence *both-member-options summary sc*
using *4.hyps*(2) *both-member-options-equiv-member* **by** *auto*
hence *sc* < 2^*m*
using *4.hyps*(2) ‹*vebt-member summary sc*› *member-bound* **by** *blast*
hence ∃ *miny*. *both-member-options* (*treeList* ! *sc*) *miny*
using *4.hyps*(7) ‹*both-member-options summary sc*› **by** *blast*
hence *fgh*:*set-vebt'* (*treeList* ! *sc*) ≠ {}
by (*metis 4.hyps*(1) *4.hyps*(4) *4.hyps*(5) *Collect-empty-eq-bot* ‹*deg div 2* = *n*› ‹*sc* < 2 ^ *m*› *bot-empty-eq empty-iff nth-mem set-vebt'-def valid-member-both-member-options*)
hence *invar-vebt* (*treeList* ! *the ?sc*) *n*
by (*simp add*: *4.hyps*(1) *4.hyps*(4) ‹*sc* < 2 ^ *m*› ‹*vebt-succ summary* (*high x* (*deg div* 2)) = *Some sc*›)
then obtain *miny* **where** *Some miny* = *vebt-mint* (*treeList* ! *sc*)
by (*metis fgh Collect-empty-eq VEBT-Member.vebt-member.simps*(2) *vebt-buildup.simps*(2) *buildup-gives-empty vebt-mint.elims set-vebt'-def*)
hence *Some miny* = *vebt-mint* (*treeList* ! *the ?sc*)
by (*simp add*: ‹*vebt-succ summary* (*high x* (*deg div* 2)) = *Some sc*›)
hence *min-in-set* (*set-vebt'* (*treeList* ! *the ?sc*)) *miny*
using ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div* 2)))) *n*› *mint-corr* **by** *auto*
hence *scmem*:*vebt-member* (*treeList* ! *the ?sc*) *miny*
using ‹*Some miny* = *vebt-mint* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div* 2))))› ‹*invar-vebt* (*treeList* ! *the*(*vebt-succ summary* (*high x* (*deg div* 2)))) *n*› *mint-member* **by** *auto*
let *?res* = *Some* (2^(*deg div* 2)) $*_o$ *?sc* $+_o$ *vebt-mint* (*treeList* !*the ?sc*)
obtain *res* **where** *res* = *the ?res* **by** *blast*
hence *res* = 2^(*deg div* 2) ∗ *sc* + *miny*
by (*metis* ‹*Some miny* = *vebt-mint* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div* 2))))› ‹*vebt-succ summary* (*high x* (*deg div* 2)) = *Some sc*› *add-def mul-def option.sel option-shift.simps*(3))
have *high res* (*deg div* 2) = *sc*
by (*metis* ‹*deg div 2* = *n*› ‹*res* = 2 ^ (*deg div* 2) ∗ *sc* + *miny*› ‹*invar-vebt* (*treeList* ! *the ?sc*) *n*› *high-inv member-bound mult.commute scmem*)
hence *res* > *x*
by (*metis is-succ-in-set-def* ‹*is-succ-in-set* (*set-vebt'* *summary*) (*high x* (*deg div* 2)) *sc*› *div-le-mono high-def not-le*)
hence *res* > *mi*
using ‹*mi* ≤ *x*› *le-less-trans* **by** *blast*
hence *res* ≤ *ma*
proof(*cases high res n* < *high ma n*)
case *True*
then show *?thesis*
by (*metis div-le-mono high-def leD nat-le-linear*)

next
case *False*
hence *mi* ≠ *ma*
by (*metis 4.hyps*(*5*) *4.hyps*(*8*) ‹∃ *miny*. *both-member-options* (*treeList* ! *sc*) *miny*› ‹*length treeList* = *2* ^ *n*› ‹*sc* < *2* ^ *m*› *nth-mem*)
have *high res n* < *2*^*m*
using ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*sc* < *2* ^ *m*› **by** *blast*
hence (∀ *x*. *high x n* = *high res n* ∧ *both-member-options* (*treeList* ! (*high res n*)) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*) **using** *4*(*11*)
using ‹*mi* ≠ *ma*› **by** *blast*
have *high res n* = *high res n* ∧ *both-member-options* (*treeList* ! (*high res n*)) (*low res n*)
by (*metis* ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*res* = *2* ^ (*deg div 2*) ∗ *sc* + *miny*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *both-member-options-equiv-member low-inv member-bound mult.commute option.sel scmem*)
then show *?thesis*
using ‹∀ *x*. *high x n* = *high res n* ∧ *both-member-options* (*treeList* ! *high res n*) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*› **by** *blast*
qed
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*the ?res*) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary res*]
by (*metis 4.hyps*(*4*) ‹*2* ≤ *deg*› ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*mi* < *res*› ‹*res* = *2* ^ (*deg div 2*) ∗ *sc* + *miny*› ‹*res* = *the* (*Some* (*2* ^ (*deg div 2*)) ∗$_o$ *vebt-succ summary* (*high x* (*deg div 2*)) +$_o$ *vebt-mint* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))))› ‹*sc* < *2* ^ *m*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *add-2-eq-Suc leD le-add-diff-inverse low-inv member-bound mult.commute not-less-iff-gr-or-eq option.sel scmem*)
have (*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* > *x*) ⟹ *z* ≥ *res* **for** *z*
proof−
fix *z*
assume *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* > *x*
hence *20*: *z* = *mi* ∨ *z* = *ma* ∨ (*high z* (*deg div 2*) < *length treeList*
∧ *vebt-member* (*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary z*]
using *member-inv* **by** *blast*
have *z* ≠ *mi*
using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x* < *z*› ‹*mi* ≤ *x*› *not-le* **by** *blast*
hence *mi* ≠ *ma*
using ‹*mi* < *res*› ‹*res* ≤ *ma*› *not-le* **by** *blast*
hence *z* < *2*^*deg*
using *4.hyps*(*10*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x* < *z*› ‹*z* ≠ *mi*› *member-inv* **by** *fastforce*
hence *aa*:*z* = *ma* ⟹ *both-member-options*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))
using *4.hyps*(*11*) *4.hyps*(*2*) *4.hyps*(*5*) *4.hyps*(*6*) ‹*mi* ≠ *ma*› *deg-not-0 exp-split-high-low*(*1*) **by** *auto*
hence *abc*:*invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n*

using *4.hyps(1) 4.hyps(2) 4.hyps(5) 4.hyps(6) ‹deg div 2 = n› ‹z < 2 ^ deg› ‹length treeList = 2 ^ n› deg-not-0 exp-split-high-low(1)* **by** *simp*

hence *abd:z = ma ⟹ vebt-member(treeList ! (high z (deg div 2))) (low z (deg div 2))*

using *aa valid-member-both-member-options* **by** *blast*

hence *abe:vebt-member(treeList ! (high z (deg div 2))) (low z (deg div 2))*

using *20 ‹z ≠ mi›* **by** *blast*

hence *abf:both-member-options(treeList ! (high z (deg div 2))) (low z (deg div 2))*

using *‹invar-vebt (treeList ! high z (deg div 2)) n› both-member-options-equiv-member* **by** *blast*

hence *abg:both-member-options summary (high z (deg div 2))*

by *(metis (full-types) 4.hyps(5) 4.hyps(6) 4.hyps(7) ‹deg div 2 = n› ‹invar-vebt (treeList ! the (vebt-succ summary (high x (deg div 2)))) n› ‹z < 2 ^ deg› deg-not-0 exp-split-high-low(1))*

hence *abh:vebt-member summary (high z (deg div 2))*

using *4.hyps(2) valid-member-both-member-options* **by** *blast*

have *aaa:(high z (deg div 2)) = (high x (deg div 2)) ⟹ vebt-member (treeList ! ?h) (low z (deg div 2))*

using *abe* **by** *auto*

have *high z(deg div 2)< sc ⟹ False*

proof−

assume *high z(deg div 2)< sc*

hence *vebt-member summary (high z(deg div 2))*

using *abh* **by** *blast*

have *aaaa:?h ≤ high z(deg div 2)*

by *(simp add: ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) z ∧ x < z› div-le-mono high-def less-imp-le-nat)*

have *bbbb:?h ≥ high z(deg div 2)*

using *‹is-succ-in-set (set-vebt′ summary) (high x (deg div 2)) sc› ‹high z (deg div 2) < sc› abh leD succ-member* **by** *auto*

hence *?h = high z (deg div 2)*

using *aaaa eq-iff* **by** *blast*

hence *vebt-member (treeList ! ?h) (low z (deg div 2))*

using *aaa* **by** *linarith*

then show *False*

by *(metis 33 ‹high x (deg div 2) = high z (deg div 2)› ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) z ∧ x < z› add-diff-cancel-left′ bit-concat-def bit-split-inv diff-diff-left zero-less-diff)*

qed

hence *high z(deg div 2) ≥ sc*

using *not-less* **by** *blast*

then show *z ≥ res*

proof(*cases high z(deg div 2) = sc*)

case *True*

hence *vebt-member (treeList ! (high z(deg div 2))) (low z (deg div 2))*

using *abe* **by** *blast*

have *low z (deg div 2) ≥ miny*

using *True ‹min-in-set (set-vebt′ (treeList ! the (vebt-succ summary (high x (deg div 2))))) miny› ‹vebt-succ summary (high x (deg div 2)) = Some sc› abe min-in-set-def set-vebt′-def* **by** *auto*

hence *z ≥ res*

```
                by (metis (full-types) True ‹res = 2 ^ (deg div 2) * sc + miny› add-le-cancel-left
bit-concat-def bit-split-inv mult.commute)
            then show ?thesis by simp
          next
            case False
            hence high z(deg div 2) > sc
              using ‹sc ≤ high z (deg div 2)› le-less by blast
            then show ?thesis
              by (metis ‹high res (deg div 2) = sc› div-le-mono high-def leD linear)
          qed
        qed
        hence is-succ-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x res
          using ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) (the (Some (2 ^ (deg
div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the (vebt-succ summary ?h))))›
            ‹res = the (Some (2 ^ (deg div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the
(vebt-succ summary ?h)))› ‹x < res› succ-member by blast
        moreover have Some res = Some (2^(deg div 2)) *o ?sc +o vebt-mint (treeList ! the ?sc)
          by (metis ‹Some miny = vebt-mint (treeList !the (vebt-succ summary (high x (deg div 2))))›
‹res = 2 ^ (deg div 2) * sc + miny› ‹vebt-succ summary (high x (deg div 2)) = Some sc› add-def
mul-def option-shift.simps(3))
        ultimately show ?thesis
          by (metis (mono-tags) is-succ-in-set-def ‹vebt-succ (Node (Some (mi, ma)) deg treeList
summary) x = Some (2 ^ (deg div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the (vebt-succ
summary ?h))› eq-iff option.inject)
      qed
    qed
  next
    case False
    hence 0:vebt-succ (Node (Some (mi, ma)) deg treeList summary) x = None
      by (simp add: ‹2 ≤ deg› ‹mi ≤ x› succ-greatereq-min)
    have 1:x ≥ 2^deg
    by (metis 4.hyps(4) 4.hyps(5) 4.hyps(6) False ‹deg div 2 = n› high-def le-less-linear less-mult-imp-div-less
mult-2 power2-eq-square power-even-eq)
    hence x ∉ set-vebt' (Node (Some (mi, ma)) deg treeList summary)
      using 4.hyps(10) 4.hyps(9) member-inv set-vebt'-def by fastforce
    hence ∄ ss. is-succ-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x ss
      using 4.hyps(10) 1 ‹mi ≤ x› member-inv succ-member by fastforce
    then show ?thesis using 0 by auto
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  hence Suc n = m and deg = n + m and length treeList = 2^m ∧ invar-vebt summary m
    by blast +
  hence n ≥ 1
    using 5.hyps(1) set-n-deg-not-0 by blast
  hence deg ≥ 2
    by (simp add: 5.hyps(5) 5.hyps(6))
  hence deg div 2 =n
```

by (*simp add*: *5.hyps*(*5*) *5.hyps*(*6*))
then show *?case* **proof**(*cases x* < *mi*)
case *True*
hence *0*: *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*
by (*simp add*: ‹*2* ≤ *deg*› *succ-min*)
have *1*:*mi* = *the* (*vebt-mint* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) **by** *simp*
hence *mi* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*metis VEBT-Member.vebt-member.simps*(*5*) ‹*2* ≤ *deg*› *add-numeral-left arith-simps*(*1*) *le-add-diff-inverse mem-Collect-eq numerals*(*1*) *plus-1-eq-Suc set-vebt'-def*)
hence *2*:*y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ *y* ≥ *x* **for** *y*
using *5.hyps*(*9*) *True member-inv set-vebt'-def* **by** *fastforce*
hence *3*: *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ (*y* > *mi* ⟹ *y* ≥ *x*) **for** *y* **by** *blast*
hence *4*: ∀ *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*). *y* > *mi* ⟶ *y* ≥ *x* **by** *blast*
hence *is-succ-in-set* (*set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x mi*
by (*metis* (*mono-tags*, *lifting*) *5.hyps*(*9*) *True* ‹*mi* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)› *eq-iff less-imp-le-nat mem-Collect-eq member-inv succ-member set-vebt'-def*)
then show *?thesis* **using** *0*
by (*metis is-succ-in-set-def antisym-conv option.inject*)
next
case *False*
hence *x* ≥ *mi***by** *simp*
then show *?thesis*
proof(*cases high x* (*deg div 2*)< *length treeList*)
case *True*
hence *high x n* < *2^m* ∧ *low x n* < *2^n*
by (*simp add*: *5.hyps*(*4*) ‹*deg div 2* = *n*› *low-def*)
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
let *?maxlow* = *vebt-maxt* (*treeList* ! *?h*)
let *?sc* = *vebt-succ summary ?h*
have *1*:*vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
(*if ?maxlow* ≠ *None* ∧ (*Some ?l* <*o* *?maxlow*) *then*
Some (*2^*(*deg div 2*)) ∗*o* *Some ?h* +*o* *vebt-succ* (*treeList* ! *?h*) *?l*
else if ?sc = *None then None*
else Some (*2^*(*deg div 2*)) ∗*o* *?sc* +*o* *vebt-mint* (*treeList* ! *the ?sc*))
by (*smt* (*verit*) *True* ‹*2* ≤ *deg*› ‹*mi* ≤ *x*› *succ-less-length-list*)
then show *?thesis*
proof(*cases ?maxlow* ≠ *None* ∧ (*Some ?l* <*o* *?maxlow*))
case *True*
then obtain *maxl* **where** *00*:*Some maxl* = *?maxlow* ∧ *?l* < *maxl* **by** *auto*
have *01*:*invar-vebt* ((*treeList* ! *?h*)) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*
using *5.hyps*(*1*) *5.hyps*(*4*) ‹*deg div 2* = *n*› ‹*high x n* < *2* ^ *m* ∧ *low x n* < *2* ^ *n*› **by** *simp*
have *02*:*vebt-member* ((*treeList* ! *?h*)) *maxl*
using *00 01 maxt-member* **by** *auto*
hence *03*: ∃ *y*. *y* > *?l* ∧ *vebt-member* ((*treeList* ! *?h*)) *y*
using *00* **by** *blast*

hence *afinite*: *finite* (*set-vebt'* (*treeList* ! *?h*))
using *01 set-vebt-finite* **by** *blast*
then obtain *succy* **where** *04*:*is-succ-in-set* (*set-vebt'* (*treeList* ! *?h*)) *?l succy*
using *00 01 maxt-corr obtain-set-succ* **by** *fastforce*
hence *05*:*Some succy* = *vebt-succ* (*treeList* ! *?h*) *?l* **using** *5*(*1*) *01* **by** *force*
hence *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some* (*2*^(*deg div 2*)∗ *?h* + *succy*)
by (*metis 1 True add-def mul-def option-shift.simps*(*3*))
hence *06*: *succy* ∈ *set-vebt'* (*treeList* ! *?h*)
using *04 is-succ-in-set-def* **by** *blast*
hence *07*: *succy* < *2*^(*deg div 2*) ∧ *?h* < *2*^*m* ∧ *Suc* (*deg div 2* + *deg div 2*) = *deg*
using *01 04 5.hyps*(*5*) *5.hyps*(*6*) ‹*high x n* < *2* ^ *m* ∧ *low x n* < *2* ^ *n*› *member-bound succ-member* **by** *auto*
let *?y* = *2*^(*deg div 2*)∗ *?h* + *succy*
have *08*: *vebt-member* (*treeList* ! *?h*) *succy*
using *06 set-vebt'-def* **by** *auto*
hence *09*: *both-member-options* (*treeList* ! *?h*) *succy*
using *01 both-member-options-equiv-member* **by** *blast*
have *10*: *high ?y* (*deg div 2*) = *?h* ∧ *low ?y* (*deg div 2*) = *succy*
by (*simp add*: *07 high-inv low-inv mult.commute*)
hence *11*: *naive-member* (*treeList* ! *?h*) *succy*
⟹ *naive-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *naive-member.simps*(*3*)[*of Some* (*mi*, *ma*) *deg*−*1 treeList summary ?y*]
using *07 5.hyps*(*4*) **by** *auto*
have *12*: *?y* ≥ *mi* ∧ *?y* ≤ *ma*
by (*metis 01 07 09 10 5.hyps*(*11*) *5.hyps*(*5*) *5.hyps*(*8*) ‹*deg div 2* = *n*› *less-imp-le-nat*)
hence *13*: *membermima* (*treeList* ! *?h*) *succy*
⟹ *membermima* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *membermima.simps*(*4*)[*of mi ma deg* −*1 treeList summary ?y*]
apply(*cases ?y* = *mi* ∨ *?y* = *ma*)
using *07* **apply** *auto*[*1*]
using *07 10 5.hyps*(*4*) **by** *auto*
hence *14*:*both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *09 11 both-member-options-def* **by** *blast*
have *15*: *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
by (*smt* (*verit*) *07 08 10 12 5.hyps*(*4*) *5.hyps*(*5*) *VEBT-Member.vebt-member.simps*(*5*) *One-nat-def Suc-1 Suc-le-eq Suc-pred* ‹*2* ≤ *deg*› ‹*deg div 2* = *n*› *add-gr-0 div-greater-zero-iff not-less zero-less-numeral*)
have *16*: *Some ?y* = *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
by (*simp add*: ‹*vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some* (*2* ^ (*deg div 2*) ∗ *high x* (*deg div 2*) + *succy*)›)
have *17*: *x* = *?h* ∗ *2*^(*deg div 2*) + *?l*
using *bit-concat-def bit-split-inv* **by** *auto*
have *18*: *?y* − *x* = *?h* ∗ *2*^(*deg div 2*) + *succy* − *?h* ∗ *2*^(*deg div 2*) − *?l*
by (*metis 17 diff-diff-add mult.commute*)
hence *?y* −*x* > *0*
using *04 is-succ-in-set-def* **by** *auto*
hence *19*: *?y* > *x*
using *zero-less-diff* **by** *blast*

have *20*: $z > x \Longrightarrow$ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) $z \Longrightarrow z \geq$ *?y* **for** z
proof−
assume $z > x$ **and** *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) z
hence *high z* (*deg div 2*) ≥ *high x* (*deg div 2*)
by (*simp add*: *div-le-mono high-def*)
then show *?thesis*
proof(*cases high z* (*deg div 2*) = *high x* (*deg div 2*))
case *True*
hence *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary z*]
by (*metis 01 07 5.hyps*(*11*) *5.hyps*(*5*) *False* ‹*deg div 2* = *n*› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z*› ‹*x* < *z*› *both-member-options-equiv-member member-inv*)
hence *succy* ≤ *low z* (*deg div 2*) **using** *04* **unfolding** *is-succ-in-set-def*
by (*metis True* ‹*x* < *z*› *add-diff-cancel-left*′ *bit-concat-def bit-split-inv diff-diff-left mem-Collect-eq set-vebt*′*-def zero-less-diff*)
hence *?y* ≤ *z*
by (*metis 10 True bit-concat-def bit-split-inv nat-add-left-cancel-le*)
then show *?thesis* **by** *blast*
next
case *False*
hence *high z* (*deg div 2*) > *high ?y* (*deg div 2*)
using *10* ‹*high x* (*deg div 2*) ≤ *high z* (*deg div 2*)› **by** *linarith*
then show *?thesis*
by (*metis div-le-mono high-def nat-le-linear not-le*)
qed
qed
hence *is-succ-in-set* (*set-vebt*′ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x ?y*
by (*simp add*: *15 19 succ-member*)
then show *?thesis* **using** *16*
by (*metis eq-iff option.inject succ-member*)
next
case *False*
hence *i1*:*?maxlow* = *None* ∨ ¬ (*Some ?l* <$_o$ *?maxlow*) **by** *simp*
hence *2*: *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if ?sc* = *None then None*
else Some (*2*^(*deg div 2*)) *+*$_o$ *?sc* *$_o$ *vebt-mint* (*treeList* ! *the ?sc*))
using *1* **by** *auto*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *5*(*1*) *True* **by** *simp*
hence *33*:∄ *u*. *vebt-member* (*treeList* ! *?h*) *u* ∧ *u* > *?l*
proof(*cases ?maxlow* = *None*)
case *True*
then show *?thesis* **using** *maxt-corr-help-empty*[*of treeList* ! *?h n*]
by (*simp add*: ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *set-vebt*′*-def*)
next
case *False*
obtain *maxilow* **where** *?maxlow* =*Some maxilow*
using *False* **by** *blast*

hence *maxilow* ≤ *?l*
using *i1* **by** *auto*
then show *?thesis*
by (*meson* ‹*vebt-maxt* (*treeList* ! *high x* (*deg div 2*)) = *Some maxilow*› ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *le-imp-less-Suc le-less-trans maxt-corr-help not-less-eq*)
qed
then show *?thesis*
proof(*cases ?sc* = *None*)
case *True*
hence *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*
by (*simp add*: *2*)
hence ∄ *i*. *is-succ-in-set* (*set-vebt′ summary*) *?h i*
using *5.hyps*(*3*) *True* **by** *force*
hence ∄ *i*. *i* > *?h* ∧ *vebt-member summary i* **using** *succ-none-empty*[*of set-vebt′ summary ?h*]
proof −
{ fix *nn* :: *nat*
have ∀ *n*. ((*is-succ-in-set* (*Collect* (*vebt-member summary*)) (*high x* (*deg div 2*)) *esk1-0* ∨ *infinite* (*Collect* (*vebt-member summary*))) ∨ *n* ∉ *Collect* (*vebt-member summary*)) ∨ ¬ *high x* (*deg div 2*) < *n*
using ‹∄ *i*. *is-succ-in-set* (*set-vebt′ summary*) (*high x* (*deg div 2*)) *i*› *succ-none-empty set-vebt′-def* **by** *auto*
then have ¬ *high x* (*deg div 2*) < *nn* ∨ ¬ *vebt-member summary nn*
using *5.hyps*(*2*) ‹∄ *i*. *is-succ-in-set* (*set-vebt′ summary*) (*high x* (*deg div 2*)) *i*› *set-vebt′-def set-vebt-finite* **by** *auto* **}**
then show *?thesis*
by *blast*
qed
hence (*i* > *x* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*) ⟹ *False* **for** *i*
proof−
fix *i*
assume *i* > *x* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*
hence *20*: *i* = *mi* ∨ *i* = *ma* ∨ (*high i* (*deg div 2*) < *length treeList*
∧ *vebt-member* (*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary i*]
using *member-inv* **by** *blast*
have *i* ≠ *mi*
using ‹*mi* ≤ *x*› ‹*x* < *i* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *not-le* **by** *blast*
hence *mi* ≠ *ma*
using ‹*x* < *i* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *member-inv not-less-iff-gr-or-eq* **by** *blast*
hence *i* < *2*^*deg*
using *5.hyps*(*10*) ‹*i* ≠ *mi*› ‹*x* < *i* ∧ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *i*› *member-inv* **by** *fastforce*
hence *aa*:*i* = *ma* ⟹ *both-member-options*(*treeList* ! (*high i* (*deg div 2*))) (*low i* (*deg div 2*))
using *5.hyps*(*11*) *5.hyps*(*2*) *5.hyps*(*6*) ‹*deg div 2* = *n*› ‹*i* ≠ *mi*› ‹*invar-vebt* (*treeList* !

high x (deg div 2)) n› deg-not-0 exp-split-high-low(1) **by** *auto*
hence *abc:invar-vebt (treeList ! (high i (deg div 2))) n*
using *5.hyps(1) 5.hyps(4) 5.hyps(5) 5.hyps(6)*
‹deg div 2 = n› ‹i < 2 ^ deg› ‹invar-vebt (treeList ! high x (deg div 2)) n›
exp-split-high-low(1) [of i n ‹Suc n›] deg-not-0 **by** *auto*
hence *abd:i = ma ⟹ vebt-member(treeList ! (high i (deg div 2))) (low i (deg div 2))*
using *aa valid-member-both-member-options* **by** *blast*
hence *abe:vebt-member(treeList ! (high i (deg div 2))) (low i (deg div 2))*
using *20 ‹i ≠ mi›* **by** *blast*
hence *abf:both-member-options(treeList ! (high i (deg div 2))) (low i (deg div 2))*
using *‹invar-vebt (treeList ! high i (deg div 2)) n› both-member-options-equiv-member* **by** *blast*
hence *abg:both-member-options summary (high i (deg div 2))*
by *(metis (full-types) 5.hyps(5) 5.hyps(6) 5.hyps(7) ‹deg div 2 = n› ‹i < 2 ^ deg› abc deg-not-0 exp-split-high-low(1) zero-less-Suc)*
hence *abh:vebt-member summary (high i (deg div 2))*
using *5.hyps(2) valid-member-both-member-options* **by** *blast*
have *aaa:(high i (deg div 2)) = (high x (deg div 2)) ⟹ vebt-member (treeList ! ?h) (low i (deg div 2))*
using *‹vebt-member (treeList ! high i (deg div 2)) (low i (deg div 2))›* **by** *auto*
have *abi:(high i (deg div 2)) = (high x (deg div 2)) ⟹ low i (deg div 2) > ?l*
by *(metis ‹x < i ∧ vebt-member (Node (Some (mi, ma)) deg treeList summary) i› add-le-cancel-left bit-concat-def bit-split-inv le-neq-implies-less less-imp-le-nat nat-neq-iff)*
hence *abj:(high i (deg div 2)) = (high x (deg div 2)) ⟹ False* **using** *33 aaa* **by** *blast*
hence *abk: (high i (deg div 2)) ∈ (set-vebt' summary) ∧ (high i (deg div 2)) > (high x (deg div 2))*
by *(metis (full-types) ‹vebt-member summary (high i (deg div 2))› ‹x < i ∧ vebt-member (Node (Some (mi, ma)) deg treeList summary) i› div-le-mono high-def le-less mem-Collect-eq set-vebt'-def)*

then show *?thesis*
using *‹¬ (∃ i>high x (deg div 2). vebt-member summary i)› abh* **by** *blast*
qed
then show *?thesis*
using *‹vebt-succ (Node (Some (mi, ma)) deg treeList summary) x = None› succ-member* **by** *auto*
next
case *False*
hence *vebt-succ (Node (Some (mi, ma)) deg treeList summary) x =*
*Some (2^(deg div 2)) *o ?sc +o vebt-mint (treeList ! the ?sc)*
by *(simp add: False 2)*
obtain *sc* **where** *?sc = Some sc*
using *False* **by** *blast*
hence *is-succ-in-set (set-vebt' summary) ?h sc*
using *5.hyps(3)* **by** *blast*
hence *vebt-member summary sc*
using *succ-member* **by** *blast*
hence *both-member-options summary sc*
using *5.hyps(2) both-member-options-equiv-member* **by** *auto*
hence *sc < 2^m*

using *5.hyps(2)* ‹*vebt-member summary sc*› *member-bound* **by** *blast*
hence ∃ *miny*. *both-member-options* (*treeList* ! *sc*) *miny*
using *5.hyps(7)* ‹*both-member-options summary sc*› **by** *blast*
hence *fgh:set-vebt'* (*treeList* ! *sc*) ≠ {}
using *5.hyps(1) 5.hyps(4)* ‹*sc* < 2 ^ *m*› *valid-member-both-member-options* [*of* ‹*treeList* ! *sc*›]
by (*auto simp add*: *set-vebt'-def*) (*metis in-set-conv-nth*)
hence *invar-vebt* (*treeList* ! *the ?sc*) *n*
by (*simp add*: *5.hyps(1) 5.hyps(4)* ‹*sc* < 2 ^ *m*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*›)
then obtain *miny* **where** *Some miny* = *vebt-mint* (*treeList* ! *sc*)
by (*metis fgh Collect-empty-eq VEBT-Member.vebt-member.simps(2) vebt-buildup.simps(2) buildup-gives-empty vebt-mint.elims set-vebt'-def*)
hence *Some miny* = *vebt-mint* (*treeList* ! *the ?sc*)
by (*simp add*: ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*›)
hence *min-in-set* (*set-vebt'* (*treeList* ! *the ?sc*)) *miny*
using ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *mint-corr* **by** *auto*
hence *scmem:vebt-member* (*treeList* ! *the ?sc*) *miny*
using ‹*Some miny* = *vebt-mint* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*))))›
‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *mint-member* **by** *auto*
let *?res* = *Some* (2^(*deg div 2*)) *$*_o_ *?sc* +_o_ *vebt-mint* (*treeList* ! *the ?sc*)
obtain *res* **where** *res* = *the ?res* **by** *blast*
hence *res* = 2^(*deg div 2*) * *sc* + *miny*
by (*metis* ‹*Some miny* = *vebt-mint* (*treeList* ! *sc*)› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› *add-shift mul-shift option.sel*)
have *high res* (*deg div 2*) = *sc*
by (*metis* ‹*deg div 2* = *n*› ‹*res* = 2 ^ (*deg div 2*) * *sc* + *miny*› ‹*invar-vebt* (*treeList* ! *the ?sc*) *n*› *high-inv member-bound mult.commute scmem*)
hence *res* > *x*
by (*metis is-succ-in-set-def* ‹*is-succ-in-set* (*set-vebt' summary*) (*high x* (*deg div 2*)) *sc*› *div-le-mono high-def not-le*)
hence *res* > *mi*
using ‹*mi* ≤ *x*› *le-less-trans* **by** *blast*
hence *res* ≤ *ma*
proof(*cases high res n* < *high ma n*)
case *True*
then show *?thesis*
by (*metis div-le-mono high-def leD nat-le-linear*)
next
case *False*
hence *mi* ≠ *ma*
by (*metis 5.hyps(4) 5.hyps(8)* ‹∃ *miny*. *both-member-options* (*treeList* ! *sc*) *miny*› ‹*sc* < 2 ^ *m*› *nth-mem*)
have *high res n* < 2^*m*
using ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*sc* < 2 ^ *m*› **by** *blast*
hence (∀ *x*. *high x n* = *high res n* ∧ *both-member-options* (*treeList* ! (*high res n*)) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*) **using** *5*(*11*)

using ‹*mi* ≠ *ma*› **by** *blast*

have *high res n* = *high res n* ∧ *both-member-options* (*treeList* ! (*high res n*)) (*low res n*)

by (*metis* ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*res* = 2 ^ (*deg div 2*) ∗ *sc* + *miny*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *both-member-options-equiv-member low-inv member-bound mult.commute option.sel scmem*)

then show *?thesis*

using ‹∀ *x*. *high x n* = *high res n* ∧ *both-member-options* (*treeList* ! *high res n*) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*› **by** *blast*

qed

hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*the ?res*) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary res*]

by (*metis 5.hyps*(*4*) ‹*2* ≤ *deg*› ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *sc*› ‹*mi* < *res*› ‹*res* = 2 ^ (*deg div 2*) ∗ *sc* + *miny*› ‹*res* = *the* (*Some* (2 ^ (*deg div 2*)) ∗$_o$ *vebt-succ summary* (*high x* (*deg div 2*)) +$_o$ *vebt-mint* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))))› ‹*sc* < 2 ^ *m*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› *add-2-eq-Suc*′ *le-add-diff-inverse2 less-imp-le low-inv member-bound mult.commute not-less option.sel scmem*)

have (*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* > *x*) ⟹ *z* ≥ *res* **for** *z*

proof−

fix *z*

assume *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* > *x*

hence *20*: *z* = *mi* ∨ *z* = *ma* ∨ (*high z* (*deg div 2*) < *length treeList*
∧ *vebt-member* (*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary z*]

using *member-inv* **by** *blast*

have *z* ≠ *mi*

using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x* < *z*› ‹*mi* ≤ *x*› *not-le* **by** *blast*

hence *mi* ≠ *ma*

using ‹*mi* < *res*› ‹*res* ≤ *ma*› *not-le* **by** *blast*

hence *z* < *2*^*deg*

using *5.hyps*(*10*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x* < *z*› ‹*z* ≠ *mi*› *member-inv* **by** *fastforce*

hence *aa*:*z* = *ma* ⟹ *both-member-options*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))

using *5.hyps*(*11*) *5.hyps*(*2*) *5.hyps*(*6*) ‹*deg div 2* = *n*› ‹*mi* ≠ *ma*› ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *deg-not-0 exp-split-high-low*(*1*) **by** *auto*

hence *abc*:*invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n*

by (*metis 20 5.hyps*(*1*) *5.hyps*(*10*) *5.hyps*(*4*) *5.hyps*(*5*) *5.hyps*(*6*) ‹*deg div 2* = *n*› ‹*invar-vebt* (*treeList* ! *the*(*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› ‹*z* ≠ *mi*› *deg-not-0 exp-split-high-low*(*1*) *nth-mem zero-less-Suc*)

hence *abd*:*z* = *ma* ⟹ *vebt-member*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))

using *aa valid-member-both-member-options* **by** *blast*

hence *abe*:*vebt-member*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))

using *20* ‹*z* ≠ *mi*› **by** *blast*

hence *abf*:*both-member-options*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))

using ‹*invar-vebt* (*treeList* ! *high z* (*deg div 2*)) *n*› *both-member-options-equiv-member* **by**

blast
hence *abg*:*both-member-options summary* (*high z* (*deg div 2*))
by (*metis* (*full-types*) *5.hyps*(*5*) *5.hyps*(*6*) *5.hyps*(*7*) ‹*deg div 2 = n*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*)))) *n*› ‹*z < 2 ^ deg*› *deg-not-0 exp-split-high-low*(*1*) *zero-less-Suc*)
hence *abh*:*vebt-member summary* (*high z* (*deg div 2*))
using *5.hyps*(*2*) *valid-member-both-member-options* **by** *blast*
have *aaa*:(*high z* (*deg div 2*)) = (*high x* (*deg div 2*)) ⟹ *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *abe* **by** *auto*
have *high z*(*deg div 2*)< *sc* ⟹ *False*
proof−
assume *high z*(*deg div 2*)< *sc*
hence *vebt-member summary* (*high z*(*deg div 2*))
using *abh* **by** *blast*
have *aaaa*:*?h* ≤ *high z*(*deg div 2*)
by (*simp add*: ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x < z*› *div-le-mono high-def less-imp-le-nat*)
have *bbbb*:*?h* ≥ *high z*(*deg div 2*)
using ‹*is-succ-in-set* (*set-vebt*′ *summary*) (*high x* (*deg div 2*)) *sc*› ‹*high z* (*deg div 2*) < *sc*› *abh leD succ-member* **by** *auto*
hence *?h* = *high z* (*deg div 2*)
using *aaaa eq-iff* **by** *blast*
hence *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *aaa* **by** *linarith*
then show *False*
by (*metis 33* ‹*high x* (*deg div 2*) = *high z* (*deg div 2*)› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *x < z*› *add-diff-cancel-left*′ *bit-concat-def bit-split-inv diff-diff-left zero-less-diff*)
qed
hence *high z*(*deg div 2*) ≥ *sc*
using *not-less* **by** *blast*
then show *z* ≥ *res*
proof(*cases high z*(*deg div 2*) = *sc*)
case *True*
hence *vebt-member* (*treeList* ! (*high z*(*deg div 2*))) (*low z* (*deg div 2*))
using *abe* **by** *blast*
have *low z* (*deg div 2*) ≥ *miny*
using *True* ‹*min-in-set* (*set-vebt*′ (*treeList* ! *the* (*vebt-succ summary* (*high x* (*deg div 2*))))) *miny*› ‹*vebt-succ summary* (*high x* (*deg div 2*)) = *Some sc*› *abe min-in-set-def set-vebt*′*-def* **by** *auto*
hence *z* ≥ *res*
by (*metis* (*full-types*) *True* ‹*res* = *2 ^* (*deg div 2*) ∗ *sc* + *miny*› *add-le-cancel-left bit-concat-def bit-split-inv mult.commute*)
then show *?thesis* **by** *simp*
next
case *False*
hence *high z*(*deg div 2*) > *sc*
using ‹*sc* ≤ *high z* (*deg div 2*)› *le-less* **by** *blast*

```
        then show ?thesis
          by (metis ‹high res (deg div 2) = sc› div-le-mono high-def leD linear)
      qed
    qed
    hence is-succ-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x res
      using ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) (the (Some (2 ^ (deg
div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the (vebt-succ summary ?h))))›
        ‹res = the (Some (2 ^ (deg div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the
(vebt-succ summary ?h)))› ‹x < res› succ-member by blast
    moreover have Some res = Some (2^(deg div 2)) *o ?sc +o vebt-mint (treeList ! the ?sc)
      by (metis ‹Some miny = vebt-mint (treeList ! the (vebt-succ summary (high x (deg div 2))))›
‹res = 2 ^ (deg div 2) * sc + miny› ‹vebt-succ summary (high x (deg div 2)) = Some sc› add-def
mul-def option-shift.simps(3))
    ultimately show ?thesis
      by (metis (mono-tags) is-succ-in-set-def ‹vebt-succ (Node (Some (mi, ma)) deg treeList
summary) x = Some (2 ^ (deg div 2)) *o vebt-succ summary ?h +o vebt-mint (treeList ! the (vebt-succ
summary ?h))› eq-iff option.inject)
  qed
 qed
 next
  case False
  hence 0:vebt-succ (Node (Some (mi, ma)) deg treeList summary) x = None
    by (simp add: ‹2 ≤ deg› ‹mi ≤ x› succ-greatereq-min)
  have 1:x ≥ 2^deg
    by (metis 5.hyps(4) 5.hyps(5) 5.hyps(6) False One-nat-def Suc-le-eq ‹1 ≤ n› ‹deg div 2 = n›
exp-split-high-low(1) leI zero-less-Suc)
  hence x ∉ set-vebt' (Node (Some (mi, ma)) deg treeList summary)
    using 5.hyps(10) 5.hyps(9) member-inv set-vebt'-def by fastforce
  hence ∄ ss. is-succ-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x ss
    using 5.hyps(10) 1 ‹mi ≤ x› member-inv succ-member by fastforce
  then show ?thesis using 0 by auto
 qed
qed
qed

corollary succ-empty: assumes invar-vebt t n
 shows (vebt-succ t x = None) = ({y. vebt-member t y ∧ y > x} = {})
proof
 show vebt-succ t x = None ⟹ {y. vebt-member t y ∧ x < y} = {}
 proof
  show vebt-succ t x = None ⟹ {y. vebt-member t y ∧ x < y} ⊆ {}
  proof−
   assume vebt-succ t x = None
   hence ∄ y. is-succ-in-set (set-vebt' t) x y
    using assms succ-corr by force
   moreover hence is-succ-in-set (set-vebt' t) x y ⟹ vebt-member t y ∧ x < y for y by auto
   ultimately show {y. vebt-member t y ∧ x < y} ⊆ {}
    using assms succ-none-empty set-vebt'-def set-vebt-finite by auto
  qed
```

show *vebt-succ t x = None ⟹ {} ⊆ {y. vebt-member t y ∧ x < y}* **by** *simp*
qed
show *{y. vebt-member t y ∧ x < y} = {} ⟹ vebt-succ t x = None*
proof−
assume *{y. vebt-member t y ∧ x < y} = {}*
hence *is-succ-in-set (set-vebt' t) x y ⟹ False* **for** *y*
using *succ-member* **by** *auto*
thus *vebt-succ t x = None*
by (*meson assms not-Some-eq succ-corr*)
qed
qed

theorem *succ-correct*: *invar-vebt t n ⟹ vebt-succ t x = Some sx ⟷is-succ-in-set (set-vebt t) x sx*
by (*simp add*: *succ-corr set-vebt-set-vebt'-valid*)

lemma *is-succ-in-set S x y ⟷ min-in-set {s . s ∈ S ∧ s > x} y*
using *is-succ-in-set-def min-in-set-def* **by** *fastforce*

lemma *helpyd*:*invar-vebt t n ⟹ vebt-succ t x = Some y ⟹ y < 2^n*
using *member-bound succ-corr succ-member* **by** *blast*

lemma *geqmaxNone*:
assumes *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *n x ≥ ma*
shows *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x = None*
proof(*rule ccontr*)
assume *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x ≠ None*
then obtain *y* **where** *vebt-succ* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x = Some y* **by** *auto*
hence *y > ma ∧ y ∈ set-vebt'* ((*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*))
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *assms*(*2*) *dual-order.strict-trans2 member-inv min-in-set-def vebt-mint.simps*(*3*) *mint-corr not-less-iff-gr-or-eq succ-corr succ-member*)
then show *False*
by (*metis assms*(*1*) *leD vebt-maxt.simps*(*3*) *maxt-corr-help mem-Collect-eq set-vebt'-def*)
qed

end
end

theory *VEBT-Pred* **imports** *VEBT-MinMax VEBT-Insert*
begin

7 The Predecessor Operation

definition *is-pred-in-set* :: *nat set ⇒ nat ⇒ nat ⇒ bool* **where**
is-pred-in-set xs x y = (y ∈ xs ∧ y < x ∧ (∀ z ∈ xs. (z < x ⟶ z ≤ y)))

context *VEBT-internal* **begin**

7.1 Lemmas on Sets and Predecessorship

corollary *pred-member*: *is-pred-in-set* (*set-vebt'* *t*) *x y* = (*vebt-member t y* ∧ *y* < *x* ∧ (∀ *z*. *vebt-member t z* ∧ *z* < *x* ⟶ *z* ≤ *y*))
using *is-pred-in-set-def set-vebt'-def* **by** *auto*

lemma *finite* (*A*:: *nat set*) ⟹ *A* ≠ {}⟹ *Max A* ∈ *A*
proof(*induction A rule*: *finite.induct*)
case *emptyI*
then show *?case* **by** *blast*
next
case (*insertI A a*)
then show *?case*
by (*meson Max-in finite-insert*)
qed

lemma *obtain-set-pred*: **assumes** (*x*::*nat*) > *z* **and** *min-in-set A z* **and** *finite A* **shows** ∃ *y*. *is-pred-in-set A x y*
proof−
have {*y* ∈ *A*. *y* < *x*} ≠ {}
using *assms*(*1*) *assms*(*2*) *min-in-set-def* **by** *auto*
hence *Max* {*y* ∈ *A*. *y* < *x*} ∈ {*y* ∈ *A*. *y* < *x*}
by (*metis* (*full-types*) *Max-eq-iff finite-M-bounded-by-nat*)
moreover have *i* ∈ *A*⟹ *i* < *x* ⟹ *i* ≤ *Max* {*y* ∈ *A*. *y* < *x*} **for** *i* **by** *simp*
ultimately have *is-pred-in-set A x* (*Max* {*y* ∈ *A*. *y* < *x*})
using *is-pred-in-set-def* **by** *auto*
then show*?thesis* **by** *auto*
qed

lemma *pred-none-empty*: **assumes** (∄ *x*. *is-pred-in-set* (*xs*) *a x*) **and** *finite xs***shows** ¬ (∃ *x* ∈ *xs*. *ord-class.less x a*)
proof−
have ∃ *x* ∈ *xs*. *ord-class.less x a* ⟹ *False*
proof−
assume ∃ *x* ∈ *xs*. *ord-class.less x a*
hence {*x* ∈ *xs*. *ord-class.less x a*} ≠ {} **by** *auto*
hence *Max* {*y* ∈ *xs*. *y* < *a*} ∈ {*y* ∈ *xs*. *y* < *a*}
by (*metis* (*full-types*) *Max-eq-iff finite-M-bounded-by-nat*)
moreover hence *i* ∈ *xs* ⟹ *ord-class.less i a*⟹
ord-class.less-eq i (*Max* {*y* ∈ *xs*. *ord-class.less y a*}) **for** *i*
by (*simp add*: *assms*(*2*))
ultimately have *is-pred-in-set xs a* (*Max* {*y* ∈ *xs*. *y* < *a*})
using *is-pred-in-set-def* **by** *auto*
then show *False*
using *assms*(*1*) **by** *blast*
qed
then show *?thesis* **by** *blast*
qed

end

7.2 The actual Function for Predecessor Search

context begin
interpretation *VEBT-internal* **.**

fun *vebt-pred* :: *VEBT* ⇒ *nat* ⇒ *nat option* **where**
vebt-pred (*Leaf* - -) *0* = *None*|
vebt-pred (*Leaf a* -) (*Suc 0*) = (*if a then Some 0 else None*)|
vebt-pred (*Leaf a b*) - = (*if b then Some 1 else if a then Some 0 else None*)|
vebt-pred (*Node None* - - -) - = *None*|
vebt-pred (*Node* - *0* - -) - = *None*|
vebt-pred (*Node* - (*Suc 0*) - -) - = *None*|
vebt-pred (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (
if x > *ma then Some ma*
else (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in*
if h < *length treeList then*
let minlow = *vebt-mint* (*treeList* ! *h*) *in* (
if minlow ≠ *None* ∧ (*Some l* >*o* *minlow*) *then*
Some (*2*^(*deg div 2*)) ∗*o Some h* +*o vebt-pred* (*treeList* ! *h*) *l*
else let pr = *vebt-pred summary h in*
if pr = *None then* (
if x > *mi then Some mi*
else None)
else Some (*2*^(*deg div 2*)) ∗*o pr* +*o vebt-maxt* (*treeList* ! *the pr*))
else None))

end

context *VEBT-internal* **begin**

7.3 Auxiliary Lemmas

lemma *pred-max*:
assumes *deg* ≥ *2* **and** (*x*::*nat*) > *ma*
shows *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some ma*
by (*metis VEBT-Pred.vebt-pred.simps*(*7*) *add-2-eq-Suc assms*(*1*) *assms*(*2*) *le-add-diff-inverse*)

lemma *pred-lesseq-max*:
assumes *deg* ≥ *2* **and** (*x*::*nat*) ≤ *ma*
shows *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in*
if h < *length treeList then*

let minlow = *vebt-mint* (*treeList* ! *h*) *in*
(*if minlow* ≠ *None* ∧ (*Some l* >*o* *minlow*) *then*
Some (*2*^(*deg div 2*)) ∗*o Some h* +*o vebt-pred* (*treeList* ! *h*) *l*
else let pr = *vebt-pred summary h in*
if pr = *None then* (*if x* > *mi then Some mi else None*)
else Some (*2*^(*deg div 2*)) ∗*o pr* +*o vebt-maxt* (*treeList* ! *the pr*))

```
                else None)
proof −
  obtain deg' where deg = Suc (Suc deg')
    using assms(1) add-2-eq-Suc le-Suc-ex by blast

  have ¬ ma < x
    using assms(2)
    by presburger

  show ?thesis
    unfolding ‹deg = Suc (Suc deg')› vebt-pred.simps if-not-P[OF ‹¬ ma < x›]
    by metis
qed

lemma pred-list-to-short:
  assumes deg ≥ 2 and ord-class.less-eq x ma and  high x (deg div 2) ≥ length treeList
  shows vebt-pred (Node (Some (mi, ma)) deg treeList summary) x = None
  by (simp add: assms(1) assms(2) assms(3) leD pred-lesseq-max)


lemma pred-less-length-list:
  assumes deg ≥ 2 and ord-class.less-eq x  ma and  high x (deg div 2) < length treeList
  shows
  vebt-pred (Node (Some (mi, ma)) deg treeList summary) x = (let l = low x (deg div 2); h = high x
(deg div 2); minlow = vebt-mint (treeList ! h) in
                        (if minlow ≠ None ∧ (Some l >o  minlow) then
                                        Some (2^(deg div 2)) *o Some h +o vebt-pred (treeList ! h) l
                         else let pr = vebt-pred summary h in
                         if pr = None then (if x > mi then Some mi else None)
                         else Some (2^(deg div 2)) *o pr +o vebt-maxt (treeList ! the pr) ))
  by (simp add: assms(1) assms(2) assms(3) pred-lesseq-max)
```

7.4 Correctness Proof

```
theorem pred-corr: invar-vebt t n ⟹ vebt-pred t x = Some px == is-pred-in-set (set-vebt' t) x px
proof(induction t n arbitrary: x px rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
  proof(cases x)
    case 0
    then show ?thesis
      by (simp add: is-pred-in-set-def)
  next
    case (Suc sucX)
    hence x ≥ 0 ∧ x = Suc sucX by auto
    then show ?thesis
    proof(cases sucX)
      case 0
      then show ?thesis
```

```
      by (simp add: Suc pred-member)
    next
      case (Suc nat)
      hence x≥ 2
        by (simp add: ‹0 ≤ x ∧ x = Suc sucX›)
      then show ?thesis
      proof(cases b)
        case True
        hence vebt-pred (Leaf a b) x = Some 1
          by (simp add: Suc ‹0 ≤ x ∧ x = Suc sucX›)
        moreover have is-pred-in-set (set-vebt' (Leaf a b)) x 1
          by (simp add: Suc True ‹0 ≤ x ∧ x = Suc sucX› pred-member)
        ultimately show ?thesis
          using pred-member by auto
      next
        case False
        hence b = False by simp
        then show ?thesis
        proof(cases a)
          case True
          hence vebt-pred (Leaf a b) x = Some 0
            by (simp add: False Suc ‹0 ≤ x ∧ x = Suc sucX›)
          moreover have is-pred-in-set (set-vebt' (Leaf a b)) x 0
            by (simp add: False True ‹0 ≤ x ∧ x = Suc sucX› pred-member)
          ultimately show ?thesis
            by (metis False VEBT-Member.vebt-member.simps(1) option.sel pred-member)
        next
          case False
          then show ?thesis
            by (simp add: Suc ‹0 ≤ x ∧ x = Suc sucX› pred-member)
        qed
      qed
    qed
  qed
next
  case (2 treeList n summary m deg)
  then show ?case
    by (simp add: pred-member)
next
  case (3 treeList n summary m deg)
  then show ?case
    by (simp add: pred-member)
next
  case (4 treeList n summary m deg mi ma)
  hence n = m and n ≥ 1 and deg ≥ 2 and deg = n + m
      apply blast+
    using 4.hyps(2) 4.hyps(5) Suc-le-eq deg-not-0 apply auto[1]
    using 4.hyps(2) 4.hyps(5) 4.hyps(6) deg-not-0 apply fastforce
    by (simp add: 4.hyps(6))
```

moreover hence *thisvalid*:*invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*
using *4 invar-vebt.intros*(*4*)[*of treeList n summary m*] **by** *blast*
ultimately have *deg div 2* =*n* **and** *length treeList* = *2*^*n*
using *add-self-div-2* **apply** *blast* **by** (*simp add*: *4.hyps*(*4*) *4.hyps*(*5*))
then show *?case*
proof(*cases x* > *ma*)
case *True*
hence *0*: *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some ma*
by (*simp add*: ‹*2* ≤ *deg*› *pred-max*)
have *1*:*ma* = *the* (*vebt-maxt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) **by** *simp*
hence *ma* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*metis VEBT-Member.vebt-member.simps*(*5*) ‹*2* ≤ *deg*› *add-numeral-left arith-simps*(*1*) *le-add-diff-inverse mem-Collect-eq numerals*(*1*) *plus-1-eq-Suc set-vebt'-def*)
hence *2*:*y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ *y* ≤ *x* **for** *y*
using *4.hyps*(*9*) *True member-inv set-vebt'-def* **by** *fastforce*
hence *3*: *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ (*y* < *ma* ⟹ *y* ≤ *x*) **for** *y* **by** *blast*
hence *4*: ∀ *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*). *y* < *ma* ⟶ *y* ≤ *x* **by** *blast*
hence *is-pred-in-set* (*set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x ma*
by (*metis 4.hyps*(*9*) *True* ‹*ma* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)› *less-or-eq-imp-le mem-Collect-eq member-inv pred-member set-vebt'-def*)
then show *?thesis*
by (*metis 0 option.sel leD le-less-Suc-eq not-less-eq pred-member*)
next
case *False*
hence *x* ≤ *ma***by** *simp*
then show *?thesis*
proof(*cases high x* (*deg div 2*)< *length treeList*)
case *True*
hence *high x n* < *2*^*n* ∧ *low x n* < *2*^*n*
by (*simp add*: ‹*deg div 2* = *n*› ‹*length treeList* = *2* ^ *n*› *low-def*)
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
let *?minlow* = *vebt-mint* (*treeList* ! *?h*)
let *?pr* = *vebt-pred summary ?h*
have *1*:*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
(*if ?minlow* ≠ *None* ∧ (*Some ?l* >*o* *?minlow*) *then*
Some (*2*^(*deg div 2*)) ∗*o* *Some ?h* +*o* *vebt-pred* (*treeList* ! *?h*) *?l*
else let pr = *vebt-pred summary ?h in*
if pr = *None then* (*if x* > *mi then Some mi else None*)
else Some (*2*^(*deg div 2*)) ∗*o* *pr* +*o* *vebt-maxt* (*treeList* ! *the pr*))
by (*smt* (*verit*) *True* ‹*2* ≤ *deg*› ‹*x* ≤ *ma*› *pred-less-length-list*)
then show *?thesis*
proof(*cases ?minlow* ≠ *None* ∧ (*Some ?l* >*o* *?minlow*))
case *True*
then obtain *minl* **where** *00*:(*Some minl* = *?minlow*) ∧ *?l* > *minl* **by** *auto*
have *01*:*invar-vebt* ((*treeList* ! *?h*)) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*

by (*simp add*: *4.hyps*(*1*) *4.hyps*(*4*) *4.hyps*(*5*) ‹*deg div 2 = n*› ‹*high x n < 2 ^ n ∧ low x n < 2 ^ n*›)
have *02*:*vebt-member* ((*treeList* ! *?h*)) *minl*
using *00 01 mint-member* **by** *auto*
hence *03*: ∃ *y*. *y < ?l ∧ vebt-member* ((*treeList* ! *?h*)) *y*
using *00* **by** *blast*
hence *afinite*: *finite* (*set-vebt'* (*treeList* ! *?h*))
using *01 set-vebt-finite* **by** *blast*
then obtain *predy* **where** *04*:*is-pred-in-set* (*set-vebt'* (*treeList* ! *?h*)) *?l predy*
using *00 01 mint-corr obtain-set-pred* **by** *fastforce*
hence *05*:*Some predy = vebt-pred* (*treeList* ! *?h*) *?l* **using** *4*(*1*) *01* **by** *force*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x = Some* (*2^*(*deg div 2*)∗ *?h + predy*)
using *1 True add-def mul-def option-shift.simps*(*3*) **by** *metis*
hence *06*: *predy* ∈ *set-vebt'* (*treeList* ! *?h*)
using *04 is-pred-in-set-def* **by** *blast*
hence *07*: *predy < 2^*(*deg div 2*) ∧ *?h < 2^*(*deg div 2*) ∧ *deg div 2 + deg div 2 = deg*
using *01 04 4.hyps*(*5*) *4.hyps*(*6*) ‹*high x n < 2 ^ n ∧ low x n < 2 ^ n*› *member-bound pred-member* **by** *auto*
let *?y = 2^*(*deg div 2*)∗ *?h + predy*
have *08*: *vebt-member* (*treeList* ! *?h*) *predy*
using *06 set-vebt'-def* **by** *auto*
hence *09*: *both-member-options* (*treeList* ! *?h*) *predy*
using *01 both-member-options-equiv-member* **by** *blast*
have *10*: *high ?y* (*deg div 2*) = *?h ∧ low ?y* (*deg div 2*) = *predy*
by (*simp add*: *07 high-inv low-inv mult.commute*)
hence *14*:*both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
by (*metis 07 09 4.hyps*(*4*) *4.hyps*(*5*) *Suc-1* ‹*2 ≤ deg*› ‹*deg div 2 = n*› *add-leD1 both-member-options-from-chilf-to-d plus-1-eq-Suc*)
have *15*: *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *14 thisvalid valid-member-both-member-options* **by** *blast*
have *16*: *Some ?y = vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
by (*simp add*: ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x = Some* (*2 ^* (*deg div 2*) ∗ *high x* (*deg div 2*) + *predy*)›)
have *17*: *x = ?h ∗ 2^*(*deg div 2*) + *?l*
using *bit-concat-def bit-split-inv* **by** *auto*
have *18*: *x − ?y = ?h ∗ 2^*(*deg div 2*) + *?l* −*?h ∗ 2^*(*deg div 2*) − *predy*
by (*metis 17 diff-diff-add mult.commute*)
hence *19*: *?y < x*
using *04 17 mult.commute nat-add-left-cancel-less pred-member* **by** *fastforce*
have *20*: *z < x* ⟹ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ⟹ *z*≤ *?y* **for** *z*
proof−
assume *z < x* **and** *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z*
hence *high z* (*deg div 2*) ≤ *high x* (*deg div 2*)
by (*simp add*: *div-le-mono high-def*)
then show *?thesis*
proof(*cases high z* (*deg div 2*) = *high x* (*deg div 2*))
case *True*

```
      hence 0000: high z (deg div 2) = high x (deg div 2) by simp
      then show ?thesis
      proof(cases z = mi)
        case True
        then show ?thesis
          using 15 vebt-mint.simps(3) mint-corr-help thisvalid by blast
      next
        case False
        hence ad:vebt-member (treeList ! ?h) (low z (deg div 2))
          using vebt-member.simps(5)[of mi ma deg−2 treeList summary z]
            by (metis True ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) z› ‹x ≤
ma› ‹z < x› leD member-inv)
        have is-pred-in-set (set-vebt' (treeList ! ?h)) ?l predy
          using 04 by blast
        have low z (deg div 2) < ?l
          by (metis (full-types) True ‹z < x› bit-concat-def bit-split-inv nat-add-left-cancel-less)
        hence predy ≥ low z (deg div 2) using 04 ad unfolding is-pred-in-set-def
          by (simp add: set-vebt'-def)
        hence ?y ≥ z
          by (metis True bit-concat-def bit-split-inv mult.commute nat-add-left-cancel-le)
        then show ?thesis by blast
      qed
    next
      case False
      hence high z (deg div 2) < high ?y (deg div 2)
        using 10 ‹high z (deg div 2) ≤ high x (deg div 2)› by linarith
      then show ?thesis
        by (metis div-le-mono high-def nat-le-linear not-le)
    qed
  qed
  hence is-pred-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x ?y
    by (simp add: 15 19 pred-member)
  then show ?thesis using 16
    by (metis eq-iff option.inject pred-member)
next
  case False
  hence i1:?minlow = None ∨ ¬ (Some ?l >o ?minlow) by simp
  hence 2: vebt-pred (Node (Some (mi, ma)) deg treeList summary) x = (
                  if ?pr = None then (if x > mi
                                      then Some mi
                                      else None)
                   else Some (2^(deg div 2)) *o ?pr +o vebt-maxt (treeList ! the ?pr))
    using 1 by auto
  have invar-vebt (treeList ! ?h) n
    using 4(1) True by simp
  hence 33:∄ u. vebt-member (treeList ! ?h) u ∧ u < ?l
  proof(cases ?minlow = None)
    case True
    then show ?thesis using mint-corr-help-empty[of treeList ! ?h n]
```

by (*simp add*: ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *set-vebt'-def*)
next
case *False*
obtain *minilow* **where** *?minlow* =*Some minilow*
using *False* **by** *blast*
hence *minilow* ≥ *?l*
using *i1* **by** *auto*
then show *?thesis*
by (*meson* ‹*vebt-mint* (*treeList* ! *high x* (*deg div 2*)) = *Some minilow*› ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *leD less-le-trans mint-corr-help*)
qed
then show *?thesis*
proof(*cases ?pr*= *None*)
case *True*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if x* > *mi then Some mi else None*)
by (*simp add*: *2*)
hence ∄ *i*. *is-pred-in-set* (*set-vebt' summary*) *?h i*
using *4*.*hyps*(*3*) *True* **by** *force*
hence ∄ *i*. *i* < *?h* ∧ *vebt-member summary i* **using** *pred-none-empty*[*of set-vebt' summary ?h*]
proof −
{ fix *nn* :: *nat*
have ∀ *n*. ((*is-pred-in-set* (*Collect* (*vebt-member summary*)) (*high x* (*deg div 2*)) *esk1-0* ∨ *infinite* (*Collect* (*vebt-member summary*))) ∨ *n* ∉ *Collect* (*vebt-member summary*)) ∨ ¬ *n* < *high x* (*deg div 2*)
using ‹∄ *i*. *is-pred-in-set* (*set-vebt' summary*) (*high x* (*deg div 2*)) *i*› *pred-none-empty set-vebt'-def* **by** *auto*
then have ¬ *nn* < *high x* (*deg div 2*) ∨ ¬ *vebt-member summary nn*
by (*metis* (*no-types*) *4*.*hyps*(*2*) ‹∄ *i*. *is-pred-in-set* (*set-vebt' summary*) (*high x* (*deg div 2*)) *i*› *mem-Collect-eq set-vebt'-def set-vebt-finite*) **}**
then show *?thesis*
by *blast*
qed
then show *?thesis*
proof(*cases x* > *mi*)
case *True*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*
by (*simp add*: ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if mi* < *x then Some mi else None*)›)
have (*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *z* > *mi*) ⟹ *False* **for** *z*
proof−
assume *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *z* > *mi*
hence *vebt-member* (*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))
using ‹*x* ≤ *ma*› *member-inv not-le* **by** *blast*
moreover hence *high z* (*deg div 2*) < *2*^*m*
using *4*.*hyps*(*4*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› ‹*x* ≤ *ma*› *member-inv* **by** *fastforce*

moreover hence *invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n* **using** *4*(*1*)
by (*simp add*: *4.hyps*(*4*))
ultimately have *vebt-member summary* (*high z* (*deg div 2*)) **using** *4*(*7*)
using *4.hyps*(*2*) *both-member-options-equiv-member* **by** *blast*
have (*high z* (*deg div 2*)) ≤ *?h*
by (*simp add*: ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› *div-le-mono high-def less-or-eq-imp-le*)
then show *False*
by (*metis 33* ‹¬ (∃ *i*<*high x* (*deg div 2*). *vebt-member summary i*)› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› ‹*vebt-member* (*treeList* ! *high z* (*deg div 2*)) (*low z* (*deg div 2*))› ‹*vebt-member summary* (*high z* (*deg div 2*))› *bit-concat-def bit-split-inv le-neq-implies-less nat-add-left-cancel-less*)
qed
hence *is-pred-in-set* (*set-vebt'* ((*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*))) *x mi*
by (*metis VEBT-Member.vebt-member.simps*(*5*) *True* ‹*2* ≤ *deg*› *add-2-eq-Suc le-add-diff-inverse le-less-linear pred-member*)
then show *?thesis*
by (*metis* ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*› ‹*x* ≤ *ma*› *option.sel leD member-inv pred-member*)
next
case *False*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*
by (*simp add*: *2 True*)
then show *?thesis*
by (*metis* (*full-types*) *False less-trans member-inv option.distinct*(*1*) *pred-max pred-member*)
qed
next
case *False*
hence *fst*:*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
Some (*2*^(*deg div 2*)) ∗*o* *?pr* +*o* *vebt-maxt* (*treeList* ! *the ?pr*)
using *2* **by** *presburger*
obtain *pr* **where** *?pr* = *Some pr*
using *False* **by** *blast*
hence *is-pred-in-set* (*set-vebt' summary*) *?h pr*
using *4.hyps*(*3*) **by** *blast*
hence *vebt-member summary pr*
using *pred-member* **by** *blast*
hence *both-member-options summary pr*
using *4.hyps*(*2*) *both-member-options-equiv-member* **by** *auto*
hence *pr* < *2*^*m*
using *4.hyps*(*2*) ‹*vebt-member summary pr*› *member-bound* **by** *blast*
hence ∃ *maxy*. *both-member-options* (*treeList* ! *pr*) *maxy*
using *4.hyps*(*7*) ‹*both-member-options summary pr*› **by** *blast*
hence *fgh*:*set-vebt'* (*treeList* ! *pr*) ≠ {}
by (*metis 4.hyps*(*1*) *4.hyps*(*2*) *4.hyps*(*4*) ‹*vebt-member summary pr*› *empty-Collect-eq member-bound nth-mem set-vebt'-def valid-member-both-member-options*)
hence *invar-vebt* (*treeList* ! *the ?pr*) *n*
by (*simp add*: *4.hyps*(*1*) *4.hyps*(*4*) ‹*pr* < *2* ^ *m*› ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*›)

then obtain *maxy* **where** *Some maxy = vebt-maxt (treeList ! pr)*
by (*metis ‹vebt-pred summary (high x (deg div 2)) = Some pr› fgh option.sel vebt-maxt.elims maxt-corr-help-empty*)
hence *Some maxy = vebt-maxt (treeList ! the ?pr)*
by (*simp add*: *‹vebt-pred summary (high x (deg div 2)) = Some pr›*)
hence *max-in-set (set-vebt' (treeList ! the ?pr)) maxy*
using *‹invar-vebt (treeList ! the (vebt-pred summary (high x (deg div 2)))) n› maxt-corr* **by** *auto*
hence *scmem:vebt-member (treeList ! the ?pr) maxy*
using *‹Some maxy = vebt-maxt (treeList ! the (vebt-pred summary (high x (deg div 2))))› ‹invar-vebt (treeList ! the (vebt-pred summary (high x (deg div 2)))) n› maxt-member* **by** *force*
let *?res = Some (2^(deg div 2)) *o ?pr +o vebt-maxt (treeList ! the ?pr)*
obtain *res* **where** *snd*: *res = the ?res* **by** *blast*
hence *res = 2^(deg div 2) * pr + maxy*
by (*metis ‹Some maxy = vebt-maxt (treeList ! pr)› ‹vebt-pred summary (high x (deg div 2)) = Some pr› add-def option.sel mul-def option-shift.simps(3)*)
have *high res (deg div 2) = pr*
by (*metis ‹deg div 2 = n› ‹res = 2 ^ (deg div 2) * pr + maxy› ‹invar-vebt (treeList ! the ?pr) n› high-inv member-bound mult.commute scmem*)
hence *res < x*
by (*metis ‹is-pred-in-set (set-vebt' summary) (high x (deg div 2)) pr› div-le-mono high-def pred-member alethe-comp-simplify1(3)*)
have *both-member-options (treeList ! (high res (deg div 2))) (low res (deg div 2))*
by (*metis ‹deg div 2 = n› ‹high res (deg div 2) = pr› ‹vebt-pred summary (high x (deg div 2)) = Some pr› ‹res = 2 ^ (deg div 2) * pr + maxy› ‹invar-vebt (treeList ! the (vebt-pred summary (high x (deg div 2)))) n› both-member-options-equiv-member option.sel low-inv member-bound mult.commute scmem*)
have *both-member-options (Node (Some (mi, ma)) deg treeList summary) res*
by (*metis 4.hyps(2) 4.hyps(4) 4.hyps(6) ‹1 ≤ n› ‹both-member-options (treeList ! high res (deg div 2)) (low res (deg div 2))› ‹high res (deg div 2) = pr› ‹vebt-member summary pr› both-member-options-from-chilf-to-complete-tree member-bound trans-le-add1*)
hence *vebt-member (Node (Some (mi, ma)) deg treeList summary) res*
using *thisvalid valid-member-both-member-options* **by** *auto*
hence *res > mi*
by (*metis 4.hyps(11) ‹both-member-options (treeList ! high res (deg div 2)) (low res (deg div 2))› ‹deg div 2 = n› ‹high res (deg div 2) = pr› ‹pr < 2 ^ m› ‹res < x› ‹x ≤ ma› less-le-trans member-inv*)
hence *res < ma*
using *‹res < x› ‹x ≤ ma› less-le-trans* **by** *blast*
have *(vebt-member (Node (Some (mi, ma)) deg treeList summary) z ∧ z < x) ⟹ z ≤ res* **for** *z*
proof−
fix *z*
assume *vebt-member (Node (Some (mi, ma)) deg treeList summary) z ∧ z < x*
hence *20*: *z = mi ∨ z = ma ∨ (high z (deg div 2) < length treeList*
∧ vebt-member (treeList ! (high z (deg div 2))) (low z (deg div 2))) **using**
vebt-member.simps(5)[of mi ma deg−2 treeList summary z]
using *member-inv* **by** *blast*
have *z ≠ ma*

using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› ‹*x* ≤ *ma*› *leD* **by** *blast*
hence *mi* ≠ *ma*
by (*metis* ‹*mi* < *res*› ‹*res* < *x*› ‹*x* ≤ *ma*› *leD less-trans*)
hence *z* < *2^deg*
using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *member-bound thisvalid* **by** *blast*
hence *abc*:*invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n*
using *4*.*hyps*(*1*) *4*.*hyps*(*2*) *4*.*hyps*(*5*) *4*.*hyps*(*6*) ‹*deg div 2* = *n*› ‹*z* < *2* ^ *deg*›
‹*length treeList* = *2* ^ *n*› *deg-not-0 exp-split-high-low*(*1*) **by** *simp*
then show *z* ≤ *res*
proof(*cases z* = *mi*)
case *True*
then show *?thesis*
using ‹*mi* < *res*› **by** *auto*
next
case *False*
hence *abe*:*vebt-member*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))
using *20* ‹*z* ≠ *ma*› **by** *blast*
hence *abh*:*vebt-member summary* (*high z* (*deg div 2*))
by (*metis 20 4*.*hyps*(*2*) *4*.*hyps*(*4*) *4*.*hyps*(*7*) *False* ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› ‹*x* ≤ *ma*› *abc both-member-options-equiv-member not-le*)
have *aaa*:(*high z* (*deg div 2*)) = (*high x* (*deg div 2*)) ⟹ *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *abe* **by** *auto*
have *high z*(*deg div 2*) > *pr* ⟹ *False*
proof−
assume *high z*(*deg div 2*) > *pr*
hence *vebt-member summary* (*high z*(*deg div 2*))
using *abh* **by** *blast*
have *aaaa*:*?h* ≤ *high z*(*deg div 2*)
by (*meson* ‹*is-pred-in-set* (*set-vebt*′ *summary*) (*high x* (*deg div 2*)) *pr*› ‹*pr* < *high z* (*deg div 2*)› *abh leD not-le-imp-less pred-member*)
have *bbbb*:*?h* ≥ *high z*(*deg div 2*)
by (*simp add*: ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *div-le-mono dual-order*.*strict-implies-order high-def*)
hence *?h* = *high z* (*deg div 2*)
using *aaaa eq-iff* **by** *blast*
hence *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *aaa* **by** *linarith*
hence (*low z* (*deg div 2*)) < *?l*
by (*metis* ‹*high x* (*deg div 2*) = *high z* (*deg div 2*)› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *add-le-cancel-left div-mult-mod-eq high-def less-le low-def*)
then show *False*
using *33* ‹*vebt-member* (*treeList* ! *high x* (*deg div 2*)) (*low z* (*deg div 2*))› **by** *blast*
qed
hence *high z*(*deg div 2*) ≤ *pr*
using *not-less* **by** *blast*
then show *z* ≤ *res*

```
            proof(cases high z(deg div 2) = pr)
              case True
              hence vebt-member (treeList ! (high z(deg div 2))) (low z (deg div 2))
                using abe by blast
              have low z (deg div 2) ≤ maxy
                using True ‹Some maxy = vebt-maxt (treeList ! pr)› abc abe maxt-corr-help by auto
              hence z ≤ res
                by (metis True ‹res = 2 ^ (deg div 2) * pr + maxy› add-le-cancel-left div-mult-mod-eq
high-def low-def mult.commute)
              then show ?thesis by simp
            next
              case False
              hence high z(deg div 2) < pr
                by (simp add: ‹high z (deg div 2) ≤ pr› less-le)
              then show ?thesis
                by (metis ‹high res (deg div 2) = pr› div-le-mono high-def leD linear)
            qed
          qed
        qed
        hence is-pred-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x res
        using ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) res› ‹res < x› pred-member
by presburger
        then show ?thesis using fst snd
        by (metis ‹Some maxy = vebt-maxt (treeList ! the (vebt-pred summary (high x (deg div 2))))›
‹vebt-pred summary (high x (deg div 2)) = Some pr› ‹res = 2 ^ (deg div 2) * pr + maxy› add-shift
dual-order.eq-iff mul-shift pred-member)
      qed
    qed
  next
    case False
    then show ?thesis
      by (metis 4.hyps(10) 4.hyps(5) 4.hyps(6) ‹1 ≤ n› ‹deg div 2 = n› ‹length treeList = 2 ^ n›
‹x ≤ ma› exp-split-high-low(1) le-less-trans le-neq-implies-less not-less not-less-zero zero-neq-one)
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  hence Suc n = m  and deg = n + m and length treeList = 2^m ∧ invar-vebt summary m
    by blast +
  hence n ≥ 1
    using 5.hyps(1) set-n-deg-not-0 by blast
  hence deg ≥ 2
    by (simp add: 5.hyps(5) 5.hyps(6))
  hence deg div 2 =n
    by (simp add: 5.hyps(5) 5.hyps(6))
  moreover hence thisvalid:invar-vebt (Node (Some (mi, ma)) deg treeList summary) deg
    using 5 invar-vebt.intros(5)[of treeList n summary m]  by blast
  ultimately have deg div 2 =n by simp
  then show ?case
```

proof(*cases x > ma*)
case *True*
hence *0*: *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some ma*
by (*simp add*: ‹*2* ≤ *deg*› *pred-max*)
have *1*:*ma* = *the* (*vebt-maxt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) **by** *simp*
hence *ma* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*metis VEBT-Member.vebt-member.simps*(*5*) ‹*2* ≤ *deg*› *add-numeral-left arith-simps*(*1*) *le-add-diff-inverse mem-Collect-eq numerals*(*1*) *plus-1-eq-Suc set-vebt'-def*)
hence *2*:*y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ *y* ≤ *x* **for** *y*
using *5.hyps*(*9*) *True member-inv set-vebt'-def* **by** *fastforce*
hence *3*: *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) ⟹ (*y* < *ma* ⟹ *y* ≤ *x*) **for** *y* **by** *blast*
hence *4*: ∀ *y* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*). *y* < *ma* ⟶ *y* ≤ *x* **by** *blast*
hence *is-pred-in-set* (*set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) *x ma*
by (*metis 5.hyps*(*9*) *True* ‹*ma* ∈ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)› *less-or-eq-imp-le mem-Collect-eq member-inv pred-member set-vebt'-def*)
then show *?thesis*
by (*metis 0 option.sel leD le-less-Suc-eq not-less-eq pred-member*)
next
case *False*
hence *x* ≤ *ma***by** *simp*
then show *?thesis*
proof(*cases high x* (*deg div 2*)< *length treeList*)
case *True*
hence *high x n* < *2^m* ∧ *low x n* < *2^n*
by (*simp add*: ‹*deg div 2* = *n*› ‹*length treeList* = *2* ^ *m*› *low-def*)
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
let *?minlow* = *vebt-mint* (*treeList* ! *?h*)
let *?pr* = *vebt-pred summary ?h*
have *1*:*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
(*if ?minlow* ≠ *None* ∧ (*Some ?l* >*o* *?minlow*) *then*
Some (*2^*(*deg div 2*)) ∗*o* *Some ?h* +*o* *vebt-pred* (*treeList* ! *?h*) *?l*
else let pr = *vebt-pred summary ?h in*
if pr = *None then* (*if x* > *mi then Some mi else None*)
else Some (*2^*(*deg div 2*)) ∗*o* *pr* +*o* *vebt-maxt* (*treeList* ! *the pr*))
by (*smt* (*verit*) *True* ‹*2* ≤ *deg*› ‹*x* ≤ *ma*› *pred-less-length-list*)
then show *?thesis*
proof(*cases ?minlow* ≠ *None* ∧ (*Some ?l* >*o* *?minlow*))
case *True*
then obtain *minl* **where** *00*:(*Some minl* = *?minlow*) ∧ *?l* > *minl* **by** *auto*
have *01*:*invar-vebt* ((*treeList* ! *?h*)) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*
using *5.hyps*(*1*) ‹*deg div 2* = *n*› ‹*high x n* < *2* ^ *m* ∧ *low x n* < *2* ^ *n*› ‹*length treeList* = *2* ^ *m* ∧ *invar-vebt summary m*›
by *simp*
have *02*:*vebt-member* ((*treeList* ! *?h*)) *minl*
using *00 01 mint-member* **by** *auto*

hence *03*: $\exists\ y.\ y < ?l \wedge$ *vebt-member* ((*treeList* ! *?h*)) *y*
using *00* **by** *blast*
hence *afinite*: *finite* (*set-vebt'* (*treeList* ! *?h*))
using *01 set-vebt-finite* **by** *blast*
then obtain *predy* **where** *04*:*is-pred-in-set* (*set-vebt'* (*treeList* ! *?h*)) *?l predy*
using *00 01 mint-corr obtain-set-pred* **by** *fastforce*
hence *05*:*Some predy* = *vebt-pred* (*treeList* ! *?h*) *?l* **using** *5*(*1*) *01* **by** *force*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some* (*2*^(*deg div 2*)∗ *?h* + *predy*)
by (*metis 1 True add-def mul-def option-shift.simps*(*3*))
hence *06*: *predy* ∈ *set-vebt'* (*treeList* ! *?h*)
using *04 is-pred-in-set-def* **by** *blast*
hence *07*: *predy* < *2*^(*deg div 2*) ∧ *?h* < *2*^(*deg div 2 +1*) ∧ *deg div 2* + *deg div 2 +1* = *deg*
using *04 5.hyps*(*5*) *5.hyps*(*6*) ‹*high x n* < *2* ^ *m* ∧ *low x n* < *2* ^ *n*› *pred-member* **by** *force*
let *?y* = *2*^(*deg div 2*)∗ *?h* + *predy*
have *08*: *vebt-member* (*treeList* ! *?h*) *predy*
using *06 set-vebt'-def* **by** *auto*
hence *09*: *both-member-options* (*treeList* ! *?h*) *predy*
using *01 both-member-options-equiv-member* **by** *blast*
have *10*: *high ?y* (*deg div 2*) = *?h* ∧ *low ?y* (*deg div 2*) = *predy*
by (*simp add*: *07 high-inv low-inv mult.commute*)
hence *14*:*both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *07 09 5.hyps*(*4*) ‹*deg div 2* = *n*› ‹*high x n* < *2* ^ *m* ∧ *low x n* < *2* ^ *n*› *both-member-options-from-chilf-to-complete-tree* **by** *auto*
have *15*: *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?y*
using *14 thisvalid valid-member-both-member-options* **by** *blast*
have *16*: *Some ?y* = *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x*
by (*simp add*: ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some* (*2* ^ (*deg div 2*) ∗ *high x* (*deg div 2*) + *predy*)›)
have *17*: *x* = *?h* ∗ *2*^(*deg div 2*) + *?l*
using *bit-concat-def bit-split-inv* **by** *auto*
have *18*: *x* − *?y* = *?h* ∗ *2*^(*deg div 2*) + *?l* − *?h* ∗ *2*^(*deg div 2*) − *predy*
by (*metis 17 diff-diff-add mult.commute*)
hence *19*: *?y* < *x*
using *04 17 mult.commute nat-add-left-cancel-less pred-member* **by** *fastforce*
have *20*: *z* < *x* ⟹ *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ⟹ *z*≤ *?y* **for** *z*
proof−
assume *z* < *x* **and** *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z*
hence *high z* (*deg div 2*) ≤ *high x* (*deg div 2*)
by (*simp add*: *div-le-mono high-def*)
then show *?thesis*
proof(*cases high z* (*deg div 2*) = *high x* (*deg div 2*))
case *True*
hence *0000*: *high z* (*deg div 2*) = *high x* (*deg div 2*) **by** *simp*
then show *?thesis*
proof(*cases z* = *mi*)
case *True*
then show *?thesis*

```
            by (metis 15 5.hyps(9) add.left-neutral le-add2 less-imp-le-nat member-inv)
          next
            case False
            hence ad:vebt-member (treeList ! ?h) (low z (deg div 2))
              using vebt-member.simps(5)[of mi ma deg−2 treeList summary z]
                by (metis True ‹vebt-member (Node (Some (mi, ma)) deg treeList summary) z› ‹x ≤
ma› ‹z < x› leD member-inv)
            have is-pred-in-set (set-vebt' (treeList ! ?h)) ?l predy
              using 04 by blast
            have low z (deg div 2) < ?l
              by (metis (full-types) True ‹z < x› bit-concat-def bit-split-inv nat-add-left-cancel-less)
            hence predy ≥ low z (deg div 2) using 04 ad unfolding is-pred-in-set-def
              by (simp add: set-vebt'-def)
            hence ?y ≥ z
              by (metis True bit-concat-def bit-split-inv mult.commute nat-add-left-cancel-le)
            then show ?thesis by blast
          qed
        next
          case False
          hence high z (deg div 2) < high ?y (deg div 2)
            using 10 ‹high z (deg div 2) ≤ high x (deg div 2)› by linarith
          then show ?thesis
            by (metis div-le-mono high-def nat-le-linear not-le)
        qed
      qed
      hence is-pred-in-set (set-vebt'(Node (Some (mi, ma)) deg treeList summary)) x ?y
        by (simp add: 15 19 pred-member)
      then show ?thesis using 16
        by (metis eq-iff option.inject pred-member)
    next
      case False
      hence i1:?minlow =  None ∨ ¬ (Some ?l >o  ?minlow) by simp
      hence 2: vebt-pred (Node (Some (mi, ma)) deg treeList summary) x =  (
                        if ?pr = None then (if x > mi
                                            then Some mi
                                            else None)
                         else Some (2^(deg div 2)) *o ?pr +o vebt-maxt (treeList ! the ?pr))
        using 1 by auto
      have  invar-vebt (treeList ! ?h) n
        using 5(1) True by simp
      hence 33:♯ u. vebt-member (treeList ! ?h) u ∧ u < ?l
      proof(cases ?minlow = None)
        case True
        then show ?thesis using mint-corr-help-empty[of treeList ! ?h n]
          by (simp add: ‹invar-vebt (treeList ! high x (deg div 2)) n› set-vebt'-def)
      next
        case False
        obtain minilow where ?minlow =Some minilow
          using False by blast
```

hence *minilow* $\geq$ *?l*
using *i1* **by** *auto*
then show *?thesis*
by (*meson* ‹*vebt-mint* (*treeList* ! *high x* (*deg div 2*)) = *Some minilow*› ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› *leD less-le-trans mint-corr-help*)
qed
then show *?thesis*
proof(*cases ?pr*= *None*)
case *True*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if x* > *mi then Some mi else None*)
by (*simp add*: *2*)
hence ∄ *i*. *is-pred-in-set* (*set-vebt′ summary*) *?h i*
using *5.hyps*(*3*) *True* **by** *force*
hence ∄ *i*. *i* < *?h* ∧ *vebt-member summary i* **using** *pred-none-empty*[*of set-vebt′ summary ?h*]
proof −
{ fix *nn* :: *nat*
have ∀ *n*. ((*is-pred-in-set* (*Collect* (*vebt-member summary*)) (*high x* (*deg div 2*)) *esk1-0* ∨ *infinite* (*Collect* (*vebt-member summary*))) ∨ *n* ∉ *Collect* (*vebt-member summary*)) ∨ ¬ *n* < *high x* (*deg div 2*)
using ‹∄ *i*. *is-pred-in-set* (*set-vebt′ summary*) (*high x* (*deg div 2*)) *i*› *pred-none-empty set-vebt′-def* **by** *auto*
then have ¬ *nn* < *high x* (*deg div 2*) ∨ ¬ *vebt-member summary nn*
by (*metis* (*no-types*) *5.hyps*(*2*) ‹∄ *i*. *is-pred-in-set* (*set-vebt′ summary*) (*high x* (*deg div 2*)) *i*› *mem-Collect-eq set-vebt′-def set-vebt-finite*) **}**
then show *?thesis*
by *blast*
qed
then show *?thesis*
proof(*cases x* > *mi*)
case *True*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*
by (*simp add*: ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*if mi* < *x then Some mi else None*)›)
have (*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *z* > *mi*) ⟹ *False* **for** *z*
proof−
assume *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *z* > *mi*
hence *vebt-member* (*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))
using ‹*x* ≤ *ma*› *member-inv not-le* **by** *blast*
moreover hence *high z* (*deg div 2*) < *2*^*m*
using *5.hyps*(*4*) ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› ‹*x* ≤ *ma*› *member-inv* **by** *fastforce*
moreover hence *invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n* **using** *5*(*1*)
by (*simp add*: *5.hyps*(*4*))
ultimately have *vebt-member summary* (*high z* (*deg div 2*)) **using** *5*(*7*)
using *5.hyps*(*2*) *both-member-options-equiv-member* **by** *blast*
have (*high z* (*deg div 2*)) ≤ *?h*

by (*simp add*: ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› *div-le-mono high-def less-or-eq-imp-le*)
then show *False*
by (*metis 33* ‹¬ (∃ *i*<*high x* (*deg div 2*). *vebt-member summary i*)› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x* ∧ *mi* < *z*› ‹*vebt-member* (*treeList* ! *high z* (*deg div 2*)) (*low z* (*deg div 2*))› ‹*vebt-member summary* (*high z* (*deg div 2*))› *bit-concat-def bit-split-inv le-neq-implies-less nat-add-left-cancel-less*)
qed
hence *is-pred-in-set* (*set-vebt'* ((*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*))) *x mi*
by (*metis VEBT-Member.vebt-member.simps*(*5*) *True* ‹*2* ≤ *deg*› *add-2-eq-Suc le-add-diff-inverse le-less-linear pred-member*)
then show *?thesis*
by (*metis* ‹*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *Some mi*› ‹*x* ≤ *ma*› *option.sel leD member-inv pred-member*)
next
case *False*
hence *vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *None*
by (*simp add*: *2 True*)
then show *?thesis*
by (*metis* (*full-types*) *False less-trans member-inv option.distinct*(*1*) *pred-max pred-member*)
qed
next
case *False*
hence *fst*:*vebt-pred* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
Some (*2*^(*deg div 2*)) ∗*o* *?pr* +*o* *vebt-maxt* (*treeList* ! *the ?pr*)
using *2* **by** *presburger*
obtain *pr* **where** *?pr* = *Some pr*
using *False* **by** *blast*
hence *is-pred-in-set* (*set-vebt' summary*) *?h pr*
using *5.hyps*(*3*) **by** *blast*
hence *vebt-member summary pr*
using *pred-member* **by** *blast*
hence *both-member-options summary pr*
using *5.hyps*(*2*) *both-member-options-equiv-member* **by** *auto*
hence *pr* < *2*^*m*
using *5.hyps*(*2*) ‹*vebt-member summary pr*› *member-bound* **by** *blast*
hence ∃ *maxy*. *both-member-options* (*treeList* ! *pr*) *maxy*
using *5.hyps*(*7*) ‹*both-member-options summary pr*› **by** *blast*
hence *fgh*:*set-vebt'* (*treeList* ! *pr*) ≠ {}
by (*metis 5.hyps*(*1*) *5.hyps*(*4*) *Collect-empty-eq* ‹*pr* < *2* ^ *m*› *nth-mem set-vebt'-def valid-member-both-member-options*)
hence *invar-vebt* (*treeList* ! *the ?pr*) *n*
by (*simp add*: *5.hyps*(*1*) *5.hyps*(*4*) ‹*pr* < *2* ^ *m*› ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*›)
then obtain *maxy* **where** *Some maxy* = *vebt-maxt* (*treeList* ! *pr*)
by (*metis* ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*› *fgh option.sel vebt-maxt.elims maxt-corr-help-empty*)
hence *Some maxy* = *vebt-maxt* (*treeList* ! *the ?pr*)
by (*simp add*: ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*›)

hence *max-in-set* (*set-vebt'* (*treeList* ! *the ?pr*)) *maxy*
using ‹*invar-vebt* (*treeList* ! *the* (*vebt-pred summary* (*high x* (*deg div 2*)))) *n*› *maxt-corr* **by** *auto*
hence *scmem*:*vebt-member* (*treeList* ! *the ?pr*) *maxy*
using ‹*Some maxy* = *vebt-maxt* (*treeList* ! *the* (*vebt-pred summary* (*high x* (*deg div 2*))))› ‹*invar-vebt* (*treeList* ! *the* (*vebt-pred summary* (*high x* (*deg div 2*)))) *n*› *maxt-member* **by** *force*
let *?res* = *Some* (*2*^(*deg div 2*)) $*_o$ *?pr* $+_o$ *vebt-maxt* (*treeList* ! *the ?pr*)
obtain *res* **where** *snd*: *res* = *the ?res* **by** *blast*
hence *res* = *2*^(*deg div 2*) * *pr* + *maxy*
by (*metis* ‹*Some maxy* = *vebt-maxt* (*treeList* ! *pr*)› ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*› *add-def option.sel mul-def option-shift.simps*(*3*))
have *high res* (*deg div 2*) = *pr*
by (*metis* ‹*deg div 2* = *n*› ‹*res* = *2* ^ (*deg div 2*) * *pr* + *maxy*› ‹*invar-vebt* (*treeList* ! *the ?pr*) *n*› *high-inv member-bound mult.commute scmem*)
hence *res* < *x*
by (*metis* ‹*is-pred-in-set* (*set-vebt' summary*) (*high x* (*deg div 2*)) *pr*› *div-le-mono high-def pred-member alethe-comp-simplify1*(*3*))
have *both-member-options* (*treeList* ! (*high res* (*deg div 2*))) (*low res* (*deg div 2*))
by (*metis* ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *pr*› ‹*vebt-pred summary* (*high x* (*deg div 2*)) = *Some pr*› ‹*res* = *2* ^ (*deg div 2*) * *pr* + *maxy*› ‹*invar-vebt* (*treeList* ! *the* (*vebt-pred summary* (*high x* (*deg div 2*)))) *n*› *both-member-options-equiv-member option.sel low-inv member-bound mult.commute scmem*)
have *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *res*
by (*metis 5.hyps*(*2*) *5.hyps*(*4*) *5.hyps*(*6*) ‹*1* ≤ *n*› ‹*both-member-options* (*treeList* ! *high res* (*deg div 2*)) (*low res* (*deg div 2*))› ‹*high res* (*deg div 2*) = *pr*› ‹*vebt-member summary pr*› *both-member-options-from-chilf-to-complete-tree member-bound trans-le-add1*)
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *res*
using *thisvalid valid-member-both-member-options* **by** *auto*
hence *res* > *mi*
by (*metis 5.hyps*(*11*) ‹*both-member-options* (*treeList* ! *high res* (*deg div 2*)) (*low res* (*deg div 2*))› ‹*deg div 2* = *n*› ‹*high res* (*deg div 2*) = *pr*› ‹*pr* < *2* ^ *m*› ‹*res* < *x*› ‹*x* ≤ *ma*› *less-le-trans member-inv*)
hence *res* < *ma*
using ‹*res* < *x*› ‹*x* ≤ *ma*› *less-le-trans* **by** *blast*
have (*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*) ⟹ *z* ≤ *res* **for** *z*
proof−
fix *z*
assume *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*
hence *20*: *z* = *mi* ∨ *z* = *ma* ∨ (*high z* (*deg div 2*) < *length treeList* ∧ *vebt-member* (*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))) **using** *vebt-member.simps*(*5*)[*of mi ma deg*−*2 treeList summary z*]
using *member-inv* **by** *blast*
have *z* ≠ *ma*
using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› ‹*x* ≤ *ma*› *leD* **by** *blast*
hence *mi* ≠ *ma*
by (*metis* ‹*mi* < *res*› ‹*res* < *x*› ‹*x* ≤ *ma*› *leD less-trans*)
hence *z* < *2*^*deg*

using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *member-bound thisvalid* **by** *blast*
hence (*high z* (*deg div 2*)) <2^*m*
by (*metis 5.hyps*(*5*) *5.hyps*(*6*) ‹*1* ≤ *n*› ‹*deg div 2* = *n*› *exp-split-high-low*(*1*) *less-le-trans numeral-One zero-less-Suc zero-less-numeral*)
hence *abc*:*invar-vebt* (*treeList* ! (*high z* (*deg div 2*))) *n*
by (*simp add*: *5.hyps*(*1*) *5.hyps*(*4*))
then show *z* ≤ *res*
proof(*cases z* = *mi*)
case *True*
then show *?thesis*
using ‹*mi* < *res*› **by** *auto*
next
case *False*
hence *abe*:*vebt-member*(*treeList* ! (*high z* (*deg div 2*))) (*low z* (*deg div 2*))
using *20* ‹*z* ≠ *ma*› **by** *blast*
hence *abh*:*vebt-member summary* (*high z* (*deg div 2*))
using *5.hyps*(*7*) ‹*high z* (*deg div 2*) < *2* ^ *m*› ‹*length treeList* = *2* ^ *m* ∧ *invar-vebt summary m*› *abc both-member-options-equiv-member* **by** *blast*
have *aaa*:(*high z* (*deg div 2*)) = (*high x* (*deg div 2*)) ⟹ *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *abe* **by** *auto*
have *high z*(*deg div 2*) > *pr* ⟹ *False*
proof−
assume *high z*(*deg div 2*) > *pr*
hence *vebt-member summary* (*high z*(*deg div 2*))
using *abh* **by** *blast*
have *aaaa*:*?h* ≤ *high z*(*deg div 2*)
by (*meson* ‹*is-pred-in-set* (*set-vebt*′ *summary*) (*high x* (*deg div 2*)) *pr*› ‹*pr* < *high z* (*deg div 2*)› *abh leD not-le-imp-less pred-member*)
have *bbbb*:*?h* ≥ *high z*(*deg div 2*)
by (*simp add*: ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *div-le-mono dual-order.strict-implies-order high-def*)
hence *?h* = *high z* (*deg div 2*)
using *aaaa eq-iff* **by** *blast*
hence *vebt-member* (*treeList* ! *?h*) (*low z* (*deg div 2*))
using *aaa* **by** *linarith*
hence (*low z* (*deg div 2*)) < *?l*
by (*metis* ‹*high x* (*deg div 2*) = *high z* (*deg div 2*)› ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *z* ∧ *z* < *x*› *add-le-cancel-left div-mult-mod-eq high-def less-le low-def*)
then show *False*
using *33* ‹*vebt-member* (*treeList* ! *high x* (*deg div 2*)) (*low z* (*deg div 2*))› **by** *blast*
qed
hence *high z*(*deg div 2*) ≤ *pr*
using *not-less* **by** *blast*
then show *z* ≤ *res*
proof(*cases high z*(*deg div 2*) = *pr*)
case *True*
hence *vebt-member* (*treeList* ! (*high z*(*deg div 2*))) (*low z* (*deg div 2*))

using *abe* **by** *blast*
have *low z (deg div 2) ≤ maxy*
using *True ‹Some maxy = vebt-maxt (treeList ! pr)› abc abe maxt-corr-help* **by** *auto*
hence *z ≤ res*
by (*metis True ‹res = 2 ^ (deg div 2) * pr + maxy› add-le-cancel-left div-mult-mod-eq high-def low-def mult.commute*)
then show *?thesis* **by** *simp*
next
case *False*
hence *high z(deg div 2) < pr*
by (*simp add*: *‹high z (deg div 2) ≤ pr› less-le*)
then show *?thesis*
by (*metis ‹high res (deg div 2) = pr› div-le-mono high-def leD linear*)
qed
qed
qed
hence *is-pred-in-set (set-vebt' (Node (Some (mi, ma)) deg treeList summary)) x res*
using *‹vebt-member (Node (Some (mi, ma)) deg treeList summary) res› ‹res < x› pred-member* **by** *presburger*
then show *?thesis* **using** *fst snd*
by (*metis ‹Some maxy = vebt-maxt (treeList ! the (vebt-pred summary (high x (deg div 2))))› ‹vebt-pred summary (high x (deg div 2)) = Some pr› ‹res = 2 ^ (deg div 2) * pr + maxy› add-shift dual-order.eq-iff mul-shift pred-member*)
qed
qed
next
case *False*
then show *?thesis*
by (*metis 5.hyps(10) 5.hyps(4) 5.hyps(5) 5.hyps(6) ‹1 ≤ n› ‹deg div 2 = n› ‹x ≤ ma› exp-split-high-low(1) le-0-eq le-less-trans alethe-comp-simplify1(3) zero-less-Suc zero-neq-one*)
qed
qed
qed

corollary *pred-empty*: **assumes** *invar-vebt t n*
shows *(vebt-pred t x = None) = ({y. vebt-member t y ∧ y < x} = {})*
proof
show *vebt-pred t x = None ⟹ {y. vebt-member t y ∧ x > y} = {}*
proof
show *vebt-pred t x = None ⟹ {y. vebt-member t y ∧ x > y} ⊆ {}*
proof−
assume *vebt-pred t x = None*
hence *∄ y. is-pred-in-set (set-vebt' t) x y*
using *assms pred-corr* **by** *force*
moreover hence *is-pred-in-set (set-vebt' t) x y ⟹ vebt-member t y ∧ x < y* **for** *y* **by** *auto*
ultimately show *{y. vebt-member t y ∧ x > y} ⊆ {}*
using *assms pred-none-empty set-vebt'-def set-vebt-finite* **by** *auto*
qed
show *vebt-pred t x = None ⟹ {} ⊆ {y. vebt-member t y ∧ x > y}* **by** *simp*

qed
show $\{y.\ vebt\text{-}member\ t\ y \wedge x > y\} = \{\} \Longrightarrow vebt\text{-}pred\ t\ x = None$
proof−
assume $\{y.\ vebt\text{-}member\ t\ y \wedge x > y\} = \{\}$
hence $is\text{-}pred\text{-}in\text{-}set\ (set\text{-}vebt'\ t)\ x\ y \Longrightarrow False$ **for** y
using *pred-member* **by** *auto*
thus $vebt\text{-}pred\ t\ x\ = None$
by (*meson assms option-shift.elims pred-corr*)
qed
qed

theorem *pred-correct*: $invar\text{-}vebt\ t\ n \Longrightarrow vebt\text{-}pred\ t\ x = Some\ sx \longleftrightarrow is\text{-}pred\text{-}in\text{-}set\ (set\text{-}vebt\ t)\ x\ sx$
by (*simp add*: *pred-corr set-vebt-set-vebt'-valid*)

lemma *helpypredd*:$invar\text{-}vebt\ t\ n \Longrightarrow vebt\text{-}pred\ t\ x = Some\ y \Longrightarrow y < 2\hat{}n$
using *member-bound pred-corr pred-member* **by** *blast*

lemma $invar\text{-}vebt\ t\ n \Longrightarrow vebt\text{-}pred\ t\ x = Some\ y \Longrightarrow y < x$
by (*simp add*: *pred-corr pred-member*)

end
end

theory *VEBT-Delete* **imports** *VEBT-Pred VEBT-Succ*
begin

8 Deletion

8.1 Function Definition

context begin
interpretation *VEBT-internal* **.**

fun $vebt\text{-}delete :: VEBT \Rightarrow nat \Rightarrow VEBT$ **where**
$vebt\text{-}delete\ (Leaf\ a\ b)\ 0 = Leaf\ False\ b|$
$vebt\text{-}delete\ (Leaf\ a\ b)\ (Suc\ 0) = Leaf\ a\ False|$
$vebt\text{-}delete\ (Leaf\ a\ b)\ (Suc\ (Suc\ n)) = Leaf\ a\ b|$
$vebt\text{-}delete\ (Node\ None\ deg\ treeList\ summary)\ \text{-} = (Node\ None\ deg\ treeList\ summary)|$
$vebt\text{-}delete\ (Node\ (Some\ (mi,\ ma))\ 0\ trLst\ smry)\ x = (Node\ (Some\ (mi,\ ma))\ 0\ trLst\ smry)\ |$
$vebt\text{-}delete\ (Node\ (Some\ (mi,\ ma))\ (Suc\ 0)\ tr\ sm)\ x = (Node\ (Some\ (mi,\ ma))\ (Suc\ 0)\ tr\ sm)\ |$
$vebt\text{-}delete\ (Node\ (Some\ (mi,\ ma))\ deg\ treeList\ summary)\ x =($
$if\ (x < mi \vee x > ma)\ then\ (Node\ (Some\ (mi,\ ma))\ deg\ treeList\ summary)$
$else\ if\ (x = mi \wedge x = ma)\ then\ (Node\ None\ deg\ treeList\ summary)$
$else\ let\ xn = (if\ x = mi$
$then\ the\ (vebt\text{-}mint\ summary) * 2\hat{}(deg\ div\ 2)$
$+\ the\ (vebt\text{-}mint\ (treeList\ !\ the\ (vebt\text{-}mint\ summary)))$
$else\ x);$
$minn = (if\ x = mi\ then\ xn\ else\ mi);$
$l = low\ xn\ (deg\ div\ 2);$

```
    h = high xn (deg div 2) in
    if h < length treeList
    then(
        let newnode = vebt-delete (treeList ! h) l;
            newlist = treeList[h:= newnode]in
            if minNull newnode
            then( let sn = vebt-delete summary h in(
                Node (Some (minn, if xn  = ma then
                                    (let maxs = vebt-maxt sn in (
                                     if maxs = None
                                     then minn
                                     else 2^(deg div 2) * the maxs
                                       + the (vebt-maxt (newlist ! the maxs))))
                                  else ma)) deg newlist sn))
            else (Node (Some (minn, (if xn = ma
                                then h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                else ma))) deg newlist summary ))
    else (Node (Some (mi, ma)) deg treeList summary))
```

end

8.2 Auxiliary Lemmas

context *VEBT-internal* **begin**

context begin

lemma *delt-out-of-range*:
assumes $x < mi \vee x > ma$ **and** $deg \geq 2$
shows
vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =(Node (Some (mi, ma)) deg treeList summary)
using *vebt-delete.simps(7)[of mi ma deg−2 treeList summary x]*
by *(metis add-2-eq-Suc assms(1) assms(2) le-add-diff-inverse)*

lemma *del-single-cont*:
assumes $x = mi \wedge x = ma$ **and** $deg \geq 2$
shows *vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (Node None deg treeList summary)*
using *vebt-delete.simps(7)[of mi ma deg−2 treeList summary x]*
by *(metis add-2-eq-Suc assms(1) assms(2) le-add-diff-inverse nat-less-le)*

lemma *del-in-range*:
assumes $x \geq mi \wedge x \leq ma$ **and** $mi \neq ma$ **and** $deg \geq 2$
shows
vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (let xn = (if x = mi
*then the (vebt-mint summary) * 2^(deg div 2)*
+ the (vebt-mint (treeList ! the (vebt-mint summary)))
else x);

```
                minn = (if x = mi then xn else mi);
                l = low xn (deg div 2);
                h = high xn (deg div 2) in
                 if h < length treeList
                    then(
                      let newnode = vebt-delete (treeList ! h) l;
                          newlist = treeList[h:= newnode] in
                        if minNull newnode
                        then(
                          let sn = vebt-delete summary h in
                          (Node (Some (minn, if xn  = ma then (let maxs = vebt-maxt sn in
                                                              (if maxs = None
                                                                 then minn
                                                                 else 2^(deg div 2) * the maxs
                                                                     + the (vebt-maxt (newlist ! the maxs))
                                                              )
                                                          )
                                                      else ma))
                                deg newlist sn)
                        )else
                          (Node (Some (minn, (if xn = ma then
                                              h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                                 else ma)))
                            deg newlist summary )
                    )else
                     (Node (Some (mi, ma)) deg treeList summary))
proof −
  obtain deg′ :: nat where deg = Suc (Suc deg′)
    using ‹2 ≤ deg› add-2-eq-Suc le-Suc-ex by blast
  then show ?thesis
    using assms(1,2) by force
qed

lemma del-x-not-mia:
  assumes x > mi ∧ x ≤ ma and mi ≠ ma  and deg ≥ 2  and high x (deg div 2) = h and
  low x (deg div 2) = land  high x (deg div 2) < length treeList
  shows
   vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (
                let  newnode = vebt-delete (treeList ! h) l;
                         newlist = treeList[h:= newnode] in
                        if minNull newnode
                        then(
                          let sn = vebt-delete summary h in
                          (Node (Some (mi, if x  = ma then (let maxs = vebt-maxt sn in
                                                              (if maxs = None
                                                                 then mi
                                                                 else 2^(deg div 2) * the maxs
                                                                     + the (vebt-maxt (newlist ! the maxs))
                                                              )
```

```
                                                    )
                                                 else ma))
                                    deg newlist sn)
                              )else
                                (Node (Some (mi, (if x = ma then
                                                    h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                                  else ma)))
                                  deg newlist summary )
)
  using del-in-range[of mi x ma deg treeList summary, OF - assms(2,3)]
  unfolding Let-def
  using assms(1,4-) by fastforce

lemma del-x-not-mi:
  assumes x > mi ∧ x ≤ ma and mi ≠ ma  and deg ≥ 2  and high x (deg div 2) = h and
  low x (deg div 2) = land  newnode = vebt-delete (treeList ! h) l
  and newlist = treeList[h:= newnode] and high x (deg div 2) < length treeList
  shows
   vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (
               if minNull newnode
                        then(
                          let sn = vebt-delete summary h in
                          (Node (Some (mi, if x  = ma then (let maxs = vebt-maxt sn in
                                                              (if maxs = None
                                                                 then mi
                                                                 else 2^(deg div 2) * the maxs
                                                                     + the (vebt-maxt (newlist ! the maxs))
                                                               )
                                                            )
                                                         else ma))
                                    deg newlist sn)
                              )else
                                (Node (Some (mi, (if x = ma then
                                                    h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                                  else ma)))
                                  deg newlist summary )
) using del-x-not-mia[of mi x ma deg h l treeList summary]
  by (smt (verit) assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) assms(8))

lemma del-x-not-mi-new-node-nil:
  assumes x > mi ∧ x ≤ ma and mi ≠ ma  and deg ≥ 2  and high x (deg div 2) = h and
    low x (deg div 2) = land  newnode = vebt-delete (treeList ! h) l and minNull newnode  and
    sn = vebt-delete summary h  and newlist =treeList[h:= newnode] and  high x (deg div 2) < length
treeList
  shows
   vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =  (Node (Some (mi, if x  = ma then
(let maxs = vebt-maxt sn in
                                                              (if maxs = None
                                                                 then mi
```

```
                                                    else 2^(deg div 2) * the maxs
                                                        + the (vebt-maxt (newlist ! the maxs))
                                                )
                                            )
                                        else ma))  deg newlist sn)
  using del-x-not-mi[of mi x ma deg h l newnode treeList]
  by (metis assms(1) assms(10) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) assms(8)
assms(9))

lemma del-x-not-mi-newnode-not-nil:
  assumes x > mi ∧ x ≤ ma and mi ≠ ma  and deg ≥ 2  and high x (deg div 2) = h and
    low x (deg div 2) = land  newnode = vebt-delete (treeList ! h) l and ¬  minNull newnode  and
     newlist = treeList[h:= newnode] andhigh x (deg div 2) < length treeList
  shows
   vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
                        (Node (Some (mi, (if x = ma then
                                        h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                            else ma)))
                          deg newlist summary )
  using del-x-not-mi[of mi x ma deg h l newnode treeList newlist summary]
  using assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) assms(8) assms(9) by
auto

lemma del-x-mia: assumes x = mi ∧ x < ma and mi ≠ ma  and deg ≥ 2
  shows vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =(
            let xn = the (vebt-mint summary) * 2^(deg div 2)
                                + the (vebt-mint (treeList ! the (vebt-mint summary)));
                                    minn = xn;
                l = low xn (deg div 2);
                h = high xn (deg div 2) in
                 if h < length treeList
                    then(
                       let newnode = vebt-delete (treeList ! h) l;
                           newlist = treeList[h:= newnode]in
                          if minNull newnode
                          then(
                             let sn = vebt-delete summary h in
                            (Node (Some (minn, if xn  = ma then (let maxs = vebt-maxt sn in
                                                    (if maxs = None
                                                      then minn
                                                      else 2^(deg div 2) * the maxs
                                                          + the (vebt-maxt (newlist ! the maxs))
                                                    )
                                                )
                                            else ma))
                                    deg newlist sn)
                          )else
                            (Node (Some (minn, (if xn = ma then
                                        h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
```

```
                                                    else ma)))
                                deg newlist summary )
                        )else
                         (Node (Some (mi, ma)) deg treeList summary)
          )
  using del-in-range[of mi x ma deg treeList summary]
  using assms(1) assms(3) nat-less-le order-refl by fastforce

lemma del-x-mi:
  assumes x = mi ∧ x < ma and mi ≠ ma  and deg ≥ 2  and high xn (deg div 2) = h and
    xn =  the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary)))
    low xn (deg div 2) = land  high xn (deg div 2) < length treeList
  shows
  vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =(
                        let newnode = vebt-delete (treeList ! h) l;
                            newlist = treeList[h:= newnode]in
                           if minNull newnode
                           then(
                              let sn = vebt-delete summary h in
                              (Node (Some (xn, if xn  = ma then (let maxs = vebt-maxt sn in
                                                        (if maxs = None
                                                           then xn
                                                           else 2^(deg div 2) * the maxs
                                                               + the (vebt-maxt (newlist ! the maxs))
                                                        )
                                                    )
                                                 else ma))
                                    deg newlist sn)
                           )else
                             (Node (Some (xn, (if xn = ma then
                                                h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                                else ma)))
                              deg newlist summary ))

  using del-x-mia[of x mi ma deg treeList summary, OF assms(1,2,3)]
  by (smt (verit) assms(4-))

lemma del-x-mi-lets-in:
  assumes x = mi ∧ x < ma and mi ≠ ma  and deg ≥ 2  and high xn (deg div 2) = h and
    xn =  the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary)))
    low xn (deg div 2) = land  high xn (deg div 2) < length treeList and
     newnode = vebt-delete (treeList ! h) l and  newlist = treeList[h:= newnode]
  shows  vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =(       if minNull newnode
                           then(
                              let sn = vebt-delete summary h in
                              (Node (Some (xn, if xn  = ma then (let maxs = vebt-maxt sn in
                                                        (if maxs = None
```

```
                                                    then xn
                                                    else 2^(deg div 2) * the maxs
                                                        + the (vebt-maxt (newlist ! the maxs))
                                                )
                                            )
                                        else ma))
                        deg newlist sn)
                    )else
                      (Node (Some (xn, (if xn = ma then
                                        h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                            else ma)))
                        deg newlist summary ))
  using del-x-mi[of x mi ma deg xn h summary treeList l, OF assms(1,2,3)]
  by (smt (verit) assms(4-))

lemma del-x-mi-lets-in-minNull:
  assumes x = mi ∧ x < ma and mi ≠ ma  and deg ≥ 2  and high xn (deg div 2) = h and
    xn =  the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary)))
    low xn (deg div 2) = land  high xn (deg div 2) < length treeList and
    newnode = vebt-delete (treeList ! h) l and  newlist =treeList[h:= newnode] and
    minNull newnode  and  sn = vebt-delete summary h
  shows
  vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
                      (Node (Some (xn, if xn  = ma then (let maxs = vebt-maxt sn in
                                                    (if maxs = None
                                                      then xn
                                                      else 2^(deg div 2) * the maxs
                                                          + the (vebt-maxt (newlist ! the maxs))
                                                    )
                                                )
                                            else ma))   deg newlist sn)
  using del-x-mi-lets-in[of x mi ma deg xn h summary treeList l newnode newlist]
  by (metis assms(1) assms(10) assms(11) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7)
assms(8) assms(9))

lemma del-x-mi-lets-in-not-minNull:
  assumes x = mi ∧ x < ma and mi ≠ ma  and deg ≥ 2  and high xn (deg div 2) = h and
    xn =  the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary)))
    low xn (deg div 2) = land  high xn (deg div 2) < length treeList and
     newnode = vebt-delete (treeList ! h) l and  newlist = treeList[h:= newnode] and
    ¬minNull newnode
  shows
  vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
                      (Node (Some (xn, (if xn = ma then
                                        h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                            else ma)))
                        deg newlist summary )
```

using *del-x-mi-lets-in*[*of x mi ma deg xn h summary treeList l newnode newlist*]
by (*meson assms*(*1*) *assms*(*10*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*6*) *assms*(*7*) *assms*(*8*) *assms*(*9*))

theorem *dele-bmo-cont-corr*:*invar-vebt t n* ⟹ (*both-member-options* (*vebt-delete t x*) *y* ⟷ *x* ≠ *y* ∧ *both-member-options t y*)
proof(*induction t n arbitrary*: *x y rule*: *invar-vebt.induct*)
case (*1 a b*)
have (*both-member-options* (*vebt-delete* (*Leaf a b*) *x*) *y*) ⟹ (*x* ≠ *y* ∧ *both-member-options* (*Leaf a b*) *y*)
by (*metis One-nat-def both-member-options-def vebt-buildup.cases vebt-delete.simps*(*1*) *vebt-delete.simps*(*2*) *vebt-delete.simps*(*3*) *membermima.simps*(*1*) *naive-member.simps*(*1*))
moreover have (*x* ≠ *y* ∧ *both-member-options* (*Leaf a b*) *y*) ⟹(*both-member-options* (*vebt-delete* (*Leaf a b*) *x*) *y*)
by (*metis One-nat-def both-member-options-def vebt-buildup.cases vebt-delete.simps*(*1*) *vebt-delete.simps*(*2*) *vebt-delete.simps*(*3*) *membermima.simps*(*1*) *naive-member.simps*(*1*))
ultimately show *?case* **by** *blast*
next
case (*2 treeList n summary m deg*)
hence *deg* ≥ *2*
by (*metis Suc-leI deg-not-0 dual-order.strict-trans2 less-add-same-cancel1 numerals*(*2*))
hence (*vebt-delete* (*Node None deg treeList summary*) *x*) = (*Node None deg treeList summary*) **by** *simp*
moreover have ¬*vebt-member* (*Node None deg treeList summary*) *y* **by** *simp*
moreover hence ¬*both-member-options* (*Node None deg treeList summary*) *y*
using *invar-vebt.intros*(*2*)[*of treeList n summary m deg*] *2*
by (*metis valid-member-both-member-options*)
moreover hence ¬*both-member-options* (*vebt-delete* (*Node None deg treeList summary*) *x*) *y* **by** *simp*
ultimately show *?case*
by *force*
next
case (*3 treeList n summary m deg*)
hence *deg* ≥ *2*
by (*metis One-nat-def add-mono le-add1 numeral-2-eq-2 plus-1-eq-Suc set-n-deg-not-0*)
hence (*vebt-delete* (*Node None deg treeList summary*) *x*) = (*Node None deg treeList summary*) **by** *simp*
moreover have ¬*vebt-member* (*Node None deg treeList summary*) *y* **by** *simp*
moreover hence ¬*both-member-options* (*Node None deg treeList summary*) *y*
using *invar-vebt.intros*(*3*)[*of treeList n summary m deg*] *3*
by (*metis valid-member-both-member-options*)
moreover hence ¬*both-member-options* (*vebt-delete* (*Node None deg treeList summary*) *x*) *y* **by** *simp*
ultimately show *?case*
by *force*
next
case (*4 treeList n summary m deg mi ma*)
then obtain *n*′ **where** *n* = *Suc n*′
by (*metis deg-not-0 gr0-implies-Suc*)

then have *deg* = *Suc* (*Suc* (*n'* + *n'*))
unfolding ‹*deg* = *n* + *m*› ‹*m* = *n*› **by** *presburger*
hence *dp*: *deg* ≥ *2*
by *linarith*
have *tvalid*: *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*
using *4 invar-vebt.intros*(*4*)[*of treeList n summary m deg mi ma*] **by** *simp*
hence *mi* ≤ *ma* **and** *deg div 2* = *n* **and** *ma* ≤ *2^deg* **using** *4*
by (*auto simp add*: *4.hyps*(*3*) *4.hyps*(*4*))
show *?case* **proof**(*cases x <mi* ∨ *x* > *ma*)
case *True*
hence *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)›)
then show *?thesis*
by (*metis 4.hyps*(*7*) *True tvalid leD member-inv not-less-iff-gr-or-eq valid-member-both-member-options*)
next
case *False*
hence *mi* ≤ *x* ∧ *x* ≤ *ma* **by** *simp*
hence *x* < *2^deg*
using *4.hyps*(*8*) *order.strict-trans1* **by** *blast*
then show *?thesis*
proof(*cases x* = *mi* ∧ *x* = *ma*)
case *True*
hence *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node None deg treeList summary*)
using *del-single-cont*[*of x mi ma deg treeList summary*] ‹*2* ≤ *deg*› **by** *blast*
moreover hence *invar-vebt* (*Node None deg treeList summary*) *deg*
using *4*(*4*) *4.IH*(*1*) *4.hyps*(*1*) *4.hyps*(*3*) *4.hyps*(*4*) *True mi-eq-ma-no-ch tvalid invar-vebt.intros*(*2*) **by** *force*
moreover hence ¬ *vebt-member* (*Node None deg treeList summary*) *y* **by** *simp*
moreover hence ¬*both-member-options* (*Node None deg treeList summary*) *y*
using *calculation*(*2*) *valid-member-both-member-options* **by** *blast*
then show *?thesis*
by (*metis True calculation*(*1*) *member-inv not-less-iff-gr-or-eq tvalid valid-member-both-member-options*)
next
case *False*
hence *mimapr*:*mi* < *ma*
by (*metis 4.hyps*(*7*) ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› *le-antisym nat-less-le*)
then show *?thesis*
proof(*cases x* ≠ *mi*)
case *True*
hence *xmi*:*x* ≠ *mi* **by** *simp*
let *?h* =*high x n*
let *?l* = *low x n*
have *?h* < *length treeList*
using *4*(*10*) *4*(*4*) *4.hyps*(*1*) *4.hyps*(*3*) *4.hyps*(*4*) ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› *deg-not-0 exp-split-high-low*(*1*) **by** *auto*
let *?newnode* = *vebt-delete* (*treeList* ! *?h*) *?l*
let *?newlist* = *treeList*[*?h*:= *?newnode*]

have *length treeList = length ?newlist* **by** *simp*
hence *hprolist*: *?newlist ! ?h = ?newnode*
by (*meson ‹high x n < length treeList› nth-list-update-eq*)
have *nothprolist*: *i ≠ ?h ∧ i < 2^m ⟹ ?newlist ! i = treeList ! i* **for** *i* **by** *auto*
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
let *?sn = vebt-delete summary ?h*
let *?newma= (if x = ma then (let maxs = vebt-maxt ?sn in (if maxs = None*
then mi
*else 2^(deg div 2) * the maxs*
+ the (vebt-maxt (?newlist ! the maxs))
)
)
else ma)
let *?delsimp = (Node (Some (mi, ?newma)) deg ?newlist ?sn)*
have *vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp*
using *del-x-not-mi-new-node-nil*[*of mi x ma deg ?h ?l ?newnode treeList ?sn summary ?newlist*]
by (*metis True ‹2 ≤ deg› ‹deg div 2 = n› ‹high x n < length treeList› ‹mi < ma› ‹mi ≤ x ∧ x ≤ ma› ‹x ≠ mi› less-not-refl3 order.not-eq-order-implies-strict*)
moreover have *both-member-options (?delsimp) y ⟹ (x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y)*
proof−
assume *assm*: *both-member-options (?delsimp) y*
then have *y = mi ∨ y = ?newma ∨*
(both-member-options (?newlist ! (high y (deg div 2))) (low y (deg div 2)) ∧ (high y (deg div 2)) < length ?newlist)
using *both-member-options-from-complete-tree-to-child*[*of deg mi ?newma ?newlist ?sn y*]
using *‹deg = Suc (Suc -)›*
by (*smt* (*verit*, *best*) *membermima.simps*(*4*) *naive-member.simps*(*3*) *both-member-options-def*)
moreover have *y = mi ⟹ ?thesis*
by (*meson ‹x ≠ mi› both-member-options-equiv-member vebt-mint.simps*(*3*) *mint-member tvalid*)
moreover have *y = ?newma ⟹ ?thesis*
proof−
assume *y = ?newma*
show *?thesis*
proof(*cases x = ma*)
case *True*
let *?maxs = vebt-maxt ?sn*
have *?newma = (if ?maxs = None then mi*
*else 2 ^ (deg div 2) * the ?maxs + the (vebt-maxt*
((treeList[(high x n):= vebt-delete (treeList ! (high x n)) (low x n)]) !
the ?maxs))) **using** *True* **by** *force*
then show *?thesis*
proof(*cases ?maxs = None*)
case *True*
then show *?thesis*

using ‹(*if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (treeList [high x n := vebt-delete (treeList ! high x n) (low x n)] ! the maxs)) else ma*) = (*if vebt-maxt (vebt-delete summary (high x n)) = None then mi else 2 ^ (deg div 2) * the (vebt-maxt (vebt-delete summary (high x n))) + the (vebt-maxt (treeList[high x n := vebt-delete (treeList ! high x n) (low x n)] ! the (vebt-maxt (vebt-delete summary (high x n)))))›› ‹*y* = (*if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (treeList [high x n := vebt-delete (treeList ! high x n) (low x n)] ! the maxs)) else ma*)› *calculation*(*2*) **by** *presburger*

next

case *False*

then obtain *maxs* **where** *Some maxs = ?maxs* **by** *force*

hence *both-member-options ?sn maxs*

by (*simp add*: *maxbmo*)

hence *both-member-options summary maxs ∧ maxs ≠ ?h*

using *4.IH*(*2*) **by** *blast*

hence *?newlist ! the ?maxs = treeList ! maxs*

by (*metis 4.hyps*(*1*) ‹*Some maxs = vebt-maxt (vebt-delete summary (high x n))*› *option.sel member-bound nothprolist valid-member-both-member-options*)

have *maxs < 2^m*

using *4.hyps*(*1*) ‹*both-member-options summary maxs ∧ maxs ≠ high x n*› *member-bound valid-member-both-member-options* **by** *blast*

hence *the (vebt-maxt (?newlist ! the ?maxs)) = the (vebt-maxt (treeList ! maxs))*

by (*simp add*: ‹*treeList*[*high x n := vebt-delete (treeList ! high x n) (low x n)*] ! *the (vebt-maxt (vebt-delete summary (high x n))) = treeList ! maxs*›)

have ∃ *z. both-member-options*(*treeList ! maxs*) *z*

by (*simp add*: *4.hyps*(*5*) ‹*both-member-options summary maxs ∧ maxs ≠ high x n*› ‹*maxs < 2 ^ m*›)

moreover have *invar-vebt* (*treeList ! maxs*) *n*

using *4* ‹*maxs < 2 ^ m*› **by** *simp*

ultimately obtain *maxi* **where** *Some maxi = (vebt-maxt (treeList ! maxs))*

by (*metis empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)

hence *maxi < 2^n*

by (*metis* ‹*invar-vebt* (*treeList ! maxs*) *n*› *maxt-member member-bound*)

hence *both-member-options* (*treeList ! maxs*) *maxi*

using ‹*Some maxi = vebt-maxt* (*treeList ! maxs*)› *maxbmo* **by** *presburger*

hence *2 ^ (deg div 2) * the ?maxs + the (vebt-maxt (?newlist ! the ?maxs)) = 2^n * maxs + maxi*

by (*metis* ‹*Some maxi = vebt-maxt* (*treeList ! maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*deg div 2 = n*› ‹*the* (*vebt-maxt* (*treeList*[*high x n := vebt-delete* (*treeList ! high x n*) (*low x n*)] ! *the* (*vebt-maxt* (*vebt-delete summary* (*high x n*))))) = *the* (*vebt-maxt* (*treeList ! maxs*))› *option.sel*)

hence *y = 2^n * maxs + maxi*

using *False True* ‹*y* = (*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) * *the maxs* + *the* (*vebt-maxt* (*treeList* [*high x n := vebt-delete* (*treeList ! high x n*) (*low x n*)] ! *the maxs*)) *else ma*)› **by** *fastforce*

hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*

by (*metis 4.hyps*(*2*) *Suc-1* ‹*both-member-options* (*treeList ! maxs*) *maxi*› ‹*deg div 2 = n*› ‹*maxi < 2 ^ n*› ‹*maxs < 2 ^ m*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp*

high-inv low-inv mult.commute plus-1-eq-Suc)
moreover hence $y \neq x$
by (*metis* ‹*both-member-options summary maxs* ∧ *maxs* ≠ *high x n*› ‹*maxi* < 2 ^ *n*› ‹*y* = 2 ^ *n* ∗ *maxs* + *maxi*› *high-inv mult.commute*)
ultimately show *?thesis* **by** *force*
qed
next
case *False*
hence *?newma* = *ma* **by** *simp*
moreover hence $y \neq x$
using *False* ‹*y* = *?newma*› **by** *presburger*
then show *?thesis*
by (*metis False* ‹*y* =*?newma*› *both-member-options-equiv-member vebt-maxt.simps*(3) *maxt-member tvalid*)
qed
qed
moreover have (*both-member-options* (*?newlist* ! (*high y* (*deg div* 2))) (*low y* (*deg div* 2)) ∧ (*high y* (*deg div* 2)) < *length ?newlist*) ⟹ *?thesis*
proof−
assume *assm*:*both-member-options* (*?newlist* ! (*high y* (*deg div* 2))) (*low y* (*deg div* 2)) ∧ (*high y* (*deg div* 2)) < *length ?newlist*
show *?thesis*
proof(*cases* (*high y* (*deg div* 2)) = *?h*)
case *True*
hence *both-member-options ?newnode* (*low y* (*deg div* 2)) **using** *hprolist* **by** (*metis assm*)
moreover hence *invar-vebt* (*treeList* ! (*high y* (*deg div* 2))) *n*
using *4.IH*(1) *True* ‹*high x n* < *length treeList*› **by** *simp*
ultimately have *both-member-options* (*treeList* ! *?h*) (*low y* (*deg div* 2)) ∧ (*low y* (*deg div* 2)) ≠ (*low x* (*deg div* 2))
using *4.IH*(1) ‹*deg div* 2 = *n*› ‹*high x n* < *length treeList*› **by** *simp*
then show *?thesis*
by (*metis Suc-1 True* ‹*high x n* < *length treeList*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp plus-1-eq-Suc*)
next
case *False*
hence $x \neq y$
using ‹*deg div* 2 = *n*› **by** *blast*
moreover hence (*?newlist* ! (*high y* (*deg div* 2))) = *treeList* ! (*high y* (*deg div* 2)) **using** *nothprolist*
using *4.hyps*(2) *False* ‹*length treeList* = *length ?newlist*› *assm* **by** *presburger*
moreover hence *both-member-options* (*treeList* ! (*high y* (*deg div* 2))) (*low y* (*deg div* 2))
using *assm* **by** *presburger*
moreover hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis One-nat-def Suc-leD* ‹*length treeList* = *length ?newlist*› *assm both-member-options-from-chilf-to-com dp numeral-2-eq-2*)
ultimately show *?thesis* **by** *blast*
qed

qed
ultimately show *?thesis* **by** *fastforce*
qed
moreover have (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*) ⟹ *both-member-options* (*?delsimp*) *y*
proof−
assume (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)
hence *aa*:*x* ≠ *y* **and** *bb*:*y* = *mi* ∨ *y* = *ma* ∨ (*both-member-options* (*treeList* ! (*high y n*)) (*low y n*) ∧ *high y n* < *length treeList*)
apply *auto*[*1*] **by** (*metis Suc-1* ‹*deg div 2* = *n*› ‹*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)
show *both-member-options* (*?delsimp*) *y*
proof−
have *y* = *mi* ⟹*both-member-options* (*?delsimp*) *y*
by (*metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps*(*4*))
moreover have *y* = *ma* ⟹ *both-member-options* (*?delsimp*) *y*
using *aa maxbmo vebt-maxt.simps*(*3*) **by** *presburger*
moreover have *both-member-options* (*treeList* ! (*high y n*)) (*low y n*) ⟹*both-member-options* (*?delsimp*) *y*
proof−
assume *assmy*: *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
then show *both-member-options* (*?delsimp*) *y*
proof(*cases high y n* = *?h*)
case *True*
moreover hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *auto*
hence *0*:*invar-vebt* (*treeList* !(*high y n*)) *n* **using** *4*
using *True* ‹*high x n* < *length treeList*› **by** *simp*
moreover have *1*:*low y n* ≠ *low x n*
by (*metis True aa bit-split-inv*)
moreover have *11*: (*treeList* !(*high y n*)) ∈ *set treeList*
using *True* ‹*high x n* < *length treeList*› **by** *simp*
ultimately have (∀ *xa*. *both-member-options ?newnode xa* =
((*low x n*) ≠ *xa* ∧ *both-member-options* (*treeList* ! *?h*) *xa*))
by (*simp add*: *4.IH*(*1*))
hence ((*low x n*) ≠ *xa* ∧ *both-member-options* (*treeList* ! *?h*) *xa*) ⟹ *both-member-options ?newnode xa* **for** *xa* **by** *blast*
moreover have ((*low x n*) ≠ (*low y n*) ∧ *both-member-options* (*treeList* ! *?h*) (*low y n*)) **using** *1*
using *True assmy* **by** *presburger*
ultimately have *both-member-options ?newnode* (*low y n*) **by** *blast*
then show *?thesis*
by (*metis One-nat-def Suc-leD True* ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *both-member-options-from-chilf-to-complete-tree dp hprolist numerals*(*2*))
next
case *False*
hence *?newlist* ! (*high y n*) = *treeList* ! (*high y n*) **by** *auto*
hence *both-member-options* (*?newlist* !(*high y n*)) (*low y n*)

```
            using assmy by presburger
          then show ?thesis
            by (smt (verit) Suc-1 Suc-le-D ‹deg div 2 = n› ‹length treeList = length ?newlist›
aa add-leD1 bb both-member-options-def both-member-options-from-chilf-to-complete-tree dp member-
mima.simps(4) plus-1-eq-Suc)
        qed
      qed
      ultimately show ?thesis using bb by fastforce
    qed
  qed
  ultimately show ?thesis by metis
next
  case False
  hence notemp:∃ z. both-member-options ?newnode z
    using not-min-Null-member by auto
  let ?newma = (if x = ma then
                                    ?h * 2^(deg div 2) + the( vebt-maxt (?newlist ! ?h))
                                  else ma)
  let ?delsimp = (Node (Some (mi, ?newma)) deg ?newlist summary)
  have vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp
  using del-x-not-mi-newnode-not-nil[of mi x ma deg ?h ?l ?newnode treeList ?newlist summary]
False xmi mimapr
      using ‹deg div 2 = n› ‹high x n < length treeList› ‹mi ≤ x ∧ x ≤ ma› dp nat-less-le
plus-1-eq-Suc by fastforce
  moreover have both-member-options ?delsimp y
          ⟹ x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y
  proof−
    assume ssms: both-member-options ?delsimp y
    hence aaaa: y = mi ∨ y = ?newma ∨ (both-member-options (?newlist ! (high y n)) (low y
n) ∧ high y n < length ?newlist)
      by (smt (verit, ccfv-SIG) membermima.simps(4) naive-member.simps(3) ‹deg = Suc (Suc
-)› ‹deg div 2 = n› both-member-options-def)
    show x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y
    proof−
      have y = mi ⟹?thesis
        by (metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps(4) xmi)
      moreover have y = ?newma ⟹ ?thesis
      proof−
        assume y = ?newma
        show ?thesis
        proof(cases x = ma)
          case True
          hence ?newma =?h * 2 ^(deg div 2) +the(vebt-maxt(?newlist ! ?h))
            by metis
          have ?newlist ! ?h = ?newnode  using hprolist by blast
          obtain maxi where maxidef:Some maxi = vebt-maxt(?newlist ! ?h)
        by (metis (full-types) False hprolist vebt-maxt.elims minNull.simps(1) minNull.simps(4))
          have aa:invar-vebt (treeList ! ?h) n
            using 4.IH(1) ‹high x n < length treeList› by simp
```

moreover hence *ab:maxi* $\neq$ *?l* $\wedge$ *both-member-options ?newnode maxi*
using *4.IH(1)* ‹*high x n* < *length treeList*› *hprolist maxidef*
by (*auto simp add*: *maxbmo*) (*metis maxbmo nth-mem*)
ultimately have *ac:maxi* $\neq$ *?l* $\wedge$ *both-member-options* (*treeList* ! *?h*) *maxi*
using *4.IH(1)* ‹*high x n* < *length treeList*› **by** *simp*
hence *ad:maxi* < *2^n*
using ‹*invar-vebt* (*treeList* ! *high x n*) *n*› *member-bound valid-member-both-member-options*
by *blast*
then show *?thesis*
by (*metis Suc-1* ‹(*if x* = *ma then high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*)) *else ma*) = *high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*))› ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› ‹*y* = (*if x* = *ma then high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*)) *else ma*)› *ac add-leD1 both-member-options-from-chilf-to-complete-tree dp option.sel high-inv low-inv maxidef plus-1-eq-Suc*)
next
case *False*
then show *?thesis*
by (*simp add*: ‹*y* = *?newma*› *maxbmo*)
qed
qed
moreover have *both-member-options* (*?newlist* ! (*high y n*)) (*low y n*) ⟹ *?thesis*
proof−
assume *assmy:both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)
then show *?thesis*
proof(*cases high y n* = *?h*)
case *True*
hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *presburger*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *4.IH(1)* ‹*high x n* < *length treeList*› **by** *simp*
hence *low y n* $\neq$ *?l* $\wedge$ *both-member-options* (*treeList* ! *?h*) (*low y n*)
using *4.IH(1) True* ‹*high x n* < *length treeList*› *assmy hprolist* **by** *simp*
then show *?thesis*
by (*metis Suc-1 True* ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› *add-leD1*
both-member-options-from-chilf-to-complete-tree dp plus-1-eq-Suc)
next
case *False*
hence *?newlist* ! (*high y n*) = *treeList* !(*high y n*) **by** *auto*
then show *?thesis*
by (*metis False Suc-1* ‹*deg div 2* = *n*› ‹*length treeList* = *length ?newlist*› *aaaa add-leD1*
both-member-options-from-chilf-to-complete-tree calculation(1) calculation(2) dp plus-1-eq-Suc)
qed
qed
ultimately show *?thesis*
using *aaaa* **by** *fastforce*
qed
qed

moreover have ($x \neq y \wedge$ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)$\Longrightarrow$

both-member-options ?delsimp y

proof−

assume *assm*: $x \neq y \wedge$ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*

hence *abcv*:$y = mi \vee y = ma \vee$ (*high y n* < *length treeList* $\wedge$ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))

by (*metis Suc-1* ‹*deg div 2* = *n*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)

thus *both-member-options ?delsimp y*

proof−

have $y = mi \Longrightarrow$ *?thesis*

by (*metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps*(*4*))

moreover have $y = ma \Longrightarrow$ *?thesis*

using *assm maxbmo vebt-maxt.simps*(*3*) **by** *presburger*

moreover have *both-member-options* (*treeList* ! (*high y n*)) (*low y n*) $\Longrightarrow$ *?thesis*

proof−

assume *myass*: *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)

thus *?thesis*

proof(*cases high y n* = *?h*)

case *True*

hence *low y n* $\neq$ *?l*

by (*metis assm bit-split-inv*)

hence *pp*:*?newlist* ! *?h* = *?newnode*

using *hprolist* **by** *blast*

hence *invar-vebt* (*treeList* ! *?h*) *n*

using *4*.*IH*(*1*) ‹*high x n* < *length treeList*› **by** *simp*

hence *both-member-options ?newnode* (*low y n*)

using *4*.*IH*(*1*) *True* ‹*high x n* < *length treeList*› ‹*low y n* $\neq$ *low x n*› *inthall myass* **by** *simp*

then show *?thesis*

by (*metis One-nat-def Suc-leD True* ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *both-member-options-from-chilf-to-complete-tree dp numerals*(*2*) *pp*)

next

case *False*

hence *pp*:*?newlist* ! (*high y n*) = *treeList* ! (*high y n*) **using** *nothprolist* **by** *auto*

then show *?thesis*

by (*metis Suc-1* ‹*deg div 2* = *n*› ‹*length treeList* = *length* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)]›› *add-leD1 assm both-member-options-from-chilf-to-complete-tree calculation*(*1*) *calculation*(*2*) *member-inv myass plus-1-eq-Suc tvalid valid-member-both-member-options*)

qed

qed

then show *?thesis*

by (*metis Suc-1 Suc-leD* ‹*deg div 2* = *n*› *assm both-member-options-from-complete-tree-to-child calculation*(*1*) *calculation*(*2*) *dp*)

qed

qed

ultimately show *?thesis* **by** *metis*
qed
next
case *False*
hence $x = mi$ **by** *simp*
have *both-member-options summary (high ma n)*
by (*metis 4(10) 4(11) 4(7) 4.hyps(4) div-eq-0-iff Suc-leI Suc-le-D div-exp-eq dual-order.irrefl high-def mimapr nat.simps(3)*)
hence *vebt-member summary (high ma n)*
using *4.hyps(1) valid-member-both-member-options* **by** *blast*
obtain *summin* **where** *Some summin = vebt-mint summary*
by (*metis 4.hyps(1) ‹vebt-member summary (high ma n)› empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def*)
hence $\exists$ *z . both-member-options (treeList ! summin) z*
by (*metis 4.hyps(1) 4.hyps(5) both-member-options-equiv-member member-bound mint-member*)
moreover have *invar-vebt (treeList ! summin) n*
by (*metis 4(4) 4.IH(1) 4.hyps(1) ‹Some summin = vebt-mint summary› member-bound mint-member nth-mem*)
ultimately obtain *lx* **where** *Some lx = vebt-mint (treeList ! summin)*
by (*metis empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
let *?xn = summin*2^n + lx*
have *xn-eq*: *?xn = (if x = mi*
*then the (vebt-mint summary) * 2^(deg div 2)*
+ the (vebt-mint (treeList ! the (vebt-mint summary)))
else x)
by (*metis False ‹Some lx = vebt-mint (treeList ! summin)› ‹Some summin = vebt-mint summary› ‹deg div 2 = n› option.sel*)
have *vebt-member (treeList ! summin) lx*
using *‹Some lx = vebt-mint (treeList ! summin)› ‹invar-vebt (treeList ! summin) n› mint-member* **by** *auto*
moreover have *summin < 2^m*
by (*metis 4.hyps(1) ‹Some summin = vebt-mint summary› member-bound mint-member*)
ultimately have *xnin*: *both-member-options (Node (Some (mi, ma)) deg treeList summary) ?xn*
by (*metis 4.hyps(2) Suc-1 ‹deg div 2 = n› ‹invar-vebt (treeList ! summin) n› add-leD1 both-member-options-equiv-member both-member-options-from-chilf-to-complete-tree dp high-inv low-inv member-bound plus-1-eq-Suc*)
let *?h =high ?xn n*
let *?l = low ?xn n*
have *xn-lt*: *?xn < 2^deg*
by (*smt (verit, ccfv-SIG) 4.hyps(1) 4.hyps(4) div-eq-0-iff ‹Some lx = vebt-mint (treeList ! summin)› ‹Some summin = vebt-mint summary› ‹invar-vebt (treeList ! summin) n› div-exp-eq high-def high-inv le-0-eq member-bound mint-member not-numeral-le-zero power-not-zero*)
hence *h-lt*: *?h < length treeList*
using *4.hyps(2) 4.hyps(3) 4.hyps(4) ‹invar-vebt (treeList ! summin) n› deg-not-0 exp-split-high-low(1)* **by** *metis*
let *?newnode = vebt-delete (treeList ! ?h) ?l*
let *?newlist = treeList[?h:= ?newnode]*
have *length treeList = length ?newlist* **by** *simp*

hence *hprolist*: *?newlist ! ?h = ?newnode*
by (*meson ‹high (summin * 2 ^ n + lx) n < length treeList› nth-list-update*)
have *nothprolist*: *i ≠ ?h ∧ i < 2^m ⟹ ?newlist ! i = treeList ! i* **for** *i* **by** *simp*
have *take ?h treeList @ [?newnode] @ drop (?h + 1) treeList =*
*treeList[high (summin * 2 ^ n + lx) n := ?newlist ! ?h]*
using *h-lt upd-conv-take-nth-drop* **by** *fastforce*
then have *firstsimp*: *vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =(*
let newnode = vebt-delete (treeList ! ?h) ?l;
newlist = (take ?h treeList @ [newnode]@drop (?h+1) treeList)in
if minNull newnode
then(
let sn = vebt-delete summary ?h in
(Node (Some (?xn, if ?xn = ma then (let maxs = vebt-maxt sn in
(if maxs = None
then ?xn
*else 2^(deg div 2) * the maxs*
+ the (vebt-maxt (newlist ! the maxs))
)
)
else ma))
deg newlist sn)
)else
(Node (Some (?xn, (if ?xn = ma then
*?h * 2^(deg div 2) + the(vebt-maxt (newlist ! ?h))*
else ma)))
deg newlist summary))
using *del-x-mi[of x mi ma deg ?xn ?h summary treeList ?l]*
using *‹deg div 2 = n› ‹x = mi› dp h-lt mimapr xn-eq* **by** *auto*
have *minxnrel*: *?xn ≠ mi*
by (*metis 4.hyps(2) 4.hyps(9) ‹high (summin * 2 ^ n + lx) n < length treeList› ‹vebt-member (treeList ! summin) lx› ‹invar-vebt (treeList ! summin) n› both-member-options-equiv-member high-inv less-not-refl low-inv member-bound mimapr*)
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
let *?sn = vebt-delete summary ?h*
let *?newma= (if ?xn= ma then (let maxs = vebt-maxt ?sn in*
(if maxs = None
then ?xn
*else 2^(deg div 2) * the maxs*
+ the (vebt-maxt (?newlist ! the maxs))
)
)
else ma)
let *?delsimp = (Node (Some (?xn, ?newma)) deg ?newlist ?sn)*
have *vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp*
using *del-x-mi-lets-in-minNull[of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist ?sn] False True ‹deg div 2 = n› ‹?h < length treeList› ‹summin * 2 ^ n + lx = (if x = mi then the (vebt-mint summary) * 2 ^ (deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary))) else*

```
x)› dp less-not-refl3 mimapr by fastforce
        moreover have both-member-options (?delsimp) y ⟹ (x ≠ y ∧ both-member-options (Node
(Some (mi, ma)) deg treeList summary) y)
        proof−
          assume both-member-options (?delsimp) y
          hence y = ?xn ∨ y = ?newma ∨
             (both-member-options (?newlist ! (high  y (deg div 2))) (low y (deg div 2)) ∧ (high y (deg
div 2)) < length ?newlist)
            using both-member-options-from-complete-tree-to-child[of deg mi ?newma ?newlist ?sn y]
            by (smt (verit, best) membermima.simps(4) naive-member.simps(3) ‹deg = Suc (Suc -)›
both-member-options-def)
          moreover have y = ?xn ⟹ ?thesis
              by (metis 4.hyps(9) False ‹vebt-member (treeList ! summin) lx› ‹summin < 2 ^ m›
‹invar-vebt (treeList ! summin) n› both-member-options-equiv-member high-inv less-not-refl low-inv
member-bound mimapr xnin)
          moreover have y = ?newma ⟹ ?thesis
          proof−
            assume asmt: y = ?newma
            show ?thesis
            proof(cases ?xn =  ma)
              case True
              let ?maxs = vebt-maxt ?sn
              have newmaext:?newma = (if ?maxs = None then ?xn
                            else 2 ^ (deg div 2) ∗ the ?maxs +   the (vebt-maxt
               ( ?newlist ! the ?maxs)))   using True by force
              then show ?thesis
              proof(cases ?maxs = None )
                case True
                hence aa:?newma = ?xn using newmaext by auto
                hence bb: ?newma ≠ x
                  using False minxnrel by presburger
                hence both-member-options (Node (Some (mi, ma)) deg treeList summary) ?xn
                  using xnin newmaext minxnrel asmt by simp
                moreover have ?xn = y using aa asmt by simp
                ultimately have both-member-options (Node (Some (mi, ma)) deg treeList summary)
y by simp
                then show ?thesis using bb
                 using ‹summin ∗ 2 ^ n + lx = y› ‹y = ?xn ⟹ x ≠ y ∧ both-member-options (Node
(Some (mi, ma)) deg treeList summary) y› by blast
              next
                case False
                then obtain maxs where Some maxs = ?maxs by force
                hence both-member-options ?sn maxs
                  by (simp add: maxbmo)
                hence both-member-options summary maxs ∧ maxs ≠ ?h
                  using 4.IH(2) by blast
                hence ?newlist ! the ?maxs = treeList ! maxs
                 by (metis 4.hyps(1) ‹Some maxs = vebt-maxt (vebt-delete summary (high (summin ∗
2 ^ n + lx) n))› option.sel member-bound nothprolist valid-member-both-member-options)
```

have *maxs* < 2^*m*
using *4.hyps*(*1*) ‹*both-member-options summary maxs* ∧ *maxs* ≠ *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*› *member-bound valid-member-both-member-options* **by** *blast*
hence *the* (*vebt-maxt* (*?newlist* ! *the ?maxs*)) = *the* (*vebt-maxt* (*treeList* ! *maxs*))
using ‹*?newlist* ! *the* (*vebt-maxt ?sn*) = *treeList* ! *maxs*› **by** *presburger*
have ∃ *z*. *both-member-options*(*treeList* ! *maxs*) *z*
using *4.hyps*(*5*) ‹*both-member-options summary maxs* ∧ *maxs* ≠*?h*› ‹*maxs* < 2 ^ *m*› **by** *blast*
moreover have *invar-vebt* (*treeList* ! *maxs*) *n* **using** *4*
using ‹*maxs* < 2 ^ *m*› **by** *simp*
ultimately obtain *maxi* **where** *Some maxi* = (*vebt-maxt* (*treeList* ! *maxs*))
by (*metis empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
hence *maxi* < 2^*n*
by (*metis* ‹*invar-vebt* (*treeList* ! *maxs*) *n*› *maxt-member member-bound*)
hence *both-member-options* (*treeList* ! *maxs*) *maxi*
using ‹*Some maxi* = *vebt-maxt* (*treeList* ! *maxs*)› *maxbmo* **by** *presburger*
hence 2 ^ (*deg div 2*) ∗ *the ?maxs* + *the*
(*vebt-maxt* (*?newlist* ! *the ?maxs*)) = 2^*n* ∗ *maxs* + *maxi*
by (*metis* ‹*Some maxi* = *vebt-maxt* (*treeList* ! *maxs*)› ‹*Some maxs* = *vebt-maxt ?sn*› ‹*deg div 2* = *n*› ‹*the* (*vebt-maxt* (*?newlist* ! *the* (*vebt-maxt ?sn*))) = *the* (*vebt-maxt* (*treeList* ! *maxs*))› *option.sel*)
hence *?newma* = 2^*n* ∗ *maxs* + *maxi*
using *False True* **by** *auto*
hence *y* = 2^*n* ∗ *maxs* + *maxi* **using** *asmt* **by** *simp*
hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis 4.hyps*(*2*) *Suc-1* ‹*both-member-options* (*treeList* ! *maxs*) *maxi*› ‹*deg div 2* = *n*› ‹*maxi* < 2 ^ *n*› ‹*maxs* < 2 ^ *m*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp high-inv low-inv mult.commute plus-1-eq-Suc*)
moreover hence *y* ≠ *x*
by (*metis 4.hyps*(*9*) *True* ‹*Some maxi* = *vebt-maxt* (*treeList* ! *maxs*)› ‹*maxi* < 2 ^ *n*› ‹*maxs* < 2 ^ *m*› ‹*x* = *mi*› ‹*y* = 2 ^ *n* ∗ *maxs* + *maxi*› *high-inv less-not-refl low-inv maxbmo minxnrel mult.commute*)
ultimately show *?thesis* **by** *force*
qed
next
case *False*
hence *?newma* = *ma* **by** *simp*
moreover hence *mi* ≠ *ma*
using *mimapr* **by** *blast*
moreover hence *y* ≠ *x*
using *False* ‹*y* = *?newma*› ‹*x* = *mi*› **by** *auto*
then show *?thesis*
by (*metis False* ‹*y* =*?newma*› *both-member-options-equiv-member vebt-maxt.simps*(*3*) *maxt-member tvalid*)
qed
qed
moreover have (*both-member-options* (*?newlist* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*)) ∧ (*high y* (*deg div 2*)) < *length ?newlist*) ⟹ *?thesis*

proof−
assume *assm*:*both-member-options* (*?newlist* ! (*high* (*y* (*deg div* 2))) (*low y* (*deg div* 2)) ∧ (*high y* (*deg div* 2)) < *length ?newlist*
show *?thesis*
proof(*cases* (*high* (*y* (*deg div* 2)) = *?h*)
case *True*
hence *000*:*both-member-options ?newnode* (*low y* (*deg div* 2)) **using** *hprolist* **by** (*metis assm*)
hence *001*:*invar-vebt* (*treeList* ! (*high y* (*deg div* 2))) *n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
then show *?thesis*
proof(*cases low y n* = *?l*)
case *True*
hence *y* = *?xn*
using *000 4.IH*(*1*) ‹*deg div* 2 = *n*› ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*›
by *simp*
then show *?thesis*
using *calculation*(*2*) **by** *blast*
next
case *False*
hence *both-member-options* (*treeList* ! *?h*) (*low y* (*deg div* 2)) ∧ (*low y* (*deg div* 2)) ≠ (*low ?xn* (*deg div* 2))
using *4.IH*(*1*) ‹*deg div* 2 = *n*› ‹*high ?xn n* < *length treeList*› *inthall 000* **by** *simp*
then show *?thesis*
by (*metis 4.hyps*(*2*) *4.hyps*(*9*) *Suc-1 Suc-leD True* ‹*deg div* 2 = *n*› ‹*length treeList* = *length ?newlist*› ‹*x* = *mi*› *assm both-member-options-from-chilf-to-complete-tree dp less-not-refl mimapr*)
qed
next
case *False*
hence *x* ≠ *y*
by (*metis 4.hyps*(*2*) *4.hyps*(*9*) ‹*deg div* 2 = *n*› ‹*length treeList* = *length ?newlist*› ‹*x* = *mi*› *assm less-not-refl mimapr nothprolist*)
moreover hence (*?newlist* ! (*high* (*y* (*deg div* 2))) = *treeList* ! (*high y* (*deg div* 2)) **using** *nothprolist*
using *4.hyps*(*2*) *False* ‹*length treeList* = *length ?newlist*› *assm* **by** *presburger*
moreover hence *both-member-options* (*treeList* ! (*high y* (*deg div* 2))) (*low y* (*deg div* 2))
using *assm* **by** *presburger*
moreover hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis One-nat-def Suc-leD* ‹*length treeList* = *length ?newlist*› *assm both-member-options-from-chilf-to-com dp numeral-2-eq-2*)
ultimately show *?thesis* **by** *blast*
qed
qed
ultimately show *?thesis* **by** *fastforce*
qed
moreover have (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)⟹

both-member-options ?delsimp y

proof−

assume *assm*: $x \neq y \wedge$ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*

hence *abcv*:$y = mi \vee y = ma \vee$ (*high y n* < *length treeList* $\wedge$ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))

by (*metis Suc-1* ‹*deg div 2* = *n*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)

thus *both-member-options ?delsimp y*

proof−

have $y = mi \Longrightarrow$ *?thesis*

using *False assm* **by** *force*

moreover have $y = ma \Longrightarrow$ *?thesis*

by (*smt* (*verit*) *Suc-le-D both-member-options-def dp membermima.simps*(*4*) *nat-1-add-1 plus-1-eq-Suc*)

moreover have *both-member-options* (*treeList* ! (*high y n*)) (*low y n*) $\Longrightarrow$ *?thesis*

proof−

assume *myass*: *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)

thus *?thesis*

proof(*cases high y n* = *?h*)

case *True*

hence *high y n* = *?h* **by** *simp*

then show *?thesis*

proof(*cases low y n* = *?l*)

case *True*

hence *y* = *?xn*

by (*metis* ‹*high y n* = *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*› *bit-split-inv*)

then show *?thesis*

by (*metis Suc-le-D both-member-options-def dp membermima.simps*(*4*) *nat-1-add-1 plus-1-eq-Suc*)

next

case *False*

hence *pp*:*?newlist* ! *?h* = *?newnode*

using *hprolist* **by** *blast*

hence *invar-vebt* (*treeList* ! *?h*) *n*

using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*

hence *both-member-options ?newnode* (*low y n*)

using *4.IH*(*1*) *False True* ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› *myass* **by** *auto*

then show *?thesis*

by (*metis True* ‹*deg div 2* = *n*› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp nat-1-add-1 pp*)

qed

next

case *False*

hence *pp*:*?newlist* ! (*high y n*) = *treeList* ! (*high y n*) **using** *nothprolist abcv*

by (*metis 4.hyps*(*1*) *4.hyps*(*3*) *4.hyps*(*4*) *assm deg-not-0 exp-split-high-low*(*1*)

member-bound tvalid valid-member-both-member-options)
then show *?thesis*
using ‹*deg* = *Suc* (*Suc* (*n'* + *n'*))› ‹*deg div 2* = *n*› *abcv*
both-member-options-from-chilf-to-complete-tree calculation(*1*,*2*)
by *fastforce*
qed
qed
then show *?thesis*
using *abcv calculation*(*1*) *calculation*(*2*) **by** *fastforce*
qed
qed
ultimately show *?thesis* **by** *metis*
next
case *False*
hence *notemp*:∃ *z*. *both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*
let *?newma* = (*if ?xn* = *ma then*
?h ∗ *2*^(*deg div 2*) + *the*(*vebt-maxt* (*?newlist* ! *?h*))
else ma)
let *?delsimp* = (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist summary*)
have *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-mi-lets-in-not-minNull*[*of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist*]
by (*metis 4*.*hyps*(*3*) *4*.*hyps*(*4*) *False* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*Some summin* = *vebt-mint summary*› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*x* = *mi*› *add-self-div-2 dp option.sel less-not-refl mimapr*)
moreover have *both-member-options ?delsimp y*
⟹ *x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
proof−
assume *ssms*: *both-member-options ?delsimp y*
hence *aaaa*: *y* = *?xn* ∨ *y* = *?newma* ∨ (*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*) ∧ *high y n* < *length ?newlist*)
by (*smt* (*verit*, *ccfv-threshold*) *membermima.simps*(*4*) *naive-member.simps*(*3*) ‹*deg* = *Suc* (*Suc* -)› ‹*deg div 2* = *n*› *both-member-options-def*)
show *x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
proof−
have *y* = *?xn* ⟹ *?thesis*
using ‹*x* = *mi*› *minxnrel xnin* **by** *blast*
moreover have *y* = *?newma* ⟹ *?thesis*
proof−
assume *y* = *?newma*
show *?thesis*
proof(*cases ?xn* = *ma*)
case *True*
hence *aaa*:*?newma* =*?h* ∗ *2* ^ (*deg div 2*) +*the*(*vebt-maxt*(*?newlist* ! *?h*))
by *metis*
have *?newlist* ! *?h* = *?newnode* **using** *hprolist* **by** *blast*
obtain *maxi* **where** *maxidef*:*Some maxi* = *vebt-maxt*(*?newlist* ! *?h*)
by (*metis* (*full-types*) *False hprolist vebt-maxt.elims minNull.simps*(*1*) *minNull.simps*(*4*))

have *aa*:*invar-vebt* (*treeList* ! *?h*) *n*
using *4.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
moreover hence *ab*:*maxi* ≠ *?l* ∧ *both-member-options ?newnode maxi*
using *4.IH*(*1*) ‹*high ?xn n* < *length treeList*› *hprolist maxidef*
by (*auto simp add*: *maxbmo*) (*metis maxbmo nth-mem*)
ultimately have *ac*:*maxi* ≠ *?l* ∧ *both-member-options* (*treeList* ! *?h*) *maxi*
using *4.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
hence *ad*:*maxi* < *2*^*n*
by (*meson aa member-bound valid-member-both-member-options*)
then show *?thesis* **using** *Suc-1 aaa* ‹*y* = *?newma*› *ac add-leD1*
using *4.hyps*(*2*) *4.hyps*(*9*) ‹*deg div 2* = *n*›
‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*x* = *mi*› *dp maxidef*
mimapr
apply *simp*
using *both-member-options-from-chilf-to-complete-tree high-inv low-inv*
apply (*metis Suc-1 Suc-leD nat-less-le option.sel*)
done
next
case *False*
then show *?thesis*
by (*metis* ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› ‹*x* = *mi*› ‹*y* = *?newma*› *both-member-options-equiv-member leD vebt-maxt.simps*(*3*) *maxt-member mimapr tvalid*)
qed
qed
moreover have (*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)∧ *high y n* < *length ?newlist*) ⟹ *?thesis*
proof−
assume *assmy*:(*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)∧ *high y n* < *length ?newlist*)
then show *?thesis*
proof(*cases high y n* = *?h*)
case *True*
hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *presburger*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *4.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
then show *?thesis*
proof(*cases low y n*= *?l*)
case *True*
hence *y* = *?xn*
using *4.IH*(*1*) ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*treeList* [*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)] ! *high y n* = *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)› *assmy* **by** *force*
then show *?thesis*
using *calculation*(*1*) **by** *blast*
next
case *False*
hence *low y n* ≠ *?l* ∧ *both-member-options* (*treeList* ! *?h*) (*low y n*) **using** *assmy*

using *4.IH(1) 4.hyps(2)* ‹*?newlist* ! *high y n* = *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*summin* < *2* ^ *m*›

by (*auto simp add*: *high-inv member-bound*)

then show *?thesis*

by (*metis 4.hyps(2) 4.hyps(9) Suc-1 Suc-leD True* ‹*deg div 2* = *n*› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› ‹*x* = *mi*› *both-member-options-from-chilf-to-complete-tree dp leD mimapr*)

qed

next

case *False*

hence *?newlist* ! (*high y n*) = *treeList* !(*high y n*)

by (*smt* (*verit*) *4.hyps(1) 4.hyps(2) 4.hyps(3) 4.hyps(4) 4.hyps(8)* ‹*length treeList* = *length ?newlist*› ‹*ma* ≤ *2* ^ *deg*› *aaaa calculation(2) deg-not-0 exp-split-high-low(1) less-le-trans member-inv mimapr nothprolist tvalid valid-member-both-member-options*)

hence *both-member-options* (*treeList* !(*high y n*)) (*low y n*)

using *assmy* **by** *presburger*

moreover have *x* ≠ *y*

by (*metis 4.hyps(1) 4.hyps(4) 4.hyps(9)* ‹*invar-vebt* (*treeList* ! *summin*) *n*› ‹*x* < *2* ^ *deg*› ‹*x* = *mi*› *calculation deg-not-0 exp-split-high-low(1) mimapr not-less-iff-gr-or-eq*)

moreover have *high y n* < *length ?newlist* **using** *assmy* **by** *blast*

moreover hence *high y n* < *length treeList*

using ‹*length treeList* = *length ?newlist*› **by** *presburger*

ultimately show *?thesis*

by (*metis One-nat-def Suc-leD* ‹*deg div 2* = *n*› *both-member-options-from-chilf-to-complete-tree dp numerals(2)*)

qed

qed

ultimately show *?thesis*

using *aaaa* **by** *fastforce*

qed

qed

moreover have (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)⟹

both-member-options ?delsimp y

proof−

assume *assm*: *x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*

hence *abcv*:*y* = *mi* ∨ *y* = *ma* ∨ (*high y n* < *length treeList* ∧ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))

by (*metis Suc-1* ‹*deg div 2* = *n*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)

thus *both-member-options ?delsimp y*

proof−

have *y* = *mi* ⟹ *?thesis*

using ‹*x* = *mi*› *assm* **by** *blast*

moreover have *y* = *ma* ⟹ *?thesis*

by (*smt* (*verit*) *Suc-1 Suc-le-D both-member-options-def dp membermima.simps*(*4*))

moreover have (*high y n* < *length treeList* ∧ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))
⟹ *?thesis*
proof−
assume *myass*: (*high y n* < *length treeList* ∧ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))
thus *?thesis*
proof(*cases high y n* = *?h*)
case *True*
then show *?thesis*
proof(*cases low y n* = *?l*)
case *True*
then show *?thesis*
by (*smt* (*verit*) *Suc-1 Suc-le-D* ‹*deg div 2* = *n*› ‹*length treeList* = *length* (*treeList* [*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*) (*low* (*summin* ∗ 2 ^ *n* + *lx*) *n*)]› *add-leD1 bit-split-inv both-member-options-def both-member-options-from-chilf-to-complete-tree dp membermima.simps*(*4*) *myass nth-list-update-neq plus-1-eq-Suc*)
next
case *False*
hence *low y n* ≠ *?l* **by** *simp*
hence *pp*:*?newlist* ! *?h* = *?newnode*
using *hprolist* **by** *blast*
hence *invar-vebt* (*treeList* ! *?h*) *n*
using *4.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
hence *both-member-options ?newnode* (*low y n*)
using *4.IH*(*1*) *True* ‹*high ?xn n* < *length treeList*› ‹*low y n* ≠ *low ?xn n*› *inthall myass* **by** *simp*
then show *?thesis*
by (*metis One-nat-def Suc-leD True* ‹*deg div 2* = *n*› ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *both-member-options-from-chilf-to-complete-tree dp numerals*(*2*) *pp*)
qed
next
case *False*
have *pp*:*?newlist* ! (*high y n*) = *treeList* ! (*high y n*)
using *nothprolist*[*of high y n*] *False*
by (*metis 4.hyps*(*1*) *4.hyps*(*3*) *4.hyps*(*4*) *assm deg-not-0 exp-split-high-low*(*1*) *member-bound tvalid valid-member-both-member-options*)
then show *?thesis*
by (*metis One-nat-def Suc-leD* ‹*deg div 2* = *n*› ‹*length treeList* = *length ?newlist*› *abcv both-member-options-from-chilf-to-complete-tree calculation*(*1*) *calculation*(*2*) *dp numerals*(*2*))
qed
qed
then show *?thesis*
using *abcv calculation*(*1*) *calculation*(*2*) **by** *fastforce*
qed
qed
ultimately show *?thesis* **by** *metis*
qed

```
    qed
   qed
  qed
next
  case (5 treeList n summary m deg mi ma)
  hence tvalid: invar-vebt (Node (Some (mi, ma)) deg treeList summary) deg
    using invar-vebt.intros(5)[of treeList n summary m deg mi ma] by simp
  hence mi ≤ ma and deg div 2 = n and ma ≤ 2^deg using 5
    by  (auto simp add: 5.hyps(3) 5.hyps(4))
  hence dp:deg ≥ 2
    by (meson vebt-maxt.simps(3) maxt-member member-inv tvalid)
  then obtain deg' :: nat where deg = Suc (Suc deg')
    using add-2-eq-Suc le-Suc-ex by blast
  hence nmpr:n≥ 1 ∧ m = Suc n
    using 5.hyps(3) ‹deg div 2 = n› by linarith
  then show ?case proof(cases x <mi ∨ x > ma)
    case True
    hence vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (Node (Some (mi, ma)) deg
treeList summary)
      using delt-out-of-range[of x mi ma deg treeList summary]  ‹2 ≤ deg› by blast
    then show ?thesis
    by (metis 5.hyps(7) True tvalid leD member-inv not-less-iff-gr-or-eq valid-member-both-member-options)
  next
    case False
    hence mi ≤ x ∧ x ≤ ma by simp
    hence xdegrel:x < 2^deg
      using 5.hyps(8) order.strict-trans1 by blast
    then show ?thesis
    proof(cases x = mi ∧ x = ma)
      case True
       hence vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = (Node None deg treeList
summary)
        using del-single-cont[of x mi ma deg treeList summary] ‹2 ≤ deg› by blast
      moreover hence invar-vebt (Node None deg treeList summary) deg
        using 5(4) 5.IH(1) 5.hyps(1) 5.hyps(3) 5.hyps(4) True mi-eq-ma-no-ch
          tvalid invar-vebt.intros(3) by force
      moreover hence ¬ vebt-member (Node None deg treeList summary) y by simp
      moreover hence ¬both-member-options (Node None deg treeList summary) y
        using calculation(2) valid-member-both-member-options by blast
      then show ?thesis
      by (metis True calculation(1) member-inv not-less-iff-gr-or-eq tvalid valid-member-both-member-options)
    next
      case False
      hence mimapr:mi < ma
        by (metis 5.hyps(7) ‹mi ≤ x ∧ x ≤ ma› le-antisym nat-less-le)
      then show ?thesis
      proof(cases x ≠ mi)
        case True
        hence xmi:x ≠ mi by simp
```

```
      let ?h =high x n
      let ?l = low x n
      have ?h < length treeList using xdegrel 5
     by (metis ‹deg div 2 = n› deg-not-0 div-greater-zero-iff dp exp-split-high-low(1) zero-less-numeral)
      let ?newnode = vebt-delete (treeList ! ?h) ?l
      let ?newlist = treeList[?h:=?newnode]
      have length treeList = length ?newlist by simp
      hence hprolist: ?newlist ! ?h = ?newnode
        by (meson ‹high x n < length treeList› nth-list-update-eq)
      have nothprolist: i ≠ ?h ∧ i < 2^m ⟹ ?newlist ! i = treeList ! i for i by simp
      then show ?thesis
      proof(cases minNull ?newnode)
        case True
        let ?sn = vebt-delete summary ?h
        let ?newma= (if x  = ma then (let maxs = vebt-maxt ?sn in
                                                    (if maxs = None
                                                       then mi
                                                       else 2^(deg div 2) * the maxs
                                                         + the (vebt-maxt (?newlist ! the maxs))
                                                     )
                                                  )
                                                else ma)
        let ?delsimp = (Node (Some (mi, ?newma))  deg ?newlist ?sn)
        have vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =  ?delsimp
            using del-x-not-mi-new-node-nil[of mi x ma deg ?h ?l ?newnode treeList ?sn summary
?newlist]
          by (metis True ‹2 ≤ deg› ‹deg div 2 = n› ‹high x n < length treeList› ‹mi < ma› ‹mi ≤ x
∧ x ≤ ma› ‹x ≠ mi› less-not-refl3 order.not-eq-order-implies-strict)
        moreover have both-member-options (?delsimp) y ⟹ (x ≠ y ∧ both-member-options (Node
(Some (mi, ma)) deg treeList summary) y)
        proof−
          assume both-member-options (?delsimp) y
          hence y = mi ∨ y = ?newma ∨
             (both-member-options (?newlist ! (high  y (deg div 2))) (low y (deg div 2)) ∧ (high y (deg
div 2)) < length ?newlist)
            using both-member-options-from-complete-tree-to-child[of deg mi ?newma ?newlist ?sn y]
          by (smt (verit, best) VEBT-internal.membermima.simps(4) VEBT-internal.naive-member.simps(3)
‹deg = Suc (Suc deg')› both-member-options-def)
          moreover have y = mi ⟹ ?thesis
            by (meson ‹x ≠ mi› both-member-options-equiv-member vebt-mint.simps(3) mint-member
tvalid)
          moreover have y = ?newma ⟹ ?thesis
          proof−
            assume y = ?newma
            show ?thesis
            proof(cases x =  ma)
              case True
              let ?maxs = vebt-maxt ?sn
              have newmapropy:?newma = (if ?maxs = None then mi
```

*else 2 ^ (deg div 2) * the ?maxs + the (vebt-maxt*
(?newlist !
the ?maxs)))) **using** *True* **by** *force*
then show *?thesis*
proof(*cases ?maxs = None*)
case *True*
then show *?thesis*
using ‹*y = (if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (treeList [high x n := vebt-delete (treeList ! high x n) (low x n)] ! the maxs)) else ma*› *calculation*(*2*) *newmapropy* **by** *presburger*
next
case *False*
then obtain *maxs* **where** *Some maxs = ?maxs* **by** *force*
hence *both-member-options ?sn maxs*
by (*simp add*: *maxbmo*)
hence *both-member-options summary maxs ∧ maxs ≠ ?h*
using *5.IH*(*2*) **by** *blast*
hence *?newlist ! the ?maxs = treeList ! maxs*
by (*metis 5.hyps*(*1*) ‹*Some maxs = vebt-maxt (vebt-delete summary (high x n))*› *option.sel member-bound nothprolist valid-member-both-member-options*)
have *maxs < 2^m*
using *5.hyps*(*1*) ‹*both-member-options summary maxs ∧ maxs ≠ high x n*› *member-bound valid-member-both-member-options* **by** *blast*
hence *the (vebt-maxt (?newlist ! the ?maxs)) = the (vebt-maxt (treeList ! maxs))*
by (*metis* ‹*Some maxs = vebt-maxt (vebt-delete summary (high x n))*› ‹*both-member-options summary maxs ∧ maxs ≠ high x n*› *option.sel nth-list-update-neq*)
have ∃ *z. both-member-options*(*treeList ! maxs*) *z*
by (*simp add*: *5.hyps*(*5*) ‹*both-member-options summary maxs ∧ maxs ≠ high x n*› ‹*maxs < 2 ^ m*›)
moreover have *invar-vebt* (*treeList ! maxs*) *n* **using** *5*
using ‹*maxs < 2 ^ m*› **by** *simp*
ultimately obtain *maxi* **where** *Some maxi = (vebt-maxt (treeList ! maxs))*
by (*metis empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
hence *maxi < 2^n*
by (*metis* ‹*invar-vebt* (*treeList ! maxs*) *n*› *maxt-member member-bound*)
hence *both-member-options* (*treeList ! maxs*) *maxi*
using ‹*Some maxi = vebt-maxt* (*treeList ! maxs*)› *maxbmo* **by** *presburger*
hence *2 ^ (deg div 2) * the ?maxs + the*
*(vebt-maxt (?newlist ! the ?maxs)) = 2^n * maxs + maxi*
by (*metis* ‹*Some maxi = vebt-maxt* (*treeList ! maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*deg div 2 = n*› ‹*the* (*vebt-maxt* (*treeList*[*high x n := vebt-delete* (*treeList ! high x n*) (*low x n*)] ! *the* (*vebt-maxt* (*vebt-delete summary* (*high x n*))))) = *the* (*vebt-maxt* (*treeList ! maxs*))› *option.sel*)
hence *y = 2^n * maxs + maxi*
using *False* ‹*y = (if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (treeList [high x n := vebt-delete (treeList ! high x n) (low x n)] ! the maxs)) else ma*› *newmapropy* **by** *presburger*

hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis 5.hyps*(*2*) *Suc-1* ‹*both-member-options* (*treeList* ! *maxs*) *maxi*› ‹*deg div 2 = n*› ‹*maxi < 2 ^ n*› ‹*maxs < 2 ^ m*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp high-inv low-inv mult.commute plus-1-eq-Suc*)
moreover hence *y* ≠ *x*
by (*metis* ‹*both-member-options summary maxs* ∧ *maxs* ≠ *high x n*› ‹*maxi < 2 ^ n*› ‹*y = 2 ^ n * maxs + maxi*› *high-inv mult.commute*)
ultimately show *?thesis* **by** *force*
qed
next
case *False*
hence *?newma = ma* **by** *simp*
moreover hence *y* ≠ *x*
using *False* ‹*y = ?newma*› **by** *presburger*
then show *?thesis*
by (*metis False* ‹*y =?newma*› *both-member-options-equiv-member vebt-maxt.simps*(*3*) *maxt-member tvalid*)
qed
qed
moreover have (*both-member-options* (*?newlist* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*)) ∧ (*high y* (*deg div 2*)) < *length ?newlist*) ⟹ *?thesis*
proof−
assume *assm*:*both-member-options* (*?newlist* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*)) ∧ (*high y* (*deg div 2*)) < *length ?newlist*
show *?thesis*
proof(*cases* (*high y* (*deg div 2*)) = *?h*)
case *True*
hence *both-member-options ?newnode* (*low y* (*deg div 2*)) **using** *hprolist* **by** (*metis assm*)
moreover hence *invar-vebt* (*treeList* ! (*high y* (*deg div 2*))) *n*
using *5.IH*(*1*) *True* ‹*high x n < length treeList*› **by** *simp*
ultimately have *both-member-options* (*treeList* ! *?h*) (*low y* (*deg div 2*)) ∧ (*low y* (*deg div 2*)) ≠ (*low x* (*deg div 2*))
using *5.IH*(*1*) ‹*deg div 2 = n*› ‹*high x n < length treeList*› **by** *simp*
then show *?thesis*
by (*metis Suc-1 True* ‹*high x n < length treeList*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp plus-1-eq-Suc*)
next
case *False*
hence *x* ≠ *y*
using ‹*deg div 2 = n*› **by** *blast*
moreover hence (*?newlist* ! (*high y* (*deg div 2*))) = *treeList* ! (*high y* (*deg div 2*)) **using** *nothprolist*
using *5.hyps*(*2*) *False* ‹*length treeList = length ?newlist*› *assm* **by** *presburger*
moreover hence *both-member-options* (*treeList* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*))
using *assm* **by** *presburger*
moreover hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis One-nat-def Suc-leD* ‹*length treeList = length ?newlist*› *assm both-member-options-from-chilf-to-com*

dp numeral-2-eq-2)
ultimately show *?thesis* **by** *blast*
qed
qed
ultimately show *?thesis* **by** *fastforce*
qed
moreover have (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*) ⟹ *both-member-options* (*?delsimp*) *y*
proof−
assume (*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)
hence *aa*:*x* ≠ *y* **and** *bb*:*y* = *mi* ∨ *y* = *ma* ∨ (*both-member-options* (*treeList* ! (*high y n*)) (*low y n*) ∧ *high y n* < *length treeList*)
apply *auto*[*1*] **by** (*metis Suc-1* ‹*deg div 2* = *n*› ‹*x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)
show *both-member-options* (*?delsimp*) *y*
proof−
have *y* = *mi* ⟹*both-member-options* (*?delsimp*) *y*
by (*metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps*(*4*))
moreover have *y* = *ma* ⟹ *both-member-options* (*?delsimp*) *y*
using *aa maxbmo vebt-maxt.simps*(*3*) **by** *presburger*
moreover have *both-member-options* (*treeList* ! (*high y n*)) (*low y n*) ⟹*both-member-options* (*?delsimp*) *y*
proof−
assume *assmy*: *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
then show *both-member-options* (*?delsimp*) *y*
proof(*cases high y n* = *?h*)
case *True*
moreover hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *auto*
hence *0*:*invar-vebt* (*treeList* !(*high y n*)) *n* **using** *5*
using *True* ‹*high x n* < *length treeList*› **by** *simp*
moreover have *1*:*low y n* ≠ *low x n*
by (*metis True aa bit-split-inv*)
moreover have *11*: (*treeList* !(*high y n*)) ∈ *set treeList*
using *True* ‹*high x n* < *length treeList*› **by** *simp*
ultimately have (∀ *xa*. *both-member-options ?newnode xa* =
((*low x n*) ≠ *xa* ∧ *both-member-options* (*treeList* ! *?h*) *xa*))
by (*simp add*: *5.IH*(*1*))
hence ((*low x n*) ≠ *xa* ∧ *both-member-options* (*treeList* ! *?h*) *xa*) ⟹ *both-member-options ?newnode xa* **for** *xa* **by** *blast*
moreover have ((*low x n*) ≠ (*low y n*) ∧ *both-member-options* (*treeList* ! *?h*) (*low y n*)) **using** *1*
using *True assmy* **by** *presburger*
ultimately have *both-member-options ?newnode* (*low y n*) **by** *blast*
then show *?thesis*
by (*metis One-nat-def Suc-leD True* ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *both-member-options-from-chilf-to-complete-tree dp hprolist numerals*(*2*))
next

case *False*
hence *?newlist* ! (*high y n*) = *treeList* ! (*high y n*) **by** *auto*
hence *both-member-options* (*?newlist* !(*high y n*)) (*low y n*)
using *assmy* **by** *presburger*
then show *?thesis*
by (*smt* (*verit*) *Suc-1 Suc-le-D* ‹*deg div 2 = n*› ‹*length treeList = length ?newlist*› *aa add-leD1 bb both-member-options-def both-member-options-from-chilf-to-complete-tree dp membermima.simps*(*4*) *plus-1-eq-Suc*)
qed
qed
ultimately show *?thesis* **using** *bb* **by** *fastforce*
qed
qed
ultimately show *?thesis* **by** *metis*
next
case *False*
hence *notemp*:∃ *z*. *both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*
let *?newma* = (*if x = ma then*
?h ∗ 2^(*deg div 2*) + *the*(*vebt-maxt* (*?newlist* ! *?h*))
else ma)
let *?delsimp* = (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist summary*)
have *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-not-mi-newnode-not-nil*[*of mi x ma deg ?h ?l ?newnode treeList ?newlist summary*] *False xmi mimapr*
using ‹*deg div 2 = n*› ‹*high x n* < *length treeList*› ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› *dp nat-less-le plus-1-eq-Suc* **by** *fastforce*
moreover have *both-member-options ?delsimp y*
⟹ *x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
proof−
assume *ssms*: *both-member-options ?delsimp y*
hence *aaaa*: *y* = *mi* ∨ *y* = *?newma* ∨ (*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*) ∧ *high y n* < *length ?newlist*)
by (*smt* (*verit*, *best*) *membermima.simps*(*4*) *naive-member.simps*(*3*) ‹*deg* = *Suc* (*Suc deg*′)› ‹*deg div 2 = n*› *both-member-options-def*)
show *x* ≠ *y* ∧ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
proof−
have *y* = *mi* ⟹*?thesis*
by (*metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps*(*4*) *xmi*)
moreover have *y* = *?newma* ⟹ *?thesis*
proof−
assume *y* = *?newma*
show *?thesis*
proof(*cases x* = *ma*)
case *True*
hence *?newma* =*?h* ∗ 2 ^(*deg div 2*) +*the*(*vebt-maxt*(*?newlist* ! *?h*))
by *metis*
have *?newlist* ! *?h* = *?newnode* **using** *hprolist* **by** *blast*
obtain *maxi* **where** *maxidef*:*Some maxi* = *vebt-maxt*(*?newlist* ! *?h*)

by (*metis False hprolist vebt-maxt.elims minNull.simps(1) minNull.simps(4)*)
have *aa:invar-vebt* (*treeList* ! *?h*) *n*
using *5.IH*(*1*) ‹*high x n* < *length treeList*› **by** *simp*
moreover hence *ab:maxi* ≠ *?l* ∧ *both-member-options ?newnode maxi*
using *5.IH*(*1*) ‹*high x n* < *length treeList*› *hprolist maxidef*
by *simp* (*metis maxbmo nth-mem*)
ultimately have *ac:maxi* ≠ *?l* ∧ *both-member-options* (*treeList* ! *?h*) *maxi*
using *5.IH*(*1*) ‹*high x n* < *length treeList*› **by** *simp*
hence *ad:maxi* < *2*^*n*
using ‹*invar-vebt* (*treeList* ! *high x n*) *n*› *member-bound valid-member-both-member-options*
by *blast*
then show *?thesis*
by (*metis Suc-1* ‹(*if x* = *ma then high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*)) *else ma*) = *high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*))› ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› ‹*y* = (*if x* = *ma then high x n* ∗ *2* ^ (*deg div 2*) + *the* (*vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *high x n*)) *else ma*)› *ac add-leD1 both-member-options-from-chilf-to-complete-tree dp option.sel high-inv low-inv maxidef plus-1-eq-Suc*)
next
case *False*
then show *?thesis*
by (*simp add*: ‹*y* = *?newma*› *maxbmo*)
qed
qed
moreover have *both-member-options* (*?newlist* ! (*high y n*)) (*low y n*) ⟹ *?thesis*
proof−
assume *assmy:both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)
then show *?thesis*
proof(*cases high y n* = *?h*)
case *True*
hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *presburger*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *5.IH*(*1*) ‹*high x n* < *length treeList*› **by** *simp*
hence *low y n* ≠ *?l* ∧ *both-member-options* (*treeList* ! *?h*) (*low y n*)
using *5.IH*(*1*) *True* ‹*high x n* < *length treeList*› *assmy hprolist* **by** *simp*
then show *?thesis*
by (*metis Suc-1 True* ‹*deg div 2* = *n*› ‹*high x n* < *length treeList*› *add-leD1*
both-member-options-from-chilf-to-complete-tree dp plus-1-eq-Suc)
next
case *False*
hence *?newlist* ! (*high y n*) = *treeList* !(*high y n*) **by** *auto*
then show *?thesis*
by (*metis False Suc-1* ‹*deg div 2* = *n*› ‹*length treeList* = *length ?newlist*› *aaaa add-leD1*
both-member-options-from-chilf-to-complete-tree calculation(*1*) *calculation*(*2*) *dp plus-1-eq-Suc*)
qed
qed
ultimately show *?thesis*

```
            using aaaa by fastforce
          qed
        qed
      moreover have (x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary)
y)⟹
                              both-member-options ?delsimp y
       proof−
        assume assm: x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary)
y
        hence abcv:y = mi ∨ y = ma ∨ ( high y n < length treeList ∧ both-member-options (treeList
! (high y n)) (low y n))
          by (metis Suc-1 ‹deg div 2 = n› add-leD1 both-member-options-from-complete-tree-to-child
member-inv plus-1-eq-Suc tvalid valid-member-both-member-options)
        thus both-member-options ?delsimp y
        proof−
          have y = mi ⟹ ?thesis
            by (metis Suc-1 Suc-le-D both-member-options-def dp membermima.simps(4))
          moreover have y = ma ⟹ ?thesis
            using assm maxbmo vebt-maxt.simps(3) by presburger
          moreover have both-member-options (treeList ! (high y n)) (low y n) ⟹ ?thesis
          proof−
            assume myass: both-member-options (treeList ! (high y n)) (low y n)
            thus ?thesis
            proof(cases high y n = ?h)
              case True
              hence low y n ≠ ?l
                by (metis assm bit-split-inv)
              hence pp:?newlist ! ?h = ?newnode
                using hprolist by blast
              hence invar-vebt (treeList ! ?h) n
                using 5.IH(1) ‹high x n < length treeList› by simp
              hence both-member-options ?newnode (low y n)
                 using 5.IH(1) True ‹high x n < length treeList› ‹low y n ≠ low x n› inthall myass
by simp
              then show ?thesis
                  using True ‹deg div 2 = n› ‹high x n < length treeList› ‹length treeList = length
?newlist› pp dp
                    both-member-options-from-chilf-to-complete-tree
                by (metis Suc-1 Suc-leD)
            next
              case False
              hence pp:?newlist ! (high y n) = treeList ! (high y n) using nothprolist abcv by auto
              then show ?thesis
                 by (metis One-nat-def Suc-leD ‹deg div 2 = n› ‹length treeList = length ?newlist›
abcv both-member-options-from-chilf-to-complete-tree calculation(1) calculation(2) dp numerals(2))
            qed
          qed
          then show ?thesis
            using abcv calculation(1) calculation(2) by fastforce
```

qed
qed
ultimately show *?thesis* **by** *metis*
qed
next
case *False*
hence $x = mi$ **by** *simp*
have *both-member-options summary (high ma n)*
by (*metis 5(10) 5(11) 5(7) 5.hyps(4) div-eq-0-iff Suc-leI Suc-le-D div-exp-eq dual-order.irrefl high-def mimapr nat.simps(3)*)
hence *vebt-member summary (high ma n)*
using *5.hyps(1) valid-member-both-member-options* **by** *blast*
obtain *summin* **where** *Some summin = vebt-mint summary*
by (*metis 5.hyps(1) ‹vebt-member summary (high ma n)› empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def*)
hence $\exists$ *z . both-member-options (treeList ! summin) z*
by (*metis 5.hyps(1) 5.hyps(5) both-member-options-equiv-member member-bound mint-member*)
moreover have *invar-vebt (treeList ! summin) n*
by (*metis 5.IH(1) 5.hyps(1) 5.hyps(2) ‹Some summin = vebt-mint summary› member-bound mint-member nth-mem*)
ultimately obtain *lx* **where** *Some lx = vebt-mint (treeList ! summin)*
by (*metis empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
let *?xn = summin*$*2\hat{}$*n + lx*
have *?xn = (if x = mi*
then the (vebt-mint summary) $*$ *2*$\hat{}$*(deg div 2)*
+ the (vebt-mint (treeList ! the (vebt-mint summary)))
else x)
by (*metis False ‹Some lx = vebt-mint (treeList ! summin)› ‹Some summin = vebt-mint summary› ‹deg div 2 = n› option.sel*)
have *vebt-member (treeList ! summin) lx*
using *‹Some lx = vebt-mint (treeList ! summin)› ‹invar-vebt (treeList ! summin) n› mint-member* **by** *auto*
moreover have *summin < 2*$\hat{}$*m*
by (*metis 5.hyps(1) ‹Some summin = vebt-mint summary› member-bound mint-member*)
ultimately have *xnin: both-member-options (Node (Some (mi, ma)) deg treeList summary) ?xn*
by (*metis 5.hyps(2) Suc-1 ‹deg div 2 = n› ‹invar-vebt (treeList ! summin) n› add-leD1 both-member-options-equiv-member both-member-options-from-chilf-to-complete-tree dp high-inv low-inv member-bound plus-1-eq-Suc*)
let *?h =high ?xn n*
let *?l = low ?xn n*
have *?xn < 2*$\hat{}$*deg*
by (*smt (verit, ccfv-SIG) 5.hyps(1) 5.hyps(4) div-eq-0-iff ‹Some lx = vebt-mint (treeList ! summin)› ‹Some summin = vebt-mint summary› ‹invar-vebt (treeList ! summin) n› div-exp-eq high-def high-inv le-0-eq member-bound mint-member not-numeral-le-zero power-not-zero*)
hence *?h < length treeList*
using *5.hyps(2) ‹vebt-member (treeList ! summin) lx› ‹summin < 2 ^ m› ‹invar-vebt (treeList ! summin) n› high-inv member-bound* **by** *presburger*
let *?newnode = vebt-delete (treeList ! ?h) ?l*

let *?newlist = treeList*[*?h*:= *?newnode*]
have *length treeList = length ?newlist* **by** *simp*
hence *hprolist*: *?newlist ! ?h = ?newnode*
by (*meson* ‹*high* (*summin * 2 ^ n + lx*) *n < length treeList*› *nth-list-update*)
have *nothprolist*: *i ≠ ?h ∧ i < 2^m ⟹ ?newlist ! i = treeList ! i* **for** *i* **by** *simp*
have *firstsimp*: *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =(
let newnode = vebt-delete (*treeList ! ?h*) *?l*;
newlist = treeList[*?h*:= *?newnode*] *in*
if minNull newnode
then(
let sn = vebt-delete summary ?h in
(*Node* (*Some* (*?xn*, *if ?xn = ma then* (*let maxs = vebt-maxt sn in*
(*if maxs = None*
then ?xn
*else 2^*(*deg div 2*) ** the maxs*
+ the (*vebt-maxt* (*newlist ! the maxs*))
)
)
else ma))
deg newlist sn)
)*else*
(*Node* (*Some* (*?xn*, (*if ?xn = ma then*
*?h * 2^*(*deg div 2*) *+ the*(*vebt-maxt* (*newlist ! ?h*))
else ma)))
deg newlist summary))
using *del-x-mi*[*of x mi ma deg ?xn ?h summary treeList ?l*]
‹*deg div 2 = n*› ‹*high* (*summin * 2 ^ n + lx*) *n < length treeList*›
‹*summin * 2 ^ n + lx = (if x = mi then the* (*vebt-mint summary*) ** 2 ^* (*deg div 2*) +
the (*vebt-mint* (*treeList ! the* (*vebt-mint summary*))) *else x*)› ‹*x = mi*› *dp mimapr nat-less-le*
by *smt*
have *minxnrel*: *?xn ≠ mi*
by (*metis 5.hyps*(*2*) *5.hyps*(*9*) ‹*high* (*summin * 2 ^ n + lx*) *n < length treeList*› ‹*vebt-member*
(*treeList ! summin*) *lx*› ‹*invar-vebt* (*treeList ! summin*) *n*› *both-member-options-equiv-member high-inv*
less-not-refl low-inv member-bound mimapr)
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
let *?sn = vebt-delete summary ?h*
let *?newma*= (*if ?xn*= *ma then* (*let maxs = vebt-maxt ?sn in*
(*if maxs = None*
then ?xn
*else 2^*(*deg div 2*) ** the maxs*
+ the (*vebt-maxt* (*?newlist ! the maxs*))
)
)
else ma)
let *?delsimp* = (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist ?sn*)
have *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-mi-lets-in-minNull*[*of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist*

?sn] *False True* ‹*deg div 2 = n*› ‹*?h < length treeList*› ‹*summin * 2 ^ n + lx = (if x = mi then the (vebt-mint summary) * 2 ^ (deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary))) else x)*› *dp less-not-refl3 mimapr* **by** *fastforce*
moreover have *both-member-options (?delsimp) y ⟹ (x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y)*
proof−
assume *both-member-options (?delsimp) y*
hence *y = ?xn ∨ y = ?newma ∨*
(both-member-options (?newlist ! (high y (deg div 2))) (low y (deg div 2)) ∧ (high y (deg div 2)) < length ?newlist)
using *both-member-options-from-complete-tree-to-child*[*of deg mi ?newma ?newlist ?sn y*]
by (*smt* (*verit*, *best*) *VEBT-internal.membermima.simps*(*4*) *VEBT-internal.naive-member.simps*(*3*) ‹*deg = Suc (Suc deg')*› *both-member-options-def*)
moreover have *y = ?xn ⟹ ?thesis*
by (*metis 5.hyps*(*9*) *False* ‹*vebt-member (treeList ! summin) lx*› ‹*summin < 2 ^ m*› ‹*invar-vebt (treeList ! summin) n*› *both-member-options-equiv-member high-inv less-not-refl low-inv member-bound mimapr xnin*)
moreover have *y = ?newma ⟹ ?thesis*
proof−
assume *asmt*: *y = ?newma*
show *?thesis*
proof(*cases ?xn = ma*)
case *True*
let *?maxs = vebt-maxt ?sn*
have *newmaext*:*?newma = (if ?maxs = None then ?xn*
*else 2 ^ (deg div 2) * the ?maxs + the (vebt-maxt*
(?newlist ! the ?maxs))) **using** *True* **by** *force*
then show *?thesis*
proof(*cases ?maxs = None*)
case *True*
hence *aa*:*?newma = ?xn* **using** *newmaext* **by** *auto*
hence *bb*: *?newma ≠ x*
using *False minxnrel* **by** *presburger*
hence *both-member-options (Node (Some (mi, ma)) deg treeList summary) ?xn*
using *xnin newmaext minxnrel asmt* **by** *simp*
moreover have *?xn = y* **using** *aa asmt* **by** *simp*
ultimately have *both-member-options (Node (Some (mi, ma)) deg treeList summary) y* **by** *simp*
then show *?thesis* **using** *bb*
using ‹*summin * 2 ^ n + lx = y*› ‹*y = ?xn ⟹ x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y*› **by** *blast*
next
case *False*
then obtain *maxs* **where** *Some maxs = ?maxs* **by** *force*
hence *both-member-options ?sn maxs*
by (*simp add*: *maxbmo*)
hence *both-member-options summary maxs ∧ maxs ≠ ?h*
using *5.IH*(*2*) **by** *blast*
hence *?newlist ! the ?maxs = treeList ! maxs*

by (*metis 5.hyps(1)* ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin* ∗ *2* ^ *n* + *lx*) *n*))› *option.sel member-bound nothprolist valid-member-both-member-options*)
have *maxs* < *2*^*m*
using *5.hyps(1)* ‹*both-member-options summary maxs* ∧ *maxs* ≠ *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*› *member-bound valid-member-both-member-options* **by** *blast*
hence *the* (*vebt-maxt* (*?newlist* ! *the ?maxs*)) = *the* (*vebt-maxt* (*treeList* ! *maxs*))
using ‹*?newlist* ! *the* (*vebt-maxt ?sn*) = *treeList* ! *maxs*› **by** *presburger*
have ∃ *z*. *both-member-options*(*treeList* ! *maxs*) *z*
using *5.hyps(5)* ‹*both-member-options summary maxs* ∧ *maxs* ≠*?h*› ‹*maxs* < *2* ^ *m*› **by** *blast*
moreover have *invar-vebt* (*treeList* ! *maxs*) *n* **using** *5*
using ‹*maxs* < *2* ^ *m*› **by** *simp*
ultimately obtain *maxi* **where** *Some maxi* = (*vebt-maxt* (*treeList* ! *maxs*))
by (*metis empty-Collect-eq maxt-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
hence *maxi* < *2*^*n*
by (*metis* ‹*invar-vebt* (*treeList* ! *maxs*) *n*› *maxt-member member-bound*)
hence *both-member-options* (*treeList* ! *maxs*) *maxi*
using ‹*Some maxi = vebt-maxt* (*treeList* ! *maxs*)› *maxbmo* **by** *presburger*
hence *2* ^ (*deg div 2*) ∗ *the ?maxs* + *the*
(*vebt-maxt* (*?newlist* ! *the ?maxs*)) = *2*^*n* ∗ *maxs* + *maxi*
by (*metis* ‹*Some maxi = vebt-maxt* (*treeList* ! *maxs*)› ‹*Some maxs = vebt-maxt ?sn*› ‹*deg div 2 = n*› ‹*the* (*vebt-maxt* (*?newlist* ! *the* (*vebt-maxt ?sn*))) = *the* (*vebt-maxt* (*treeList* ! *maxs*))› *option.sel*)
hence *?newma* = *2*^*n* ∗ *maxs* + *maxi*
using *False True* **by** *auto*
hence *y* = *2*^*n* ∗ *maxs* + *maxi* **using** *asmt* **by** *simp*
hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis 5.hyps(2) Suc-1* ‹*both-member-options* (*treeList* ! *maxs*) *maxi*› ‹*deg div 2 = n*› ‹*maxi* < *2* ^ *n*› ‹*maxs* < *2* ^ *m*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp high-inv low-inv mult.commute plus-1-eq-Suc*)
moreover hence *y* ≠ *x*
by (*metis 5.hyps(9) True* ‹*Some maxi = vebt-maxt* (*treeList* ! *maxs*)› ‹*maxi* < *2* ^ *n*› ‹*maxs* < *2* ^ *m*› ‹*x = mi*› ‹*y = 2* ^ *n* ∗ *maxs* + *maxi*› *high-inv less-not-refl low-inv maxbmo minxnrel mult.commute*)
ultimately show *?thesis* **by** *force*
qed
next
case *False*
hence *?newma = ma* **by** *simp*
moreover hence *mi* ≠ *ma*
using *mimapr* **by** *blast*
moreover hence *y* ≠ *x*
using *False* ‹*y = ?newma*› ‹*x = mi*› **by** *auto*
then show *?thesis*
by (*metis False* ‹*y =?newma*› *both-member-options-equiv-member vebt-maxt.simps(3) maxt-member tvalid*)
qed
qed

moreover have (*both-member-options* (*?newlist* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*)) ∧ (*high y* (*deg div 2*)) < *length ?newlist*) ⟹ *?thesis*
proof −
assume *assm*:*both-member-options* (*?newlist* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*)) ∧ (*high y* (*deg div 2*)) < *length ?newlist*
show *?thesis*
proof(*cases* (*high y* (*deg div 2*)) = *?h*)
case *True*
hence *000*:*both-member-options ?newnode* (*low y* (*deg div 2*)) **using** *hprolist* **by** (*metis assm*)
hence *001*:*invar-vebt* (*treeList* ! (*high y* (*deg div 2*))) *n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
then show *?thesis*
proof(*cases low y n* = *?l*)
case *True*
hence *y* = *?xn*
using *000 5.IH*(*1*) ‹*deg div 2* = *n*› ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*› **by** *simp*
then show *?thesis*
using *calculation*(*2*) **by** *blast*
next
case *False*
hence *both-member-options* (*treeList* ! *?h*) (*low y* (*deg div 2*)) ∧ (*low y* (*deg div 2*)) ≠ (*low ?xn* (*deg div 2*))
using *5.IH*(*1*) ‹*deg div 2* = *n*› ‹*high ?xn n* < *length treeList*› *000* **by** *simp*
then show *?thesis*
by (*metis 5.hyps*(*2*) *5.hyps*(*9*) *Suc-1 Suc-leD True* ‹*deg div 2* = *n*› ‹*length treeList* = *length ?newlist*› ‹*x* = *mi*› *assm both-member-options-from-chilf-to-complete-tree dp less-not-refl mimapr*)
qed
next
case *False*
hence *x* ≠ *y*
by (*metis 5.hyps*(*2*) *5.hyps*(*9*) ‹*deg div 2* = *n*› ‹*length treeList* = *length ?newlist*› ‹*x* = *mi*› *assm less-not-refl mimapr nothprolist*)
moreover hence (*?newlist* ! (*high y* (*deg div 2*))) = *treeList* ! (*high y* (*deg div 2*)) **using** *nothprolist*
using *5.hyps*(*2*) *False* ‹*length treeList* = *length ?newlist*› *assm* **by** *presburger*
moreover hence *both-member-options* (*treeList* ! (*high y* (*deg div 2*))) (*low y* (*deg div 2*))
using *assm* **by** *presburger*
moreover hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
by (*metis One-nat-def Suc-leD* ‹*length treeList* = *length ?newlist*› *assm both-member-options-from-chilf-to-com dp numeral-2-eq-2*)
ultimately show *?thesis* **by** *blast*
qed
qed
ultimately show *?thesis* **by** *fastforce*
qed

moreover have ($x \neq y \wedge$ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*)$\Longrightarrow$
both-member-options ?delsimp y
proof−
assume *assm*: $x \neq y \wedge$ *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *y*
hence *abcv*:$y = mi \vee y = ma \vee ($ *high y n* < *length treeList* $\wedge$ *both-member-options* (*treeList* ! (*high y n*)) (*low y n*))
by (*metis Suc-1* ‹*deg div 2* = *n*› *add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options*)
thus *both-member-options ?delsimp y*
proof−
have $y = mi \Longrightarrow$ *?thesis*
using *False assm* **by** *force*
moreover have $y = ma \Longrightarrow$ *?thesis*
by (*smt* (*verit*) *Suc-le-D both-member-options-def dp membermima.simps*(*4*) *nat-1-add-1 plus-1-eq-Suc*)
moreover have *both-member-options* (*treeList* ! (*high y n*)) (*low y n*) $\Longrightarrow$ *?thesis*
proof−
assume *myass*: *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
thus *?thesis*
proof(*cases high y n* = *?h*)
case *True*
hence *high y n* = *?h* **by** *simp*
then show *?thesis*
proof(*cases low y n* = *?l*)
case *True*
hence *y* = *?xn*
by (*metis* ‹*high y n* = *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*› *bit-split-inv*)
then show *?thesis*
by (*metis Suc-le-D both-member-options-def dp membermima.simps*(*4*) *nat-1-add-1 plus-1-eq-Suc*)
next
case *False*
hence *low y n* ≠ *?l*
by (*metis assm bit-split-inv*)
hence *pp*:*?newlist* ! *?h* = *?newnode*
using *hprolist* **by** *blast*
hence *invar-vebt* (*treeList* ! *?h*) *n*
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
hence *both-member-options ?newnode* (*low y n*)
using *5.IH*(*1*) *False True* ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*› *myass* **by** *force*
then show *?thesis*
by (*metis True* ‹*deg div 2* = *n*› ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*› ‹*length treeList* = *length ?newlist*› *add-leD1 both-member-options-from-chilf-to-complete-tree dp nat-1-add-1 pp*)
qed

```
            next
              case False
              hence pp:?newlist ! (high y n) = treeList ! (high y n) using nothprolist abcv by auto
              then show ?thesis
                using ‹deg = Suc (Suc deg')› ‹deg div 2 = n› abcv
                  both-member-options-from-chilf-to-complete-tree calculation(1,2)
                by fastforce
            qed
          qed
          then show ?thesis
            using abcv calculation(1) calculation(2) by fastforce
        qed
      qed
      ultimately show ?thesis by metis
    next
      case False
      hence notemp:∃ z. both-member-options ?newnode z
        using not-min-Null-member by auto
      let ?newma = (if ?xn = ma then
                                        ?h * 2^(deg div 2) + the( vebt-maxt (?newlist ! ?h))
                                      else ma)
      let ?delsimp = (Node (Some (?xn, ?newma)) deg ?newlist summary)
      have vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp
         using del-x-mi-lets-in-not-minNull[of x mi ma deg ?xn ?h summary treeList ?l ?newnode
?newlist]
        using False ‹deg div 2 = n› ‹high (summin * 2 ^ n + lx) n < length treeList› ‹summin *
2 ^ n + lx = (if x = mi then the (vebt-mint summary) * 2 ^ (deg div 2) + the (vebt-mint (treeList !
the (vebt-mint summary))) else x)› ‹x = mi› dp mimapr nat-less-le by fastforce
      moreover have both-member-options ?delsimp y
              ⟹ x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y
      proof−
        assume ssms: both-member-options ?delsimp y
        hence aaaa: y = ?xn ∨ y = ?newma ∨ (both-member-options (?newlist ! (high y n)) (low y
n) ∧ high y n < length ?newlist)
        by (smt (verit, best) membermima.simps(4) naive-member.simps(3) ‹deg = Suc (Suc deg')›
‹deg div 2 = n› both-member-options-def)
        show x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y
        proof−
          have y = ?xn ⟹?thesis
            using ‹x = mi› minxnrel xnin by blast
          moreover have y = ?newma ⟹ ?thesis
          proof−
            assume y = ?newma
            show ?thesis
            proof(cases ?xn = ma)
              case True
              hence aaa:?newma =?h * 2 ^ (deg div 2) +the(vebt-maxt(?newlist ! ?h))
                by metis
              have ?newlist ! ?h = ?newnode  using hprolist by blast
```

obtain *maxi* **where** *maxidef*:*Some maxi* = *vebt-maxt*(*?newlist* ! *?h*)
by (*metis* (*full-types*) *False hprolist vebt-maxt.elims minNull.simps*(*1*) *minNull.simps*(*4*))
have *aa*:*invar-vebt* (*treeList* ! *?h*) *n*
using *5.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
moreover hence *ab*:*maxi* ≠ *?l* ∧ *both-member-options ?newnode maxi*
using *5.IH*(*1*) ‹*high ?xn n* < *length treeList*› *hprolist maxidef*
by *simp* (*metis in-set-conv-nth maxbmo*)
ultimately have *ac*:*maxi* ≠ *?l* ∧ *both-member-options* (*treeList* ! *?h*) *maxi*
using *5.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
hence *ad*:*maxi* < *2*^*n*
using ‹*invar-vebt* (*treeList* ! *high ?xn n*) *n*› *member-bound valid-member-both-member-options*
by *blast*
then show *?thesis* **using** *Suc-1 aaa* ‹*y* = *?newma*› *ac add-leD1*
both-member-options-from-chilf-to-complete-tree dp option.sel high-inv low-inv maxidef
plus-1-eq-Suc
by (*metis* (*no-types*, *lifting*) *True* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)›
‹*deg div 2* = *n*› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*›
‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*›
‹*x* = *mi*› *leD member-bound mimapr mint-corr-help nat-add-left-cancel-le*
valid-member-both-member-options)
next
case *False*
then show *?thesis*
by (*metis* ‹*mi* ≤ *x* ∧ *x* ≤ *ma*› ‹*x* = *mi*› ‹*y* = *?newma*› *both-member-options-equiv-member*
leD vebt-maxt.simps(*3*) *maxt-member mimapr tvalid*)
qed
qed
moreover have (*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)∧ *high y n* < *length*
?newlist) ⟹ *?thesis*
proof−
assume *assmy*:(*both-member-options* (*?newlist* ! (*high y n*)) (*low y n*)∧ *high y n* < *length*
?newlist)
then show *?thesis*
proof(*cases high y n* = *?h*)
case *True*
hence *?newlist* ! (*high y n*) = *?newnode*
using *hprolist* **by** *presburger*
have *invar-vebt* (*treeList* ! *?h*) *n*
using *5.IH*(*1*) ‹*high ?xn n* < *length treeList*› **by** *simp*
then show *?thesis*
proof(*cases low y n*= *?l*)
case *True*
hence *y* = *?xn*
using *5.IH*(*1*) ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*treeList* [*high*
(*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2*
^ *n* + *lx*) *n*)] ! *high y n* = *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^
n + *lx*) *n*)› *assmy* **by** *force*
then show *?thesis*
using *calculation*(*1*) **by** *blast*

next
case *False*
hence *low y n ≠ ?l ∧ both-member-options (treeList ! ?h) (low y n)* **using** *assmy*
using *5.IH(1) 5.hyps(2) ‹?newlist ! high y n = vebt-delete (treeList ! high (summin * 2 ^ n + lx) n) (low (summin * 2 ^ n + lx) n)› ‹vebt-member (treeList ! summin) lx› ‹summin < 2 ^ m› high-inv inthall member-bound* **by** *simp*
then show *?thesis*
by *(metis 5.hyps(2) 5.hyps(9) Suc-1 Suc-leD True ‹deg div 2 = n› ‹high (summin * 2 ^ n + lx) n < length treeList› ‹mi ≤ x ∧ x ≤ ma› ‹x = mi› both-member-options-from-chilf-to-complete-tree dp leD mimapr)*
qed
next
case *False*
hence *?newlist ! (high y n) = treeList !(high y n)*
using *5.hyps(2) ‹length treeList = length ?newlist› assmy nothprolist* **by** *presburger*
hence *both-member-options (treeList !(high y n)) (low y n)*
using *assmy* **by** *presburger*
moreover have *x ≠ y*
by *(metis 5.hyps(1) 5.hyps(4) 5.hyps(9) ‹invar-vebt (treeList ! summin) n› ‹x < 2 ^ deg› ‹x = mi› calculation deg-not-0 exp-split-high-low(1) mimapr not-less-iff-gr-or-eq)*
moreover have *high y n < length ?newlist* **using** *assmy* **by** *blast*
moreover hence *high y n < length treeList*
using *‹length treeList = length ?newlist›* **by** *presburger*
ultimately show *?thesis*
by *(metis One-nat-def Suc-leD ‹deg div 2 = n› both-member-options-from-chilf-to-complete-tree dp numerals(2))*
qed
qed
ultimately show *?thesis*
using *aaaa* **by** *fastforce*
qed
qed
moreover have *(x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y)⟹*
both-member-options ?delsimp y
proof−
assume *assm: x ≠ y ∧ both-member-options (Node (Some (mi, ma)) deg treeList summary) y*
hence *abcv:y = mi ∨ y = ma ∨ (high y n < length treeList ∧ both-member-options (treeList ! (high y n)) (low y n))*
by *(metis Suc-1 ‹deg div 2 = n› add-leD1 both-member-options-from-complete-tree-to-child member-inv plus-1-eq-Suc tvalid valid-member-both-member-options)*
thus *both-member-options ?delsimp y*
proof−
have *y = mi ⟹ ?thesis*
using *‹x = mi› assm* **by** *blast*
moreover have *y = ma ⟹ ?thesis*
by *(smt (verit) Suc-1 Suc-le-D both-member-options-def dp membermima.simps(4))*

```
          moreover have  ( high y n < length treeList ∧ both-member-options (treeList ! (high y
n)) (low y n))
                                ⟹ ?thesis
        proof−
          assume myass: (  high y n < length treeList ∧ both-member-options (treeList ! (high y
n)) (low y n))
          thus ?thesis
          proof(cases high y n = ?h)
            case True
            then show ?thesis
            proof(cases low y n = ?l)
              case True
              then show ?thesis
                  by (smt (verit) 5.hyps(3) 5.hyps(4) Suc-1 ‹deg div 2 = n› ‹length treeList =
length (treeList [high (summin * 2 ^ n + lx) n := vebt-delete (treeList ! high (summin * 2 ^ n +
lx) n) (low (summin * 2 ^ n + lx) n)])› add-Suc-right add-leD1 bit-split-inv both-member-options-def
both-member-options-from-chilf-to-complete-tree dp membermima.simps(4) myass nth-list-update-neq
plus-1-eq-Suc)
            next
              case False
              hence low y n ≠ ?l by simp
              hence pp:?newlist ! ?h = ?newnode
                using hprolist by blast
              hence invar-vebt (treeList ! ?h) n
                using 5.IH(1) ‹high ?xn n < length treeList› by simp
              hence both-member-options ?newnode (low y n)
                 using 5.IH(1) True ‹high ?xn n < length treeList› ‹low y n ≠ low ?xn n› myass
by simp
              then show ?thesis
               by (metis One-nat-def Suc-leD True ‹deg div 2 = n› ‹high (summin * 2 ^ n + lx) n
< length treeList› ‹length treeList = length ?newlist› both-member-options-from-chilf-to-complete-tree
dp numerals(2) pp)
            qed
          next
            case False
            have pp:?newlist ! (high y n) = treeList ! (high y n)
              using nothprolist[of high y n] False 5.hyps(2) myass by presburger
            then show ?thesis
               by (metis One-nat-def Suc-leD ‹deg div 2 = n› ‹length treeList = length ?newlist›
abcv both-member-options-from-chilf-to-complete-tree calculation(1) calculation(2) dp numerals(2))
          qed
        qed
        then show ?thesis
          using abcv calculation(1) calculation(2) by fastforce
      qed
    qed
    ultimately show ?thesis by metis
  qed
qed
```

```
    qed
  qed
qed

end

corollary invar-vebt t n ⟹ both-member-options t x ⟹ x ≠ y ⟹ both-member-options (vebt-delete
t y) x
  using dele-bmo-cont-corr by blast

corollary invar-vebt t n ⟹ both-member-options t x ⟹ ¬ both-member-options (vebt-delete t x) x
  by (simp add: dele-bmo-cont-corr)

corollary invar-vebt t n ⟹ both-member-options (vebt-delete t y) x ⟹ both-member-options t x ∧
x ≠ y
  using dele-bmo-cont-corr by blast

end
end

theory VEBT-DeleteCorrectness imports VEBT-Delete
begin

context VEBT-internal begin
```

8.3 Validness Preservation

```
theorem delete-pres-valid: invar-vebt t n ⟹ invar-vebt (vebt-delete t x) n
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
  proof(cases x)
    case 0
    then show ?thesis
      by (simp add: invar-vebt.intros(1))
  next
    case (Suc prex)
    hence x = Suc prex by simp
    then show ?thesis
    proof(cases prex)
      case 0
      then show ?thesis
        by (simp add: Suc invar-vebt.intros(1))
    next
      case (Suc preprex)
      then show ?thesis
        by (simp add: ‹x = Suc prex› invar-vebt.intros(1))
    qed
  qed
```

next
case (*2 treeList n summary m deg*)
then show *?case*
using *invar-vebt.intros*(*2*) **by** *force*
next
case (*3 treeList n summary m deg*)
then show *?case*
using *invar-vebt.intros*(*3*) **by** *auto*
next
case (*4 treeList n summary m deg mi ma*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary m* **and** *2*:*length treeList* = *2*^*m* **and** *3*: *deg* = *n*+*m* **and**
4: (∀ *i* < *2*^*m*. (∃ *y*. *both-member-options* (*treeList* ! *i*) *y*) ⟷ (*both-member-options summary i*)) **and**
5: (*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)) **and** *6*:*mi* ≤ *ma* ∧ *ma* < *2*^*deg* **and**
7: (*mi* ≠ *ma* ⟶ (∀ *i* < *2*^*m*. (*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *ma*)))
and *8*: *n* = *m* **and** *9*: *deg div 2* = *n* **using** *4 add-self-div-2* **by** *auto*
hence *10*: *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*
using *invar-vebt.intros*(*4*)[*of treeList n summary m deg mi ma*] **by** *blast*
hence *11*:*n* ≥ *1* **and** *12*: *deg* ≥ *2*
by (*metis 1 8 9 One-nat-def Suc-leI deg-not-0 div-greater-zero-iff*)+
then show *?case*
proof(*cases* (*x* < *mi* ∨ *x* > *ma*))
case *True*
hence *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
using *delt-out-of-range*[*of x mi ma deg treeList summary*]
using *1 4*.*hyps*(*3*) *9 deg-not-0 div-greater-zero-iff* **by** *blast*
then show *?thesis*
by (*simp add*: *10*)
next
case *False*
hence *inrg*: *mi*≤ *x* ∧ *x* ≤ *ma* **by** *simp*
then show *?thesis*
proof(*cases x* = *mi* ∧ *x* = *ma*)
case *True*
hence (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)
using *5* **by** *blast*
moreover have *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node None deg treeList summary*)
using *del-single-cont*[*of x mi ma deg treeList summary*] *1 8 9 True deg-not-0 div-greater-zero-iff* **by** *blast*
moreover have (∄ *i*. *both-member-options summary i*)
using *10 True mi-eq-ma-no-ch* **by** *blast*
ultimately show *?thesis*

```
      using 0 1 2 3 4.hyps(3) invar-vebt.intros(2) by force
    next
      case False
      hence x ≠ mi ∨ x ≠ ma by simp
      hence mi ≠ ma ∧ x < 2^deg
        by (metis 6 inrg le-antisym le-less-trans)
      hence 7b: (∀ i < 2^m. (high ma n = i ⟶ both-member-options (treeList ! i) (low ma n)) ∧
            (∀ y. (high y n = i ∧ both-member-options (treeList ! i) (low y n) ) ⟶ mi < y ∧ y ≤
ma))
        using 7 by fastforce
      hence both-member-options (treeList ! (high ma n)) (low ma n)
        using 1 3 6 8 deg-not-0 exp-split-high-low(1) by blast
      hence yhelper:both-member-options (treeList ! (high y n)) (low y n)
              ⟹ high y n < 2^m ⟹ mi < y ∧ y ≤ ma ∧ low y n < 2^n for y
        by (simp add: 7b low-def)
      then show ?thesis
      proof(cases x ≠ mi)
        case True
        hence xnotmi: x ≠ mi by simp
        let ?h = high x n
        let ?l = low x n
        have hlbound:?h < 2^m ∧ ?l < 2^n
          using 1 3 8 ‹mi ≠ ma ∧ x < 2 ^ deg› deg-not-0 exp-split-high-low(1) exp-split-high-low(2)
by blast
        let ?newnode = vebt-delete (treeList ! ?h) ?l
        have treeList ! ?h ∈ set treeList
          using 2 hlbound by simp
        hence nnvalid: invar-vebt ?newnode n
          by (simp add: 4.IH(1))
        let ?newlist = treeList[?h:= ?newnode]
        have hlist:?newlist ! ?h = ?newnode
          by (simp add: 2 hlbound)
        have nothlist:i ≠ ?h ⟹ i < 2^m ⟹ ?newlist ! i = treeList ! i for i by simp
        have allvalidinlist:∀ t ∈ set ?newlist. invar-vebt t n
        proof
          fix t
          assume t ∈ set ?newlist
          then obtain i where i< 2^m ∧ ?newlist ! i = t
            by (metis 2 in-set-conv-nth length-list-update)
          show invar-vebt t n
            by (metis 0 2 ‹i < 2 ^ m ∧ treeList[high x n := vebt-delete (treeList ! high x n) (low x n)] !
i = t› hlist nnvalid nth-list-update-neq nth-mem)
        qed
        have newlistlength: length ?newlist = 2^m using 2 by auto
        then show ?thesis
        proof(cases minNull ?newnode)
          case True
          hence ninNullc: minNull ?newnode by simp
          let ?sn = vebt-delete summary ?h
```

```
      let ?newma= (if x  = ma then (let maxs = vebt-maxt ?sn in
                                                    (if maxs = None
                                                      then mi
                                                      else 2^(deg div 2) * the maxs
                                                        + the (vebt-maxt (?newlist ! the maxs))
                                                    )
                                                  )
                                                else ma)
      let ?delsimp = (Node (Some (mi, ?newma))  deg ?newlist ?sn)
      have dsimp:vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =  ?delsimp
        using del-x-not-mi-new-node-nil[of mi x ma deg ?h ?l ?newnode treeList ?sn summary
?newlist]
        hlbound 9 11 12 True 2 inrg xnotmi by simp
      have newsummvalid: invar-vebt ?sn m
        by (simp add: 4.IH(2))
     have 111: (∀ i < 2^m. (∃ x. both-member-options (?newlist ! i) x) ⟷ ( both-member-options
?sn i))
      proof
        fix i
        show  i < 2^m ⟶ ((∃ x. both-member-options (?newlist ! i) x) = ( both-member-options
?sn i))
        proof
          assume i < 2^m
          show (∃ x. both-member-options (?newlist ! i) x) = ( both-member-options ?sn i)
          proof(cases i = ?h)
            case True
            hence 1000:?newlist ! i = ?newnode
              using hlist by blast
            hence  1001:∄ x. vebt-member (?newlist ! i) x
              by (simp add: min-Null-member ninNullc)
            hence 1002: ∄ x. both-member-options (?newlist ! i) x
              using 1000 nnvalid valid-member-both-member-options by auto
            have 1003: ¬ both-member-options ?sn i
              using 1 True dele-bmo-cont-corr by auto
            then show ?thesis
              using 1002 by blast
          next
            case False
            hence 1000:?newlist ! i = treeList ! i
              using ‹i < 2 ^ m› nothlist by blast
            hence both-member-options (?newlist ! i) y ⟹ both-member-options ?sn i for y
            proof−
              assume both-member-options (?newlist ! i) y
              hence both-member-options summary i
                using 1000 4 ‹i < 2 ^ m› by auto
              thus both-member-options ?sn i
                using 1 False dele-bmo-cont-corr by blast
            qed
            moreover have both-member-options ?sn i ⟹ ∃ y. both-member-options (?newlist ! i)
```

y

proof−
assume *both-member-options ?sn i*
hence *both-member-options summary i*
using *1 dele-bmo-cont-corr* **by** *auto*
thus ∃ *y. both-member-options* (*?newlist* ! *i*) *y*
using *1000 4* ‹*i* < *2* ^ *m*› **by** *presburger*
qed
then show *?thesis*
using *calculation* **by** *blast*
qed
qed
qed
have *112*: (*mi* = *?newma* ⟶ (∀ *t* ∈ *set ?newlist*. ∄ *x. both-member-options t x*))
proof
assume *aampt*:*mi* = *?newma*
show (∀ *t* ∈ *set ?newlist*. ∄ *y. both-member-options t y*)
proof(*cases x* = *ma*)
case *True*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
hence *aa*:∄ *y. vebt-member ?sn y*
using *maxt-corr-help-empty newsummvalid set-vebt'-def* **by** *auto*
hence ∄ *y. both-member-options ?sn y*
using *newsummvalid valid-member-both-member-options* **by** *blast*
hence *t* ∈ *set ?newlist* ⟹ ∄ *y. both-member-options t y* **for** *t*
proof−
assume *t* ∈ *set ?newlist*
then obtain *i* **where** *?newlist* ! *i* = *t* ∧ *i*< *2*^*m*
by (*metis in-set-conv-nth newlistlength*)
thus ∄ *y. both-member-options t y*
using *111* ‹∄ *y. both-member-options* (*vebt-delete summary* (*high x n*)) *y*› **by** *blast*
qed
then show *?thesis* **by** *blast*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *auto*
hence ∃ *y. both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*) **using**

‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*›
empty-Collect-eq option.sel maxt-corr-help-empty set-vebt'-def valid-member-both-member-options
by *fastforce*
hence *maxs* = *high mi n* ∧ *both-member-options* (*?newlist* ! *maxs*) (*low mi n*)
by (*smt* (*verit*) *9 True* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x n*))›
‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*› *aampt option.sel*
high-inv low-inv maxbmo member-bound mult.commute option.distinct(*1*) *valid-member-both-member-options*)

hence *False*
by (*metis bb nat-less-le nothlist yhelper*)
then show *?thesis* **by** *simp*
qed
next
case *False*
then show *?thesis*
using ‹*mi* ≠ *ma* ∧ *x* < *2* ^ *deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma* < *2*^*deg* ∧ *mi* ≤ *?newma*
proof(*cases x* = *ma*)
case *True*
hence *x* = *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
then show *?thesis*
using *6* **by** *fastforce*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound*
valid-member-both-member-options)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *auto*
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
by (*metis* ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*)
n› *empty-Collect-eq maxt-corr-help-empty option-shift.elims set-vebt'-def valid-member-both-member-options*)
then show *?thesis*
by (*smt* (*verit*, *best*) *6 9* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x*
n))› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *bb option.sel high-inv less-le-trans low-inv maxbmo maxt-member*
member-bound mult.commute not-less-iff-gr-or-eq nothlist alethe-comp-simplify1(*3*) *yhelper*)
qed
next

```
          case False
          then show ?thesis
            using 6 by auto
        qed
        have 115: mi ≠ ?newma ⟶
              (∀ i < 2^m.
              (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
              (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma))
        proof
          assume mi ≠ ?newma
          show (∀ i < 2^m.
              (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
              (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma))
          proof
            fix i
            show i < 2^m ⟶
              (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
              (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma)
            proof
              assume assumption:i < 2^m
              show (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
                (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma)
              proof−
                have (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n))
                proof
                  assume newmaassm: high ?newma n = i
                  thus both-member-options (?newlist ! i) (low ?newma n)
                  proof(cases x = ma )
                    case True
                    let ?maxs = vebt-maxt ?sn
                    show ?thesis
                    proof(cases ?maxs = None)
                      case True
                      then show ?thesis
                        by (smt (verit) 0 ‹both-member-options (treeList ! high ma n) (low ma n)› ‹mi
≠ (if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi
else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (treeList [high x n := vebt-delete (treeList ! high x
n) (low x n)] ! the maxs)) else ma)› ‹treeList ! high x n ∈ set treeList› bit-split-inv dele-bmo-cont-corr
hlist newmaassm nth-list-update-neq)
                    next
                      case False
                      then obtain maxs where Some maxs = ?maxs
                        by fastforce
                      hence both-member-options summary maxs
                        by (metis 1 dele-bmo-cont-corr maxbmo)
```

have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *auto*
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*) **using**
‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*›
empty-Collect-eq maxt-corr-help-empty set-vebt'-def valid-member-both-member-options
by *fastforce*
then show *?thesis*
by (*smt* (*verit*) *9 False True* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*› *option.sel high-inv low-inv maxbmo maxt-member member-bound mult.commute newmaassm*)
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption bit-split-inv dele-bmo-cont-corr hlist newmaassm nothlist*)
qed
qed
moreover have (∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *mi* < *y*
proof(*cases i* = *?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr hlist yassm* **by** *auto*
then show *?thesis*
by (*simp add*: *assumption yassm yhelper*)
next
case *False*
then show *?thesis*
using *assumption nothlist yassm yhelper* **by** *presburger*
qed
moreover have *y* ≤ *?newma*
proof(*cases x* = *ma*)
case *True*
hence *x*= *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*

show *?thesis*
proof(*cases ?maxs = None*)
case *True*
then show *?thesis*
using ‹*mi ≠ ?newma*› ‹*x = ma*› **by** *presburger*
next
case *False*
then obtain *maxs* **where** *Some maxs = ?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs ≠ ?h ∧ maxs < 2^m*
by (*metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *auto*
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb ‹both-member-options summary maxs› nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi = vebt-maxt* (*?newlist* ! *maxs*)
by (*metis 2 4*.*IH*(*1*) *Collect-empty-eq bb both-member-options-equiv-member maxt-corr-help-empty nth-list-update-neq nth-mem option.exhaust set-vebt'-def*)
hence *maxs < 2^m ∧ maxi < 2^n*
by (*metis ‹invar-vebt* (*?newlist* ! *maxs*) *n› bb maxt-member member-bound*)
hence *?newma = 2^n∗ maxs + maxi*
by (*smt* (*verit*) *9 False True ‹Some maxi = vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› *option.sel*)
hence *low ?newma n = maxi ∧ high ?newma n = maxs*
by (*simp add*: ‹*maxs < 2 ^ m ∧ maxi < 2 ^ n› high-inv low-inv mult.commute*)
hence *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
by (*metis 0 ‹treeList* ! *high x n ∈ set treeList› assumption dele-bmo-cont-corr hlist nothlist yassm*)
hence *hleqdraft*:*high y n > maxs ⟹ False*
proof−
assume *high y n > maxs*
have *both-member-options summary* (*high y n*)
using *1 111 assumption dele-bmo-cont-corr yassm* **by** *blast*
moreover have *both-member-options ?sn* (*high y n*)
using *111 assumption yassm* **by** *blast*
ultimately show *False*
by (*metis ‹Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*maxs < high y n› leD maxt-corr-help newsummvalid valid-member-both-member-options*)
qed
hence *hleqmaxs*: *high y n ≤ maxs* **by** *presburger*
then show *?thesis*
proof(*cases high y n = maxs*)
case *True*
hence *low y n ≤ maxi*
by (*metis ‹Some maxi = vebt-maxt* (*treeList*[*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*)› ‹*invar-vebt* (*treeList*[*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n› maxt-corr-help valid-member-both-member-options yassm*)

then show *?thesis*
by (*smt* (*verit*) *True* ‹(*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* ((*?newlist*) ! *the maxs*)) *else ma*) = *2 ^ n* ∗ *maxs* + *maxi*› *add-le-cancel-left bit-concat-def bit-split-inv mult.commute*)
next
case *False*
then show *?thesis*
by (*metis* ‹*low* (*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* ((*?newlist*) ! *the maxs*)) *else ma*) *n* = *maxi* ∧ *high* (*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* ((*?newlist*) ! *the maxs*)) *else ma*) *n* = *maxs*› *div-le-mono high-def hleqdraft le-neq-implies-less nat-le-linear*)
qed
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption dele-bmo-cont-corr hlist nothlist yassm yhelper*)
qed
ultimately show *mi* < *y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* < *2^deg* **and** *118*: *mi* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist ?sn*) *deg*
using *invar-vebt.intros*(*4*)[*of ?newlist n ?sn m deg mi ?newma*]
using *3 allvalidinlist newlistlength newsummvalid 4.hyps*(*3*) *111 112 118 117 115* **by** *fastforce*
show *?thesis*
using *116 dsimp* **by** *presburger*
next
case *False*
hence *notemp*:∃ *z*. *both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*

obtain *maxi* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
by (*metis* (*full-types*) *False VEBT-internal.minNull.simps*(*1*,*4*) *hlist vebt-maxt.elims*)
then have *both-member-options* (*?newlist* ! *?h*) *maxi*
using *maxbmo* **by** *metis*
then have *both-member-options ?newnode maxi*
by (*simp only*: *hlist*)
then have *both-member-options* (*treeList* ! *?h*) *maxi*
using *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr* **by** *blast*

let *?newma* = (*if x = ma then*
*?h * 2^(deg div 2) + the(vebt-maxt (?newlist ! ?h))*
else ma)
let *?delsimp* = (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist summary*)
have *dsimp:vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-not-mi-newnode-not-nil*[*of mi x ma deg ?h ?l ?newnode treeList ?newlist summary*]
by (*metis 12 2 9 False dual-order.eq-iff hlbound inrg order.not-eq-order-implies-strict xnotmi*)
have *111*: (∀ *i* < *2^m*. (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) ⟷ (*both-member-options summary i*))
proof
fix *i*
show *i* < *2^m* ⟶ ((∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*))
proof
assume *i* < *2^m*
show (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*)
proof(*cases i* = *?h*)
case *True*
hence *1000*:*?newlist* ! *i* = *?newnode*
using *hlist* **by** *blast*
hence *1001*:∃ *x*. *vebt-member* (*?newlist* ! *i*) *x*
using *nnvalid notemp valid-member-both-member-options* **by** *auto*
hence *1002*: ∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*
using *1000 notemp* **by** *presburger*
have *1003*: *both-member-options summary i*
using *0 1000 1002 4 True* ‹*i* < *2 ^ m*› ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr* **by** *fastforce*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000*:*?newlist* ! *i* = *treeList* ! *i*
using ‹*i* < *2 ^ m*› *nothlist* **by** *blast*
then show *?thesis*
using *4* ‹*i* < *2 ^ m*› **by** *presburger*
qed
qed
qed
have *112*: (*mi* = *?newma* ⟶ (∀ *t* ∈ *set ?newlist*. ∄ *x*. *both-member-options t x*))
proof
assume *aampt*:*mi* = *?newma*
show (∀ *t* ∈ *set ?newlist*. ∄ *y*. *both-member-options t y*)
proof(*cases x* = *ma*)
case *True*
hence *False*
using ‹*both-member-options* (*treeList* ! *?h*) *maxi*›
by (*metis 9* ‹*both-member-options ?newnode maxi*› ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *aampt option.sel high-inv hlbound low-inv member-bound nnvalid not-less-iff-gr-or-eq*

valid-member-both-member-options yhelper)
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
using ‹*mi* ≠ *ma* ∧ *x* < 2 ^ *deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma* < 2^*deg* ∧ *mi* ≤ *?newma*
proof(*cases x* = *ma*)
case *True*
hence *x* = *ma* **by** *simp*
hence *maxi* < 2^*n*
using ‹*both-member-options?newnode maxi*› *member-bound nnvalid valid-member-both-member-options* **by** *blast*
show *?thesis*
by (*smt* (*verit*) *3 9 div-eq-0-iff True* ‹*both-member-options* (*treeList* ! *high x n*) *maxi*› ‹*maxi* < 2 ^ *n*› ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *add.right-neutral div-exp-eq div-mult-self3 option.sel high-inv hlbound le-0-eq less-imp-le-nat low-inv power-not-zero rel-simps*(*28*) *yhelper*)
next
case *False*
then show *?thesis*
using *6* **by** *auto*
qed
have *115*: *mi* ≠ *?newma* ⟶
(∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*))
proof
assume *mi* ≠ *?newma*
show (∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*))
proof
fix *i*
show *i* < 2^*m* ⟶
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof
assume *assumption*:*i* < 2^*m*
show (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof−
have (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*))
proof

assume *newmaassm*: *high ?newma n = i*
thus *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)
proof(*cases x = ma*)
case *True*
then show *?thesis*
using ‹*both-member-options* (*?newlist* ! *?h*) *maxi*›
by (*smt* (*verit*) *9* ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *option.sel high-inv hlist low-inv maxt-member member-bound newmaassm nnvalid*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption bit-split-inv dele-bmo-cont-corr hlist newmaassm nothlist*)
qed
qed
moreover have (∀ *y*. (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi < y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi < y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *mi < y*
proof(*cases i = ?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr hlist yassm* **by** *auto*
then show *?thesis*
by (*simp add*: *assumption yassm yhelper*)
next
case *False*
then show *?thesis*
using *assumption nothlist yassm yhelper* **by** *presburger*
qed
moreover have *y* ≤ *?newma*
proof(*cases x = ma*)
case *True*
hence *x*= *ma* **by** *simp*
have *high y n* ≤ *?h*
by (*metis 7b True assumption div-le-mono high-def nothlist yassm*)
then show *?thesis*
proof(*cases high y n = ?h*)
case *True*
have *low y n > maxi* ⟹ *False*
by (*metis True* ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› *hlist leD maxt-corr-help nnvalid valid-member-both-member-options yassm*)
then show *?thesis*
by (*smt* (*verit*) *9 True* ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› ‹*x = ma*›

add-le-cancel-left div-mult-mod-eq option.sel high-def low-def nat-le-linear nat-less-le)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *9 True* ‹*both-member-options* (*?newlist* ! *high x n*) *maxi*› ‹*high y n* ≤ *high x n*› ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *div-le-mono option.sel high-def high-inv hlist le-antisym member-bound nat-le-linear nnvalid valid-member-both-member-options*)
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption dele-bmo-cont-corr hlist nothlist yassm yhelper*)
qed
ultimately show *mi* < *y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* < *2*^*deg* **and** *118*: *mi* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist summary*) *deg*
using *invar-vebt.intros*(*4*)[*of ?newlist n summary m deg mi ?newma*] *allvalidinlist 1 newlistlength 8 3 111 112 117 118 115* **by** *fastforce*
then show *?thesis*
using *dsimp* **by** *presburger*
qed
next
case *False*
hence *xmi*:*x* = *mi* **by** *simp*
have *both-member-options summary* (*high ma n*)
using *1 3 4 4.hyps*(*3*) *6* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› *deg-not-0 exp-split-high-low*(*1*) **by** *blast*
hence *vebt-member summary* (*high ma n*)
using *4.hyps*(*1*) *valid-member-both-member-options* **by** *blast*
obtain *summin* **where** *Some summin* = *vebt-mint summary*
by (*metis 4.hyps*(*1*) ‹*vebt-member summary* (*high ma n*)› *empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def*)
hence ∃ *z* . *both-member-options* (*treeList* ! *summin*) *z*
by (*metis 4.hyps*(*1*) *4.hyps*(*5*) *both-member-options-equiv-member member-bound mint-member*)
moreover have *invar-vebt* (*treeList* ! *summin*) *n*
by (*metis 0 1 2* ‹*Some summin* = *vebt-mint summary*› *member-bound mint-member nth-mem*)
ultimately obtain *lx* **where** *Some lx* = *vebt-mint* (*treeList* ! *summin*)
by (*metis empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
let *?xn* = *summin*∗*2*^*n* + *lx*
have *?xn* = (*if x* = *mi*
then the (*vebt-mint summary*) ∗ *2*^(*deg div 2*)

```
                                + the (vebt-mint (treeList ! the (vebt-mint summary)))
                        else x)
          by (metis False ‹Some lx = vebt-mint (treeList ! summin)› ‹Some summin = vebt-mint
summary› ‹deg div 2 = n› option.sel)
        have vebt-member (treeList ! summin) lx
            using ‹Some lx = vebt-mint (treeList ! summin)› ‹invar-vebt (treeList ! summin) n›
mint-member by auto
        moreover have summin < 2^m
          by (metis 4.hyps(1) ‹Some summin = vebt-mint summary› member-bound mint-member)
         ultimately have xnin: both-member-options (Node (Some (mi, ma)) deg treeList summary)
?xn
       by (metis 12 2 9 ‹invar-vebt (treeList ! summin) n› add-leD1 both-member-options-equiv-member
both-member-options-from-chilf-to-complete-tree high-inv low-inv member-bound numeral-2-eq-2 plus-1-eq-Suc)
        let ?h =high ?xn n
        let ?l = low ?xn n
        have ?xn < 2^deg
          by (smt (verit, ccfv-SIG) 4.hyps(1) 4.hyps(4) div-eq-0-iff ‹Some lx = vebt-mint (treeList !
summin)› ‹Some summin = vebt-mint summary› ‹invar-vebt (treeList ! summin) n› div-exp-eq high-def
high-inv le-0-eq member-bound mint-member not-numeral-le-zero power-not-zero)
        hence ?h < length treeList
       using 4.hyps(2) 4.hyps(3) 4.hyps(4) ‹invar-vebt (treeList ! summin) n› deg-not-0 exp-split-high-low(1)
by metis
        let ?newnode = vebt-delete (treeList ! ?h) ?l
        let ?newlist = treeList[?h:= ?newnode]
        have length treeList = length ?newlist by simp
        hence hprolist: ?newlist ! ?h = ?newnode
          by (meson ‹high (summin * 2 ^ n + lx) n < length treeList› nth-list-update-eq)
        have nothprolist: i ≠ ?h ∧ i < 2^m ⟹ ?newlist ! i = treeList ! i for i by simp
        have hlbound:?h < 2^m ∧ ?l < 2^n
          using 1 2 3 8 ‹high (summin * 2 ^ n + lx) n < length treeList› ‹summin * 2 ^ n + lx < 2 ^
deg› deg-not-0 exp-split-high-low(2) by presburger
        hence nnvalid: invar-vebt ?newnode n
          using 4.IH(1) ‹high (summin * 2 ^ n + lx) n < length treeList› by simp
        have allvalidinlist:∀ t ∈ set ?newlist. invar-vebt t n
        proof
          fix t
          assume t ∈ set ?newlist
          then obtain i where i < 2^m ∧ ?newlist ! i = t
            by (metis 2 ‹length treeList = length (treeList [high (summin * 2 ^ n + lx) n := vebt-delete
(treeList ! high (summin * 2 ^ n + lx) n) (low (summin * 2 ^ n + lx) n)])› in-set-conv-nth)
          then show invar-vebt t n
            by (metis 0 2 hprolist nnvalid nth-list-update-neq nth-mem)
        qed
        have newlistlength: length ?newlist = 2^m
          by (simp add: 2)
        then show ?thesis
        proof(cases minNull ?newnode)
          case True
          hence ninNullc: minNull ?newnode by simp
```

let *?sn = vebt-delete summary ?h*
let *?newma= (if ?xn = ma then (let maxs = vebt-maxt ?sn in*
(if maxs = None
then ?xn
*else 2^(deg div 2) * the maxs*
+ the (vebt-maxt (?newlist ! the maxs))
)
)
else ma)
let *?delsimp = (Node (Some (?xn, ?newma)) deg ?newlist ?sn)*
have *dsimp:vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp*
using *del-x-mi-lets-in-minNull[of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist ?sn]*
by *(metis 12 9 ‹high (summin * 2 ^ n + lx) n < length treeList› ‹summin * 2 ^ n + lx = (if x = mi then the (vebt-mint summary) * 2 ^ (deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary))) else x)› ‹x = mi› ‹x ≠ mi ∨ x ≠ ma› inrg nat-less-le ninNullc)*
have *newsummvalid: invar-vebt ?sn m*
by *(simp add: 4.IH(2))*
have *111: (∀ i < 2^m. (∃ x. both-member-options (?newlist ! i) x) ⟷ (both-member-options ?sn i))*
proof
fix *i*
show *i < 2^m ⟶ ((∃ x. both-member-options (?newlist ! i) x) = (both-member-options ?sn i))*
proof
assume *i < 2^m*
show *(∃ x. both-member-options (?newlist ! i) x) = (both-member-options ?sn i)*
proof(*cases i = ?h*)
case *True*
hence *1000:?newlist ! i = ?newnode*
using *hprolist* **by** *fastforce*
hence *1001:∄ x. vebt-member (?newlist ! i) x*
by *(simp add: min-Null-member ninNullc)*
hence *1002: ∄ x. both-member-options (?newlist ! i) x*
using *1000 nnvalid valid-member-both-member-options* **by** *auto*
have *1003: ¬ both-member-options ?sn i*
using *1 True dele-bmo-cont-corr* **by** *auto*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000:?newlist ! i = treeList ! i*
using *‹i < 2 ^ m› nothprolist* **by** *blast*
hence *both-member-options (?newlist ! i) y ⟹ both-member-options ?sn i* **for** *y*
proof−
assume *both-member-options (?newlist ! i) y*
hence *both-member-options summary i*
using *1000 4 ‹i < 2 ^ m›* **by** *auto*
thus *both-member-options ?sn i*

```
            using 1 False dele-bmo-cont-corr by blast
          qed
          moreover have both-member-options ?sn i ⟹ ∃ y. both-member-options (?newlist ! i)
y
          proof−
            assume both-member-options ?sn i
            hence both-member-options summary i
              using 1 dele-bmo-cont-corr by auto
            thus ∃ y. both-member-options (?newlist ! i) y
              using 1000 4 ‹i < 2 ^ m› by presburger
          qed
          then show ?thesis
            using calculation by blast
        qed
      qed
      have 112: (?xn = ?newma ⟶ (∀ t ∈ set ?newlist. ∄ x. both-member-options t x))
      proof
        assume aampt:?xn = ?newma
        show (∀ t ∈ set ?newlist. ∄ y. both-member-options t y)
        proof(cases ?xn = ma)
          case True
          let ?maxs = vebt-maxt ?sn
          show ?thesis
          proof(cases ?maxs = None)
            case True
            hence aa:∄ y. vebt-member ?sn y
              using maxt-corr-help-empty newsummvalid set-vebt'-def by auto
            hence ∄ y. both-member-options ?sn y
              using newsummvalid valid-member-both-member-options by blast
            hence t ∈ set ?newlist ⟹ ∄y. both-member-options t y for t
            proof−
              assume t ∈ set ?newlist
              then obtain i where ?newlist ! i = t ∧ i< 2^m
                  by (metis 2 ‹length treeList = length (treeList [high (summin * 2 ^ n + lx) n :=
vebt-delete (treeList ! high (summin * 2 ^ n + lx) n) (low (summin * 2 ^ n + lx) n)])› in-set-conv-nth)
              thus ∄y. both-member-options t y
                using 111 ‹∄y. both-member-options (vebt-delete summary (high (summin * 2 ^ n +
lx) n)) y› by blast
            qed
            then show ?thesis by blast
          next
            case False
            then obtain maxs where Some maxs = ?maxs
              by fastforce
            hence both-member-options summary maxs
              by (metis 1 dele-bmo-cont-corr maxbmo)
            have bb:maxs ≠ ?h ∧ maxs < 2^m
              by (metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo member-bound
```

valid-member-both-member-options)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0*
by (*simp add*: *2 allvalidinlist*)
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
using ‹*invar-vebt* (*treeList* [*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)] ! *maxs*) *n*› *maxt-corr-help-empty set-vebt'-def valid-member-both-member-options* **by** *fastforce*
hence *maxs* = *high ?xn n* ∧ *both-member-options* (*?newlist* ! *maxs*) (*low ?xn n*)
by (*smt* (*verit*) *9 False True* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary ?h*)› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *aampt option.sel high-inv low-inv maxbmo maxt-member member-bound mult.commute*)
hence *False*
using *bb* **by** *blast*
then show *?thesis* **by** *simp*
qed
next
case *False*
hence *?xn* ≠ *?newma* **by** *simp*
hence *False* **using** *aampt* **by** *simp*
then show *?thesis* **by** *blast*
qed
qed
have *114*: *?newma* < *2*^*deg* ∧ *?xn* ≤ *?newma*
proof(*cases ?xn* = *ma*)
case *True*
hence *?xn* = *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
then show *?thesis*
using *4*.*hyps*(*8*) ‹*?xn* = *ma*› **by** *force*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0* **by** (*simp add*: *2 allvalidinlist*)
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4* ‹*both-member-options summary maxs*› *bb nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
using ‹*invar-vebt* (*treeList* [*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)] ! *maxs*) *n*› *empty-Collect-eq*

maxt-corr-help-empty not-Some-eq set-vebt'-def valid-member-both-member-options **by** *fastforce*
hence *abc:?newma = 2^n * maxs + maxi*
by (*smt* (*verit*) *9 True* ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin * 2 ^ n + lx*) *n*))› *option.sel not-None-eq*)
have *abd:maxi < 2^n*
by (*metis* ‹*Some maxi = vebt-maxt* (*?newlist ! maxs*)› ‹*invar-vebt* (*?newlist ! maxs*) *n*› *maxt-member member-bound*)
have *high ?xn n ≤ maxs*
using *1* ‹*Some summin = vebt-mint summary*› ‹*both-member-options summary maxs*› ‹*vebt-member* (*treeList ! summin*) *lx*› ‹*invar-vebt* (*treeList ! summin*) *n*› *high-inv member-bound mint-corr-help valid-member-both-member-options* **by** *force*
then show *?thesis*
proof(*cases high ?xn n = maxs*)
case *True*
then show *?thesis*
using *bb* **by** *fastforce*
next
case *False*
hence *high ?xn n < maxs*
by (*simp add*: ‹*high* (*summin * 2 ^ n + lx*) *n ≤ maxs*› *order.not-eq-order-implies-strict*)
hence *?newma < 2^deg***using**
1 10 9 True ‹*both-member-options summary maxs*› ‹*mi ≠ ma ∧ x < 2 ^ deg*›
equals0D leD maxt-corr-help maxt-corr-help-empty mem-Collect-eq summaxma set-vebt'-def
valid-member-both-member-options
by (*metis option.exhaust-sel*)
moreover have *high ?xn n < high ?newma n*
by (*smt* (*verit*) *9 True* ‹*Some maxi = vebt-maxt* (*treeList* [*high* (*summin * 2 ^ n + lx*) *n := vebt-delete* (*treeList ! high* (*summin * 2 ^ n + lx*) *n*) (*low* (*summin * 2 ^ n + lx*) *n*)] *! maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin * 2 ^ n + lx*) *n*))› ‹*high* (*summin * 2 ^ n + lx*) *n < maxs*› *abd option.sel high-inv mult.commute option.discI*)
ultimately show *?thesis*
by (*metis div-le-mono high-def linear not-less*)
qed
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *12 4.hyps*(*7*) *4.hyps*(*8*) *9 both-member-options-from-complete-tree-to-child dual-order.trans hlbound one-le-numeral xnin yhelper*)
qed
have *115*: *?xn ≠ ?newma ⟶*
(*∀ i < 2^m.*
(*high ?newma n = i ⟶ both-member-options* (*?newlist ! i*) (*low ?newma n*)) ∧
(*∀ y.* (*high y n = i ∧ both-member-options* (*?newlist ! i*) (*low y n*)) *⟶ ?xn < y ∧ y ≤ ?newma*))
proof
assume *assumption0*:*?xn ≠ ?newma*
show (*∀ i < 2^m.*

```
                (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
                (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)  ) ⟶ ?xn < y ∧
y ≤ ?newma))
          proof
            fix i
            show i < 2^m ⟶
                (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
                (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)  ) ⟶ ?xn < y ∧
y ≤ ?newma)
            proof
              assume assumption:i < 2^m
              show  (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
                (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)  ) ⟶ ?xn < y ∧
y ≤ ?newma)
              proof−
                have (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n))
                proof
                  assume newmaassm: high ?newma n = i
                  thus  both-member-options (?newlist ! i) (low ?newma n)
                  proof(cases ?xn = ma )
                    case True
                    hence bb:?xn = ma by simp
                    let ?maxs = vebt-maxt ?sn
                    show ?thesis
                    proof(cases ?maxs = None)
                      case True
                      hence ?newma = ?xn  using assumption Let-def bb by simp
                      hence False using assumption0 by simp
                      then show ?thesis by simp
                    next
                      case False
                      then obtain maxs where Some maxs = ?maxs
                        by fastforce
                      hence both-member-options summary maxs
                        by (metis 1 dele-bmo-cont-corr maxbmo)
                      have bb:maxs ≠ ?h ∧ maxs < 2^m
                              by (metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo
member-bound valid-member-both-member-options)
                      hence invar-vebt (?newlist ! maxs) nusing 0    by (simp add: 2 allvalidinlist)
                      hence ∃ y. both-member-options (?newlist ! maxs) y
                        using 4 ‹both-member-options summary maxs› bb nothprolist by presburger
                      then obtain maxi where Some maxi = vebt-maxt (?newlist ! maxs) using
                        ‹invar-vebt (treeList [high (summin * 2 ^ n + lx) n :=
                        vebt-delete (treeList ! high (summin * 2 ^ n + lx) n) (low (summin * 2 ^ n +
lx) n)] ! maxs) n›
                         equals0D  maxt-corr-help-empty mem-Collect-eq set-vebt'-def
                         valid-member-both-member-options
                        by (metis option.collapse)
                      then show ?thesis using  1 10 9 True ‹Some summin = vebt-mint summary›
```

‹both-member-options summary maxs› ‹vebt-member (treeList ! summin) lx› ‹mi ≠ ma ∧ x < 2 ^ deg›
‹invar-vebt (treeList ! summin) n› bb equals0D high-inv maxt-corr-help maxt-corr-help-empty
mem-Collect-eq member-bound mint-corr-help nat-less-le summaxma set-vebt'-def
valid-member-both-member-options alethe-comp-simplify1(3)
by (metis option.collapse)
qed
next
case *False*
hence *ccc*:*?newma = ma* **by** *simp*
then show *?thesis*
proof(*cases ?xn = ma*)
case *True*
hence *?xn = ?newma*
using *False* **by** *blast*
hence *False*
using *False* **by** *auto*
then show *?thesis* **by** *simp*
next
case *False*
hence *both-member-options* (*?newlist* ! *high ma n*) (*low ma n*)
by (*metis 1* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*vebt-member summary* (*high ma n*)› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *bit-split-inv dele-bmo-cont-corr high-inv hprolist member-bound nothprolist*)
moreover have *high ma n = i ∧ low ma n = low ?newma n* **using** *ccc newmaassm* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
moreover have (∀ *y*. (*high y n = i ∧ both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn < y ∧ y ≤ ?newma*)
proof
fix *y*
show (*high y n = i ∧ both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn < y ∧ y ≤ ?newma*
proof
assume *yassm*: (*high y n = i ∧ both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *?xn < y*
proof(*cases i = ?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *auto*
then show *?thesis*
using *True hprolist min-Null-member ninNullc nnvalid valid-member-both-member-options yassm* **by** *fastforce*
next

case *False*
hence $i \leq ?h \Longrightarrow$ *False*
by (*metis 1 111* ‹*Some summin = vebt-mint summary*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption dele-bmo-cont-corr high-inv le-antisym member-bound mint-corr-help valid-member-both-member-options yassm*)
hence $i > ?h$
using *leI* **by** *blast*
then show *?thesis*
by (*metis div-le-mono high-def not-less yassm*)
qed
moreover have $y \leq ?newma$
proof(*cases ?xn = ma*)
case *True*
hence *?xn= ma* **by** *simp*
let *?maxs = vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs = None*)
case *True*
then show *?thesis*
using *1 111 assumption dele-bmo-cont-corr nothprolist yassm yhelper* **by** *auto*
next
case *False*
then obtain *maxs* **where** *Some maxs = ?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:$maxs \neq ?h \wedge maxs < 2\hat{}m$
by (*metis 1* ‹*Some maxs = vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0*
by (*simp add*: *2 allvalidinlist*)
hence $\exists$ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4* ‹*both-member-options summary maxs*› *bb nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi = vebt-maxt* (*?newlist* ! *maxs*)
by (*metis True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption calculation dele-bmo-cont-corr high-inv hprolist leD member-bound nth-list-update-neq yassm yhelper*)
hence $maxs < 2\hat{}m \wedge maxi < 2\hat{}n$
by (*metis* ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *bb maxt-member member-bound*)
hence $?newma = 2\hat{}n * maxs + maxi$
by (*smt* (*verit*) *9 False True* ‹*Some maxi = vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high ?xn n*))› *option.sel*)
hence *low ?newma n = maxi* $\wedge$ *high ?newma n = maxs*
by (*simp add*: ‹$maxs < 2\hat{}m \wedge maxi < 2\hat{}n$› *high-inv low-inv mult.commute*)
hence *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
by (*metis 1 111 assumption dele-bmo-cont-corr nothprolist yassm*)
hence *hleqdraft*:$high\ y\ n > maxs \Longrightarrow$ *False*
proof−
assume *high y n > maxs*

have *both-member-options summary* (*high y n*)
using *1 111 assumption dele-bmo-cont-corr yassm* **by** *blast*
moreover have *both-member-options ?sn* (*high y n*)
using *111 assumption yassm* **by** *blast*
ultimately show *False*
using *True* ‹*both-member-options* (*treeList* ! *high y n*) (*low y n*)› ‹*summin* ∗ *2* ^ *n* + *lx* < *y*› *assumption leD yassm yhelper* **by** *blast*
qed
hence *hleqmaxs*: *high y n* ≤ *maxs* **by** *presburger*
then show *?thesis*
using ‹*both-member-options* (*treeList* ! *high y n*) (*low y n*)› *assumption calculation dual-order.strict-trans1 yassm yhelper* **by** *auto*
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption dele-bmo-cont-corr high-inv hprolist member-bound nothprolist yassm yhelper*)
qed
ultimately show *?xn* < *y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* < *2*^*deg* **and** *118*: *?xn* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist ?sn*) *deg*
using *invar-vebt.intros*(*4*)[*of ?newlist n ?sn m deg ?xn ?newma*]
using *3 allvalidinlist newlistlength newsummvalid 4.hyps*(*3*) *111 112 118 117 115* **by** *fastforce*
show *?thesis*
using *116 dsimp* **by** *presburger*
next
case *False*
hence *notemp*:∃ *z*. *both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*
let *?newma* = (*if ?xn* = *ma then*
?h ∗ *2*^(*deg div 2*) + *the*(*vebt-maxt* (*?newlist* ! *?h*))
else ma)
let *?delsimp* = (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist summary*)
have *dsimp*:*vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-mi-lets-in-not-minNull*[*of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist*]
12 2 9 False dual-order.eq-iff hlbound inrg order.not-eq-order-implies-strict xmi
by (*metis* ‹*summin* ∗ *2* ^ *n* + *lx* = (*if x* = *mi then the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*))) *else x*)› ‹*x* ≠ *mi* ∨ *x* ≠ *ma*›)
have *111*: (∀ *i* < *2*^*m*. (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) ⟷ (*both-member-options*

summary i))
proof
fix *i*
show *i* < 2^*m* ⟶ ((∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*))
proof
assume *i* < 2^*m*
show (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*)
proof(*cases i* = *?h*)
case *True*
hence *1000*:*?newlist* ! *i* = *?newnode*
using *hprolist* **by** *blast*
hence *1001*:∃ *x*. *vebt-member* (*?newlist* ! *i*) *x*
using *nnvalid notemp valid-member-both-member-options* **by** *auto*
hence *1002*: ∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*
using *1000 notemp* **by** *presburger*
have *1003*: *both-member-options summary i*
using *4 True* ‹∃ *z*. *both-member-options* (*treeList* ! *summin*) *z*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*summin* < 2 ^ *m*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *auto*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000*:*?newlist* ! *i* = *treeList* ! *i*
using ‹*i* < 2 ^ *m*› *nothprolist* **by** *blast*
then show *?thesis*
using *4* ‹*i* < 2 ^ *m*› **by** *presburger*
qed
qed
qed
have *112*: (*?xn* = *?newma* ⟶ (∀ *t* ∈ *set ?newlist*. ∄ *x*. *both-member-options t x*))
proof
assume *aampt*:*?xn* = *?newma*
show (∀ *t* ∈ *set ?newlist*. ∄ *y*. *both-member-options t y*)
proof(*cases ?xn* = *ma*)
case *True*
obtain *maxi* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
by (*metis Collect-empty-eq False hprolist maxt-corr-help-empty nnvalid not-None-eq not-min-Null-member set-vebt'-def valid-member-both-member-options*)
hence *both-member-options ?newnode maxi*
using *hprolist maxbmo* **by** *auto*
hence *both-member-options* (*treeList* ! *?h*) *maxi*
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv member-bound* **by** *force*
hence *False*
by (*metis 9* ‹*both-member-options* (*vebt-delete* (*treeList* ! *high* (*summin* * 2 ^ *n* + *lx*) *n*) (*low* (*summin* * 2 ^ *n* + *lx*) *n*)) *maxi*› ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *aampt add-diff-cancel-left' dele-bmo-cont-corr option.sel high-inv low-inv member-bound*)

then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
using ‹*mi* ≠ *ma* ∧ *x* < 2 ^ *deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma* < 2^*deg* ∧ *?xn* ≤ *?newma*
proof(*cases ?xn* = *ma*)
case *True*
hence *?xn* = *ma* **by** *simp*
obtain *maxi* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
by (*metis 111 2 4 Collect-empty-eq True* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* < *length treeList*› *hprolist maxt-corr-help-empty nnvalid not-None-eq set-vebt'-def valid-member-both-member-options*)
hence *both-member-options ?newnode maxi*
using *hprolist maxbmo* **by** *auto*
hence *both-member-options* (*treeList* ! *?h*) *maxi*
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv member-bound* **by** *force*
hence *maxi* < 2^*n*
using ‹*both-member-options?newnode maxi*› *member-bound nnvalid valid-member-both-member-options* **by** *blast*
show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *3 9 div-eq-0-iff True* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*both-member-options* (*treeList* ! *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*) *maxi*› ‹*vebt-maxt* (*?newlist* ! *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*) = *Some maxi*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *add.right-neutral add-left-mono div-mult2-eq div-mult-self3 option.sel high-inv hlbound le-0-eq member-bound mint-corr-help power-add power-not-zero rel-simps*(*28*) *valid-member-both-member-options*)
next
case *False*
then show *?thesis*
using *10 4.hyps*(*8*) *maxt-corr-help valid-member-both-member-options xnin* **by** *force*

qed
have *115*: *?xn* ≠ *?newma* ⟶
(∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*))
proof
assume *xnmassm*:*?xn* ≠ *?newma*
show (∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*))
proof
fix *i*
show *i* < 2^*m* ⟶

(*high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧*
(∀ *y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma*)
proof
assume *assumption:i < 2^m*
show (*high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧*
(∀ *y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma*)
proof−
have (*high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)*)
proof
assume *newmaassm*: *high ?newma n = i*
thus *both-member-options (?newlist ! i) (low ?newma n)*
proof(*cases ?xn = ma*)
case *True*
obtain *maxi* **where** *vebt-maxt (?newlist ! ?h) = Some maxi*
by (*metis Collect-empty-eq both-member-options-equiv-member hprolist maxt-corr-help-empty nnvalid not-Some-eq notemp set-vebt'-def*)
hence *both-member-options (?newlist ! ?h) maxi*
using *maxbmo* **by** *blast*
then show *?thesis*
by (*smt (verit) 2 9 True ‹Some lx = vebt-mint (treeList ! summin)› ‹high (summin * 2 ^ n + lx) n < length treeList› ‹vebt-member (treeList ! summin) lx› ‹invar-vebt (treeList ! summin) n› add-left-mono dele-bmo-cont-corr eq-iff high-inv hprolist low-inv member-bound mint-corr-help valid-member-both-member-options yhelper*)
next
case *False*
hence *abcd:?newma = ma* **by** *simp*
then show *?thesis*
proof(*cases high ma n = ?h*)
case *True*
hence *?newlist ! high ma n = ?newnode*
using *hprolist* **by** *presburger*
then show *?thesis*
by (*smt (verit) False True ‹both-member-options (treeList ! high ma n) (low ma n)› ‹vebt-member (treeList ! summin) lx› ‹invar-vebt (treeList ! summin) n› bit-split-inv dele-bmo-cont-corr high-inv member-bound newmaassm*)
next
case *False*
hence *?newlist ! high ma n = treeList ! high ma n*
using *1 ‹vebt-member summary (high ma n)› member-bound nothprolist* **by** *blast*
moreover hence *both-member-options (treeList ! high ma n) (low ma n)*
using *‹both-member-options (treeList ! high ma n) (low ma n)›* **by** *blast*
ultimately show *?thesis* **using** *abcd newmaassm* **by** *simp*
qed
qed
qed
moreover have (∀ *y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma*)

proof
fix *y*
show (*high y n = i ∧ both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn < y ∧ y ≤ ?newma*
proof
assume *yassm*: (*high y n = i ∧ both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *?xn < y*
proof(*cases i = ?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *force*
moreover have *vebt-mint* (*treeList* ! *i*) = *Some* (*low ?xn n*)
using *True* ‹*Some lx = vebt-mint* (*treeList* ! *summin*)› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv low-inv member-bound* **by** *presburger*
moreover hence *low y n ≥ low ?xn n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *calculation*(*1*) *high-inv member-bound mint-corr-help valid-member-both-member-options* **by** *auto*
moreover have *low y n ≠ low ?xn n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *auto*
ultimately have *low y n > low ?xn n* **by** *simp*
show *?thesis*
by (*metis True* ‹*low* (*summin ∗ 2 ^ n + lx*) *n ≤ low y n*› ‹*low y n ≠ low* (*summin ∗ 2 ^ n + lx*) *n*› *bit-concat-def bit-split-inv leD linorder-neqE-nat nat-add-left-cancel-less yassm*)
next
case *False*
have *Some* (*high ?xn n*) = *vebt-mint summary*
using ‹*Some summin = vebt-mint summary*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
moreover hence *high y n ≥ high ?xn n*
by (*metis 1 111 assumption mint-corr-help valid-member-both-member-options yassm*)
ultimately show *?thesis*
by (*metis False div-le-mono high-def leI le-antisym yassm*)
qed
moreover have *y ≤ ?newma*
by (*smt* (*verit*) ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption calculation dele-bmo-cont-corr high-inv hprolist leD member-bound nothprolist yassm yhelper*)
ultimately show *?xn < y ∧ y ≤ ?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma < 2^deg* **and** *118*: *?xn ≤ ?newma* **using** *114* **by** *auto*

have *116*: *invar-vebt* (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist summary*) *deg*
using *invar-vebt.intros*(*4*)[*of ?newlist n summary m deg ?xn ?newma*] *allvalidinlist*
1 newlistlength 8 3 111 112 117 118 115 **by** *fastforce*
hence *invar-vebt* (*?delsimp*) *deg* **by** *simp*
moreover obtain *delsimp* **where** *118*:*delsimp* = *?delsimp* **by** *simp*
ultimately have *119*:*invar-vebt delsimp deg* **by** *simp*
have *vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* = *delsimp* **using** *dsimp 118*
by *simp*
hence *delsimp* = *vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* **by** *simp*
then show *?thesis* **using** *119*
using *xmi* **by** *auto*
qed
qed
qed
qed
next
case (*5 treeList n summary m deg mi ma*)
hence *0*: (∀ *t* ∈ *set treeList*. *invar-vebt t n*) **and** *1*: *invar-vebt summary m* **and** *2*:*length treeList* = *2^m* **and** *3*: *deg* = *n*+*m* **and**
4: (∀ *i* < *2^m*. (∃ *y*. *both-member-options* (*treeList* ! *i*) *y*) ⟷ (*both-member-options summary i*)) **and**
5: (*mi* = *ma* ⟶ (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)) **and** *6*:*mi* ≤ *ma* ∧ *ma* < *2^deg* **and**
7: (*mi* ≠ *ma* ⟶ (∀ *i* < *2^m*. (*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *ma*)))
and *8*: *Suc n* = *m* **and** *9*: *deg div 2* = *n* **using** *5 add-self-div-2* **by** *auto*
hence *10*: *invar-vebt* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg*
using *invar-vebt.intros*(*5*)[*of treeList n summary m deg mi ma*] **by** *blast*
hence *11*:*n* ≥ *1* **and** *12*: *deg* ≥ *2*
by (*metis 0 2 9 One-nat-def deg-not-0 div-greater-zero-iff le-0-eq numeral-2-eq-2 set-n-deg-not-0*)+
then show *?case*
proof(*cases* (*x* < *mi* ∨ *x* > *ma*))
case *True*
hence *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
using *delt-out-of-range*[*of x mi ma deg treeList summary*]
using *12* **by** *fastforce*
then show *?thesis*
by (*simp add*: *10*)
next
case *False*
hence *inrg*: *mi*≤ *x* ∧ *x* ≤ *ma* **by** *simp*
then show *?thesis*
proof(*cases x* = *mi* ∧ *x* = *ma*)
case *True*
hence (∀ *t* ∈ *set treeList*. ∄ *y*. *both-member-options t y*)
using *5* **by** *blast*

moreover have *vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = (*Node None deg treeList summary*)
using *del-single-cont*[*of x mi ma deg treeList summary*] *1 8 9 True deg-not-0 div-greater-zero-iff 12* **by** *fastforce*
moreover have (∄ *i*. *both-member-options summary i*)
using *10 True mi-eq-ma-no-ch* **by** *blast*
ultimately show *?thesis*
using *0 1 2 3 8 invar-vebt.intros*(*3*) **by** *force*
next
case *False*
hence *x* ≠ *mi* ∨ *x* ≠ *ma* **by** *simp*
hence *mi* ≠ *ma* ∧ *x* < *2^deg*
by (*metis 6 inrg le-antisym le-less-trans*)
hence *7b*: (∀ *i* < *2^m*. (*high ma n* = *i* ⟶ *both-member-options* (*treeList* ! *i*) (*low ma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*treeList* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *ma*))
using *7* **by** *fastforce*
hence *both-member-options* (*treeList* ! (*high ma n*)) (*low ma n*)
by (*metis 1 12 3 6 9 deg-not-0 div-greater-zero-iff exp-split-high-low*(*1*) *zero-less-numeral*)
hence *yhelper*:*both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
⟹ *high y n* < *2^m* ⟹ *mi* < *y* ∧ *y* ≤ *ma* ∧ *low y n* < *2^n* **for** *y*
by (*simp add*: *7b low-def*)
then show *?thesis*
proof(*cases x* ≠ *mi*)
case *True*
hence *xnotmi*: *x* ≠ *mi* **by** *simp*
let *?h* = *high x n*
let *?l* = *low x n*
have *hlbound*:*?h* < *2^m* ∧ *?l* < *2^n*
by (*metis 1 11 3 One-nat-def* ‹*mi* ≠ *ma* ∧ *x* < *2* ^ *deg*› *deg-not-0 dual-order.strict-trans1 exp-split-high-low*(*1*) *exp-split-high-low*(*2*) *zero-less-Suc*)
let *?newnode* = *vebt-delete* (*treeList* ! *?h*) *?l*
have *treeList* ! *?h* ∈ *set treeList*
using *2 hlbound* **by** *simp*
hence *nnvalid*: *invar-vebt ?newnode n*
by (*simp add*: *5.IH*(*1*))
let *?newlist* = *treeList*[*?h*:= *?newnode*]
have *hlist*:*?newlist* ! *?h* = *?newnode*
by (*simp add*: *2 hlbound*)
have *nothlist*:*i* ≠ *?h* ⟹ *i* < *2^m* ⟹ *?newlist* ! *i* = *treeList* ! *i* **for** *i* **by** *simp*
have *allvalidinlist*:∀ *t* ∈ *set ?newlist*. *invar-vebt t n*
proof
fix *t*
assume *t* ∈ *set ?newlist*
then obtain *i* **where** *i*< *2^m* ∧ *?newlist* ! *i* = *t*
by (*metis 2 in-set-conv-nth length-list-update*)
then show *invar-vebt t n*
by (*metis 0 2 hlist nnvalid nth-list-update-neq nth-mem*)
qed

have *newlistlength*: *length ?newlist = 2^m*
by (*simp add*: *2*)
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
hence *ninNullc*: *minNull ?newnode* **by** *simp*
let *?sn = vebt-delete summary ?h*
let *?newma*= (*if x = ma then* (*let maxs = vebt-maxt ?sn in*
(*if maxs = None*
then mi
*else 2^*(*deg div 2*) ∗ *the maxs*
+ *the* (*vebt-maxt* (*?newlist* ! *the maxs*))
)
)
else ma)
let *?delsimp* = (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist ?sn*)
have *dsimp*:*vebt-delete* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-not-mi-new-node-nil*[*of mi x ma deg ?h ?l ?newnode treeList ?sn summary ?newlist*]
hlbound 9 11 12 True 2 inrg xnotmi **by** *simp*
have *newsummvalid*: *invar-vebt ?sn m*
by (*simp add*: *5.IH*(*2*))
have *111*: (∀ *i* < *2^m*. (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) ⟷ (*both-member-options ?sn i*))
proof
fix *i*
show *i* < *2^m* ⟶ ((∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options ?sn i*))
proof
assume *i* < *2^m*
show (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options ?sn i*)
proof(*cases i* = *?h*)
case *True*
hence *1000*:*?newlist* ! *i* = *?newnode*
using *hlist* **by** *blast*
hence *1001*:∄ *x*. *vebt-member* (*?newlist* ! *i*) *x*
by (*simp add*: *min-Null-member ninNullc*)
hence *1002*: ∄ *x*. *both-member-options* (*?newlist* ! *i*) *x*
using *1000 nnvalid valid-member-both-member-options* **by** *auto*
have *1003*: ¬ *both-member-options ?sn i*
using *1 True dele-bmo-cont-corr* **by** *auto*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000*:*?newlist* ! *i* = *treeList* ! *i*
using ‹*i* < *2* ^ *m*› *nothlist* **by** *blast*
hence *both-member-options* (*?newlist* ! *i*) *y* ⟹ *both-member-options ?sn i* **for** *y*
by (*metis 1 4 False* ‹*i* < *2* ^ *m*› *dele-bmo-cont-corr*)

```
            moreover have both-member-options ?sn i ⟹ ∃ y. both-member-options (?newlist ! i)
y
              using 1 4 ‹i < 2 ^ m› dele-bmo-cont-corr by force
            then show ?thesis
              using calculation by blast
          qed
        qed
      qed
      have 112: (mi = ?newma ⟶ (∀ t ∈ set ?newlist. ∄ x. both-member-options t x))
      proof
        assume aampt:mi = ?newma
        show (∀ t ∈ set ?newlist. ∄ y. both-member-options t y)
        proof(cases x = ma)
          case True
          let ?maxs = vebt-maxt ?sn
          show ?thesis
          proof(cases ?maxs = None)
            case True
            hence aa:∄ y. vebt-member ?sn y
              using maxt-corr-help-empty newsummvalid set-vebt'-def by auto
            hence ∄ y. both-member-options ?sn y
              using newsummvalid valid-member-both-member-options by blast
            hence t ∈ set ?newlist ⟹ ∄y. both-member-options t y for t
            proof−
              assume t ∈ set ?newlist
              then obtain i where ?newlist ! i = t ∧ i< 2^m
                by (metis in-set-conv-nth newlistlength)
              thus ∄y. both-member-options t y
                using 111 ‹∄y. both-member-options (vebt-delete summary (high x n)) y› by blast
            qed
            then show ?thesis by blast
          next
            case False
            then obtain maxs where Some maxs = ?maxs
              by fastforce
            hence both-member-options summary maxs
              by (metis 1 dele-bmo-cont-corr maxbmo)
            have bb:maxs ≠ ?h ∧ maxs < 2^m
              by (metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo member-bound
valid-member-both-member-options)
            hence invar-vebt (?newlist ! maxs) nusing 0
              by (metis allvalidinlist newlistlength nth-mem)
            hence ∃ y. both-member-options (?newlist ! maxs) y
              using 4 bb ‹both-member-options summary maxs› nothlist by presburger
            then obtain maxi where Some maxi = vebt-maxt (?newlist ! maxs)
                by (metis Collect-empty-eq-bot ‹invar-vebt (treeList[high x n := vebt-delete (treeList
! high x n) (low x n)] ! maxs) n› bb bot-empty-eq equals0D maxt-corr-help-empty nth-list-update-neq
option-shift.elims set-vebt'-def valid-member-both-member-options)
            hence maxs = high mi n ∧ both-member-options (?newlist ! maxs) (low mi n)
```

by (*smt* (*verit*) *9 False True* ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *aampt option.sel high-inv low-inv maxbmo maxt-member member-bound mult.commute*)
hence *False*
by (*metis bb nat-less-le nothlist yhelper*)
then show *?thesis* **by** *simp*
qed
next
case *False*
then show *?thesis*
using ‹*mi ≠ ma ∧ x < 2 ^ deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma < 2^deg ∧ mi ≤ ?newma*
proof(*cases x = ma*)
case *True*
hence *x = ma* **by** *simp*
let *?maxs = vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs = None*)
case *True*
then show *?thesis*
using *6* **by** *fastforce*
next
case *False*
then obtain *maxs* **where** *Some maxs = ?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs ≠ ?h ∧ maxs < 2^m*
by (*metis 1* ‹*Some maxs = vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0*
by (*metis allvalidinlist newlistlength nth-mem*)
hence ∃ *y. both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi = vebt-maxt* (*?newlist* ! *maxs*)
by (*metis* ‹*invar-vebt* (*treeList*[*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*›
maxt-corr-help-empty member-correct minNullmin min-Null-member mint-corr-help-empty ninNullc nnvalid
option.exhaust set-vebt-set-vebt'-valid valid-member-both-member-options)
then show *?thesis*
by (*smt* (*verit*, *best*) *6 9* ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *bb option.sel high-inv less-le-trans low-inv maxbmo maxt-member member-bound mult.commute not-less-iff-gr-or-eq nothlist alethe-comp-simplify1*(*3*) *yhelper*)
qed
next
case *False*

```
        then show ?thesis
          using 6 by auto
      qed
      have 115: mi ≠ ?newma ⟶
            (∀ i < 2^m.
            (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
           (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma))
      proof
        assume mi ≠ ?newma
        show (∀ i < 2^m.
            (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
           (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma))
        proof
          fix i
          show i < 2^m ⟶
            (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
           (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma)
          proof
            assume assumption:i < 2^m
            show (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
              (∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n) ) ⟶ mi < y ∧ y
≤ ?newma)
            proof−
              have (high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n))
              proof
                assume newmaassm: high ?newma n = i
                thus both-member-options (?newlist ! i) (low ?newma n)
                proof(cases x = ma )
                  case True
                  let ?maxs = vebt-maxt ?sn
                  show ?thesis
                  proof(cases ?maxs = None)
                    case True
                    then show ?thesis
                      by (smt (verit) 0 ‹both-member-options (treeList ! high ma n) (low ma n)› ‹mi
≠ (if x = ma then let maxs = vebt-maxt (vebt-delete summary (high x n)) in if maxs = None then mi
else 2 ^ (deg div 2) * the maxs + the (vebt-maxt (?newlist ! the maxs)) else ma)› ‹treeList ! high x n
∈ set treeList› assumption bit-split-inv dele-bmo-cont-corr hlist newmaassm nothlist)
                  next
                    case False
                    then obtain maxs where Some maxs = ?maxs
                      by fastforce
                    hence both-member-options summary maxs
                      by (metis 1 dele-bmo-cont-corr maxbmo)
                    have bb:maxs ≠ ?h ∧ maxs < 2^m
                        by (metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo
```

member-bound valid-member-both-member-options)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *auto*
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
by (*metis* ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*›
maxt-corr-help-empty member-correct minNullmin min-Null-member mint-corr-help-empty ninNullc nnvalid
not-None-eq set-vebt-set-vebt'-valid valid-member-both-member-options)
then show *?thesis*
by (*smt* (*verit*) *9 True* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*› *option.sel high-inv low-inv maxbmo maxt-member member-bound mult.commute newmaassm option.distinct*(*1*))
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption bit-split-inv dele-bmo-cont-corr hlist newmaassm nothlist*)
qed
qed
moreover have (∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *mi* < *y*
proof(*cases i* = *?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr hlist yassm* **by** *auto*
then show *?thesis*
by (*simp add*: *assumption yassm yhelper*)
next
case *False*
then show *?thesis*
using *assumption nothlist yassm yhelper* **by** *presburger*
qed
moreover have *y* ≤ *?newma*
proof(*cases x* = *ma*)
case *True*
hence *x*= *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)

case *True*
then show *?thesis*
using ‹*mi* ≠ *?newma*› ‹*x* = *ma*› **by** *presburger*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0 2* **by** *fastforce*
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 bb* ‹*both-member-options summary maxs*› *nothlist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
by (*metis* ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*› *equals0D maxt-corr-help-empty mem-Collect-eq option-shift.elims set-vebt'-def valid-member-both-member-options*)
hence *maxs* < *2*^*m* ∧ *maxi* < *2*^*n*
by (*metis* ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *bb maxt-member member-bound*)
hence *?newma* = *2*^*n*∗ *maxs* + *maxi*
by (*smt* (*verit*) *9 False True* ‹*Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x n*))› *option.sel*)
hence *low ?newma n* = *maxi* ∧ *high ?newma n* = *maxs*
by (*simp add*: ‹*maxs* < *2* ^ *m* ∧ *maxi* < *2* ^ *n*› *high-inv low-inv mult.commute*)
hence *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
by (*metis 0* ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption dele-bmo-cont-corr hlist nothlist yassm*)
hence *hleqdraft*:*high y n* > *maxs* ⟹ *False*
proof−
assume *high y n* > *maxs*
have *both-member-options summary* (*high y n*)
using *1 111 assumption dele-bmo-cont-corr yassm* **by** *blast*
moreover have *both-member-options ?sn* (*high y n*)
using *111 assumption yassm* **by** *blast*
ultimately show *False*
by (*metis* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high x n*))› ‹*maxs* < *high y n*› *leD maxt-corr-help newsummvalid valid-member-both-member-options*)
qed
hence *hleqmaxs*: *high y n* ≤ *maxs* **by** *presburger*
then show *?thesis*
proof(*cases high y n* = *maxs*)
case *True*
hence *low y n* ≤ *maxi*
by (*metis* ‹*Some maxi* = *vebt-maxt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*)› ‹*invar-vebt* (*treeList*[*high x n* := *vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *maxs*) *n*› *maxt-corr-help valid-member-both-member-options yassm*)
then show *?thesis*

by (*smt* (*verit*) *True* ‹(*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* (*treeList* [*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *the maxs*)) *else ma*) = *2 ^ n* ∗ *maxs* + *maxi*› *add-le-cancel-left bit-concat-def bit-split-inv mult.commute*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) ‹*low* (*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* (*treeList* [*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *the maxs*)) *else ma*) *n = maxi* ∧ *high* (*if x = ma then let maxs = vebt-maxt* (*vebt-delete summary* (*high x n*)) *in if maxs = None then mi else 2 ^* (*deg div 2*) ∗ *the maxs* + *the* (*vebt-maxt* (*treeList* [*high x n := vebt-delete* (*treeList* ! *high x n*) (*low x n*)] ! *the maxs*)) *else ma*) *n = maxs*› *div-le-mono high-def hleqmaxs le-antisym nat-le-linear*)
qed
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption dele-bmo-cont-corr hlist nothlist yassm yhelper*)
qed
ultimately show *mi < y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma < 2^deg* **and** *118*: *mi* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist ?sn*) *deg*
using *invar-vebt.intros*(*5*)[*of ?newlist n ?sn m deg mi ?newma*]
using *3 allvalidinlist newlistlength newsummvalid 5.hyps*(*3*) *111 112 118 117 115* **by** *fastforce*
show *?thesis*
using *116 dsimp* **by** *presburger*
next
case *False*
hence *notemp*:∃ *z. both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*

obtain *maxi* :: *nat* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
using *False*
by (*metis* (*full-types*) *VEBT-internal.minNull.simps*(*1*,*4*) *hlist vebt-maxt.elims*)
then have *both-member-options* (*?newlist* ! *?h*) *maxi*
using *maxbmo* **by** *blast*
then have *both-member-options ?newnode maxi*
by (*simp only*: *hlist*)
then have *both-member-options* (*treeList* ! *?h*) *maxi*

```
      using 0 ‹treeList ! high x n ∈ set treeList› dele-bmo-cont-corr by blast

    let ?newma = (if x = ma then
                                        ?h * 2^(deg div 2) + the( vebt-maxt (?newlist ! ?h))
                                    else ma)
    let ?delsimp = (Node (Some (mi, ?newma)) deg ?newlist summary)
    have dsimp:vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp
    using del-x-not-mi-newnode-not-nil[of mi x ma deg ?h ?l ?newnode treeList ?newlist summary]
     by (metis 12 2 9 False dual-order.eq-iff hlbound inrg order.not-eq-order-implies-strict xnotmi)
   have 111: (∀ i < 2^m. (∃ x. both-member-options (?newlist ! i) x) ⟷ ( both-member-options
summary i))
    proof
      fix i
      show  i < 2^m ⟶ ((∃ x. both-member-options (?newlist ! i) x) = ( both-member-options
summary i))
      proof
        assume i < 2^m
        show (∃ x. both-member-options (?newlist ! i) x) = ( both-member-options summary i)
        proof(cases i = ?h)
          case True
          hence 1000:?newlist ! i = ?newnode
            using hlist by blast
          hence  1001:∃ x. vebt-member (?newlist ! i) x
            using nnvalid notemp valid-member-both-member-options by auto
          hence 1002: ∃ x. both-member-options (?newlist ! i) x
            using 1000 notemp by presburger
          have 1003: both-member-options summary i
        using 0 1000 1002 4 True ‹i < 2 ^ m› ‹treeList ! high x n ∈ set treeList› dele-bmo-cont-corr
by fastforce
          then show ?thesis
            using 1002 by blast
        next
          case False
          hence 1000:?newlist ! i = treeList ! i
            using ‹i < 2 ^ m› nothlist by blast
          then show ?thesis
            using 4 ‹i < 2 ^ m› by presburger
        qed
      qed
    qed
    have 112: (mi = ?newma ⟶ (∀ t ∈ set ?newlist. ∄ x. both-member-options t x))
    proof
      assume aampt:mi = ?newma
      show (∀ t ∈ set ?newlist. ∄ y. both-member-options t y)
      proof(cases x = ma)
        case True
        hence False
          using ‹both-member-options (treeList ! ?h) maxi›
         by (metis 9 True ‹both-member-options ?newnode maxi› ‹vebt-maxt ( ?newlist ! high x n)
```

= Some maxi› aampt option.sel high-inv hlbound low-inv member-bound nnvalid not-less-iff-gr-or-eq valid-member-both-member-options yhelper)
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
using ‹*mi* ≠ *ma* ∧ *x* < 2 ^ *deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma* < 2^*deg* ∧ *mi* ≤ *?newma*
proof(*cases x* = *ma*)
case *True*
hence *x* = *ma* **by** *simp*
hence *maxi* < 2^*n*
using ‹*both-member-options?newnode maxi*› *member-bound nnvalid valid-member-both-member-options* **by** *blast*
show *?thesis*
by (*smt* (*verit*) *3 9 div-eq-0-iff True* ‹*both-member-options* (*treeList* ! *high x n*) *maxi*› ‹*maxi* < 2 ^ *n*› ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *add.right-neutral div-exp-eq div-mult-self3 option.sel high-inv hlbound le-0-eq less-imp-le-nat low-inv power-not-zero rel-simps*(*28*) *yhelper*)
next
case *False*
then show *?thesis*
using *6* **by** *auto*
qed
have *115*: *mi* ≠ *?newma* ⟶
(∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*))
proof
assume *mi* ≠ *?newma*
show (∀ *i* < 2^*m*.
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*))
proof
fix *i*
show *i* < 2^*m* ⟶
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof
assume *assumption*:*i* < 2^*m*
show (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi* < *y* ∧ *y* ≤ *?newma*)
proof−
have (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*))

proof
assume *newmaassm*: *high ?newma n = i*
thus *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)
proof(*cases x = ma*)
case *True*
then show *?thesis*
using ‹*both-member-options* (*?newlist* ! *?h*) *maxi*›
by (*smt* (*verit*) *9 True* ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *option.sel high-inv hlist low-inv maxt-member member-bound newmaassm nnvalid*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption bit-split-inv dele-bmo-cont-corr hlist newmaassm nothlist*)
qed
qed
moreover have (∀ *y*. (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi < y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *mi < y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *mi < y*
proof(*cases i = ?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *dele-bmo-cont-corr hlist yassm* **by** *auto*
then show *?thesis*
by (*simp add*: *assumption yassm yhelper*)
next
case *False*
then show *?thesis*
using *assumption nothlist yassm yhelper* **by** *presburger*
qed
moreover have *y* ≤ *?newma*
proof(*cases x = ma*)
case *True*
hence *x*= *ma* **by** *simp*
have *high y n* ≤ *?h*
by (*metis 7b True assumption div-le-mono high-def nothlist yassm*)
then show *?thesis*
proof(*cases high y n = ?h*)
case *True*
have *low y n > maxi* ⟹ *False*
by (*metis True* ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› *hlist leD maxt-corr-help nnvalid valid-member-both-member-options yassm*)
then show *?thesis*

by (*smt* (*verit*) *9 True* ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› ‹*x* = *ma*› *add-le-cancel-left div-mult-mod-eq option.sel high-def low-def nat-le-linear nat-less-le*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *9 True* ‹*both-member-options* (*?newlist* ! *high x n*) *maxi*› ‹*high y n* ≤ *high x n*› ‹*vebt-maxt* (*?newlist* ! *high x n*) = *Some maxi*› *div-le-mono option.sel high-def high-inv hlist le-antisym member-bound nat-le-linear nnvalid valid-member-both-member-options*)
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *0* ‹*treeList* ! *high x n* ∈ *set treeList*› *assumption dele-bmo-cont-corr hlist nothlist yassm yhelper*)
qed
ultimately show *mi* < *y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* < *2*^*deg* **and** *118*: *mi* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*mi*, *?newma*)) *deg ?newlist summary*) *deg*
using *invar-vebt.intros*(*5*)[*of ?newlist n summary m deg mi ?newma*] *allvalidinlist 1 newlistlength 8 3 111 112 117 118 115* **by** *fastforce*
then show *?thesis*
using *dsimp* **by** *presburger*
qed
next
case *False*
hence *xmi*:*x* = *mi* **by** *simp*
have *both-member-options summary* (*high ma n*)
by (*metis 1 11 3 4 6 One-nat-def Suc-le-eq* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› *deg-not-0 exp-split-high-low*(*1*))
hence *vebt-member summary* (*high ma n*)
using *5.hyps*(*1*) *valid-member-both-member-options* **by** *blast*
obtain *summin* **where** *Some summin* = *vebt-mint summary*
by (*metis 5.hyps*(*1*) ‹*vebt-member summary* (*high ma n*)› *empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def*)
hence ∃ *z* . *both-member-options* (*treeList* ! *summin*) *z*
by (*metis 5.hyps*(*1*) *5.hyps*(*5*) *both-member-options-equiv-member member-bound mint-member*)
moreover have *invar-vebt* (*treeList* ! *summin*) *n*
by (*metis 0 1 2* ‹*Some summin* = *vebt-mint summary*› *member-bound mint-member nth-mem*)
ultimately obtain *lx* **where** *Some lx* = *vebt-mint* (*treeList* ! *summin*)
by (*metis empty-Collect-eq mint-corr-help-empty not-None-eq set-vebt'-def valid-member-both-member-options*)
let *?xn* = *summin*∗*2*^*n* + *lx*
have *?xn* = (*if x* = *mi*

then the (*vebt-mint summary*) ∗ 2^(*deg div 2*)
+ *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))
else x)
by (*metis False* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*Some summin* = *vebt-mint summary*› ‹*deg div 2* = *n*› *option.sel*)
have *vebt-member* (*treeList* ! *summin*) *lx*
using ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *mint-member* **by** *auto*
moreover have *summin* < 2^*m*
by (*metis 5.hyps*(*1*) ‹*Some summin* = *vebt-mint summary*› *member-bound mint-member*)
ultimately have *xnin*: *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *?xn*
by (*metis 12 2 9* ‹*invar-vebt* (*treeList* ! *summin*) *n*› *add-leD1 both-member-options-equiv-member both-member-options-from-chilf-to-complete-tree high-inv low-inv member-bound numeral-2-eq-2 plus-1-eq-Suc*)
let *?h* =*high ?xn n*
let *?l* = *low ?xn n*
have *?xn* < 2^*deg*
by (*smt* (*verit*, *ccfv-SIG*) *5.hyps*(*1*) *5.hyps*(*4*) *div-eq-0-iff* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*Some summin* = *vebt-mint summary*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *div-exp-eq high-def high-inv le-0-eq member-bound mint-member not-numeral-le-zero power-not-zero*)
hence *?h* < *length treeList*
using *2* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*summin* < *2* ^ *m*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
let *?newnode* = *vebt-delete* (*treeList* ! *?h*) *?l*
let *?newlist* = *treeList*[*?h*:= *?newnode*]
have *length treeList* = *length ?newlist* **by** *auto*
hence *hprolist*: *?newlist* ! *?h* = *?newnode*
by (*meson* ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› *nth-list-update-eq*)
have *nothprolist*: *i* ≠ *?h* ∧ *i* < 2^*m* ⟹ *?newlist* ! *i* = *treeList* ! *i* **for** *i* **by** *auto*
have *hlbound*:*?h* < 2^*m* ∧ *?l* < 2^*n*
using *2* ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *low-inv member-bound* **by** *presburger*
hence *nnvalid*: *invar-vebt ?newnode n*
using *5.IH*(*1*) ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› **by** *simp*
have *allvalidinlist*:∀ *t* ∈ *set ?newlist*. *invar-vebt t n*
proof
fix *t*
assume *t* ∈ *set ?newlist*
then obtain *i* **where** *i* < 2^*m* ∧ *?newlist* ! *i* = *t*
by (*metis 2 in-set-conv-nth length-list-update*)
then show *invar-vebt t n*
by (*metis 0 2 hprolist nnvalid nth-list-update-neq nth-mem*)
qed
have *newlistlength*: *length ?newlist* = 2^*m*
by (*simp add*: *2*)
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
hence *ninNullc*: *minNull ?newnode* **by** *simp*

let *?sn = vebt-delete summary ?h*
let *?newma= (if ?xn = ma then (let maxs = vebt-maxt ?sn in*
(if maxs = None
then ?xn
*else 2^(deg div 2) * the maxs*
+ the (vebt-maxt (?newlist ! the maxs))
)
)
else ma)
let *?delsimp = (Node (Some (?xn, ?newma)) deg ?newlist ?sn)*
have *dsimp:vebt-delete (Node (Some (mi, ma)) deg treeList summary) x = ?delsimp*
using *del-x-mi-lets-in-minNull[of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist ?sn]*
by *(metis 12 9 ‹high (summin * 2 ^ n + lx) n < length treeList› ‹summin * 2 ^ n + lx = (if x = mi then the (vebt-mint summary) * 2 ^ (deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary))) else x)› ‹x = mi› ‹x ≠ mi ∨ x ≠ ma› inrg nat-less-le ninNullc)*
have *newsummvalid: invar-vebt ?sn m*
by *(simp add: 5.IH(2))*
have *111: (∀ i < 2^m. (∃ x. both-member-options (?newlist ! i) x) ⟷ (both-member-options ?sn i))*
proof
fix *i*
show *i < 2^m ⟶ ((∃ x. both-member-options (?newlist ! i) x) = (both-member-options ?sn i))*
proof
assume *i < 2^m*
show *(∃ x. both-member-options (?newlist ! i) x) = (both-member-options ?sn i)*
proof(*cases i = ?h*)
case *True*
hence *1000:?newlist ! i = ?newnode*
using *hprolist* **by** *fastforce*
hence *1001:∄ x. vebt-member (?newlist ! i) x*
by *(simp add: min-Null-member ninNullc)*
hence *1002: ∄ x. both-member-options (?newlist ! i) x*
using *1000 nnvalid valid-member-both-member-options* **by** *auto*
have *1003: ¬ both-member-options ?sn i*
using *1 True dele-bmo-cont-corr* **by** *auto*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000:?newlist ! i = treeList ! i*
using *‹i < 2 ^ m› nothprolist* **by** *blast*
hence *both-member-options (?newlist ! i) y ⟹ both-member-options ?sn i* **for** *y*
using *1 4 False ‹i < 2 ^ m› dele-bmo-cont-corr* **by** *auto*
moreover have *both-member-options ?sn i ⟹ ∃ y. both-member-options (?newlist ! i) y*
proof−
assume *both-member-options ?sn i*

```
            hence both-member-options summary i
              using 1 dele-bmo-cont-corr by auto
            thus ∃ y. both-member-options (?newlist ! i) y
              using 1000 4 ‹i < 2 ^ m› by presburger
          qed
          then show ?thesis
            using calculation by blast
        qed
      qed
    qed
    have 112: (?xn = ?newma ⟶ (∀ t ∈ set ?newlist. ∄ x. both-member-options t x))
    proof
      assume aampt:?xn = ?newma
      show (∀ t ∈ set ?newlist. ∄ y. both-member-options t y)
      proof(cases ?xn = ma)
        case True
        let ?maxs = vebt-maxt ?sn
        show ?thesis
        proof(cases ?maxs = None)
          case True
          hence aa:∄ y. vebt-member ?sn y
            using maxt-corr-help-empty newsummvalid set-vebt'-def by auto
          hence ∄ y. both-member-options ?sn y
            using newsummvalid valid-member-both-member-options by blast
          hence t ∈ set ?newlist ⟹ ∄y. both-member-options t y for t
          proof−
            assume t ∈ set ?newlist
            then obtain i where ?newlist ! i = t ∧ i< 2^m
                by (metis 2 ‹length treeList = length (treeList [high (summin * 2 ^ n + lx) n :=
vebt-delete (treeList ! high (summin * 2 ^ n + lx) n) (low (summin * 2 ^ n + lx) n)])› in-set-conv-nth)
            thus ∄y. both-member-options t y
              using 111 ‹∄y. both-member-options (vebt-delete summary (high (summin * 2 ^ n +
lx) n)) y› by blast
          qed
          then show ?thesis by blast
        next
          case False
          then obtain maxs where Some maxs = ?maxs
            by fastforce
          hence both-member-options summary maxs
            by (metis 1 dele-bmo-cont-corr maxbmo)
          have bb:maxs ≠ ?h ∧ maxs < 2^m
            by (metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo member-bound
valid-member-both-member-options)
          hence invar-vebt (?newlist ! maxs) nusing 0
            by (simp add: 2 allvalidinlist)
          hence ∃ y. both-member-options (?newlist ! maxs) y
            using 4 bb ‹both-member-options summary maxs› nothprolist by presburger
          then obtain maxi where Some maxi = vebt-maxt (?newlist ! maxs)
```

by (*metis*
‹*invar-vebt* (*treeList* [*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)] ! *maxs*) *n*›
both-member-options-equiv-member buildup-gives-empty buildup-gives-valid buildup-nothing-in-leaf
buildup-nothing-in-min-max deg-not-0 maxt-corr-help-empty mem-Collect-eq
member-valid-both-member-options option.exhaust set-vebt′-def)
hence *maxs* = *high ?xn n* ∧ *both-member-options* (*?newlist* ! *maxs*) (*low ?xn n*)
by (*smt* (*verit*) *9 False True* ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary ?h*)› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *aampt option.sel high-inv low-inv maxbmo maxt-member member-bound mult.commute*)
hence *False*
using *bb* **by** *blast*
then show *?thesis* **by** *simp*
qed
next
case *False*
hence *?xn* ≠ *?newma* **by** *simp*
hence *False* **using** *aampt* **by** *simp*
then show *?thesis* **by** *blast*
qed
qed
have *114*: *?newma* < *2*^*deg* ∧ *?xn* ≤ *?newma*
proof(*cases ?xn* = *ma*)
case *True*
hence *?xn* = *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
then show *?thesis*
using *5.hyps*(*8*) ‹*?xn* = *ma*› **by** *force*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0* **by** (*simp add*: *2 allvalidinlist*)
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4* ‹*both-member-options summary maxs*› *bb nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
using ‹*invar-vebt* (*treeList* [*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)] ! *maxs*) *n*› *maxt-corr-help-empty set-vebt′-def valid-member-both-member-options* **by** *fastforce*
hence *abc*:*?newma* = *2*^*n* ∗ *maxs* + *maxi*

by (*smt* (*verit*) *9 True* ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin* ∗ *2* ^ *n* + *lx*) *n*))› *option.sel not-None-eq*)
have *abd*:*maxi* < *2*^*n*
by (*metis* ‹*Some maxi = vebt-maxt* (*?newlist* ! *maxs*)› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *maxt-member member-bound*)
have *high ?xn n* ≤ *maxs*
using *1* ‹*Some summin = vebt-mint summary*› ‹*both-member-options summary maxs*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound mint-corr-help valid-member-both-member-options* **by** *force*
then show *?thesis*
proof(*cases high ?xn n = maxs*)
case *True*
then show *?thesis*
using *bb* **by** *fastforce*
next
case *False*
hence *high ?xn n < maxs*
by (*simp add*: ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* ≤ *maxs*› *order.not-eq-order-implies-strict*)
hence *?newma* < *2*^*deg*
by (*smt* (*verit*) *5.hyps*(*8*) *9* ‹*Some maxi = vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin* ∗ *2* ^ *n* + *lx*) *n*))› ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *abd bb both-member-options-equiv-member option.sel high-inv less-le-trans low-inv maxt-member mult.commute nothprolist alethe-comp-simplify1*(*3*) *yhelper*)
moreover have *high ?xn n < high ?newma n*
by (*smt* (*verit*) *9 True* ‹*Some maxi = vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs = vebt-maxt* (*vebt-delete summary* (*high* (*summin* ∗ *2* ^ *n* + *lx*) *n*))› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *maxs*› *abd option.sel high-inv mult.commute option.discI*)
ultimately show *?thesis*
by (*metis div-le-mono high-def linear not-less*)
qed
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *12 5.hyps*(*7*) *5.hyps*(*8*) *9 both-member-options-from-complete-tree-to-child dual-order.trans hlbound one-le-numeral xnin yhelper*)
qed
have *115*: *?xn* ≠ *?newma* ⟶
(∀ *i* < *2*^*m*.
(*high ?newma n = i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*))
proof
assume *assumption0*:*?xn* ≠ *?newma*
show (∀ *i* < *2*^*m*.
(*high ?newma n = i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n = i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*))
proof

fix *i*
show *i* < 2^*m* ⟶
(*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*)
proof
assume *assumption*:*i* < 2^*m*
show (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)) ∧
(∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*)
proof−
have (*high ?newma n* = *i* ⟶ *both-member-options* (*?newlist* ! *i*) (*low ?newma n*))
proof
assume *newmaassm*: *high ?newma n* = *i*
thus *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)
proof(*cases ?xn* = *ma*)
case *True*
hence *bb*:*?xn* = *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
hence *?newma* = *?xn* **using** *assumption Let-def bb* **by** *simp*
hence *False* **using** *assumption0* **by** *simp*
then show *?thesis* **by** *simp*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < 2^*m*
by (*metis 1 ‹Some maxs = vebt-maxt ?sn› dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0* **by** (*simp add*: *2 allvalidinlist*)
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4 ‹both-member-options summary maxs› bb nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
using ‹*invar-vebt* (*treeList* [*high* (*summin* ∗ 2 ^ *n* + *lx*) *n* := *vebt-delete* (*treeList* ! *high* (*summin* ∗ 2 ^ *n* + *lx*) *n*) (*low* (*summin* ∗ 2 ^ *n* + *lx*) *n*)] ! *maxs*) *n*› *maxt-corr-help-empty set-vebt'-def valid-member-both-member-options* **by** *fastforce*
then show *?thesis*
by (*metis 1 10 9 True ‹Some summin = vebt-mint summary› ‹both-member-options summary maxs› ‹vebt-member* (*treeList* ! *summin*) *lx› ‹mi* ≠ *ma* ∧ *x* < 2 ^ *deg› ‹invar-vebt* (*treeList* ! *summin*) *n› bb equals0D high-inv le-antisym maxt-corr-help maxt-corr-help-empty mem-Collect-eq member-bound mint-corr-help option.collapse summaxma set-vebt'-def valid-member-both-member-options*)

qed
next

case *False*
hence *ccc*:*?newma* = *ma* **by** *simp*
then show *?thesis*
proof(*cases ?xn* = *ma*)
case *True*
hence *?xn* = *?newma*
using *False* **by** *blast*
hence *False*
using *False* **by** *auto*
then show *?thesis* **by** *simp*
next
case *False*
hence *both-member-options* (*?newlist* ! *high ma n*) (*low ma n*)
by (*metis 1* ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*vebt-member summary* (*high ma n*)› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *bit-split-inv dele-bmo-cont-corr high-inv hprolist member-bound nothprolist*)
moreover have *high ma n* = *i* ∧ *low ma n* = *low ?newma n* **using** *ccc newmaassm* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
moreover have (∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *?xn* < *y*
proof(*cases i* = *?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *auto*
then show *?thesis*
using *True hprolist min-Null-member ninNullc nnvalid valid-member-both-member-options yassm* **by** *fastforce*
next
case *False*
hence *i* ≤ *?h* ⟹ *False*
by (*metis 1 111* ‹*Some summin* = *vebt-mint summary*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption dele-bmo-cont-corr high-inv le-antisym member-bound mint-corr-help valid-member-both-member-options yassm*)
hence *i* > *?h*
using *leI* **by** *blast*
then show *?thesis*
by (*metis div-le-mono high-def not-less yassm*)

qed
moreover have *y* ≤ *?newma*
proof(*cases ?xn* = *ma*)
case *True*
hence *?xn*= *ma* **by** *simp*
let *?maxs* = *vebt-maxt ?sn*
show *?thesis*
proof(*cases ?maxs* = *None*)
case *True*
then show *?thesis*
using *1 111 assumption dele-bmo-cont-corr nothprolist yassm yhelper* **by** *auto*
next
case *False*
then obtain *maxs* **where** *Some maxs* = *?maxs*
by *fastforce*
hence *both-member-options summary maxs*
by (*metis 1 dele-bmo-cont-corr maxbmo*)
have *bb*:*maxs* ≠ *?h* ∧ *maxs* < *2*^*m*
by (*metis 1* ‹*Some maxs* = *vebt-maxt ?sn*› *dele-bmo-cont-corr maxbmo member-bound valid-member-both-member-options*)
hence *invar-vebt* (*?newlist* ! *maxs*) *n***using** *0* **by** (*simp add*: *2 allvalidinlist*)
hence ∃ *y*. *both-member-options* (*?newlist* ! *maxs*) *y*
using *4* ‹*both-member-options summary maxs*› *bb nothprolist* **by** *presburger*
then obtain *maxi* **where** *Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)
by (*metis True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption calculation dele-bmo-cont-corr high-inv hprolist leD member-bound nth-list-update-neq yassm yhelper*)
hence *maxs* < *2*^*m* ∧ *maxi* < *2*^*n*
by (*metis* ‹*invar-vebt* (*?newlist* ! *maxs*) *n*› *bb maxt-member member-bound*)
hence *?newma* = *2*^*n*∗ *maxs* + *maxi*
by (*smt* (*verit*) *9 False True* ‹*Some maxi* = *vebt-maxt* (*?newlist* ! *maxs*)› ‹*Some maxs* = *vebt-maxt* (*vebt-delete summary* (*high ?xn n*))› *option.sel*)
hence *low ?newma n* = *maxi* ∧ *high ?newma n* = *maxs*
by (*simp add*: ‹*maxs* < *2* ^ *m* ∧ *maxi* < *2* ^ *n*› *high-inv low-inv mult.commute*)
hence *both-member-options* (*treeList* ! (*high y n*)) (*low y n*)
by (*metis 1 111 assumption dele-bmo-cont-corr nothprolist yassm*)
hence *hleqdraft*:*high y n* > *maxs* ⟹ *False*
proof−
assume *high y n* > *maxs*
have *both-member-options summary* (*high y n*)
using *1 111 assumption dele-bmo-cont-corr yassm* **by** *blast*
moreover have *both-member-options ?sn* (*high y n*)
using *111 assumption yassm* **by** *blast*
ultimately show *False*
using *True* ‹*both-member-options* (*treeList* ! *high y n*) (*low y n*)› ‹*summin* ∗ *2* ^ *n* + *lx* < *y*› *assumption leD yassm yhelper* **by** *blast*
qed
hence *hleqmaxs*: *high y n* ≤ *maxs* **by** *presburger*
then show *?thesis*

using ‹*both-member-options* (*treeList* ! *high y n*) (*low y n*)› *assumption calculation dual-order.strict-trans1 yassm yhelper* **by** *auto*
qed
next
case *False*
then show *?thesis*
by (*smt* (*verit*) ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption dele-bmo-cont-corr high-inv hprolist member-bound nothprolist yassm yhelper*)
qed
ultimately show *?xn* < *y* ∧ *y* ≤ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* < *2^deg* **and** *118*: *?xn* ≤ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist ?sn*) *deg*
using *invar-vebt.intros*(*5*)[*of ?newlist n ?sn m deg ?xn ?newma*]
using *3 allvalidinlist newlistlength newsummvalid 5.hyps*(*3*) *111 112 118 117 115* **by** *fastforce*
show *?thesis*
using *116 dsimp* **by** *presburger*
next
case *False*
hence *notemp*:∃ *z*. *both-member-options ?newnode z*
using *not-min-Null-member* **by** *auto*
let *?newma* = (*if ?xn* = *ma then*
?h ∗ *2^*(*deg div 2*) + *the*(*vebt-maxt* (*?newlist* ! *?h*))
else ma)
let *?delsimp* = (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist summary*)
have *dsimp*:*vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* = *?delsimp*
using *del-x-mi-lets-in-not-minNull*[*of x mi ma deg ?xn ?h summary treeList ?l ?newnode ?newlist*]
12 2 9 False dual-order.eq-iff hlbound inrg order.not-eq-order-implies-strict xmi
by (*metis* ‹*summin* ∗ *2* ^ *n* + *lx* = (*if x* = *mi then the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*))) *else x*)› ‹*x* ≠ *mi* ∨ *x* ≠ *ma*›)
have *111*: (∀ *i* < *2^m*. (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) ⟷ (*both-member-options summary i*))
proof
fix *i*
show *i* < *2^m* ⟶ ((∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*))
proof
assume *i* < *2^m*
show (∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*) = (*both-member-options summary i*)
proof(*cases i* = *?h*)
case *True*

hence *1000*:*?newlist* ! *i* = *?newnode*
using *hprolist* **by** *blast*
hence *1001*:∃ *x*. *vebt-member* (*?newlist* ! *i*) *x*
using *nnvalid notemp valid-member-both-member-options* **by** *auto*
hence *1002*: ∃ *x*. *both-member-options* (*?newlist* ! *i*) *x*
using *1000 notemp* **by** *presburger*
have *1003*: *both-member-options summary i*
using *4 True* ‹∃ *z*. *both-member-options* (*treeList* ! *summin*) *z*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*summin* < 2 ^ *m*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *auto*
then show *?thesis*
using *1002* **by** *blast*
next
case *False*
hence *1000*:*?newlist* ! *i* = *treeList* ! *i*
using ‹*i* < 2 ^ *m*› *nothprolist* **by** *blast*
then show *?thesis*
using *4* ‹*i* < 2 ^ *m*› **by** *presburger*
qed
qed
qed
have *112*: (*?xn* = *?newma* ⟶ (∀ *t* ∈ *set ?newlist*. ∄ *x*. *both-member-options t x*))
proof
assume *aampt*:*?xn* = *?newma*
show (∀ *t* ∈ *set ?newlist*. ∄ *y*. *both-member-options t y*)
proof(*cases ?xn* = *ma*)
case *True*
obtain *maxi* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
by (*metis Collect-empty-eq False hprolist maxt-corr-help-empty nnvalid not-None-eq not-min-Null-member set-vebt'-def valid-member-both-member-options*)
hence *both-member-options ?newnode maxi*
using *hprolist maxbmo* **by** *auto*
hence *both-member-options* (*treeList* ! *?h*) *maxi*
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv member-bound* **by** *force*
hence *False*
by (*metis 9* ‹*both-member-options* (*vebt-delete* (*treeList* ! *high* (*summin* * 2 ^ *n* + *lx*) *n*) (*low* (*summin* * 2 ^ *n* + *lx*) *n*)) *maxi*› ‹*vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *aampt add-diff-cancel-left' dele-bmo-cont-corr option.sel high-inv low-inv member-bound*)
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
using ‹*mi* ≠ *ma* ∧ *x* < 2 ^ *deg*› *aampt* **by** *presburger*
qed
qed
have *114*: *?newma* < 2^*deg* ∧ *?xn* ≤ *?newma*
proof(*cases ?xn* = *ma*)
case *True*

hence *?xn = ma* **by** *simp*
obtain *maxi* **where** *vebt-maxt (?newlist ! ?h) = Some maxi*
by (*metis 111 2 4 Collect-empty-eq True ‹both-member-options (treeList ! high ma n) (low ma n)› ‹high (summin * 2 ^ n + lx) n < length treeList› hprolist maxt-corr-help-empty nnvalid not-None-eq set-vebt'-def valid-member-both-member-options*)
hence *both-member-options ?newnode maxi*
using *hprolist maxbmo* **by** *auto*
hence *both-member-options (treeList ! ?h) maxi*
using ‹*vebt-member (treeList ! summin) lx*› ‹*invar-vebt (treeList ! summin) n*› *dele-bmo-cont-corr high-inv member-bound* **by** *force*
hence *maxi < 2^n*
using ‹*both-member-options?newnode maxi*› *member-bound nnvalid valid-member-both-member-options* **by** *blast*
show *?thesis*
by (*smt (verit, ccfv-threshold) 3 9 div-eq-0-iff True ‹Some lx = vebt-mint (treeList ! summin)› ‹both-member-options (treeList ! high (summin * 2 ^ n + lx) n) maxi› ‹vebt-maxt (?newlist ! high (summin * 2 ^ n + lx) n) = Some maxi› ‹vebt-member (treeList ! summin) lx› ‹invar-vebt (treeList ! summin) n› add.right-neutral add-left-mono div-mult2-eq div-mult-self3 option.sel high-inv hlbound le-0-eq member-bound mint-corr-help power-add power-not-zero rel-simps(28) valid-member-both-member-options*)
next
case *False*
then show *?thesis*
using *10 5.hyps(8) maxt-corr-help valid-member-both-member-options xnin* **by** *force*

qed
have *115*: *?xn ≠ ?newma ⟶*
(∀ i < 2^m.
(high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
(∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma))
proof
assume *xnmassm*:*?xn ≠ ?newma*
show *(∀ i < 2^m.*
(high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
(∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma))
proof
fix *i*
show *i < 2^m ⟶*
(high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧
(∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma)
proof
assume *assumption*:*i < 2^m*
show *(high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n)) ∧*
(∀ y. (high y n = i ∧ both-member-options (?newlist ! i) (low y n)) ⟶ ?xn < y ∧ y ≤ ?newma)
proof−
have *(high ?newma n = i ⟶ both-member-options (?newlist ! i) (low ?newma n))*

proof
assume *newmaassm*: *high ?newma n = i*
thus *both-member-options* (*?newlist* ! *i*) (*low ?newma n*)
proof(*cases ?xn = ma*)
case *True*
obtain *maxi* **where** *vebt-maxt* (*?newlist* ! *?h*) = *Some maxi*
by (*metis Collect-empty-eq both-member-options-equiv-member hprolist maxt-corr-help-empty nnvalid not-Some-eq notemp set-vebt'-def*)
hence *both-member-options* (*?newlist* ! *?h*) *maxi*
using *maxbmo* **by** *blast*
then show *?thesis*
by (*smt* (*verit*) *2 9 True* ‹*Some lx = vebt-mint* (*treeList* ! *summin*)› ‹*high* (*summin* ∗ *2* ^ *n* + *lx*) *n* < *length treeList*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *add-left-mono dele-bmo-cont-corr eq-iff high-inv hprolist low-inv member-bound mint-corr-help valid-member-both-member-options yhelper*)
next
case *False*
hence *abcd*:*?newma = ma* **by** *simp*
then show *?thesis*
proof(*cases high ma n = ?h*)
case *True*
hence *?newlist* ! *high ma n = ?newnode*
using *hprolist* **by** *presburger*
then show *?thesis*
proof(*cases low ma n = ?l*)
case *True*
hence *?newma = ?xn*
by (*metis 1 False* ‹*?newlist* ! *high ma n = vebt-delete* (*treeList* ! *high* (*summin* ∗ *2* ^ *n* + *lx*) *n*) (*low* (*summin* ∗ *2* ^ *n* + *lx*) *n*)› ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)›
‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*vebt-member summary* (*high ma n*)› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *bit-split-inv dele-bmo-cont-corr high-inv member-bound nothprolist*)
hence *False*
using *False* **by** *presburger*
then show *?thesis* **by** *simp*
next
case *False*
have *both-member-options* (*treeList* ! *high ma n*) (*low ma n*)
by (*simp add*: ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)›)
hence *both-member-options ?newnode* (*low ma n*)
using *False True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv member-bound* **by** *force*
hence *both-member-options* (*?newlist* ! *high ma n*) (*low ma n*)
using *True hprolist* **by** *presburger*
then show *?thesis* **using** *abcd newmaassm* **by** *simp*
qed
next
case *False*
hence *?newlist* ! *high ma n = treeList* ! *high ma n*

using *1* ‹*vebt-member summary* (*high ma n*)› *member-bound nothprolist* **by** *blast*
moreover hence *both-member-options* (*treeList* ! *high ma n*) (*low ma n*)
using ‹*both-member-options* (*treeList* ! *high ma n*) (*low ma n*)› **by** *blast*
ultimately show *?thesis* **using** *abcd newmaassm* **by** *simp*
qed
qed
qed
moreover have (∀ *y*. (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*)
proof
fix *y*
show (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*)) ⟶ *?xn* < *y* ∧ *y* ≤ *?newma*
proof
assume *yassm*: (*high y n* = *i* ∧ *both-member-options* (*?newlist* ! *i*) (*low y n*))
hence *?xn* < *y*
proof(*cases i* = *?h*)
case *True*
hence *both-member-options* (*treeList* ! *i*) (*low y n*)
using ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *force*
moreover have *vebt-mint* (*treeList* ! *i*) = *Some* (*low ?xn n*)
using *True* ‹*Some lx* = *vebt-mint* (*treeList* ! *summin*)› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv low-inv member-bound* **by** *presburger*
moreover hence *low y n* ≥ *low ?xn n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *calculation*(*1*) *high-inv member-bound mint-corr-help valid-member-both-member-options* **by** *auto*
moreover have *low y n* ≠ *low ?xn n*
using *True* ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *dele-bmo-cont-corr high-inv hprolist member-bound yassm* **by** *auto*
ultimately have *low y n* > *low ?xn n* **by** *simp*
show *?thesis*
by (*metis True* ‹*low* (*summin* ∗ *2* ^ *n* + *lx*) *n* ≤ *low y n*› ‹*low y n* ≠ *low* (*summin* ∗ *2* ^ *n* + *lx*) *n*› *bit-concat-def bit-split-inv leD linorder-neqE-nat nat-add-left-cancel-less yassm*)
next
case *False*
have *Some* (*high ?xn n*) = *vebt-mint summary*
using ‹*Some summin* = *vebt-mint summary*› ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *high-inv member-bound* **by** *presburger*
moreover hence *high y n* ≥ *high ?xn n*
by (*metis 1 111 assumption mint-corr-help valid-member-both-member-options yassm*)
ultimately show *?thesis*
by (*metis False div-le-mono high-def leI le-antisym yassm*)
qed
moreover have *y* ≤ *?newma*
by (*smt* (*verit*) ‹*vebt-member* (*treeList* ! *summin*) *lx*› ‹*invar-vebt* (*treeList* ! *summin*) *n*› *assumption calculation dele-bmo-cont-corr high-inv hprolist leD member-bound nothprolist yassm yhelper*)

ultimately show *?xn* $<$ *y* $\wedge$ *y* $\leq$ *?newma* **by** *simp*
qed
qed
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed
hence *117*: *?newma* $<$ *2^deg* **and** *118*: *?xn* $\leq$ *?newma* **using** *114* **by** *auto*
have *116*: *invar-vebt* (*Node* (*Some* (*?xn*, *?newma*)) *deg ?newlist summary*) *deg*
using *invar-vebt.intros*(*5*)[*of ?newlist n summary m deg ?xn ?newma*] *allvalidinlist*
1 newlistlength 8 3 111 112 117 118 115 **by** *fastforce*
hence *invar-vebt* (*?delsimp*) *deg* **by** *simp*
moreover obtain *delsimp* **where** *118*:*delsimp* = *?delsimp* **by** *simp*
ultimately have *119*:*invar-vebt delsimp deg* **by** *simp*
have *vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* = *delsimp* **using** *dsimp 118*
by *simp*
hence *delsimp* = *vebt-delete* (*Node* (*Some* (*x*, *ma*)) *deg treeList summary*) *x* **by** *simp*
then show *?thesis* **using** *119*
using *xmi* **by** *auto*
qed
qed
qed
qed
qed

corollary *dele-member-cont-corr*:*invar-vebt t n* $\Longrightarrow$ (*vebt-member* (*vebt-delete t x*) *y* $\longleftrightarrow$ *x* $\neq$ *y* $\wedge$ *vebt-member t y*)
by (*meson both-member-options-equiv-member dele-bmo-cont-corr delete-pres-valid*)

8.4 Correctness with Respect to Set Interpretation

theorem *delete-correct′*: **assumes** *invar-vebt t n*
shows *set-vebt′* (*vebt-delete t x*) = *set-vebt′ t* $-$ {*x*}
using *assms* **by**(*auto simp add*: *set-vebt′-def dele-member-cont-corr*)

corollary *delete-correct*: **assumes** *invar-vebt t n*
shows *set-vebt′* (*vebt-delete t x*) = *set-vebt t* $-$ {*x*}
using *assms delete-correct′ set-vebt-set-vebt′-valid* **by** *auto*

end
end

theory *VEBT-Uniqueness* **imports** *VEBT-InsertCorrectness VEBT-Succ VEBT-Pred VEBT-DeleteCorrectness*
begin

context *VEBT-internal* **begin**

9 Uniqueness Property of valid Trees

Two valid van Emde Boas trees having equal degree number and representing the same integer set are equal.

theorem *uniquetree*: *invar-vebt t n* $\Longrightarrow$ *invar-vebt s n* $\Longrightarrow$ *set-vebt' t = set-vebt' s* $\Longrightarrow$ *s = t*
proof(*induction t n arbitrary*: *s rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case*
by (*metis One-nat-def VEBT.inject*(*2*) *deg1Leaf member-correct n-not-Suc-n set-vebt-set-vebt'-valid vebt-member.simps*(*1*))
next
case (*2 treeList n summary m deg*)
from *2*(*9*) **obtain** *treeList' summary'* **where** *sprop*:*s = Node None deg treeList' summary'* $\wedge$ *deg = n+m*
$\wedge$ *length treeList' =2^m* $\wedge$ *invar-vebt summary' m* $\wedge$ ($\forall$ *t* $\in$ *set treeList'*. *invar-vebt t n*) $\wedge$
($\nexists$ *i*. *both-member-options summary' i*)
apply(*cases*)
using *2.hyps*(*3*) *2.hyps*(*4*) *one-is-add* **apply** *force*
apply (*metis 2.hyps*(*3*) *2.hyps*(*4*) *add-self-div-2*)
apply (*metis 2.hyps*(*3*) *2.hyps*(*4*) *One-nat-def add-self-div-2 div-greater-zero-iff even-Suc-div-two not-numeral-le-zero odd-add order.not-eq-order-implies-strict plus-1-eq-Suc zero-le-one zero-neq-one*)
apply (*metis 2.prems*(*1*) *2.prems*(*2*) *VEBT-Member.vebt-member.simps*(*2*) *Suc-1 add-leD1 add-self-div-2 both-member-options-def deg-not-0 div-greater-zero-iff empty-Collect-eq membermima.simps*(*4*) *nat-le-iff-add plus-1-eq-Suc set-vebt'-def valid-member-both-member-options*)
apply (*metis 2.hyps*(*3*) *2.hyps*(*4*) *add-self-div-2 div2-Suc-Suc even-Suc-div-two odd-add one-is-add plus-1-eq-Suc zero-neq-one*)
done
from *2*(*9*) **have** *aa*:$\forall$ *t* $\in$ *set treeList'*. *invar-vebt t n* **using** *sprop* **by** *simp*
have *ac*:*deg* $\geq$ *2*
by (*metis 2.hyps*(*3*) *add-self-div-2 deg-not-0 div-greater-zero-iff sprop*)
hence *ab*:$\forall$ *t* $\in$ *set treeList*. *set-vebt' t* = {}
by (*metis 2.hyps*(*6*) *empty-Collect-eq min-Null-member not-min-Null-member set-vebt'-def*)
hence *ac*:*length treeList' =length treeList*
by (*simp add*: *2.hyps*(*2*) *sprop*)
hence *membercongy*:*i* $<$ *2^m* $\Longrightarrow$ *vebt-member* (*treeList*! *i*) *x* $\longleftrightarrow$ *vebt-member* (*treeList'* ! *i*) *x* **for** *i x*
proof−
assume *i* $<$ *2^m*
show *vebt-member* (*treeList*! *i*) *x* $\longleftrightarrow$ *vebt-member* (*treeList'* ! *i*) *x*
proof
show *vebt-member* (*treeList* ! *i*) *x* $\Longrightarrow$ *vebt-member* (*treeList'* ! *i*) *x*
by (*metis 2.hyps*(*6*) ‹*i* $<$ *2* ^ *m*› *ac min-Null-member not-min-Null-member nth-mem sprop*)
show *vebt-member* (*treeList'* ! *i*) *x* $\Longrightarrow$ *vebt-member* (*treeList* ! *i*) *x*
proof−
assume *vebt-member* (*treeList'* ! *i*) *x*
hence *both-member-options* (*treeList'* ! *i*) *x*
by (*metis* ‹*i* $<$ *2* ^ *m*› *both-member-options-equiv-member nth-mem sprop*)
hence *membermima* (*treeList'*! *i*) *x* $\vee$ *naive-member* (*treeList'*! *i*) *x* **unfolding** *both-member-options-def*

by *auto*
moreover have *membermima (treeList' ! i) x ⟹ membermima s (2^m*i+x)*
using *membermima.simps(5)[of deg−1 treeList' summary' (2^m*i+x)] sprop ac*
apply *auto*
apply *(metis One-nat-def Suc-diff-1 ‹membermima (Node None (Suc (deg − 1)) treeList' summary') (2 ^ m * i + x) = (let pos = high (2 ^ m * i + x) (Suc (deg − 1) div 2) in if pos < length treeList' then membermima (treeList' ! pos) (low (2 ^ m * i + x) (Suc (deg − 1) div 2)) else False)› add.commute deg-not-0 neq0-conv not-add-less1)*
by *(smt (verit) 2.hyps(3) Nat.add-0-right Suc-pred ‹i < 2 ^ m› ‹vebt-member (treeList' ! i) x› add-gr-0 add-self-div-2 deg-not-0 div-less div-mult-self4 high-def low-inv member-bound mult.commute nth-mem power-not-zero zero-neq-numeral)*
moreover have *naive-member (treeList' ! i) x ⟹ naive-member s (2^m*i+x)*
using *naive-member.simps(3)[of None deg−1 treeList' summary' (2^m*i+x)] sprop ac*
apply *auto*
apply *(metis One-nat-def Suc-pred' ‹naive-member (Node None (Suc (deg − 1)) treeList' summary') (2 ^ m * i + x) = (let pos = high (2 ^ m * i + x) (Suc (deg − 1) div 2) in if pos < length treeList' then naive-member (treeList' ! pos) (low (2 ^ m * i + x) (Suc (deg − 1) div 2)) else False)› add-gr-0 deg-not-0)*
by *(smt (verit) 2.hyps(3) Nat.add-0-right Suc-pred ‹i < 2 ^ m› ‹vebt-member (treeList' ! i) x› add-gr-0 add-self-div-2 deg-not-0 div-less div-mult-self4 high-def low-inv member-bound mult.commute nth-mem power-not-zero zero-neq-numeral)*
ultimately have *both-member-options s (2^m*i +x)* **unfolding** *both-member-options-def* **by** *auto*
hence *False*
using *2.prems(1) VEBT-Member.vebt-member.simps(2) sprop valid-member-both-member-options*
by *blast*
then show *?thesis* **by** *simp*
qed
qed
qed
hence *ad:i<2^m ⟹ set-vebt' (treeList' ! i) = {}* **for** *i*
proof−
assume *assm:i < 2^m*
show *set-vebt' (treeList' ! i) = {}*
proof(*rule ccontr*)
assume *set-vebt' (treeList' ! i) ≠ {}*
then obtain *y* **where** *vebt-member (treeList' ! i) y*
using *set-vebt'-def* **by** *fastforce*
thus *False*
using *ab ac assm membercongy sprop set-vebt'-def* **by** *force*
qed
qed
hence *ae:i< 2^m ⟹ treeList' ! i = treeList ! i* **for** *i*
by *(simp add: 2.IH(1) 2.hyps(2) ab sprop)*
then show *?case*
by *(metis 2.IH(2) 2.hyps(1) 2.hyps(5) ac both-member-options-equiv-member empty-Collect-eq list-eq-iff-nth-eq sprop set-vebt'-def)*
next
case *(3 treeList n summary m deg)*

from *3(9)* **obtain** *treeList' summary'* **where** *sprop:s = Node None deg treeList' summary'* ∧ *deg = n+m*
∧ *length treeList' =2^m* ∧ *invar-vebt summary' m* ∧ (∀ *t* ∈ *set treeList'. invar-vebt t n)* ∧
(∄ *i. both-member-options summary' i*)
apply(*cases*)
apply (*metis 3.IH(1) 3.hyps(2) 3.hyps(3) 3.hyps(4) One-nat-def Suc-1 not-one-le-zero one-is-add set-n-deg-not-0 zero-neq-numeral*)
apply (*metis 3.hyps(3) 3.hyps(4) add-self-div-2 div2-Suc-Suc even-Suc-div-two odd-add plus-1-eq-Suc*)
apply (*metis 3.hyps(3) 3.hyps(4) Suc-inject add-Suc-right add-self-div-2*)
apply (*metis 3.hyps(3) 3.hyps(4) add-Suc-right add-self-div-2 even-Suc-div-two le-add2 le-less-Suc-eq odd-add order.strict-iff-order plus-1-eq-Suc*)
apply (*metis 3.prems(1) 3.prems(2) VEBT-Member.vebt-member.simps(2) Suc-pred' both-member-options-def deg-not-0 mem-Collect-eq membermima.simps(4) set-vebt'-def valid-member-both-member-options*)
done
have *ac:deg* ≥ *2*
by (*metis 3.hyps(3) One-nat-def add-le-mono le-add1 numeral-2-eq-2 plus-1-eq-Suc set-n-deg-not-0 sprop*)
hence *ab:*∀ *t* ∈ *set treeList. set-vebt' t = {}*
by (*metis 3.hyps(6) empty-Collect-eq min-Null-member not-min-Null-member set-vebt'-def*)
hence *ac:length treeList' =length treeList*
by (*simp add*: *3.hyps(2) sprop*)
hence *membercongy:i < 2^m* ⟹ *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x* **for** *i x*
proof−
assume *i < 2^m*
show *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x*
proof
show *vebt-member* (*treeList* ! *i*) *x* ⟹ *vebt-member* (*treeList'* ! *i*) *x*
by (*metis 3.hyps(6)* ‹*i < 2 ^ m*› *ac min-Null-member not-min-Null-member nth-mem sprop*)
show *vebt-member* (*treeList'* ! *i*) *x* ⟹ *vebt-member* (*treeList* ! *i*) *x*
proof−
assume *vebt-member* (*treeList'* ! *i*) *x*
hence *both-member-options* (*treeList'* ! *i*) *x*
by (*metis* ‹*i < 2 ^ m*› *both-member-options-equiv-member nth-mem sprop*)
hence *membermima* (*treeList'* ! *i*) *x* ∨ *naive-member* (*treeList'* ! *i*) *x*
unfolding *both-member-options-def* **by** *auto*
moreover have *membermima* (*treeList'* ! *i*) *x* ⟹ *membermima s* (*2^n∗i+x*)
using *membermima.simps(5)*[*of deg−1 treeList' summary'* (*2^n∗i+x*)] *sprop ac*
by (*smt* (*verit*) *3.hyps(3) 3.prems(1) Nat.add-diff-assoc Suc-pred* ‹*i < 2 ^ m*› ‹*vebt-member* (*treeList'* ! *i*) *x*› *add-diff-cancel-left' add-self-div-2 deg-not-0 even-Suc high-inv le-add1 low-inv member-bound mult.commute mult-2 nth-mem odd-two-times-div-two-nat plus-1-eq-Suc*)
moreover have *naive-member* (*treeList'* ! *i*) *x* ⟹ *naive-member s* (*2^n∗i+x*)
using *naive-member.simps(3)*[*of None deg−1 treeList' summary'* (*2^n∗i+x*)] *sprop ac*
by (*smt* (*verit*) *3.hyps(3) 3.prems(1) Nat.add-0-right Nat.add-diff-assoc Suc-pred* ‹*i < 2 ^ m*› ‹*vebt-member* (*treeList'* ! *i*) *x*› *add-self-div-2 deg-not-0 div-less div-mult-self4 even-Suc-div-two high-def le-add1 low-inv member-bound mult.commute nth-mem odd-add plus-1-eq-Suc power-not-zero zero-neq-numeral*)
ultimately have *both-member-options s* (*2^n∗i +x*) **unfolding** *both-member-options-def*

```
        by auto
       hence False
      using 3.prems(1) VEBT-Member.vebt-member.simps(2) sprop valid-member-both-member-options

        by blast
       then show ?thesis by simp
      qed
    qed
  qed
  hence ad:i<2^m ⟹ set-vebt' (treeList' ! i ) = {} for i
  proof−
    assume assm:i < 2^m
    show set-vebt' (treeList' ! i ) = {}
    proof(rule ccontr)
      assume set-vebt' (treeList' ! i ) ≠ {}
      then obtain y where vebt-member (treeList' ! i) y
        using set-vebt'-def by fastforce
      thus False
        using ab ac assm membercongy sprop set-vebt'-def by force
    qed
  qed
  hence ae:i< 2^m ⟹ treeList' ! i = treeList ! i  for i
    by (simp add: 3.IH(1) 3.hyps(2) ab sprop)
  then show ?case
     by (metis 3.IH(2) 3.hyps(1) 3.hyps(5) Collect-empty-eq ac both-member-options-equiv-member
list-eq-iff-nth-eq sprop set-vebt'-def)
next
  case (4 treeList n summary m deg mi ma)
  note case4= this
  hence set-vebt' (Node (Some (mi, ma)) deg treeList summary) = set-vebt' s by simp
  hence a0:deg ≥ 2 using 4
    by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  hence aa:{mi, ma} ⊆ set-vebt' (Node (Some (mi, ma)) deg treeList summary)
    apply auto using vebt-member.simps(5)[of mi ma deg −2 treeList summary mi]
    apply (metis add-2-eq-Suc' le-add-diff-inverse2 mem-Collect-eq set-vebt'-def)
    using vebt-member.simps(5)[of mi ma deg −2 treeList summary ma]
    apply (metis add-2-eq-Suc' le-add-diff-inverse2 mem-Collect-eq set-vebt'-def)
    done
  from 4(12) obtain treeList' summary' info where sprop1:s = Node info deg treeList' summary' ∧
deg = n+m
                    ∧ length treeList' =2^m ∧ invar-vebt summary' m ∧ (∀ t ∈ set treeList'. invar-vebt
t n)
    apply cases
    using 4.hyps(3) 4.hyps(4) one-is-add apply force
    apply (metis 4.hyps(3) 4.hyps(4) add-self-div-2)
    apply (metis 4.hyps(3) 4.hyps(4) even-Suc odd-add)
    apply (metis 4.hyps(3) 4.hyps(4) add-self-div-2)
    apply (metis 4.hyps(3) 4.hyps(4) even-Suc odd-add)
    done
```

have *ac*:*invar-vebt t h* $\Longrightarrow$ *invar-vebt k h* $\Longrightarrow$ *set-vebt′ t* = *set-vebt′ k* $\Longrightarrow$ *vebt-mint t* = *vebt-mint k* **for** *t k h*
proof−
assume *assms*: *invar-vebt t h invar-vebt k h set-vebt′ t* = *set-vebt′ k*
have ¬ *vebt-mint t* = *vebt-mint k* $\Longrightarrow$ *False*
proof−
assume *vebt-mint t* ≠ *vebt-mint k*
then obtain *a b* **where** *abdef*:*vebt-mint t* = *None* ∧ *vebt-mint k* = *Some b* ∨
vebt-mint t = *Some a* ∧ *vebt-mint k* = *None* ∨
a < *b* ∧ *Some a* = *vebt-mint t* ∧ *Some b* = *vebt-mint k* ∨
b < *a* ∧ *Some a* = *vebt-mint t* ∧ *Some b* = *vebt-mint k*
by (*metis linorder-neqE-nat option.exhaust*)
show *False*
apply(*cases vebt-mint t* = *None* ∧ *vebt-mint k* = *Some b*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
apply(*cases vebt-mint t* = *Some a* ∧ *vebt-mint k* = *None*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
apply (*cases a* < *b* ∧ *Some a* = *vebt-mint t* ∧ *Some b* = *vebt-mint k*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *abdef assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
done
qed
thus *vebt-mint t* = *vebt-mint k* **by** *auto*
qed
have *ad*:*invar-vebt t h* $\Longrightarrow$ *invar-vebt k h* $\Longrightarrow$ *set-vebt′ t* = *set-vebt′ k* $\Longrightarrow$ *vebt-maxt t* = *vebt-maxt k* **for** *t k h*
proof−
assume *assms*: *invar-vebt t h invar-vebt k h set-vebt′ t* = *set-vebt′ k*
have ¬ *vebt-maxt t* = *vebt-maxt k* $\Longrightarrow$ *False*
proof−
assume *vebt-maxt t* ≠ *vebt-maxt k*
then obtain *a b* **where** *abdef*:*vebt-maxt t* = *None* ∧ *vebt-maxt k* = *Some b* ∨
vebt-maxt t = *Some a* ∧ *vebt-maxt k* = *None* ∨
a < *b* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k* ∨
b < *a* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k*
by (*metis linorder-neqE-nat option.exhaust*)
show *False* **apply**(*cases vebt-maxt t* = *None* ∧ *vebt-maxt k* = *Some b*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
apply(*cases vebt-maxt t* = *Some a* ∧ *vebt-maxt k* = *None*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
apply (*cases a* < *b* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
by (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *abdef assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
qed
thus *vebt-maxt t* = *vebt-maxt k* **by** *auto*
qed
have *infsplit*: *info* = *Some* (*mi* ,*ma*) **using** *4*(*12*)
proof *cases*

case (*1 a b*)
then show *?thesis*
using *sprop1* **by** *blast*
next
case (*2 treeList n summary m*)
then show *?thesis*
by (*metis 4.prems*(*2*) *Collect-empty-eq VEBT-Member.vebt-member.simps*(*2*) *aa empty-iff insert-subset set-vebt'-def*)
next
case (*3 treeList n summary m*)
then show *?thesis*
by (*metis 4.prems*(*2*) *Collect-empty-eq VEBT-Member.vebt-member.simps*(*2*) *aa empty-iff insert-subset set-vebt'-def*)
next
case (*4 treeList' n' summary' m' mi' ma'*)
have *vebt-mint s* = *Some mi'*
by (*simp add*: *4*(*1*))
hence *mi'* = *mi*
by (*smt* (*verit*, *ccfv-threshold*) *4.hyps*(*7*) *4.prems*(*1*) *4.prems*(*2*) *VEBT-Member.vebt-member.simps*(*5*) *One-nat-def a0 aa add.assoc eq-iff insert-subset leI le-add-diff-inverse less-imp-le-nat mem-Collect-eq min-in-set-def mint-sound numeral-2-eq-2 option.sel order.not-eq-order-implies-strict plus-1-eq-Suc set-vebt'-def*)
have *vebt-maxt s* = *Some ma'*
by (*simp add*: *4*(*1*))
hence *ma'* < *ma* ⟹ *ma*∉ *set-vebt' s*
by (*meson 4.prems*(*1*) *leD max-in-set-def maxt-corr*)
moreover have *ma* < *ma'* ⟹ *ma'* ∉ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) **using** *case4*
by (*metis dual-order.strict-trans2 mem-Collect-eq member-inv not-less-iff-gr-or-eq set-vebt'-def*)
ultimately have *ma'*=*ma*
by (*metis* ‹*vebt-maxt s* = *Some ma'*› *aa case4*(*12*) *case4*(*13*) *insert-subset max-in-set-def maxt-corr not-less-iff-gr-or-eq*)
then show *?thesis*
using *4*(*1*) ‹*mi'* = *mi*› *sprop1* **by** *force*
next
case (*5 treeList n summary m mi' ma'*)
have *vebt-mint s* = *Some mi'*
by (*simp add*: *5*(*1*))
hence *mi'* = *mi*
by (*smt* (*verit*, *ccfv-threshold*) *4.hyps*(*7*) *4.prems*(*1*) *4.prems*(*2*) *VEBT-Member.vebt-member.simps*(*5*) *One-nat-def a0 aa add.assoc eq-iff insert-subset leI le-add-diff-inverse less-imp-le-nat mem-Collect-eq min-in-set-def mint-sound numeral-2-eq-2 option.sel order.not-eq-order-implies-strict plus-1-eq-Suc set-vebt'-def*)
have *vebt-maxt s* = *Some ma'*
by (*simp add*: *5*(*1*))
hence *ma'* < *ma* ⟹ *ma*∉ *set-vebt' s*
by (*meson 4.prems*(*1*) *leD max-in-set-def maxt-corr*)
moreover have *ma* < *ma'* ⟹ *ma'* ∉ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) **using** *case4*
by (*metis dual-order.strict-trans2 mem-Collect-eq member-inv not-less-iff-gr-or-eq set-vebt'-def*)
ultimately have *ma'*=*ma*

by (*metis 5(5) 5(6) case4(5) case4(6) even-Suc odd-add*)
then show *?thesis*
using *5(1)* ‹*mi' = mi*› *sprop1* **by** *force*
qed
from *4(12)* **have** *acd:mi* ≠ *ma* ⟶
(∀ *i*<*2* ^ *m*.
(*high ma n* = *i* ⟶ *both-member-options* (*treeList'* ! *i*) (*low ma n*)) ∧
(∀ *x*. *high x n* = *i* ∧ *both-member-options* (*treeList'* ! *i*) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*))
apply *cases* **using** *sprop1* **apply** *simp*
using *sprop1 infsplit* **apply** *simp*
using *sprop1 infsplit* **apply** *simp*
apply (*metis VEBT.inject(1) add-self-div-2 case4(5) infsplit option.inject prod.inject sprop1*)
by (*metis case4(5) case4(6) even-Suc odd-add*)
hence *length treeList'* = *2*^*m*
using *sprop1* **by** *fastforce*
hence *aca:length treeList'* =*length treeList* **using** *4.hyps(2)*
by (*simp add*: *4.hyps(2) sprop1*)
from *4(12)* **have** *sumtreelistcong*: ∀ *i*<*2* ^ *m*. (∃ *x*. *both-member-options* (*treeList'* ! *i*) *x*) = *both-member-options summary' i*
apply *cases*
using *a0* **apply** *linarith*
apply (*metis VEBT.inject(1) nth-mem sprop1*)
using *infsplit sprop1* **apply** *force*
apply (*metis VEBT.inject(1) sprop1*)
using *sprop1* **by** *auto*
hence *membercongy:i* < *2*^*m* ⟹ *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x* **for** *i x*
proof−
assume *i* < *2*^*m*
show *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x*
proof
show *vebt-member* (*treeList* ! *i*) *x* ⟹ *vebt-member* (*treeList'* ! *i*) *x*
proof−
assume *vebt-member* (*treeList* ! *i*) *x*
hence *aaa:both-member-options* (*treeList* ! *i*) *x*
by (*metis* ‹*i* < *2* ^ *m*› *both-member-options-equiv-member case4(1) case4(4) nth-mem*)
have *x* < *2*^*n*
by (*metis* ‹*i* < *2* ^ *m*› ‹*vebt-member* (*treeList* ! *i*) *x*› *case4(1) case4(4) member-bound nth-mem*)
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2*^*n*∗*i*+*x*)
using *both-member-options-from-chilf-to-complete-tree*
[*of* (*2*^*n*∗*i*+*x*) *deg treeList mi ma summary*] *aaa high-inv*[*of x n i*]
by (*smt* (*verit*) ‹*i* < *2* ^ *m*› *a0 add-leD1 add-self-div-2 both-member-options-equiv-member case4(1,10,11,3,4,5,7,8,9) invar-vebt.intros(4) low-inv mult.commute one-add-one sprop1*)
have *mi* < (*2*^*n*∗*i*+*x*) ∧ (*2*^*n*∗*i*+*x*) ≤ *ma* **using** *vebt-mint.simps(3)*[*of mi ma deg treeList summary*]
by (*metis* ‹*i* < *2* ^ *m*› ‹*x* < *2* ^ *n*› *aaa case4(11) case4(4) case4(8) high-inv low-inv mult.commute nth-mem*)
moreover have *both-member-options s* (*2*^*m*∗*i* +*x*)

using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (2 ^ *n* ∗ *i* + *x*)› *both-member-options-equiv-member case4*(*12*) *case4*(*13*) *case4*(*5*) *set-vebt'-def* **by** *auto*

hence *both-member-options* (*treeList'* ! *i*) *x*

by (*metis Groups.mult-ac*(*2*) *Suc-1 VEBT-internal.deg-not-0* ‹*x* < 2 ^ *n*› *a0 acd add-leD1 add-self-div-2*

both-member-options-from-complete-tree-to-child calculation case4(*10*,*5*) *exp-split-high-low*(*1*)

high-inv infsplit low-inv nat-less-le plus-1-eq-Suc sprop1)

then show *?thesis*

by (*metis* ‹*i* < 2 ^ *m*› *nth-mem sprop1 valid-member-both-member-options*)

qed

show *vebt-member* (*treeList'* ! *i*) *x* ⟹ *vebt-member* (*treeList* ! *i*) *x*

proof−

assume *vebt-member* (*treeList'* ! *i*) *x*

hence *vebt-member s* (2^*n*∗*i* +*x*) **using** *sprop1 both-member-options-from-chilf-to-complete-tree*

[*of* (2^*n*∗*i* +*x*) *deg treeList' mi ma summary'*]

by (*smt* (*verit*) *Nat.add-0-right* ‹*i* < 2 ^ *m*› *a0 add-leD1 add-self-div-2 both-member-options-equiv-member case4*(*12*) *case4*(*5*) *div-less div-mult-self4 high-def infsplit low-inv member-bound mult.commute nat-1-add-1 nth-mem power-not-zero zero-neq-numeral*)

hence *mi* < (2^*n*∗*i* +*x*) ∧ (2^*n*∗*i* +*x*) ≤ *ma*

using *vebt-mint.simps*(*3*)[*of mi ma deg treeList' summary'*] *vebt-maxt.simps*(*3*)[*of mi ma deg treeList' summary'*]

by (*metis* ‹*i* < 2 ^ *m*› ‹*vebt-member* (*treeList'* ! *i*) *x*› *acd both-member-options-equiv-member case4*(*12*) *high-inv infsplit low-inv member-bound mi-eq-ma-no-ch mult.commute nth-mem sprop1*)

moreover have *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (2^*m*∗*i* +*x*)

by (*metis* ‹*vebt-member s* (2 ^ *n* ∗ *i* + *x*)› *add-leD1 both-member-options-equiv-member both-member-options-from-chilf-to-complete-tree calculation case4*(*1*) *case4*(*13*) *case4*(*5*) *maxbmo vebt-maxt.simps*(*3*) *mem-Collect-eq member-inv nat-neq-iff nth-mem one-add-one set-vebt'-def*)

hence *both-member-options* (*treeList* ! *i*) *x*

using *both-member-options-from-complete-tree-to-child*[*of deg mi ma treeList summary* (2^*n*∗*i* +*x*)]

by (*smt* (*verit*) *Nat.add-0-right Suc-leD* ‹*i* < 2 ^ *m*› ‹*vebt-member* (*treeList'* ! *i*) *x*› *a0 add-self-div-2 calculation case4*(*11*) *case4*(*5*) *div-less div-mult-self4 high-def low-inv member-bound mult.commute nat-1-add-1 nat-neq-iff nth-mem plus-1-eq-Suc power-not-zero sprop1 zero-neq-numeral*)

then show *?thesis*

by (*metis* ‹*i* < 2 ^ *m*› *aca case4*(*1*) *nth-mem sprop1 valid-member-both-member-options*)

qed

qed

qed

hence *setcongy*: *i* < 2^*m* ⟹ *set-vebt'* (*treeList* ! *i*) = *set-vebt'* (*treeList'* ! *i*) **for** *i* **unfolding** *set-vebt'-def* **by** *presburger*

hence *treecongy*: *i* < 2^*m* ⟹ *treeList* ! *i* = *treeList'* ! *i* **for** *i*

by (*metis case4*(*1*) *case4*(*4*) *nth-mem sprop1*)

hence *treeList* = *treeList'*

by (*metis aca case4*(*4*) *nth-equalityI*)

have *vebt-member summary x* ⟷ *vebt-member summary' x* **for** *x*

by (*metis* ‹*treeList* = *treeList'*› *both-member-options-equiv-member case4*(*3*) *case4*(*7*) *member-bound sprop1 sumtreelistcong*)

hence *set-vebt' summary* = *set-vebt' summary'* **unfolding** *set-vebt'-def* **by** *auto*

hence *summary = summary'*
using *case4(2) sprop1* **by** *blast*
then show *?case*
using ‹*treeList = treeList'*› *infsplit sprop1* **by** *fastforce*
next
case (*5 treeList n summary m deg mi ma*)
note *case4= this*
hence *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) = *set-vebt' s* **by** *simp*
hence *a0*:*deg* ≥ *2* **using** *5*
by (*metis Suc-leI add-le-mono diff-Suc-1 less-add-same-cancel1 not-add-less1 not-less-iff-gr-or-eq numeral-2-eq-2 plus-1-eq-Suc set-n-deg-not-0*)
hence *aa*:{*mi*, *ma*} ⊆ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
apply *auto* **using** *vebt-member.simps*(*5*)[*of mi ma deg −2 treeList summary mi*]
apply (*metis add-2-eq-Suc' le-add-diff-inverse2 mem-Collect-eq set-vebt'-def*)
using *vebt-member.simps*(*5*)[*of mi ma deg −2 treeList summary ma*]
apply (*metis add-2-eq-Suc' le-add-diff-inverse2 mem-Collect-eq set-vebt'-def*)
done
from *5*(*12*) **obtain** *treeList' summary' info* **where**
sprop1:*s = Node info deg treeList' summary' deg = n+m length treeList' =2^m*
invar-vebt summary' m ∀ *t* ∈ *set treeList'. invar-vebt t n*
apply *cases*
using *a0* **apply** *linarith*
apply (*metis case4*(*5*) *case4*(*6*) *even-Suc odd-add add-self-div-2*)
apply (*metis Suc-inject add-Suc-right add-self-div-2 case4*(*5*) *case4*(*6*))
apply (*metis case4*(*5*) *case4*(*6*) *even-Suc odd-add*)
apply (*metis Suc-inject add-Suc-right add-self-div-2 case4*(*5*) *case4*(*6*))
done
have *ac*:*invar-vebt t h* ⟹ *invar-vebt k h* ⟹ *set-vebt' t = set-vebt' k* ⟹ *vebt-mint t = vebt-mint k* **for** *t k h*
proof−
assume *assms*: *invar-vebt t h invar-vebt k h set-vebt' t = set-vebt' k*
have ¬ *vebt-mint t = vebt-mint k* ⟹ *False*
proof−
assume *vebt-mint t* ≠ *vebt-mint k*
then obtain *a b* **where** *abdef*:*vebt-mint t = None* ∧ *vebt-mint k = Some b* ∨
vebt-mint t = Some a ∧ *vebt-mint k = None* ∨
a < b ∧ *Some a = vebt-mint t* ∧ *Some b = vebt-mint k* ∨
b < a ∧ *Some a = vebt-mint t* ∧ *Some b = vebt-mint k*
by (*metis linorder-neqE-nat option.exhaust*)
show *False* **apply**(*cases vebt-mint t = None* ∧ *vebt-mint k = Some b*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
apply(*cases vebt-mint t = Some a* ∧ *vebt-mint k = None*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
apply (*cases a < b* ∧ *Some a = vebt-mint t* ∧ *Some b = vebt-mint k*)
apply (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
by (*metis* ‹*vebt-mint t* ≠ *vebt-mint k*› *abdef assms*(*1*) *assms*(*2*) *assms*(*3*) *mint-corr mint-sound*)
qed
thus *vebt-mint t = vebt-mint k* **by** *auto*
qed

have *ad*:*invar-vebt t h* $\Longrightarrow$ *invar-vebt k h* $\Longrightarrow$ *set-vebt' t* = *set-vebt' k* $\Longrightarrow$ *vebt-maxt t* = *vebt-maxt k* **for** *t k h*
proof−
assume *assms*: *invar-vebt t h invar-vebt k h set-vebt' t* = *set-vebt' k*
have ¬ *vebt-maxt t* = *vebt-maxt k* $\Longrightarrow$ *False*
proof−
assume *vebt-maxt t* ≠ *vebt-maxt k*
then obtain *a b* **where** *abdef*:*vebt-maxt t* = *None* ∧ *vebt-maxt k* = *Some b* ∨
vebt-maxt t = *Some a* ∧ *vebt-maxt k* = *None* ∨
a < *b* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k* ∨
b < *a* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k*
by (*metis linorder-neqE-nat option.exhaust*)
show *False*
apply(*cases vebt-maxt t* = *None* ∧ *vebt-maxt k* = *Some b*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
apply(*cases vebt-maxt t* = *Some a* ∧ *vebt-maxt k* = *None*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
apply (*cases a* < *b* ∧ *Some a* = *vebt-maxt t* ∧ *Some b* = *vebt-maxt k*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
apply (*metis* ‹*vebt-maxt t* ≠ *vebt-maxt k*› *abdef assms*(*1*) *assms*(*2*) *assms*(*3*) *maxt-corr maxt-sound*)
done
qed
thus *vebt-maxt t* = *vebt-maxt k* **by** *auto*
qed
have *infsplit*: *info* = *Some* (*mi* ,*ma*) **using** *5*(*12*)
proof *cases*
case (*1 a b*)
then show *?thesis*
using *sprop1* **by** *blast*
next
case (*2 treeList n summary m*)
then show *?thesis*
by (*metis 5.prems*(*2*) *Collect-empty-eq VEBT-Member.vebt-member.simps*(*2*) *aa empty-iff insert-subset set-vebt'-def*)
next
case (*3 treeList n summary m*)
then show *?thesis*
by (*metis 5.prems*(*2*) *Collect-empty-eq VEBT-Member.vebt-member.simps*(*2*) *aa empty-iff insert-subset set-vebt'-def*)
next
case (*4 treeList' n' summary' m' mi' ma'*)
have *vebt-mint s* = *Some mi'*
by (*simp add*: *4*(*1*))
hence *mi'* = *mi*
by (*smt* (*verit*, *ccfv-threshold*) *5.hyps*(*7*) *5.prems*(*1*) *5.prems*(*2*) *VEBT-Member.vebt-member.simps*(*5*) *One-nat-def a0 aa add.assoc eq-iff insert-subset leI le-add-diff-inverse less-imp-le-nat mem-Collect-eq min-in-set-def mint-sound numeral-2-eq-2 option.sel order.not-eq-order-implies-strict plus-1-eq-Suc set-vebt'-def*)
have *vebt-maxt s* = *Some ma'*

by (*simp add*: *4*(*1*))
hence *ma'* < *ma* ⟹ *ma*∉ *set-vebt' s*
by (*meson 5.prems*(*1*) *leD max-in-set-def maxt-corr*)
moreover have *ma* < *ma'* ⟹ *ma'* ∉ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
using *case4*
by (*metis dual-order.strict-trans2 mem-Collect-eq member-inv not-less-iff-gr-or-eq set-vebt'-def*)
ultimately have *ma'*=*ma*
by (*metis* ‹*vebt-maxt s* = *Some ma'*› *aa case4*(*12*) *case4*(*13*) *insert-subset max-in-set-def maxt-corr not-less-iff-gr-or-eq*)
then show *?thesis*
using *4*(*1*) ‹*mi'* = *mi*› *sprop1* **by** *force*
next
case (*5 treeList' n' summary' m' mi' ma'*)
have *vebt-mint s* = *Some mi'*
by (*simp add*: *5*(*1*))
hence *mi'* = *mi*
by (*smt* (*verit*, *ccfv-threshold*) *5.hyps*(*7*) *5.prems*(*1*) *5.prems*(*2*) *VEBT-Member.vebt-member.simps*(*5*) *One-nat-def a0 aa add.assoc eq-iff insert-subset leI le-add-diff-inverse less-imp-le-nat mem-Collect-eq min-in-set-def mint-sound numeral-2-eq-2 option.sel order.not-eq-order-implies-strict plus-1-eq-Suc set-vebt'-def*)
have *vebt-maxt s* = *Some ma'*
by (*simp add*: *5*(*1*))
hence *ma'* < *ma* ⟹ *ma*∉ *set-vebt' s*
by (*meson 5.prems*(*1*) *leD max-in-set-def maxt-corr*)
moreover have *ma* < *ma'* ⟹ *ma'* ∉ *set-vebt'* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
using *case4*
by (*metis dual-order.strict-trans2 mem-Collect-eq member-inv not-less-iff-gr-or-eq set-vebt'-def*)
ultimately have *ma'*=*ma* **using** *case4*(*13*) *5*
by (*metis* ‹*vebt-maxt s* = *Some ma'*› *aa both-member-options-equiv-member case4*(*12*) *insert-subset maxbmo mem-Collect-eq not-less-iff-gr-or-eq set-vebt'-def*)
then show *?thesis*
using *5*(*1*) ‹*mi'* = *mi*› *sprop1* **by** *force*
qed
from *5*(*12*) **have** *acd*:*mi* ≠ *ma* ⟶
(∀ *i*<*2* ^ *m*.
(*high ma n* = *i* ⟶ *both-member-options* (*treeList'* ! *i*) (*low ma n*)) ∧
(∀ *x*. *high x n* = *i* ∧ *both-member-options* (*treeList'* ! *i*) (*low x n*) ⟶ *mi* < *x* ∧ *x* ≤ *ma*))
unfolding *sprop1*
proof (*cases rule*: *invar-vebt.cases*)
case (*2 n m*)
then show *?thesis*
using *infsplit* **by** *simp*
next
case (*3 n m*)
then show *?thesis*
using *infsplit* **by** *simp*
next
case (*4 n m mi ma*)
then show *?thesis*
using *infsplit*

by (*metis case4*(*5*) *even-Suc odd-add*)
next
case (*5 n' m' mi' ma'*)
then have ∗: *mi'* = *mi ma'* = *ma*
using *infsplit* **by** *simp-all*
show *?thesis*
using *5 sprop1*
unfolding ∗
unfolding *case4*(*5*)
by (*metis Suc-1 lessI nat.inject power-inject-exp*)
qed
hence *length treeList'* = *2^m*
using *sprop1* **by** *fastforce*
hence *aca*:*length treeList'* =*length treeList* **using** *5.hyps*(*2*)
by (*simp add*: *5.hyps*(*2*) *sprop1*)
from *5*(*12*) **have** *sumtreelistcong*: ∀ *i*<*2* ^ *m*. (∃ *x*. *both-member-options* (*treeList'* ! *i*) *x*) = *both-member-options summary' i*
apply *cases*
using *a0* **apply** *linarith*
apply (*metis VEBT.inject*(*1*) *nth-mem sprop1*)
using *infsplit sprop1* **apply** *force*
apply (*metis VEBT.inject*(*1*) *sprop1*)
using *sprop1* **apply** *auto*
done
hence *membercongy*:*i* < *2^m* ⟹ *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x* **for** *i x*
proof−
assume *i* < *2^m*
show *vebt-member* (*treeList*! *i*) *x* ⟷ *vebt-member* (*treeList'* ! *i*) *x*
proof
show *vebt-member* (*treeList* ! *i*) *x* ⟹ *vebt-member* (*treeList'* ! *i*) *x*
proof−
assume *vebt-member* (*treeList* ! *i*) *x*
hence *aaa*:*both-member-options* (*treeList* ! *i*) *x*
by (*metis* ‹*i* < *2* ^ *m*› *both-member-options-equiv-member case4*(*1*) *case4*(*4*) *nth-mem*)
have *x* < *2^n*
by (*metis* ‹*i* < *2* ^ *m*› ‹*vebt-member* (*treeList* ! *i*) *x*› *case4*(*1*) *case4*(*4*) *member-bound nth-mem*)
hence *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2^n*∗*i*+*x*)
using *both-member-options-from-chilf-to-complete-tree*
[*of* (*2^n*∗*i*+*x*) *deg treeList mi ma summary*] *aaa high-inv*[*of x n i*]
‹*i* < *2* ^ *m*› ‹*vebt-member* (*treeList* ! *i*) *x*› *low-inv*[*of x n i*]
by (*simp add*: *case4*(*4*) *case4*(*5*) *mult.commute sprop1*)
hence *vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2^n*∗*i*+*x*) **using**
valid-member-both-member-options[*of* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *deg 2^n*∗*i*+*x*]
invar-vebt.intros(*5*)[*of treeList n summary m deg mi ma*] *case4* **by** *fastforce*
hence *mi* < (*2^n*∗*i*+*x*) ∧ (*2^n*∗*i*+*x*) ≤ *ma* **using** *vebt-mint.simps*(*3*)[*of mi ma deg treeList summary*]

by (*metis* ‹$i < 2 \hat{} m$› ‹$x < 2 \hat{} n$› *aaa case4*(*11*) *case4*(*4*) *case4*(*8*) *high-inv low-inv mult.commute nth-mem*)

moreover have *both-member-options s* (*2^n*∗*i* +*x*)

using ‹*vebt-member* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) (*2* $\hat{}$ *n* ∗ *i* + *x*)› *both-member-options-equiv-member case4*(*12*) *case4*(*13*) *case4*(*5*) *set-vebt'-def* **by** *auto*

have *acffs*:*both-member-options* (*treeList'* ! (*high ma n*)) (*low ma n*)

using *acd calculation case4*(*10*) *high-bound-aux sprop1 alethe-comp-simplify1*(*3*) **by** *blast*

hence *both-member-options* (*treeList'* ! *i*) *x*

using *both-member-options-from-complete-tree-to-child*[*of deg mi ma treeList' summary' 2^n*∗*i*+*x*]

low-inv[*of x n i*] *high-inv*[*of x n i*]

by (*smt* (*verit*, *ccfv-threshold*) *Suc-1* ‹*both-member-options s* (*2* $\hat{}$ *n* ∗ *i* + *x*)› ‹$x < 2 \hat{} n$›

add.right-neutral add-Suc-right add-leD1 calculation case4(*5*,*6*) *div-mult-self2 infsplit*

le-imp-less-Suc mult.commute mult-2 nat-less-le not-le numeral-One one-div-two-eq-zero plus-1-eq-Suc

sprop1(*1*))

then show *vebt-member* (*treeList'* ! *i*) *x*

by (*metis* ‹$i < 2 \hat{} m$› *nth-mem sprop1 valid-member-both-member-options*)

qed

show *vebt-member* (*treeList'* ! *i*) *x* ⟹ *vebt-member* (*treeList* ! *i*) *x*

proof−

assume *assm*: *vebt-member* (*treeList'* ! *i*) *x*

have *both-member-options* (*Node* (*Some* (*mi*, *ma*)) *deg treeList' summary'*) (*2* $\hat{}$ *n* ∗ *i* + *x*)

proof (*rule both-member-options-from-chilf-to-complete-tree*)

show *1* ≤ *deg*

using *a0* **by** *presburger*

have $x < 2 \hat{} n$

using *assm* **by** (*simp add*: ‹$i < 2 \hat{} m$› *member-bound sprop1*(*3*,*5*))

then have *high* (*2* $\hat{}$ *n* ∗ *i* + *x*) *n* = *i*

by (*simp add*: *high-inv mult.commute*)

moreover have *deg div 2* = *n*

by (*simp add*: *case4*(*5*) *sprop1*(*2*))

ultimately show *high* (*2* $\hat{}$ *n* ∗ *i* + *x*) (*deg div 2*) < *length treeList'*

by (*metis sprop1*(*3*) ‹$i < 2 \hat{} m$›)

then show *both-member-options* (*treeList'* ! *high* (*2* $\hat{}$ *n* ∗ *i* + *x*) (*deg div 2*))

(*low* (*2* $\hat{}$ *n* ∗ *i* + *x*) (*deg div 2*))

by (*metis* ‹*deg div 2* = *n*› ‹*high* (*2* $\hat{}$ *n* ∗ *i* + *x*) *n* = *i*› ‹*vebt-member* (*treeList'* ! *i*) *x*› ‹$x < 2 \hat{} n$›

both-member-options-equiv-member low-inv mult.commute nth-mem sprop1(*5*))

qed

then have *vebt-member s* (*2^n*∗*i* +*x*)

by (*metis sprop1*(*1*) *valid-member-both-member-options case4*(*12*) *infsplit*)

hence *mi* < (*2^n*∗*i* +*x*) ∧ (*2^n*∗*i* +*x*) ≤ *ma*

using *vebt-mint.simps*(*3*)[*of mi ma deg treeList' summary'*] *vebt-maxt.simps*(*3*)[*of mi ma deg treeList' summary'*]

```
        by (metis ‹i < 2 ^ m› ‹vebt-member (treeList' ! i) x› acd both-member-options-equiv-member
case4(12) high-inv infsplit low-inv member-bound mi-eq-ma-no-ch mult.commute nth-mem sprop1)
       moreover have both-member-options (Node (Some (mi, ma)) deg treeList summary) (2^n*i
+x)
         by (metis ‹vebt-member s (2 ^ n * i + x)› add-leD1 both-member-options-equiv-member
both-member-options-from-chilf-to-complete-tree calculation case4(1) case4(13) case4(5) maxbmo vebt-maxt.simps(3)
mem-Collect-eq member-inv nat-neq-iff nth-mem one-add-one set-vebt'-def)
       have invar-vebt (treeList' ! i) n
         by (simp add: ‹i < 2 ^ m› sprop1)
       hence x < 2^n
         using ‹vebt-member (treeList' ! i) x› member-bound by auto
       then have low (2 ^ n * i + x) (deg div 2) = x high (2 ^ n * i + x) (deg div 2) = i
         using low-inv[of x n i] high-inv[of x n i]
         by (simp-all add: case4(5) mult.commute sprop1(2))
       hence both-member-options (treeList ! i) x
        using both-member-options-from-complete-tree-to-child[of deg mi ma treeList summary (2^n*i
+x)]
         by (metis ‹both-member-options (Node (Some (mi, ma)) deg treeList summary) (2 ^ n * i +
x)›
           ‹i < 2 ^ m› ‹x < 2 ^ n› a0 add-leD1 calculation case4(11) high-inv low-inv mult.commute
nat-less-le
            one-add-one)
       then show ?thesis
         by (metis ‹i < 2 ^ m› aca case4(1) nth-mem sprop1 valid-member-both-member-options)
     qed
   qed
 qed
 hence setcongy: i < 2^m ⟹ set-vebt' (treeList ! i) = set-vebt' (treeList' ! i) for i unfolding
set-vebt'-def by presburger
 hence treecongy: i < 2^m ⟹ treeList ! i = treeList' ! i for i
   by (metis case4(1) case4(4) nth-mem sprop1)
 hence treeList = treeList'
   by (metis aca case4(4) nth-equalityI)
 have vebt-member summary x ⟷ vebt-member summary' x for x
  by (metis ‹treeList = treeList'› both-member-options-equiv-member case4(3) case4(7) member-bound
sprop1 sumtreelistcong)
 hence set-vebt' summary = set-vebt' summary' unfolding set-vebt'-def by auto
 hence summary = summary'
   using case4(2) sprop1 by blast
 then show ?case
   using ‹treeList = treeList'› infsplit sprop1 by fastforce
qed

corollary invar-vebt t n ⟹ set-vebt' t = {} ⟹ t = vebt-buildup n
 by (metis buildup-gives-empty buildup-gives-valid deg-not-0 uniquetree)

corollary unique-tree: invar-vebt t n ⟹ invar-vebt s n ⟹ set-vebt t = set-vebt s ⟹ s = t
 by (simp add: set-vebt-set-vebt'-valid uniquetree)
```

corollary *invar-vebt t n* $\Longrightarrow$ *set-vebt t* $= \{\} \Longrightarrow$ *t = vebt-buildup n*
by (*metis buildup-gives-empty buildup-gives-valid deg-not-0 uniquetree set-vebt-set-vebt'-valid*)

All valid trees can be generated by $vebt - insertion$ chains on an empty tree with same degree parameter:

inductive *perInsTrans*::*VEBT* $\Rightarrow$ *VEBT* $\Rightarrow$ *bool* **where**
perInsTrans t t|
(*t = vebt-insert s x*) $\Longrightarrow$ *perInsTrans t u* $\Longrightarrow$ *perInsTrans s u*

lemma *perIT-concat*: *perInsTrans s t* $\Longrightarrow$ *perInsTrans t u* $\Longrightarrow$ *perInsTrans s u*
by (*induction s t rule*: *perInsTrans.induct*) (*simp add*: *perInsTrans.intros*)+

lemma assumes *invar-vebt t n* **shows**
perInsTrans (*vebt-buildup n*) *t*
proof−
have *finite A* $\Longrightarrow$*invar-vebt s n* $\Longrightarrow$*set-vebt' s = B* $\Longrightarrow$ *B*$\subseteq$ *A* $\Longrightarrow$ *perInsTrans* (*vebt-buildup n*) *s*
for *s A B*
proof (*induction card B arbitrary*: *s B*)
case *0*
then show *?case*
by (*metis buildup-gives-empty buildup-gives-valid card-eq-0-iff deg-not-0 perInsTrans.intros*(*1*) *set-vebt-finite uniquetree*)
next
case (*Suc car*)
hence *finite B*
by (*meson rev-finite-subset*)
obtain *x b* **where** *B = insert x b* $\wedge$ *x* $\notin$ *b*
by (*metis Suc.hyps*(*2*) *card-Suc-eq*)
have *set-vebt'* (*vebt-delete s x*) *= b*
using *Suc.prems*(*2*) *Suc.prems*(*3*) ‹*B = insert x b* $\wedge$ *x* $\notin$ *b*› *delete-correct'* **by** *auto*
moreover hence *perInsTrans* (*vebt-buildup n*) (*vebt-delete s x*)
by (*metis Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*4*) ‹*B = insert x b* $\wedge$ *x* $\notin$ *b*› ‹*finite B*› *card-insert-disjoint delete-pres-valid finite-insert nat.inject subset-insertI subset-trans*)
hence *set-vebt'* (*vebt-insert* (*vebt-delete s x*) *x*) *= set-vebt' s*
by (*metis Diff-insert-absorb Suc.prems*(*2*) *Suc.prems*(*3*) *Un-insert-right* ‹*B = insert x b* $\wedge$ *x* $\notin$ *b*› *boolean-algebra-cancel.sup0 delete-pres-valid delete-correct' insertI1 insert-corr mem-Collect-eq member-bound set-vebt'-def*)
have *invar-vebt* (*vebt-insert* (*vebt-delete s x*) *x*) *n*
by (*metis Suc.prems*(*2*) *Suc.prems*(*3*) ‹*B = insert x b* $\wedge$ *x* $\notin$ *b*› *delete-pres-valid insertI1 mem-Collect-eq member-bound set-vebt'-def valid-pres-insert*)
moreover hence *vebt-insert* (*vebt-delete s x*) *x = s*
using *Suc.prems*(*2*) ‹*set-vebt'* (*VEBT-Insert.vebt-insert* (*vebt-delete s x*) *x*) *= set-vebt' s*› *uniquetree* **by** *force*
ultimately show *?case*
by (*metis* ‹*perInsTrans* (*vebt-buildup n*) (*vebt-delete s x*)› *perIT-concat perInsTrans.intros*(*1*) *perInsTrans.intros*(*2*))
qed
then show *?thesis*
by (*meson assms equalityD1 set-vebt-finite*)

qed

end
end

theory *VEBT-Height* **imports** *VEBT-Definitions Complex-Main*
begin

context *VEBT-internal* **begin**

10 Heights of van Emde Boas Trees

fun *height*::*VEBT* ⇒ *nat* **where**
height (*Leaf a b*) = *0*|
height (*Node - deg treeList summary*) = (*1*+ *Max* (*height* ‘ (*insert summary* (*set treeList*))))

abbreviation *lb x* ≡ *log 2 x*

lemma *setceilmax*: *invar-vebt s m* ⟹∀ *t* ∈ *set listy*. *invar-vebt t n*
⟹*m* = *Suc n* ⟹(∀ *t* ∈ *set listy*. *height t* = ⌈*lb n* ⌉) ⟹ *height s* = ⌈*lb m*⌉
⟹ *Max* (*height* ‘ (*insert s* (*set listy*))) = ⌈*lb m*⌉
proof(*induction listy*)
case *Nil*
hence *Max* (*height* ‘ (*insert s*(*set* []))) = *height s* **by** *simp*
then show *?case* **using** *Nil* **by** *simp*
next
case (*Cons a list*)
have *Max* (*height* ‘ *insert s* (*set* (*a* # *list*))) =
max (*height a*) (*Max* (*height* ‘ *insert s* (*set* (*list*))))
by (*simp add*: *insert-commute*)
moreover have *max* (*height a*) (*Max* (*height* ‘ *insert s* (*set* (*list*)))) = *max* (*height a*) ⌈*lb m* ⌉
using *Cons insert-iff list.simps*(*15*) *max-def of-nat-max* **by** *force*
moreover have ∀ *t* ∈ *set* (*a*#*list*). *invar-vebt t n* **using** *Cons* **by** *simp*
moreover hence *invar-vebt a n* **by** *simp*
hence *m* ≥ *n*
by (*simp add*: *Cons.prems*(*3*))
hence *lb m* ≥ *lb n*
using *deg-not-0* ‹*invar-vebt a n*› **by** *fastforce*
hence ⌈ *lb m*⌉ ≥ ⌈ *lb n*⌉
by (*simp add*: *ceiling-mono*)
moreover hence *max* ⌈*log 2 n* ⌉ ⌈*log 2 m* ⌉ = ⌈*log 2 m* ⌉ **by** *simp*
ultimately show *?case*
using *Cons.prems*(*4*) ‹*invar-vebt a n*›
by (*metis list.set-intros*(*1*))
qed

lemma *log-ceil-idem*:
assumes(*x*::*real*) ≥ *1*
shows ⌈*lb x* ⌉ = ⌈*lb* ⌈*x*⌉⌉

proof−
have $\lceil log\ 2\ x\ \rceil \geq 0$
by (*smt* (*verit*, *ccfv-SIG*) *assms zero-le-ceiling zero-le-log-cancel-iff*)
have $\lceil log\ 2\ x\ \rceil - 1 < log\ 2\ x \wedge log\ 2\ x \leq \lceil log\ 2\ x\ \rceil$
by *linarith*
moreover hence $2\ powr\ (\lceil log\ 2\ x\ \rceil - 1) < x \wedge x \leq 2\ powr\ (\lceil log\ 2\ x\ \rceil)$
by (*smt* (*verit*, *ccfv-SIG*) *assms less-log-iff real-nat-ceiling-ge*)
moreover hence $2\ powr\ ((\lceil log\ 2\ x\ \rceil - 1)) < \lceil x \rceil$ **and** $\lceil x \rceil \leq 2\ powr\ (\lceil log\ 2\ x\ \rceil)$
apply *linarith*
using ‹$0 \leq \lceil log\ 2\ x \rceil$› *calculation*(2) *ceiling-mono powr-int* **by** *fastforce*
moreover hence $\lceil log\ 2\ x\ \rceil - 1 < log\ 2\ \lceil\ x\ \rceil \wedge log\ 2\ \lceil x \rceil \leq \lceil log\ 2\ x\ \rceil$
by (*smt* (*verit*, *best*) *assms ceiling-correct less-log-iff*)
ultimately show *?thesis*
by *linarith*
qed

lemma *heigt-uplog-rel*:*invar-vebt t n* $\Longrightarrow$ $(height\ t) = \lceil lb\ n \rceil$
proof(*induction t n rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case* **by** *simp*
next
case (*2 treeList n summary m deg*)
hence $m \geq n$ **by** *simp*
hence $log\ 2\ m \geq log\ 2\ n$
by (*simp add*: *2.hyps*(3))
hence $\lceil\ log\ 2\ m \rceil \geq \lceil\ log\ 2\ n \rceil$
by (*simp add*: *2.hyps*(3))
have $Max\ (height\ `\ (insert\ summary\ (set\ treeList))) = \lceil\ log\ 2\ m \rceil$
by (*smt* (*verit*, *best*) *2.IH*(1) *2.IH*(2) *2.hyps*(3) *List.finite-set Max-in empty-is-image finite-imageI finite-insert image-iff insert-iff insert-not-empty*)
hence $height\ (Node\ None\ deg\ treeList\ summary) = 1 + \lceil\ log\ 2\ m \rceil$ **by** *simp*
moreover have $1 + \lceil\ log\ 2\ m \rceil = \lceil 1 + log\ 2\ m \rceil$ **by** *linarith*
moreover have $1 + log\ 2\ m = log\ 2\ (2 * m)$
using *2.hyps*(1) *deg-not-0 log-mult* **by** *force*
moreover hence $\lceil 1 + log\ 2\ m \rceil = \lceil log\ 2\ (2 * m) \rceil$ **by** *simp*
moreover hence $\lceil log\ 2\ (2 * m) \rceil = \lceil log\ 2\ (n + m) \rceil$
using *2.hyps*(3) **by** *force*
ultimately show *?case*
using *2.hyps*(4) **by** *metis*
next
case (*3 treeList n summary m deg*)
hence *00*: $n \geq 1 \wedge Suc\ n = m$
using *set-n-deg-not-0* **by** *blast*
hence *0*:$m \geq n$**using** *3* **by** *simp*
hence *1*:$log\ 2\ m \geq log\ 2\ n$
using *3.IH*(1) *3.hyps*(2) *set-n-deg-not-0* **by** *fastforce*
hence *2*:$\lceil\ log\ 2\ m \rceil \geq \lceil\ log\ 2\ n \rceil$
by (*simp add*: *ceiling-mono*)
have *3*: $Max\ (height\ `\ (insert\ summary\ (set\ treeList))) = \lceil\ log\ 2\ m \rceil$

```
    using 3.IH(1) 3.IH(2) 3.hyps(3) List.finite-set Max-in empty-is-image
      finite-imageI finite-insert image-iff insert-iff insert-not-empty  3.hyps(1) setceilmax by auto
  hence 4:height (Node None deg treeList summary) = 1+ ⌈ log 2 m⌉ by simp
  have 5:1+ ⌈ log 2 m⌉ = ⌈1+ log 2 m⌉ by linarith
  have 6:1+ log 2 m = log 2 (m+m)
    using 3.hyps(1) deg-not-0 log-mult by force
  hence 7:log 2 (m+n) = 1+log 2 ((n+m) / 2)
    by (simp add: 3.hyps(3) log-divide)
  have 8:log 2 ((n+m) / 2) = log 2 (n + 1/2)
    by (smt (verit, best) 3.hyps(3) field-sum-of-halves of-nat-Suc of-nat-add)
  have 9 : ⌈ log 2 (n + 1/2) ⌉ = ⌈ log 2 ⌈n + 1/2 ⌉ ⌉
    by (smt (verit) 00 field-sum-of-halves log-ceil-idem of-nat-1 of-nat-mono)
  hence 10: ⌈n + 1/2 ⌉ = m using 00 by linarith
  hence 11: ⌈ log 2 (n + 1/2) ⌉ = ⌈ log 2 m ⌉ using 9 by simp
  hence 12:⌈ 1+ log 2 (n + 1/2) ⌉ = ⌈1+ log 2 m ⌉
    by (smt (verit) ceiling-add-one)
  hence ⌈ log 2 (n + n+1) ⌉ = ⌈ log 2 (m+m) ⌉
    using 3.hyps(3) 6 7 8 by force
  then show ?case
    by (metis 12 3.hyps(4) 4 5 7 8 add.commute)
next
  case (4 treeList n summary m deg mi ma)
  hence m ≥ n by simp
  hence log 2 m ≥ log 2 n
    by (simp add: 4.hyps(3))
  hence ⌈ log 2 m⌉ ≥ ⌈ log 2 n⌉
    by (simp add: 4.hyps(3))
  have Max (height ' (insert summary (set treeList))) = ⌈ log 2 m⌉
   by (smt (verit, best) 4.IH(1) 4.IH(2) 4.hyps(3) List.finite-set Max-in empty-is-image finite-imageI
finite-insert image-iff insert-iff insert-not-empty)
  hence height (Node None deg treeList summary) = 1+ ⌈ log 2 m⌉ by simp
  moreover have 1+ ⌈ log 2 m⌉ = ⌈1+ log 2 m⌉ by linarith
  moreover have 1+ log 2 m = log 2 (2*m)
    using 4.hyps(1) deg-not-0 log-mult by force
  moreover hence ⌈1+ log 2 m⌉ = ⌈log 2 (2*m)⌉ by simp
  moreover hence ⌈log 2 (2*m)⌉ = ⌈log 2 (n+m)⌉
    using 4.hyps(3) by force
  ultimately show ?case
    by (metis 4.hyps(4) height.simps(2))
next
  case (5 treeList n summary m deg mi ma)
  hence 00: n ≥ 1 ∧ Suc n = m
    using set-n-deg-not-0 by blast
  hence 0:m ≥ nusing 5 by simp
  hence 1:log 2 m ≥ log 2 n
    using 5.IH(1) 5.hyps(2) set-n-deg-not-0 by fastforce
  hence 2:⌈ log 2 m⌉ ≥ ⌈ log 2 n⌉
    by (simp add: ceiling-mono)
  have 3: Max (height ' (insert summary (set treeList))) = ⌈ log 2 m⌉
```

using *5.IH(1) 5.IH(2) 5.hyps(3) List.finite-set Max-in empty-is-image finite-imageI finite-insert image-iff insert-iff insert-not-empty 5.hyps(1) setceilmax* **by** *auto*
hence *4:height (Node None deg treeList summary) = 1+ ⌈ log 2 m⌉* **by** *simp*
have *5:1+ ⌈ log 2 m⌉ = ⌈1+ log 2 m⌉* **by** *linarith*
have *6:1+ log 2 m = log 2 (m+m)*
using *5.hyps(1) deg-not-0 log-mult* **by** *force*
hence *7:log 2 (m+n) = 1+log 2 ((n+m) / 2)*
by (*simp add*: *5.hyps(3) log-divide*)
have *8:log 2 ((n+m) / 2) = log 2 (n + 1/2)*
by (*smt* (*verit*, *best*) *5.hyps(3) field-sum-of-halves of-nat-Suc of-nat-add*)
have *9* : ⌈ *log 2 (n + 1/2)* ⌉ = ⌈ *log 2* ⌈*n + 1/2* ⌉ ⌉
by (*smt* (*verit*) *00 field-sum-of-halves log-ceil-idem of-nat-1 of-nat-mono*)
hence *10*: ⌈*n + 1/2* ⌉ = *m* **using** *00* **by** *linarith*
hence *11*: ⌈ *log 2 (n + 1/2)* ⌉ = ⌈ *log 2 m* ⌉ **using** *9* **by** *simp*
hence *12*:⌈ *1+ log 2 (n + 1/2)* ⌉ = ⌈*1+ log 2 m* ⌉
by (*smt* (*verit*) *ceiling-add-one*)
hence ⌈ *log 2 (n + n+1)* ⌉ = ⌈ *log 2 (m+m)* ⌉
using *5.hyps(3) 6 7 8* **by** *force*
then show *?case*
using *4 5 5.hyps(3) 5.hyps(4) 6* **by** *force*
qed

lemma *two-powr-height-bound-deg*:
assumes *invar-vebt t n*
shows *2^(height t) ≤ 2∗(n::nat)*
proof−
have *(height t) = ⌈ log 2 n⌉*
by (*simp add*: *assms heigt-uplog-rel*)
moreover have ⌈ *log 2 n⌉ ≤ log 2 n +1* **by** *simp*
moreover hence *2 powr* ⌈ *log 2 n⌉ ≤ 2 powr (log 2 n +1)* **by** *simp*
moreover have *2 powr (log 2 n +1) = 2 powr 1 ∗ 2 powr (log 2 n)*
by (*simp add*: *powr-add*)
moreover hence *2 powr (log 2 n +1) = 2 ∗ n*
using *assms deg-not-0* **by** *force*
ultimately show *?thesis*
by (*metis linorder-not-less not-one-le-zero of-int-0 of-int-less-iff of-int-numeral of-int-of-nat-eq of-nat-le-iff one-add-one order-less-le powr-realpow real-of-nat-eq-numeral-power-cancel-iff zle-add1-eq-le*)
qed

Main Theorem

theorem *height-double-log-univ-size*:
assumes *u = 2^deg* **and** *invar-vebt t deg*
shows *height t ≤ 1 + lb (lb u)*
proof−
have *(height t) = ⌈lb deg⌉*
by (*simp add*: *assms(2) heigt-uplog-rel*)
have *2^(height t) ≤ 2 ∗ deg* **using** *assms(2) two-powr-height-bound-deg*[*of t deg*]
by (*meson dual-order.eq-iff dual-order.trans self-le-ge2-pow*)
hence *height t ≤ 1 + lb deg*

using ‹*int* (*height t*) = ⌈*lb* (*real deg*)⌉› **by** *linarith*
hence *height t* ≤ *1* + *lb* (*lb u*) **using** *assms* **by** *simp*
thus *?thesis* **by** *simp*
qed

lemma *height-compose-list*: *t*∈ *set treeList* ⟹
Max (*height* ‘ (*insert summary* (*set treeList*))) ≥ *height t*
apply(*induction treeList*) **apply** *simp*
by (*meson List.finite-set Max-ge finite-imageI finite-insert image-eqI subsetD subset-insertI*)

lemma *height-compose-child*: *t*∈ *set treeList* ⟹
height (*Node info deg treeList summary*) ≥ *1*+ *height t* **by** *simp*

lemma *height-compose-summary*: *height* (*Node info deg treeList summary*) ≥ *1*+ *height summary*
by *simp*

lemma *height-i-max*: *i* < *length x13* ⟹
height (*x13* ! *i*) ≤ *max foo* (*Max* (*height* ‘ *set x13*))
by (*meson List.finite-set Max-ge finite-imageI max.coboundedI2 nth-mem rev-image-eqI*)

lemma *max-ins-scaled*: *n*∗ *height x14* ≤ *m* + *n*∗ *Max* (*insert* (*height x14*) (*height* ‘ *set x13*))
by (*meson List.finite-set Max-ge finite-imageI finite-insert insertI1 mult-le-mono2 trans-le-add2*)

lemma *max-idx-list*:
assumes *i* < *length x13*
shows *n* ∗ *height* (*x13* !*i*) ≤ *Suc* (*Suc* (*n* ∗ *max* (*height x14*) (*Max* (*height* ‘ *set x13*))))
by (*metis assms height-i-max less-Suc-eq mult-le-mono2 nat-less-le*)

end
end

theory *VEBT-Bounds* **imports** *VEBT-Height VEBT-Member VEBT-Insert VEBT-Succ VEBT-Pred*
begin

11 Upper Bounds for canonical Functions: Relationships between Run Time and Tree Heights

11.1 Membership test

context begin

interpretation *VEBT-internal* **.**

fun T_{member}::*VEBT* ⇒ *nat* ⇒ *nat* **where**
T_{member} (*Leaf a b*) *x* = *2* + (*if x* = *0 then 1 else 1* +(*if x*=*1 then 1 else 1*))|
T_{member} (*Node None - - -*) *x* = *2*|
T_{member} (*Node - 0 - -*) *x* = *2*|
T_{member} (*Node - (Suc 0) - -*) *x* = *2*|
T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *2* + (

```
  if x = mi then 1 else 1+ (
  if x = ma then 1  else 1+(
  if x < mi then 1  else 1+ (
 if x > ma then 1 else 9 +
  (let
       h = high x (deg div 2);
      l = low x (deg div 2) in
    (if h < length treeList
       then 1 + T_member (treeList ! h) l
       else 1))))))


fun T_member'::VEBT ⇒ nat ⇒ nat where
  T_member' (Leaf a b) x = 1|
  T_member' (Node None - - -) x = 1|
  T_member' (Node - 0 - -) x = 1|
  T_member' (Node - (Suc 0) - -) x = 1|
  T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1+(
 if x = mi then 0 else (
  if x = ma then 0  else (
  if x < mi then 0  else  (
 if x > ma then 0 else if (x>mi ∧ x < ma) then
  (let
       h = high x (deg div 2);
      l = low x (deg div 2) in
       (if h < length treeList
        then  T_member' (treeList ! h) l
        else 0))
else 0))))


lemma height-node: invar-vebt  (Node (Some (mi, ma)) deg treeList summary) n
                ⟹ height  (Node (Some (mi, ma)) deg treeList summary) >= 1
  using height.simps(2) by presburger

theorem member-bound-height: invar-vebt t n ⟹ T_member t x ≤ (1+height t)*15
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then show ?case by simp
next
  case (2 treeList n summary m deg)
  then show ?case by simp
next
  case (3 treeList n summary m deg)
  then show ?case by simp
next
  case (4 treeList n summary m deg mi ma)
  then obtain k :: nat where
    n = Suc k and m = Suc k
```

```
  by (metis Suc-leI add-Suc-right add-Suc-shift deg-not-0 le-Suc-ex)
then have deg = Suc (Suc (k + k))
  using ‹deg = n + m› by presburger
then have deg ≥ 2
  by presburger
then show ?case
proof(cases x = mi)
  case True
  hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 3
    by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis
    by presburger
next
  case False
  hence x ≠ mi by simp
  hence 1:T_member (Node (Some (mi, ma)) deg treeList summary) x = 3 + (
         if x = ma then 1  else 1+(
         if x < mi then 1  else 1+ (
         if x > ma then 1 else 9 +
        (let h = high x (deg div 2); l = low x (deg div 2) in
            (if h < length treeList
             then 1 + T_member (treeList ! h) l
             else 1)))))
    by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis
  proof(cases x = ma)
    case True
    hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 4 using 1 by auto
    then show ?thesis by simp
  next
    case False
    hence x ≠ ma by simp
    hence 2:T_member (Node (Some (mi, ma)) deg treeList summary) x = 4 +(
              if x < mi then 1  else 1+ (
              if x > ma then 1 else 9 +
              (let
              h = high x (deg div 2);
              l = low x (deg div 2) in
             (if h < length treeList
              then 1 + T_member (treeList ! h) l
              else 1))))
      using 1 by simp
    then show ?thesis
    proof(cases x < mi)
      case True
      hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 5 using 2 by auto
      then show ?thesis by simp
    next
      case False
```

hence $x > mi$
using ‹$x \neq mi$› *antisym-conv3* **by** *blast*
hence *3*:T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x = *5* + (
if $x > ma$ *then 1 else 9* +
(*let* h = *high* x (*deg div 2*); l = *low* x (*deg div 2*) *in*
(*if* $h <$ *length treeList*
then 1 + T_{member} (*treeList* ! h) l
else 1)))
using *2* **by** *simp*
then show *?thesis*
proof(*cases* $x > ma$)
case *True*
hence T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x = *6* **using** *3* **by** *simp*
then show *?thesis* **by** *simp*
next
case *False*
hence $x < ma$
by (*meson* ‹$x \neq ma$› *nat-neq-iff*)
hence *4*:T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x = *14*+
(*let* h = *high* x (*deg div 2*); l = *low* x (*deg div 2*) *in*
(*if* $h <$ *length treeList*
then 1 + T_{member} (*treeList* ! h) l
else 1))
using *3* **by** *simp*
let *?h* = *high* x (*deg div 2*)
let *?l* = *low* x (*deg div 2*)
have *?h* < *length treeList*
using *4.hyps*(*2*) *4.hyps*(*3*) *4.hyps*(*4*) *4.hyps*(*8*) ‹$x < ma$› *high-bound-aux* **by** *force*
hence *5*:T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x = *15* + T_{member} (*treeList* ! *?h*) *?l*
using *4* **by** *presburger*
moreover have *invar-vebt* (*treeList* ! *?h*) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*
using *4.IH*(*1*) ‹*high x* (*deg div 2*) < *length treeList*› *nth-mem* **by** *blast*
moreover hence T_{member} (*treeList* ! *?h*) *?l* ≤ (*1* + *height* (*treeList* ! *?h*))∗*15* **using** *4.IH*(*1*) **by** *simp*
ultimately have *6*:T_{member} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) x ≤
15 + *15* ∗ (*1* + *height* (*treeList* ! *?h*)) **by** *simp*
moreover have *i*< *length treeList* ⟹
height (*treeList* ! *i*) ≤ *Max* (*height* ‘ (*insert summary* (*set treeList*))) **for** *i*
apply (*induction treeList arbitrary*: *i*)
apply *simp*
apply (*meson List.finite-set Max-ge finite-imageI finite-insert image-iff nth-mem subsetD subset-insertI*)
done
moreover hence (*1* + *height* (*treeList* ! *?h*)) ≤ *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*simp add*: ‹*high x* (*deg div 2*) < *length treeList*›)
moreover hence *14* ∗ (*1* + *height* (*treeList* ! *?h*)) ≤ *14* ∗ *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) **by** *simp*

```
        ultimately show ?thesis
          using 6 by simp
      qed
    qed
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  then have 1 ≤ n
    by (metis set-n-deg-not-0)
  then obtain n' :: nat where n = Suc n'
    by (metis le-Suc-ex plus-1-eq-Suc)
  then have deg = Suc (Suc (Suc (n' + n')))
    using ‹deg = n + m› ‹m = Suc n› by presburger
  then show ?case
  proof(cases x = mi)
    case True
    hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 3
      unfolding ‹deg = Suc (Suc -)›
      by simp
    then show ?thesis by simp
  next
    case False
    hence x ≠ mi by simp
    hence 1:T_member (Node (Some (mi, ma)) deg treeList summary) x = 3 + (
              if x = ma then 1  else 1+(
              if x < mi then 1  else 1+ (
              if x > ma then 1 else 9 +
             (let h = high x (deg div 2); l = low x (deg div 2) in
              (if h < length treeList
               then 1 + T_member (treeList ! h) l
               else 1)))))
      unfolding ‹deg = Suc (Suc -)›
      by simp
    then show ?thesis
    proof(cases x = ma)
      case True
      hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 4 using 1 by auto
      then show ?thesis by simp
    next
      case False
      hence x ≠ ma by simp
      hence 2:T_member (Node (Some (mi, ma)) deg treeList summary) x = 4 +(
               if x < mi then 1  else 1+ (
               if x > ma then 1 else 9 +
              (let h = high x (deg div 2); l = low x (deg div 2) in
              (if h < length treeList
               then 1 + T_member (treeList ! h) l
               else 1))))
```

```
    using 1 by simp
  then show ?thesis
  proof(cases x < mi)
    case True
    hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 5 using 2 by auto
    then show ?thesis by simp
  next
    case False
    hence x > mi
      using ‹x ≠ mi› antisym-conv3 by blast
    hence 3:T_member (Node (Some (mi, ma)) deg treeList summary) x = 5 + (
            if x > ma then 1 else 9 +
            (let h = high x (deg div 2); l = low x (deg div 2) in
            (if h < length treeList  then 1 + T_member (treeList ! h) l else 1)))
      using 2 by simp
    then show ?thesis
    proof(cases x > ma)
      case True
      hence T_member (Node (Some (mi, ma)) deg treeList summary) x = 6 using 3 by simp
      then show ?thesis by simp
    next
      case False
      hence x < ma
        by (meson ‹x ≠ ma› nat-neq-iff)
      hence 4:T_member (Node (Some (mi, ma)) deg treeList summary) x = 14+
              (let  h = high x (deg div 2); l = low x (deg div 2) in
              (if h < length treeList
               then 1 + T_member (treeList ! h) l
               else 1))
        using 3 by simp
      let ?h = high x (deg div 2)
      let ?l =  low x (deg div 2)
      have ?h < length treeList
          using 5(10,4) 5.hyps(4) ‹deg = Suc (Suc (Suc (n' + n')))› ‹n = Suc n'› ‹x < ma›
high-bound-aux by auto
     hence 5:T_member (Node (Some (mi, ma)) deg treeList summary) x = 15 + T_member (treeList
! ?h) ?l
        using 4 by presburger
      moreover have invar-vebt (treeList ! ?h) n ∧  (treeList ! ?h) ∈ set treeList
        using 5.IH(1) ‹high x (deg div 2) < length treeList› nth-mem by blast
        moreover hence  T_member (treeList ! ?h) ?l ≤  (1 + height (treeList ! ?h))*15 using
5.IH(1) by simp
      ultimately have 6:T_member (Node (Some (mi, ma)) deg treeList summary) x ≤
                    15  + 15 * (1 + height (treeList ! ?h))
        by simp
      moreover  have i< length treeList ⟹
          height (treeList ! i) ≤ Max (height ' (insert summary (set treeList))) for i
        apply (induction treeList arbitrary: i)
         apply simp
```

```
            apply (meson List.finite-set Max-ge finite-imageI finite-insert image-iff nth-mem subsetD
subset-insertI)
          done
        moreover hence (1 + height (treeList ! ?h)) ≤ height (Node (Some (mi, ma)) deg treeList
summary)
          by (simp add: ‹high x (deg div 2) < length treeList›)
        moreover hence 15 * (1 + height (treeList ! ?h)) ≤ 15 * height (Node (Some (mi, ma))
deg treeList summary) by simp
        ultimately show ?thesis
          using 6 by simp
      qed
    qed
  qed
 qed
qed

theorem member-bound-height': invar-vebt t n ⟹ T_member' t x ≤ (1+height t)
proof(induction t n arbitrary: x rule: invar-vebt.induct)
 case (4 treeList n summary m deg mi ma)
 then obtain n' :: nat where n = Suc n'
  by (metis deg-not-0 less-imp-Suc-add)
 then have deg = Suc (Suc (n' + n'))
  using 4 by linarith
 then have deg ≥ 2
  by linarith

 have n ≥ 1 ∧ m ≥ 1
  using 4 ‹n = Suc n'› by linarith

 show ?case
 proof(cases x = mi)
  case True
  hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1
   by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis by simp
 next
  case False
  hence x ≠ mi by simp
  hence 1:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 + (
            if x = ma then 0  else (
            if x < mi then 0  else (
            if x > ma then 0 else
            (let h = high x (deg div 2);  l = low x (deg div 2) in
            (if h < length treeList
               then T_member' (treeList ! h) l
               else 0)))))
   by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis
  proof(cases x = ma)
```

```
  case True
  hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 1 by auto
  then show ?thesis by simp
next
  case False
  hence x ≠ ma by simp
  hence 2:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 +(
              if x < mi then 0  else (
              if x > ma then 0 else
              (let h = high x (deg div 2); l = low x (deg div 2) in
              (if h < length treeList
               then  T_member' (treeList ! h) l
               else 0))))
    using 1 by simp
  then show ?thesis
  proof(cases x < mi)
    case True
    hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 2 by auto
    then show ?thesis by simp
  next
    case False
    hence x > mi
      using ‹x ≠ mi› antisym-conv3 by blast
    hence 3:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 + (
              if x > ma then 0 else
             (let h = high x (deg div 2); l = low x (deg div 2) in
             (if h < length treeList
              then T_member' (treeList ! h) l
              else 0)))
      using 2 by simp
    then show ?thesis
    proof(cases x > ma)
      case True
      hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 3 by simp
      then show ?thesis by simp
    next
      case False
      hence x < ma
        by (meson ‹x ≠ ma› nat-neq-iff)
      hence 4:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1+
                 (let h = high x (deg div 2); l = low x (deg div 2) in
                 (if h < length treeList
                  then T_member' (treeList ! h) l
                  else 0))
        using 3 by simp
      let ?h = high x (deg div 2)
      let ?l =  low x (deg div 2)
      have ?h < length treeList
        using 4.hyps(2) 4.hyps(3) 4.hyps(4) 4.hyps(8) ‹x < ma› high-bound-aux by force
```

```
        hence 5:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1+ T_member' (treeList
! ?h) ?l
          using 4 by presburger
        moreover have invar-vebt (treeList ! ?h) n ∧ (treeList ! ?h) ∈ set treeList
          using 4.IH(1) ‹high x (deg div 2) < length treeList› nth-mem by blast
         moreover hence  T_member' (treeList ! ?h) ?l ≤  (1 + height (treeList ! ?h))*1 using
4.IH(1) by simp
        ultimately have 6:T_member' (Node (Some (mi, ma)) deg treeList summary) x ≤
                        1 + (1 + height (treeList ! ?h)) by simp
        moreover  have i< length treeList ⟹
            height (treeList ! i) ≤ Max (height ' (insert summary (set treeList))) for i
          apply (induction treeList arbitrary: i)
           apply simp
            apply (meson List.finite-set Max-ge finite-imageI finite-insert image-iff nth-mem subsetD
subset-insertI)
          done
        moreover hence  (1 + height (treeList ! ?h)) ≤ height  (Node (Some (mi, ma)) deg treeList
summary)
          by (simp add: ‹high x (deg div 2) < length treeList›)
         moreover hence  14 * (1 + height (treeList ! ?h)) ≤ 14 * height  (Node (Some (mi, ma))
deg treeList summary) by simp
        ultimately show ?thesis
          using 6 by presburger
      qed
    qed
  qed
 qed
next
 case (5 treeList n summary m deg mi ma)
 then obtain n' :: nat where n = Suc n'
  by (metis nat-le-iff-add plus-1-eq-Suc set-n-deg-not-0)
 then have deg = Suc (Suc (Suc (n' + n')))
  using 5 by linarith
 then have deg ≥ 2
  by linarith

 have n ≥ 1 ∧ m ≥ 1
  using 5 ‹n = Suc n'› by linarith

 show ?case
 proof(cases x = mi)
  case True
  hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1
   by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis by simp
 next
  case False
  hence x ≠ mi by simp
  hence 1:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 + (
```

```
          if x = ma then 0  else (
          if x < mi then 0  else (
          if x > ma then 0 else
         (let h = high x (deg div 2); l = low x (deg div 2) in
         (if h < length treeList
          then T_member' (treeList ! h) l
           else 0)))))
  by (simp add: ‹deg = Suc (Suc -)›)
then show ?thesis
proof(cases x = ma)
  case True
  hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 1 by auto
  then show ?thesis by simp
next
  case False
  hence x ≠ ma by simp
  hence 2:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 +(
              if x < mi then 0  else (
              if x > ma then 0 else
             (let h = high x (deg div 2); l = low x (deg div 2) in
             (if h < length treeList
              then  T_member' (treeList ! h) l
              else 0))))
    using 1 by simp
  then show ?thesis
  proof(cases x < mi)
    case True
    hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 2 by auto
    then show ?thesis by simp
  next
    case False
    hence x > mi
      using ‹x ≠ mi› antisym-conv3 by blast
    hence 3:T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 + (
              if x > ma then 0 else
              (let h = high x (deg div 2); l = low x (deg div 2) in
              (if h < length treeList
               then T_member' (treeList ! h) l
               else 0)))
      using 2 by simp
    then show ?thesis
    proof(cases x > ma)
      case True
      hence T_member' (Node (Some (mi, ma)) deg treeList summary) x = 1 using 3 by simp
      then show ?thesis by simp
    next
      case False
      hence x < ma
        by (meson ‹x ≠ ma› nat-neq-iff)
```

hence *4*:$T_{member}{}'$ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *1*+
(*let h* = *high x* (*deg div 2*); *l* = *low x* (*deg div 2*) *in*
(*if h* < *length treeList*
then $T_{member}{}'$ (*treeList* ! *h*) *l*
else 0))
using *3* **by** *simp*
let *?h* = *high x* (*deg div 2*)
let *?l* = *low x* (*deg div 2*)
have *?h* < *length treeList*
using *5.hyps*(*2*) *5.hyps*(*3*) *5.hyps*(*4*) *5.hyps*(*8*) ‹*x* < *ma*› *high-bound-aux*
by (*metis add-Suc-right add-self-div-2 even-Suc-div-two odd-add order.strict-trans*)
hence *5*:$T_{member}{}'$ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *1*+ $T_{member}{}'$ (*treeList* ! *?h*) *?l*
using *4* **by** *presburger*
moreover have *invar-vebt* (*treeList* ! *?h*) *n* ∧ (*treeList* ! *?h*) ∈ *set treeList*
using *5.IH*(*1*) ‹*high x* (*deg div 2*) < *length treeList*› *nth-mem* **by** *blast*
moreover hence $T_{member}{}'$ (*treeList* ! *?h*) *?l* ≤ (*1* + *height* (*treeList* ! *?h*))∗*1* **using** *5.IH*(*1*) **by** *simp*
ultimately have *6*:$T_{member}{}'$ (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
1 + (*1* + *height* (*treeList* ! *?h*)) **by** *simp*
moreover have *i*< *length treeList* ⟹
height (*treeList* ! *i*) ≤ *Max* (*height* ‘ (*insert summary* (*set treeList*))) **for** *i*
apply (*induction treeList arbitrary*: *i*)
apply *simp*
apply (*meson List.finite-set Max-ge finite-imageI finite-insert image-iff nth-mem subsetD subset-insertI*)
done
moreover hence (*1* + *height* (*treeList* ! *?h*)) ≤ *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)
by (*simp add*: ‹*high x* (*deg div 2*) < *length treeList*›)
moreover hence *14* ∗ (*1* + *height* (*treeList* ! *?h*)) ≤ *14* ∗ *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) **by** *simp*
ultimately show *?thesis*
using *6* **by** *presburger*
qed
qed
qed
qed
qed *simp*+

theorem *member-bound-size-univ*: *invar-vebt t n* ⟹ *u* = *2*^*n* ⟹ T_{member} *t x* ≤ *30* + *15* ∗ *lb* (*lb u*)
using *member-bound-height*[*of t n x*] *height-double-log-univ-size*[*of u n t*] *algebra-simps* **by** *simp*

11.2 Minimum, Maximum, Emptiness Test

fun T_{mint}::*VEBT* ⇒ *nat* **where**
T_{mint} (*Leaf a b*) = (*1*+ (*if a then 0 else 1* + (*if b then 1 else 1*)))|
T_{mint} (*Node None* - - -) = *1*|

T_{mint} (*Node* (*Some* (*mi*,*ma*)) - - -) = *1*

lemma *mint-bound*: T_{mint} *t* ≤ *3* **by** (*induction t rule*: T_{mint}.*induct*) *auto*

fun T_{maxt}::*VEBT* ⇒ *nat* **where**
T_{maxt} (*Leaf a b*) = (*1*+ (*if b then 1 else 1* +(*if a then 1 else 1*)))|
T_{maxt} (*Node None* - - -) = *1*|
T_{maxt} (*Node* (*Some* (*mi*,*ma*)) - - -) = *1*

lemma *maxt-bound*: T_{maxt} *t* ≤ *3* **by** (*induction t rule*: T_{maxt}.*induct*) *auto*

fun $T_{minNull}$::*VEBT* ⇒ *nat* **where**
$T_{minNull}$ (*Leaf False False*) = *1*|
$T_{minNull}$ (*Leaf* - -) = *1*|
$T_{minNull}$ (*Node None* - - -) = *1*|
$T_{minNull}$ (*Node* (*Some* -) - - -) = *1*

lemma *minNull-bound*: $T_{minNull}$ *t* ≤ *1*
by (*metis* $T_{minNull}$.*elims order-refl*)

11.3 Insertion

fun T_{insert}::*VEBT* ⇒ *nat* ⇒*nat* **where**
T_{insert} (*Leaf a b*) *x* = *1*+ (*if x*=*0 then 1 else 1* + (*if x*=*1 then 1 else 1*))|
T_{insert} (*Node info 0 ts s*) *x* = *1*|
T_{insert} (*Node info* (*Suc 0*) *ts s*) *x* = *1*|
T_{insert} (*Node None* (*Suc deg*) *treeList summary*) *x* = *2*|
T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *19*+
(*let xn* = (*if x* < *mi then mi else x*); *minn* = (*if x*< *mi then x else mi*);
l= *low xn* (*deg div 2*); *h* = *high xn* (*deg div 2*)
in
(*if h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
T_{insert} (*treeList* ! *h*) *l* + $T_{minNull}$ (*treeList* ! *h*)+
(*if minNull* (*treeList* ! *h*) *then* T_{insert} *summary h else 1*)
else 1))

fun T_{insert}'::*VEBT* ⇒ *nat* ⇒*nat* **where**
T_{insert}' (*Leaf a b*) *x* = *1*|
T_{insert}' (*Node info 0 ts s*) *x* = *1*|
T_{insert}' (*Node info* (*Suc 0*) *ts s*) *x* = *1*|
T_{insert}' (*Node None* (*Suc deg*) *treeList summary*) *x* = *1*|
T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
(*let xn* = (*if x* < *mi then mi else x*); *minn* = (*if x*< *mi then x else mi*);
l= *low xn* (*deg div 2*); *h* = *high xn* (*deg div 2*)
in (*if h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
T_{insert}' (*treeList* ! *h*) *l* +

(*if minNull* (*treeList* ! *h*) *then* $T_{insert}{}'$ *summary h else 1*) *else 1*))

lemma *insersimp*:**assumes** *invar-vebt t n* **and** $\nexists$ *x. both-member-options t x* **shows** T_{insert} *t y* ≤ *3*
proof−
from *assms*(*1*) **show** *?thesis*
proof(*cases*)
case (*1 a b*)
then show *?thesis* **by** *simp*
next
case (*2 treeList n summary m*)
hence *n*+*m* ≥ *2*
by (*metis add-self-div-2 deg-not-0 div-greater-zero-iff*)
then show *?thesis* **using** T_{insert}*.simps*(*4*)[*of n*+*m*−*2 treeList summary y*]
by (*metis 2*(*1*) *2*(*6*) *add.commute add-2-eq-Suc le-add2 numeral-3-eq-3 ordered-cancel-comm-monoid-diff-class.add-d*
next
case (*3 treeList n summary m*)
hence *n*+*m* ≥ *2*
by (*metis add-mono-thms-linordered-semiring*(*1*) *le-add1 nat-1-add-1 plus-1-eq-Suc set-n-deg-not-0*)
then show *?thesis* **using** T_{insert}*.simps*(*4*)[*of n*+*m*−*2 treeList summary y*]
by (*metis 3*(*1*) *3*(*6*) *add.commute add-2-eq-Suc le-add2 numeral-3-eq-3 ordered-cancel-comm-monoid-diff-class.add-d*
next
case (*4 treeList n summary m mi ma*)
hence *membermima* (*Node* (*Some* (*mi*, *ma*)) (*n*+*m*) *treeList summary*) *mi*
by (*metis Suc-pred assms*(*1*) *deg-not-0 membermima.simps*(*4*))
hence *False*
using *4*(*1*) *4*(*6*) *assms*(*2*) *both-member-options-def* **by** *blast*
then show *?thesis* **by** *simp*
next
case (*5 treeList n summary m mi ma*)
hence *membermima* (*Node* (*Some* (*mi*, *ma*)) (*n*+*m*) *treeList summary*) *mi*
by (*metis Suc-pred assms*(*1*) *deg-not-0 membermima.simps*(*4*))
hence *False*
using *5*(*1*) *5*(*6*) *assms*(*2*) *both-member-options-def* **by** *blast*
then show *?thesis* **by** *simp*
qed
qed

lemma *insertsimp*: *invar-vebt t n* ⟹ *minNull t* ⟹ T_{insert} *t l* ≤ *3*
using *insersimp min-Null-member valid-member-both-member-options* **by** *blast*

lemma *insersimp′*:**assumes** *invar-vebt t n* **and** $\nexists$ *x. both-member-options t x* **shows** $T_{insert}{}'$ *t y* ≤ *1*
using *assms*(*1*)
proof (*cases rule*: *invar-vebt.cases*)
case (*1 a b*)
then show *?thesis* **by** *simp*
next
case (*2 treeList k summary m*)

then obtain k' :: *nat* **where** $k = Suc\ k'$
by (*metis Suc-lessE Suc-lessI deg-not-0*)
then show *?thesis*
using *2* **by** *simp*
next
case (*3 treeList k summary m*)
then obtain k' :: *nat* **where** $k = Suc\ k'$
by (*metis Suc-1 lessI list-decode.cases nth-mem power-Suc0-right valid-tree-deg-neq-0*)
then show *?thesis*
using *3* **by** *simp*
next
case (*4 treeList k summary m mi ma*)
then obtain k' :: *nat* **where** $k = Suc\ k'$
by (*metis list-decode.cases valid-tree-deg-neq-0*)
then show *?thesis*
using *4 assms*(*2*) *not-min-Null-member* **by** *simp*
next
case (*5 treeList k summary m mi ma*)
then obtain k' :: *nat* **where** $k = Suc\ k'$
by (*metis add.right-neutral add-Suc-right nth-mem old.nat.exhaust valid-0-not*)
then show *?thesis*
using *5 assms*(*2*) *not-min-Null-member* **by** *simp*
qed

lemma *insertsimp'*: *invar-vebt t n* $\Longrightarrow$ *minNull t* $\Longrightarrow$ $T_{insert}'\ t\ \ l \leq 1$
using *insersimp' min-Null-member valid-member-both-member-options* **by** *blast*

theorem *insert-bound-height*: *invar-vebt t n* $\Longrightarrow$ $T_{insert}\ t\ x \leq (1+height\ t)*23$
proof(*induction t n arbitrary*: *x rule*: *invar-vebt.induct*)
case (*1 a b*)
then show *?case*
using $T_{insert}.simps(1)[of\ a\ b\ x]$ *height.simps*(*1*)[*of a b*] **by** *simp+*
next
case (*2 treeList n summary m deg*)
hence *deg* $\geq$ *2*
by (*metis add-self-div-2 deg-not-0 div-greater-zero-iff*)
then obtain *deg'* :: *nat* **where** *deg* = *Suc* (*Suc deg'*)
using *add-2-eq-Suc le-Suc-ex* **by** *blast*
moreover have *height* (*Node None deg treeList summary*) $\geq$ *1* **using** $height.simps(2)[of\ None\ deg\ treeList\ summary]$
by *simp*
ultimately show *?case*
by *simp*
next
case (*3 treeList n summary m deg*)
hence *deg* $\geq$ *2*
by (*metis Suc-1 add-mono-thms-linordered-semiring*(*1*) *le-add1 plus-1-eq-Suc set-n-deg-not-0*)
then obtain *deg'* :: *nat* **where** *deg* = *Suc* (*Suc deg'*)
using *add-2-eq-Suc le-Suc-ex* **by** *blast*

moreover have *height* (*Node None deg treeList summary*) $\geq$ *1* **using** *height.simps*(*2*)[*of None deg treeList summary*] **by** *simp*
ultimately show *?case*
by *simp*
next
case (*4 treeList n summary m deg mi ma*)
hence *deg* $\geq$ *2*
by (*metis add-self-div-2 deg-not-0 div-greater-zero-iff*)
then obtain *deg'* :: *nat* **where** *deg* = *Suc* (*Suc deg'*)
using *add-2-eq-Suc le-Suc-ex* **by** *blast*
let *?xn* = (*if x* < *mi then mi else x*)
let *?minn* = (*if x*< *mi then x else mi*)
let *?l*= *low ?xn* (*deg div 2*)
let *?h* = *high ?xn* (*deg div 2*)
show *?case*
proof(*cases x* < *mi*)
case *True*
hence *0*:T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *19*+
(*if ?h* < *length treeList* $\wedge$ $\neg$ (*x* = *mi* $\vee$ *x* = *ma*)*then*
T_{insert} (*treeList* ! *?h*) *?l* + $T_{minNull}$ (*treeList* ! *?h*)+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*) *else 1*)
by (*simp add*: ‹*deg* = *Suc* (*Suc deg'*)› *Let-def*)
then show *?thesis*
proof(*cases ?h* < *length treeList* $\wedge$ $\neg$ (*x* = *mi* $\vee$ *x* = *ma*))
case *True*
hence *1*: T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *19*+
T_{insert} (*treeList* ! *?h*) *?l* + $T_{minNull}$ (*treeList* ! *?h*)+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *0* **by** *simp*
then show *?thesis*
proof(*cases minNull* (*treeList* ! *?h*))
case *True*
hence T_{insert} (*treeList* ! *?h*) *?l* $\leq$ *3*
using *4.IH*(*1*) ‹*?h* < *length treeList* $\wedge$ -›
by (*simp add*: *insertsimp*[*of* - *n*])
hence *2*: T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* $\leq$ *22* +
$T_{minNull}$ (*treeList* ! *?h*)+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *1* **by** *linarith*
also have ... $\leq$ *23* + (*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *minNull-bound* **by** *force*
also have ... $\leq$ *23* + T_{insert} *summary ?h*
using *True* **by** *simp*
also have ... $\leq$ *23* + (*height summary* +*1*)∗*23*
using *4.IH*(*2*)[*of ?h*] **by** *presburger*
also have ... = ((*1*+ *height summary*)+*1*)∗*23*
by *presburger*
also have ... $\leq$ (*1* + *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *23*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*

```
    finally show ?thesis .
  next
    case False
    hence 2:T_insert (Node (Some (mi,ma)) deg treeList summary) x = 20+
          T_insert (treeList ! ?h) ?l + T_minNull  (treeList ! ?h) using 1 by simp
    also have ... ≤ 23 + T_insert (treeList ! ?h) ?l
      using minNull-bound[of treeList ! ?h] by linarith
    also have ... ≤ 23 + (1 + height (treeList ! ?h)) * 23
      using 4.IH(1) True by simp
    also have ... ≤ ((1 + height (treeList!?h)) + 1) * 23
      by simp
    also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 23
      using height-compose-child[of treeList!?h treeList Some (mi, ma) deg summary]
      by (metis Suc-eq-plus1 True mult-le-mono1 nat-add-left-cancel-le nth-mem plus-1-eq-Suc)
    finally show ?thesis .
  qed
 next
  case False
  then show ?thesis
    using True
    by (auto simp add: ‹deg = Suc (Suc -)›)
 qed
next
 case False
 hence 0:T_insert (Node (Some (mi,ma)) deg treeList summary) x = 19+
        ( if ?h < length treeList ∧ ¬ (x = mi ∨ x = ma)then
          T_insert (treeList ! ?h) ?l + T_minNull  (treeList ! ?h)+
                    (if minNull (treeList ! ?h) then  T_insert summary ?h else 1) else 1)
  by (simp add: ‹deg = Suc (Suc -)› Let-def)
 then show ?thesis
 proof(cases  ?h < length treeList ∧ ¬ (x = mi ∨ x = ma))
  case True
  hence 1: T_insert (Node (Some (mi,ma)) deg treeList summary) x = 19+
          T_insert (treeList ! ?h) ?l + T_minNull  (treeList ! ?h)+
                    (if minNull (treeList ! ?h) then  T_insert summary ?h else 1)
    using 0 by simp
  then show ?thesis
  proof(cases  minNull (treeList ! ?h))
    case True
    hence  T_insert (treeList ! ?h) ?l ≤ 3
      using 4.IH(1) ‹?h <  length treeList ∧ -›
      by (simp add: insertsimp[of - n])
    hence 2: T_insert (Node (Some (mi,ma)) deg treeList summary) x ≤ 22 +
          T_minNull  (treeList ! ?h)+
                    (if minNull (treeList ! ?h) then  T_insert summary ?h else 1)
      using 1 by simp
    also have ... ≤ 23 + (if minNull (treeList ! ?h) then T_insert summary ?h else 1)
      using minNull-bound by force
    also have ... ≤ 23 + T_insert summary ?h
```

```
        using True by simp
      also have ... ≤ 23 + (height summary + 1) * 23
        using 4.IH(2) by simp
      also have ... ≤ ((1 + height summary) + 1 ) * 23
        by simp
      also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 23
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis .
    next
      case False
      hence 2:T_insert (Node (Some (mi,ma)) deg treeList summary) x = 20+
            T_insert (treeList ! ?h) ?l + T_minNull (treeList ! ?h) using 1 by simp
      also have ... ≤ 23 + T_insert (treeList ! ?h) ?l
        using minNull-bound[of treeList ! ?h] by linarith
      also have ... ≤ 23 + (1 + height (treeList ! ?h)) * 23
        using 4.IH(1) True by simp
      also have ... ≤ ((1 + height (treeList!?h)) + 1) * 23
        by simp
      also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 23
        using height-compose-child[of treeList!?h treeList Some (mi, ma) deg summary]
        by (metis Suc-eq-plus1 True mult-le-mono1 nat-add-left-cancel-le nth-mem plus-1-eq-Suc)
      finally show ?thesis .
    qed
  next
    case False
    then show ?thesis
      using 0 by force
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis One-nat-def Suc-eq-plus1 less-natE not-gr0 nth-mem valid-0-not)
  then have deg = Suc (Suc (Suc (n' + n')))
    unfolding ‹deg = n + m› ‹m = Suc n› by presburger
  then have 2 ≤ deg
    by linarith
  let ?xn = (if x < mi then mi else x)
  let ?minn = (if x< mi then x else mi)
  let ?l= low ?xn (deg div 2)
  let ?h = high ?xn (deg div 2)
  show ?case
  proof(cases x < mi)
    case True
    hence 0:T_insert (Node (Some (mi,ma)) deg treeList summary) x = 19+
        ( if ?h < length treeList ∧ ¬ (x = mi ∨ x = ma)then
        T_insert (treeList ! ?h) ?l + T_minNull (treeList ! ?h)+
        (if minNull (treeList ! ?h) then T_insert summary ?h else 1) else 1)
      by (simp add: ‹deg = Suc (Suc -)› Let-def)
```

```
  show ?thesis
  proof(cases  ?h < length treeList ∧ ¬ (x = mi ∨ x = ma))
    case True
    hence 1: T_insert (Node (Some (mi,ma)) deg treeList summary) x = 19+
           T_insert (treeList ! ?h) ?l + T_minNull  (treeList ! ?h)+
           (if minNull (treeList ! ?h) then  T_insert summary ?h else 1)
      using 0 by simp
    then show ?thesis
    proof(cases  minNull (treeList ! ?h))
      case True
      hence  T_insert (treeList ! ?h) ?l ≤ 3
        using 5.IH(1) ‹?h < length treeList ∧ -› 
        by (simp add: insertsimp[of - n])
      hence 2: T_insert (Node (Some (mi,ma)) deg treeList summary) x ≤ 22 +
           T_minNull  (treeList ! ?h)+(if minNull (treeList ! ?h) then  T_insert summary ?h else 1)
        using 1 by linarith
      also have ... ≤ 23 + (if minNull (treeList ! ?h) then T_insert summary ?h else 1)
        using minNull-bound[of treeList ! ?h] by linarith
      also have ... ≤ 23 + T_insert summary ?h
        using True by simp
      also have ... ≤ 23 + (height summary + 1) * 23
        using 5.IH(2) by simp
      also have ... ≤ ((1 + height summary) + 1) *23
        by simp
      also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 23
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis .
    next
      case False
      hence 2:T_insert (Node (Some (mi,ma)) deg treeList summary) x = 20+
           T_insert (treeList ! ?h) ?l + T_minNull  (treeList ! ?h) using 1 by simp
      also have ... ≤ 23 + T_insert (treeList ! ?h) ?l
        using minNull-bound[of treeList ! ?h] by linarith
      also have ... ≤ 23 + (1 + height (treeList ! ?h)) * 23
        using 5.IH(1) True by simp
      also have ... ≤ ((1 + height (treeList!?h)) + 1) * 23
        by simp
      also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 23
        using height-compose-child[of treeList!?h treeList Some (mi, ma) deg summary]
        by (metis Suc-eq-plus1 True mult-le-mono1 nat-add-left-cancel-le nth-mem plus-1-eq-Suc)
      finally show ?thesis .
    qed
  next
    case False
    then show ?thesis
      using 0 by presburger
  qed
next
  case False
```

hence *0*:T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *19*+
(*if ?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
T_{insert} (*treeList* ! *?h*) *?l* + $T_{minNull}$ (*treeList* ! *?h*)+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*) *else 1*)
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)› *Let-def*)
show *?thesis*
proof(*cases* *?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*))
case *True*
hence *1*: T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *19*+
T_{insert} (*treeList* ! *?h*) *?l* + $T_{minNull}$ (*treeList* ! *?h*)+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *0* **by** *simp*
then show *?thesis*
proof(*cases* *minNull* (*treeList* ! *?h*))
case *True*
hence T_{insert} (*treeList* ! *?h*) *?l* ≤ *3*
using *5.IH*(*1*) ‹*?h* < *length treeList* ∧ -›
by (*simp add*: *insertsimp*[*of* - *n*])
hence *2*: T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *22* +
$T_{minNull}$ (*treeList* ! *?h*)+(*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *1* **by** *linarith*
also have ... ≤ *23* + (*if minNull* (*treeList* ! *?h*) *then* T_{insert} *summary ?h else 1*)
using *minNull-bound*[*of treeList* ! *?h*] **by** *linarith*
also have ... ≤ *23* + T_{insert} *summary ?h*
using *True* **by** *simp*
also have ... ≤ *23* + (*height summary* + *1*) ∗ *23*
using *5.IH*(*2*) **by** *simp*
also have ... ≤ ((*1* + *height summary*) + *1*) ∗ *23*
by *simp*
also have ... ≤ (*1* + *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *23*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*
finally show *?thesis* **.**
next
case *False*
hence *2*:T_{insert} (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *20*+
T_{insert} (*treeList* ! *?h*) *?l* + $T_{minNull}$ (*treeList* ! *?h*) **using** *1* **by** *simp*
also have ... ≤ *23* + T_{insert} (*treeList* ! *?h*) *?l*
using *minNull-bound*[*of treeList* ! *?h*] **by** *linarith*
also have ... ≤ *23* + (*1* + *height* (*treeList* ! *?h*)) ∗ *23*
using *5.IH*(*1*) *True* **by** *simp*
also have ... ≤ ((*1* + *height* (*treeList*!*?h*)) + *1*) ∗ *23*
by *simp*
also have ... ≤ (*1* + *height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *23*
using *True* **by** *simp*
finally show *?thesis* **.**
qed
next
case *False*
then show *?thesis*

```
      using 0 by force
    qed
  qed
qed


theorem insert-bound-size-univ: invar-vebt t n ⟹ u = 2^n ⟹ T_insert t x ≤ 46 + 23 * lb (lb u)
  using insert-bound-height[of t n x] height-double-log-univ-size[of u n t] algebra-simps by simp

theorem insert'-bound-height: invar-vebt t n ⟹ T_insert' t x ≤ (1+height t)
proof(induction t n arbitrary: x rule: invar-vebt.induct )
  case (2 treeList n summary m deg)
  show ?case
  proof (cases deg ≥ 2)
    case True
    then obtain deg' :: nat where deg = Suc (Suc deg')
      using add-2-eq-Suc le-Suc-ex by blast
    then show ?thesis
      by simp
  next
    case False
    then show ?thesis
      using 2 by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  qed
next
  case (3 treeList n summary m deg)
  show ?case
  proof (cases deg ≥ 2)
    case True
    then obtain deg' :: nat where deg = Suc (Suc deg')
      using add-2-eq-Suc le-Suc-ex by blast
    then show ?thesis
      by simp
  next
    case False
    then show ?thesis
      using 3 by (metis Suc-1 add-leD2 not-less-eq-eq set-n-deg-not-0)
  qed
next
  case (4 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis deg-not-0 less-natE)
  then have deg = Suc (Suc (n' + n'))
    using 4 by linarith
  then have deg ≥ 2
    by linarith
  let ?xn = (if x < mi then mi else x)
  let ?minn = (if x< mi then x else mi)
  let ?l= low ?xn (deg div 2)
```

```
 let ?h = high ?xn (deg div 2)
 show ?case
 proof(cases x < mi)
   case True
   hence 0:T_insert' (Node (Some (mi,ma)) deg treeList summary) x =
           ( if ?h < length treeList ∧ ¬ (x = mi ∨ x = ma)then
             T_insert' (treeList ! ?h) ?l + (if minNull (treeList ! ?h) then  T_insert' summary ?h else
1) else 1)
     by (simp add: ‹deg = Suc (Suc -)› Let-def)
   then show ?thesis
   proof(cases  ?h <  length treeList ∧ ¬ (x = mi ∨ x = ma))
     case True
     hence 1: T_insert' (Node (Some (mi,ma)) deg treeList summary) x =
     T_insert' (treeList ! ?h) ?l +(if minNull (treeList ! ?h) then  T_insert' summary ?h else 1)
       using 0 by simp
     then show ?thesis
     proof(cases  minNull (treeList ! ?h))
       case True
       hence  T_insert' (treeList ! ?h) ?l ≤ 1
         by (metis 0 1 4.IH(1) insertsimp' nat-le-iff-add nth-mem)
       hence 2: T_insert' (Node (Some (mi,ma)) deg treeList summary) x ≤ 1+
                           (if minNull (treeList ! ?h) then  T_insert' summary ?h else 1)
         using 1 by linarith
       also have ... ≤ 1 + T_insert' summary ?h
         using True by simp
       also have ... ≤ 1 + (height summary +1)
         using 4.IH(2) by simp
       finally show ?thesis
         using height-compose-summary[of summary Some (mi, ma) deg treeList] by linarith
     next
       case False
       hence 2:T_insert' (Node (Some (mi,ma)) deg treeList summary) x =
          1+   T_insert' (treeList ! ?h) ?l  using 1 by simp
        hence T_insert' (Node (Some (mi,ma)) deg treeList summary) x ≤ 1+ (1+ height (treeList !
?h))
         using 4.IH(1) True by force
       moreover have  (treeList!?h) ∈ set treeList
         using True nth-mem by blast
      ultimately show ?thesis using height-compose-child[of treeList!?h treeList Some (mi, ma) deg
summary] algebra-simps
          by (smt (verit, ccfv-SIG) Suc-leI add.right-neutral le-add1 le-imp-less-Suc mult-le-mono
order-trans plus-1-eq-Suc)
     qed
   next
     case False
     then show ?thesis   using 0 Suc-eq-plus1 le-add2 plus-1-eq-Suc by presburger
   qed
 next
   case False
```

hence *0*:T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
(*if ?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
T_{insert}' (*treeList* ! *?h*) *?l* + (*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*) *else 1*)
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)› *Let-def*)
then show *?thesis*
proof(*cases* *?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*))
case *True*
hence *1*: T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
T_{insert}' (*treeList* ! *?h*) *?l* +
(*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*)
using *0* **by** *simp*
then show *?thesis*
proof(*cases* *minNull* (*treeList* ! *?h*))
case *True*
hence T_{insert}' (*treeList* ! *?h*) *?l* ≤ *1*
using *4*.*IH*(*1*) ‹*?h* < *length treeList* ∧ -› *insertsimp'*[*of* - *n*] **by** *simp*
hence *2*: T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+
(*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*)
using *1* **by** *linarith*
hence T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1* + T_{insert}' *summary ?h*
using *True* **by** *simp*
then show *?thesis* **using** *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*]
algebra-simps
by (*smt* (*verit*) *1 4*.*IH*(*2*) *True* ‹T_{insert}' (*treeList* ! *high* (*if x* < *mi then mi else x*) (*deg div 2*)) (*low* (*if x* < *mi then mi else x*) (*deg div 2*)) ≤ *1*› *add-mono order-trans*)
next
case *False*
hence *2*:T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *1*+
T_{insert}' (*treeList* ! *?h*) *?l* **using** *1* **by** *simp*
hence T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+ T_{insert}' (*treeList* ! *?h*)
?l
using *minNull-bound*[*of treeList* ! *?h*] **by** *linarith*
hence T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+ (*1*+ *height* (*treeList* !
?h))
by (*meson 4*.*IH*(*1*) *True nat-add-left-cancel-le nth-mem order-trans*)
moreover have (*treeList*!*?h*) ∈ *set treeList*
using *True nth-mem* **by** *blast*
ultimately show *?thesis* **using** *height-compose-child*[*of treeList*!*?h treeList Some* (*mi*, *ma*) *deg*
summary] *algebra-simps*
by (*smt* (*verit*, *ccfv-SIG*) *Suc-leI add.right-neutral le-add1 le-imp-less-Suc mult-le-mono*
order-trans plus-1-eq-Suc)
qed
next
case *False*
then show *?thesis*
using *0* **by** *force*
qed
qed

next
case (*5 treeList n summary m deg mi ma*)
then obtain n' :: *nat* **where** $n = Suc\ n'$
by (*metis add.right-neutral add-Suc-right nth-mem old.nat.exhaust valid-0-not*)
then have $deg = Suc\ (Suc\ (Suc\ (n' + n')))$
using *5* **by** *linarith*
then have $deg \geq 2$
by *linarith*
let *?xn* = (*if* $x < mi$ *then mi else x*)
let *?minn* = (*if* $x < mi$ *then x else mi*)
let *?l*= *low ?xn* (*deg div 2*)
let *?h* = *high ?xn* (*deg div 2*)
show *?case*
proof(*cases* $x < mi$)
case *True*
hence *0*:T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) x =
(*if ?h* < *length treeList* ∧ ¬ ($x = mi \vee x = ma$)*then*
T_{insert}' (*treeList* ! *?h*) *?l* +
(*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*)
else 1)
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)› *Let-def*)
then show *?thesis*
proof(*cases* *?h* < *length treeList* ∧ ¬ ($x = mi \vee x = ma$))
case *True*
hence *1*: T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) x =
T_{insert}' (*treeList* ! *?h*) *?l* + (*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*)
using *0* **by** *simp*
then show *?thesis*
proof(*cases* *minNull* (*treeList* ! *?h*))
case *True*
hence T_{insert}' (*treeList* ! *?h*) *?l* ≤ *1*
by (*metis 0 1 5.IH*(*1*) *insertsimp' nat-le-iff-add nth-mem*)
hence *2*: T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) $x \leq 1+$
(*if minNull* (*treeList* ! *?h*) *then* T_{insert}' *summary ?h else 1*)
using *1* **by** *linarith*
also have ... ≤ *1* + T_{insert}' *summary ?h*
using *True* **by** *simp*
also have ... ≤ *1* + (*height summary* +*1*)
using *5.IH*(*2*) **by** *simp*
finally show *?thesis*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *linarith*
next
case *False*
hence *2*:T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) x =
1+ T_{insert}' (*treeList* ! *?h*) *?l* **using** *1* **by** *simp*
hence T_{insert}' (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) $x \leq 1+$ (*1*+ *height* (*treeList* !
?h))
using *5.IH*(*1*) *True* **by** *force*
moreover have (*treeList*!*?h*) ∈ *set treeList*

using *True nth-mem* **by** *blast*
ultimately show *?thesis* **using** *height-compose-child*[*of treeList*!*?h treeList Some* (*mi*, *ma*) *deg summary*] *algebra-simps*
by (*smt* (*verit*, *ccfv-SIG*) *Suc-leI add.right-neutral le-add1 le-imp-less-Suc mult-le-mono order-trans plus-1-eq-Suc*)
qed
next
case *False*
then show *?thesis* **using** *0 Suc-eq-plus1 le-add2 plus-1-eq-Suc* **by** *presburger*
qed
next
case *False*
hence *0*:$T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
(*if ?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*)*then*
$T_{insert}{}'$ (*treeList* ! *?h*) *?l* + (*if minNull* (*treeList* ! *?h*) *then* $T_{insert}{}'$ *summary ?h else 1*) *else 1*)
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)› *Let-def*)
then show *?thesis*
proof(*cases* *?h* < *length treeList* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*))
case *True*
hence *1*: $T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* =
$T_{insert}{}'$ (*treeList* ! *?h*) *?l* + (*if minNull* (*treeList* ! *?h*) *then* $T_{insert}{}'$ *summary ?h else 1*)
using *0* **by** *simp*
then show *?thesis*
proof(*cases* *minNull* (*treeList* ! *?h*))
case *True*
hence $T_{insert}{}'$ (*treeList* ! *?h*) *?l* ≤ *1*
by (*smt* (*verit*) *0 1 5.IH*(*1*) *insertsimp*′ *le-add1 nat-add-left-cancel-le nth-mem numeral-3-eq-3 order-trans plus-1-eq-Suc*)
hence *2*: $T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+
(*if minNull* (*treeList* ! *?h*) *then* $T_{insert}{}'$ *summary ?h else 1*)
using *1* **by** *linarith*
hence $T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1* + $T_{insert}{}'$ *summary ?h*
using *True* **by** *simp*
then show *?thesis*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] *algebra-simps*
by (*smt* (*verit*) *1 5.IH*(*2*) *True* ‹$T_{insert}{}'$ (*treeList* ! *high* (*if x* < *mi then mi else x*) (*deg div 2*)) (*low* (*if x* < *mi then mi else x*) (*deg div 2*)) ≤ *1*› *add-mono order-trans*)
next
case *False*
hence *2*:$T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* = *1*+
$T_{insert}{}'$ (*treeList* ! *?h*) *?l* **using** *1* **by** *simp*
hence $T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+ $T_{insert}{}'$ (*treeList* ! *?h*) *?l*
using *minNull-bound*[*of treeList* ! *?h*] **by** *linarith*
hence $T_{insert}{}'$ (*Node* (*Some* (*mi*,*ma*)) *deg treeList summary*) *x* ≤ *1*+ (*1*+ *height* (*treeList* ! *?h*))
by (*meson 5.IH*(*1*) *True nat-add-left-cancel-le nth-mem order-trans*)

moreover have (*treeList*!*?h*) ∈ *set treeList*
using *True nth-mem* **by** *blast*
ultimately show *?thesis* **using** *height-compose-child*[*of treeList*!*?h treeList Some* (*mi*, *ma*) *deg summary*] *algebra-simps*
by (*smt* (*verit*, *ccfv-SIG*) *Suc-leI add.right-neutral le-add1 le-imp-less-Suc mult-le-mono order-trans plus-1-eq-Suc*)
qed
next
case *False*
then show *?thesis*
using *0* **by** *force*
qed
qed
qed *simp+*

11.4 Successor Function

fun T_{succ}::*VEBT* ⇒ *nat* ⇒ *nat* **where**
T_{succ} (*Leaf - b*) *0* = *1+* (*if b then 1 else 1*)|
T_{succ} (*Leaf - -*) (*Suc n*) = *1*|
T_{succ} (*Node None - - -*) *- =* 1|
T_{succ} (*Node - 0 - -*) *- =* 1|
T_{succ} (*Node - (Suc 0) - -*) *- =* 1|
T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *1+* (
if x < *mi then 1*
else (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in 10* +
(*if h* < *length treeList then 1+* T_{maxt} (*treeList* ! *h*) + (
let maxlow = *vebt-maxt* (*treeList* ! *h*) *in 3* +
(*if maxlow* ≠ *None* ∧ (*Some l* <$_o$ *maxlow*) *then*
4 + T_{succ} (*treeList* ! *h*) *l*
else let sc = *vebt-succ summary h in 1+* T_{succ} *summary h* + *1* + (
if sc = *None then 1*
else (*4* + T_{mint} (*treeList* ! *the sc*)))))

else 1)))

fun T_{succ}'::*VEBT* ⇒ *nat* ⇒ *nat* **where**
T_{succ}' (*Leaf - b*) *0* = *1*|
T_{succ}' (*Leaf - -*) (*Suc n*) = *1*|
T_{succ}' (*Node None - - -*) *- =* 1|
T_{succ}' (*Node - 0 - -*) *- =* 1|
T_{succ}' (*Node - (Suc 0) - -*) *- =* 1|
T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =(
if x < *mi then 1*
else (*let l* = *low x* (*deg div 2*); *h* = *high x* (*deg div 2*) *in*
(*if h* < *length treeList then* (
let maxlow = *vebt-maxt* (*treeList* ! *h*) *in*
(*if maxlow* ≠ *None* ∧ (*Some l* <$_o$ *maxlow*) *then*
1+ T_{succ}' (*treeList* ! *h*) *l*

```
                    else let sc = vebt-succ summary h in  T_succ' summary h + (
                    if sc = None then 1
                    else 1 )))
                else 1)))

theorem succ-bound-height: invar-vebt t n ⟹ T_succ t x ≤ (1+height t)*27
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then show ?case using  T_succ.simps(1)[of a b]
  proof −
    have ∀ b v ba n. T_succ v n = 1 ∨ Leaf b ba ≠ v ∨ 0 = n
      using T_succ.elims by blast
    then show ?thesis
    by (metis (no-types) Nat.add-0-right ‹T_succ (Leaf a b) 0 = 1 + (if b then 1 else 1)› height.simps(1)
nat-mult-1 numeral-le-iff one-add-one one-le-numeral semiring-norm(68) semiring-norm(72))
  qed
next
  case (2 treeList n summary m deg)
  then show ?case by simp
next
  case (3 treeList n summary m deg)
  then show ?case  by simp
next
  case (4 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis deg-not-0 less-imp-Suc-add)
  then have deg = Suc (Suc (n' + n'))
    unfolding ‹deg = n + m› ‹m = n› by presburger
  then show ?case
  proof(cases x < mi)
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    show ?thesis
    proof(cases ?h < length treeList)
      case True
      hence ?h < length treeList by simp
      hence 0:T_succ (Node (Some (mi, ma)) deg treeList summary) x =12 + T_maxt  (treeList ! ?h)
+ (
                    let maxlow = vebt-maxt (treeList ! ?h) in 3 +
                    (if maxlow ≠ None ∧ (Some ?l <_o  maxlow) then
                                4 +  T_succ (treeList ! ?h) ?l
                     else let sc = vebt-succ summary ?h in 1+  T_succ summary ?h + 1 + (
                     if sc = None then 1
                     else (4 + T_mint (treeList ! the sc) ))))
```

```
    using False
    by (simp add: ‹deg = Suc (Suc -)› Let-def)
  let ?maxlow= vebt-maxt (treeList ! ?h)
  let ?sc=vebt-succ summary ?h
  have 1:T_succ (Node (Some (mi, ma)) deg treeList summary) x =15 + T_maxt (treeList ! ?h) +
                  (if ?maxlow ≠ None ∧ (Some ?l <_o ?maxlow) then
                                  4 + T_succ (treeList ! ?h) ?l
                   else 2+ T_succ summary ?h + (
                   if ?sc = None then 1
                   else (4 + T_mint (treeList ! the ?sc)))) using 0 by auto
  then show ?thesis
  proof(cases ?maxlow ≠ None ∧ (Some ?l <_o ?maxlow))
    case True
    hence T_succ (Node (Some (mi, ma)) deg treeList summary) x =
          19 + T_maxt (treeList ! ?h) + T_succ (treeList ! ?h) ?l
      using 1 by simp
    hence T_succ (Node (Some (mi, ma)) deg treeList summary) x ≤
         22 + T_succ (treeList ! ?h) ?l using maxt-bound[of treeList ! ?h]
      by simp
    moreover have a:treeList ! ?h ∈ set treeList
      by (simp add: ‹high x (deg div 2) < length treeList›)
    ultimately have T_succ (Node (Some (mi, ma)) deg treeList summary) x ≤
          22 + (1+ height (treeList ! ?h))*27
      by (meson 4.IH(1) nat-add-left-cancel-le order-trans)
    hence T_succ (Node (Some (mi, ma)) deg treeList summary) x ≤
          ((1+ height (treeList ! ?h))+1)*27 by simp
    then show ?thesis
      using height-compose-child[of treeList ! ?h treeList Some (mi, ma) deg summary] a
      by (smt (verit) Suc-leI add.commute dual-order.strict-trans2 le-imp-less-Suc linorder-not-less
mult.commute mult-le-mono2 plus-1-eq-Suc)
  next
    case False
    then have 2:T_succ (Node (Some (mi, ma)) deg treeList summary) x =17 + T_maxt (treeList
! ?h) +
                  T_succ summary ?h + (
                  if ?sc = None then 1
                  else (4 + T_mint (treeList ! the ?sc)))
      using 1 by auto
    then show ?thesis
    proof(cases ?sc = None)
      case True
      hence 3:T_succ (Node (Some (mi, ma)) deg treeList summary) x =
            18 + T_maxt (treeList ! ?h) + T_succ summary ?h
        using 2 by simp
      hence T_succ (Node (Some (mi, ma)) deg treeList summary) x ≤ 21 + T_succ summary ?h
        using maxt-bound[of treeList ! ?h] by simp
      also have ... ≤ 21 + (1 + height summary) * 27
        using 4.IH(2) by simp
      also have ... ≤ 21 + height (Node (Some (mi, ma)) deg treeList summary) * 27
```

```
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by presburger
    next
      case False
      then have T_succ (Node (Some (mi, ma)) deg treeList summary) x =
        21 + T_maxt (treeList ! ?h) + T_succ summary ?h + T_mint (treeList ! the ?sc)
        using 2 by simp
      also have ... ≤ 27 + T_succ summary ?h
        using maxt-bound[of treeList ! ?h] mint-bound[of treeList ! the ?sc] by linarith
      also have ... ≤ 27 + (1 + height summary) * 27
        using 4.IH(2) by simp
      also have ... ≤ ((1 + height summary) + 1) * 27
        by simp
      also have ... ≤ (1 + height (Node (Some (mi, ma)) deg treeList summary)) * 27
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis .
    qed
  qed
next
  case False
  hence  T_succ (Node (Some (mi, ma)) deg treeList summary) x = 12
    using ‹¬ x < mi›
    by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis
    by simp
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis One-nat-def Suc-eq-plus1 less-iff-Suc-add not-gr0 nth-mem valid-0-not)
  then have deg = Suc (Suc (Suc (n' + n')))
    unfolding ‹deg = n + m› ‹m = Suc n› by presburger
  then have deg ≥ 2
    by linarith
  then show ?case
  proof(cases x < mi)
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    show ?thesis
    proof(cases ?h < length treeList)
      case True
      hence ?h < length treeList by simp
```

hence *0*:T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =*12* + T_{maxt} (*treeList* ! *?h*) + (
let maxlow = *vebt-maxt* (*treeList* ! *?h*) *in 3* +
(*if maxlow* ≠ *None* ∧ (*Some ?l* <$_o$ *maxlow*) *then*
4 + T_{succ} (*treeList* ! *?h*) *?l*
else let sc = *vebt-succ summary ?h in 1*+ T_{succ} *summary ?h* + *1* + (
if sc = *None then 1*
else (*4* + T_{mint} (*treeList* ! *the sc*)))))
using *False*
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)› *Let-def*)
let *?maxlow*= *vebt-maxt* (*treeList* ! *?h*)
let *?sc*=*vebt-succ summary ?h*
have *1*:T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =*15* + T_{maxt} (*treeList* ! *?h*) +
(*if ?maxlow* ≠ *None* ∧ (*Some ?l* <$_o$ *?maxlow*) *then*
4 + T_{succ} (*treeList* ! *?h*) *?l*
else 2+ T_{succ} *summary ?h* + (
if ?sc = *None then 1*
else (*4* + T_{mint} (*treeList* ! *the ?sc*))))
by (*simp add*: *0 Let-def*)
then show *?thesis*
proof(*cases* *?maxlow* ≠ *None* ∧ (*Some ?l* <$_o$ *?maxlow*))
case *True*
hence T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =
19 + T_{maxt} (*treeList* ! *?h*) + T_{succ} (*treeList* ! *?h*) *?l* **using** *1* **by** *simp*
hence T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
22 + T_{succ} (*treeList* ! *?h*) *?l* **using** *maxt-bound*[*of treeList* ! *?h*] **by** *simp*
moreover have *a*:*treeList* ! *?h* ∈ *set treeList*
by (*simp add*: ‹*high x* (*deg div 2*) < *length treeList*›)
ultimately have T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
22 + (*1*+ *height* (*treeList* ! *?h*))∗*27*
by (*meson 5.IH*(*1*) *nat-add-left-cancel-le order-trans*)
hence T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
((*1*+ *height* (*treeList* ! *?h*))+*1*)∗*27* **by** *simp*
then show *?thesis* **using** *height-compose-child*[*of treeList* ! *?h treeList Some* (*mi*, *ma*) *deg summary*] *a*
by (*smt* (*verit*) *Suc-leI add.commute dual-order.strict-trans2 le-imp-less-Suc linorder-not-less mult.commute mult-le-mono2 plus-1-eq-Suc*)
next
case *False*
then have *2*:T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =*17* + T_{maxt} (*treeList* ! *?h*) +
T_{succ} *summary ?h* + (
if ?sc = *None then 1*
else (*4* + T_{mint} (*treeList* ! *the ?sc*)))
using *1* **by** *auto*
then show *?thesis*
proof(*cases* *?sc* = *None*)
case *True*
hence *3*:T_{succ} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =*18* + T_{maxt} (*treeList* !

```
?h) +
                        T_succ summary ?h  using 2 by simp
      also have ... ≤ 21 + T_succ summary ?h
        using maxt-bound[of treeList ! ?h] by simp
      also have ... ≤ 21 + (1 + height summary) * 27
        using 5.IH(2) by simp
      also have ... ≤ 21 + height (Node (Some (mi, ma)) deg treeList summary) * 27
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by presburger
    next
      case False
      hence 3:T_succ (Node (Some (mi, ma)) deg treeList summary) x =21 + T_maxt  (treeList !
?h) +
                        T_succ summary ?h +   T_mint (treeList ! the ?sc) using 2 by simp
      also have ... ≤ 27 + T_succ summary ?h
        using maxt-bound[of treeList ! ?h] mint-bound[of treeList ! the ?sc] by simp
      also have ... ≤ 27 + (1 + height summary) * 27
        using 5.IH(2) by simp
      also have ... ≤ ((1 + height summary) + 1) * 27
        by simp
      also have ... ≤ (height (Node (Some (mi, ma)) deg treeList summary) + 1) * 27
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by presburger
    qed
  qed
 next
  case False
  hence  T_succ (Node (Some (mi, ma)) deg treeList summary) x = 12
    using ‹¬ x < mi›
    by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis
    by auto
 qed
qed

theorem succ-bound-size-univ: invar-vebt t n ⟹ u =  2^n ⟹   T_succ t x ≤ 54 + 27 * lb (lb u)
  using succ-bound-height[of t n x] height-double-log-univ-size[of u n t] by simp

theorem succ'-bound-height: invar-vebt t n ⟹ T_succ' t x ≤ (1+height t)
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
   by (metis One-nat-def T_succ'.simps(1) T_succ'.simps(2) height.simps(1) le-add2 le-add-same-cancel2
le-neq-implies-less less-imp-Suc-add order-refl plus-1-eq-Suc)
next
  case (4 treeList n summary m deg mi ma)
```

then obtain n' :: *nat* **where** $n = Suc\ n'$
by (*metis deg-not-0 less-imp-Suc-add*)
then have $deg = Suc\ (Suc\ (n' + n'))$
unfolding ‹$deg = n + m$› ‹$m = n$› **by** *presburger*
hence *degprop*: $deg \geq 2$
by *linarith*
then show *?case*
proof(*cases* $x <$ *mi*)
case *True*
then show *?thesis*
by (*simp add*: ‹$deg = Suc\ (Suc$ -)›)
next
case *False*
hence $x \geq mi$ **by** *simp*
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
show *?thesis*
proof(*cases ?h* < *length treeList*)
case *True*
hence *hprop*: *?h* < *length treeList* **by** *simp*
let *?maxlow* = *vebt-maxt* (*treeList* ! *?h*)
show *?thesis*
proof(*cases ?maxlow* ≠ *None* ∧ (*Some ?l* <$_o$ *?maxlow*))
case *True*
hence T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = 1+ T_{succ}' (*treeList* ! *?h*) *?l*
using ‹¬ $x < mi$› *hprop*
by (*simp add*: ‹$deg = Suc\ (Suc$ -)›)
moreover have (*treeList* ! *?h*) ∈ *set treeList*
using *hprop nth-mem* **by** *blast*
moreover have T_{succ}' (*treeList* ! *?h*) *?l* ≤ 1+ *height* (*treeList* ! *?h*) **using** *4*(*1*) *calculation*
by *blast*
ultimately have T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *1* + *1*+ *height* (*treeList* ! *?h*) **by** *simp*
then show *?thesis*
by (*smt* (*verit*) *Suc-le-mono* ‹T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *1* + T_{succ}' (*treeList* ! *high x* (*deg div 2*)) (*low x* (*deg div 2*))› ‹T_{succ}' (*treeList* ! *high x* (*deg div 2*)) (*low x* (*deg div 2*)) ≤ *1* + *height* (*treeList* ! *high x* (*deg div 2*))› ‹*treeList* ! *high x* (*deg div 2*) ∈ *set treeList*› *height-compose-child le-trans plus-1-eq-Suc*)
next
case *False*
hence T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = 1+ T_{succ}' *summary ?h*
using ‹$mi \leq x$› *hprop*
by (*auto simp add*: ‹$deg = Suc\ (Suc$ -)›)
moreover have T_{succ}' *summary ?h* ≤ 1+ *height summary* **using** *4*(*2*) *calculation* **by** *blast*
ultimately have T_{succ}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *1* + *1*+ *height summary* **by** *simp*
then show *?thesis*
by (*simp add*: *le-trans*)
qed

```
    next
      case False
      then show ?thesis
        by (simp add: ‹deg = Suc (Suc -)›)
    qed
  qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis add-0 less-natE not-gr0 nth-mem valid-0-not)
  then have deg = Suc (Suc (Suc (n' + n')))
    unfolding ‹deg = n + m› ‹m = Suc n› by presburger
  hence degprop: deg ≥ 2
    by linarith
  show ?case
  proof (cases x< mi)
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    hence x ≥ mi by simp
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    show ?thesis
    proof(cases ?h < length treeList)
      case True
      hence hprop: ?h < length treeList by simp
      let ?maxlow = vebt-maxt (treeList ! ?h)
      show ?thesis
      proof(cases  ?maxlow ≠ None ∧ (Some ?l <o ?maxlow))
        case True
        have T_succ' (Node (Some (mi, ma)) deg treeList summary) x =  1+ T_succ' (treeList ! ?h) ?l
          using ‹¬ x < mi› hprop True
          by (simp add: ‹deg = Suc (Suc -)›)
        also have ... ≤ 1 + 1+ height (treeList ! ?h)
        proof −
          have T_succ' (treeList ! ?h) ?l ≤ 1 + height (treeList ! ?h)
            using 5.IH(1) hprop nth-mem by blast
          then show ?thesis
            by linarith
        qed
        also have ... ≤ 1 + height (Node (Some (mi, ma)) deg treeList summary)
          by (simp add: hprop)
        finally show ?thesis .
      next
        case False
        hence T_succ' (Node (Some (mi, ma)) deg treeList summary) x =  1+ T_succ' summary ?h
          using ‹mi ≤ x› hprop
```

```
      by (auto simp add: ‹deg = Suc (Suc -)›)
    moreover have T_succ' summary ?h ≤ 1+ height summary using 5(2) calculation by blast
     ultimately have T_succ' (Node (Some (mi, ma)) deg treeList summary) x ≤ 1 + 1+ height
summary by simp
    then show ?thesis
      by (simp add: le-trans)
   qed
  next
   case False
   then show ?thesis
    by (simp add: ‹deg = Suc (Suc -)›)
  qed
 qed
qed simp+

theorem succ-bound-size-univ': invar-vebt t n ⟹ u = 2^n ⟹ T_succ' t x ≤ 2 + lb (lb u)
 using succ'-bound-height[of t n x] height-double-log-univ-size[of u n t] by simp
```

11.5 Predecessor Function

```
fun T_pred::VEBT ⇒ nat ⇒ nat where
 T_pred (Leaf - -) 0 = 1|
 T_pred (Leaf a -) (Suc 0) = 1+ (if a then 1 else 1)|
 T_pred (Leaf a b) - = 1+ (if b then 1 else 1+ (if a then 1 else 1))|

T_pred (Node None - - -) - = 1|
T_pred (Node - 0 - -) - = 1|
T_pred (Node - (Suc 0) - -) - = 1|
T_pred (Node (Some (mi, ma)) deg treeList summary) x = 1+ (
                  if x > ma then 1
                  else (let l = low x (deg div 2); h = high x (deg div 2) in 10 + 1+
                  (if h < length treeList then

                      let minlow = vebt-mint (treeList ! h) in 2 + T_mint(treeList ! h) + 3 +
                      (if minlow ≠ None ∧ (Some l >_o minlow) then
                                        4 + T_pred (treeList ! h) l
                       else let pr = vebt-pred summary h in 1 + T_pred summary h+ 1 + (
                       if pr = None then 1 + (if x > mi then 1 else 1)
                       else 4 + T_maxt (treeList ! the pr) ))
                else 1)))

theorem pred-bound-height: invar-vebt t n ⟹ T_pred t x ≤ (1+height t)*29
proof(induction t n arbitrary: x rule: invar-vebt.induct)
 case (1 a b)
 consider x = 0 | x = Suc 0 | x' where x = Suc (Suc x')
  by (metis nat.exhaust)
 then show ?case
  by cases simp-all
next
```

```
  case (2 treeList n summary m deg)
  then show ?case by simp
next
  case (3 treeList n summary m deg)
  then show ?case by simp
next
  case (4 treeList n summary m deg mi ma)
  hence deg ≥ 2
    by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  then obtain deg' :: nat where deg = Suc (Suc deg')
    using add-2-eq-Suc le-Suc-ex by blast
  then show ?case
  proof(cases x > ma)
    case True
    hence  T_pred (Node (Some (mi, ma)) deg treeList summary) x =2
      by (simp add: ‹deg = Suc (Suc -)›)
    then show ?thesis by simp
  next
    case False
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    have 0: T_pred (Node (Some (mi, ma)) deg treeList summary) x = 1 + 10 +1 +
                    (if ?h < length treeList then

                        let minlow = vebt-mint (treeList ! ?h) in 2 + T_mint(treeList ! ?h) + 3 +
                        (if minlow ≠ None ∧ (Some ?l >_o  minlow) then
                                        4 +  T_pred (treeList ! ?h) ?l
                         else let pr = vebt-pred summary ?h in  1 + T_pred summary ?h+ 1 + (
                         if pr = None then 1 + (if x > mi then 1 else 1)
                         else 4 +  T_maxt (treeList ! the pr) ))
                  else 1)
      using False by (simp add: ‹deg = Suc (Suc -)› Let-def)
    then show ?thesis
    proof(cases  ?h < length treeList)
      case True
      let ?minlow = vebt-mint (treeList ! ?h)
      have 1: T_pred (Node (Some (mi, ma)) deg treeList summary) x =17 +  T_mint(treeList ! ?h) +
                        (if ?minlow ≠ None ∧ (Some ?l >_o  ?minlow) then
                                        4 +  T_pred (treeList ! ?h) ?l
                         else let pr = vebt-pred summary ?h in  1 + T_pred summary ?h+ 1 + (
                         if pr = None then 1 + (if x > mi then 1 else 1)
                         else 4 +  T_maxt (treeList ! the pr) )) using True 0 by simp
      then show ?thesis
      proof(cases ?minlow ≠ None ∧ (Some ?l >_o  ?minlow))
        case True
        have 2: T_pred (Node (Some (mi, ma)) deg treeList summary) x =21 +  T_mint(treeList ! ?h)
+
                          T_pred (treeList ! ?h) ?l using True 1 by simp
        hence T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤ 24 +  T_pred (treeList ! ?h)
```

```
?l using mint-bound by simp
      moreover hence (treeList ! ?h) ∈ set treeList
        using 4.hyps(2) 4.hyps(3) 4.hyps(4) 4.hyps(8) False high-bound-aux by force
        ultimately have T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤ 24 + (1 +
height(treeList ! ?h))*29
        using 4.IH by (meson nat-add-left-cancel-le order-trans)
      hence T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤
              24 + (height (Node (Some (mi, ma)) deg treeList summary))*29
        using height-compose-child[of treeList ! ?h treeList Some(mi, ma) deg summary]
        by (meson ‹treeList ! high x (deg div 2) ∈ set treeList› add-le-cancel-left le-refl mult-le-mono
order-trans)
      then show ?thesis by simp
    next
      case False
      let ?pr = vebt-pred summary ?h
      have 2: T_pred (Node (Some (mi, ma)) deg treeList summary) x =19 + T_mint(treeList ! ?h)
+
                        T_pred summary ?h+ (
                        if ?pr = None then 1 + (if x > mi then 1 else 1)
                        else 4 + T_maxt (treeList ! the ?pr)) using False 1 by auto
      also have ... ≤22 +
                        T_pred summary ?h+ (
                        if ?pr = None then 1 + (if x > mi then 1 else 1)
                        else 4 + T_maxt (treeList ! the ?pr))
        using mint-bound by simp
      also have ... ≤ 29 + T_pred summary ?h
        using maxt-bound by (cases ?pr = None) auto
      also have ... ≤ 29 + (1 + height summary) * 29
        using 4.IH(2)[of ?h] by simp
      also have ... ≤ 29 + (height (Node (Some (mi, ma)) deg treeList summary)) * 29
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by presburger
    qed
  next
    case False
    then show ?thesis using 0 by simp
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  hence deg ≥ 2
    by (metis Suc-1 add-mono-thms-linordered-semiring(1) le-add1 plus-1-eq-Suc set-n-deg-not-0)
  then obtain deg' :: nat where deg = Suc (Suc deg')
    using add-2-eq-Suc le-Suc-ex by blast
  then show ?case
  proof(cases x > ma)
    case True
    hence T_pred (Node (Some (mi, ma)) deg treeList summary) x =2
```

```
    by (simp add: ‹deg = Suc (Suc -)›)
  then show ?thesis by simp
 next
  case False
  let ?l = low x (deg div 2)
  let ?h = high x (deg div 2)
  have 0: T_pred (Node (Some (mi, ma)) deg treeList summary) x = 1 + 10 +1 +
                (if ?h < length treeList then

                    let minlow = vebt-mint (treeList ! ?h) in 2 + T_mint(treeList ! ?h) + 3 +
                    (if minlow ≠ None ∧ (Some ?l >_o minlow) then
                                    4 + T_pred (treeList ! ?h) ?l
                     else let pr = vebt-pred summary ?h in 1 + T_pred summary ?h+ 1 + (
                     if pr = None then 1 + (if x > mi then 1 else 1)
                     else 4 + T_maxt (treeList ! the pr) ))
               else 1)
    using False by (simp add: ‹deg = Suc (Suc -)› Let-def)
  then show ?thesis
  proof(cases ?h < length treeList)
    case True
    hence ?h < length treeList by simp
    let ?minlow = vebt-mint (treeList ! ?h)
    have 1: T_pred (Node (Some (mi, ma)) deg treeList summary) x =17 + T_mint(treeList ! ?h) +
                    (if ?minlow ≠ None ∧ (Some ?l >_o ?minlow) then
                                    4 + T_pred (treeList ! ?h) ?l
                     else let pr = vebt-pred summary ?h in 1 + T_pred summary ?h+ 1 + (
                     if pr = None then 1 + (if x > mi then 1 else 1)
                     else 4 + T_maxt (treeList ! the pr) ))
      using True 0 by simp
    then show ?thesis
    proof(cases ?minlow ≠ None ∧ (Some ?l >_o ?minlow))
      case True
      have 2: T_pred (Node (Some (mi, ma)) deg treeList summary) x =21 + T_mint(treeList ! ?h)
+
                        T_pred (treeList ! ?h) ?l using True 1 by simp
      hence T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤ 24 + T_pred (treeList ! ?h)
?l using mint-bound by simp
      moreover hence (treeList ! ?h) ∈ set treeList
        by (meson ‹high x (deg div 2) < length treeList› nth-mem)
       ultimately have T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤ 24 + (1 +
height(treeList ! ?h))*29
        using 5.IH by (meson nat-add-left-cancel-le order-trans)
      hence T_pred (Node (Some (mi, ma)) deg treeList summary) x ≤
              24 + (height (Node (Some (mi, ma)) deg treeList summary))*29
        using height-compose-child[of treeList ! ?h treeList Some(mi, ma) deg summary]
        by (meson ‹treeList ! high x (deg div 2) ∈ set treeList› add-le-cancel-left le-refl mult-le-mono
order-trans)
      then show ?thesis by simp
    next
```

```
      case False
      let ?pr = vebt-pred summary ?h
      have 2: T_pred (Node (Some (mi, ma)) deg treeList summary) x =19 +  T_mint(treeList ! ?h)
+
                        T_pred summary ?h+ (
                        if ?pr = None then 1 + (if x > mi then 1 else 1)
                        else 4 +  T_maxt (treeList ! the ?pr))
        using False 1 by auto
      also have ... ≤ 22  +
                        T_pred summary ?h+ (
                        if ?pr = None then 1 + (if x > mi then 1 else 1)
                        else 4 +  T_maxt (treeList ! the ?pr))
        using mint-bound[of treeList ! ?h] by simp
      also have ... ≤ 29 + T_pred summary ?h
        using maxt-bound by (cases  ?pr = None) auto
      also have ... ≤ 29 + (1 + height summary) * 29
        using 5.IH(2)[of ?h] by simp
      also have ... ≤ 29 + (height (Node (Some (mi, ma)) deg treeList summary)) * 29
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by presburger
    qed
   next
    case False
    then show ?thesis using 0 by simp
   qed
  qed
qed


theorem pred-bound-size-univ: invar-vebt t n ⟹ u =  2^n ⟹   T_pred t x ≤ 58 + 29 * lb (lb u)
  using pred-bound-height[of t n x] height-double-log-univ-size[of u n t] by simp

fun T_pred'::VEBT ⇒ nat ⇒ nat  where
  T_pred' (Leaf - -) 0 = 1|
  T_pred' (Leaf a -) (Suc 0) = 1|
  T_pred' (Leaf a b) - = 1|

T_pred' (Node None - - -) - = 1|
T_pred' (Node - 0 - -) - = 1|
T_pred' (Node - (Suc 0) - -) - = 1|
T_pred' (Node (Some (mi, ma)) deg treeList summary) x = (
                    if x > ma then 1
                    else (let l = low x (deg div 2); h = high x (deg div 2) in
                    (if h < length treeList then

                        let minlow = vebt-mint (treeList ! h) in
                        (if minlow ≠ None ∧ (Some l >_o  minlow) then
                                        1+  T_pred' (treeList ! h) l
```

```
                    else let pr = vebt-pred summary h in    T_pred' summary h+ (
                    if pr = None then 1
                    else 1 ))
                else 1)))

theorem pred-bound-height': invar-vebt t n⟹ T_pred' t x ≤ (1 + height t)
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  consider x = 0 | x = Suc 0 | x' where x = Suc (Suc x')
    by (metis nat.exhaust)
  then show ?case
    by cases simp-all
next
  case (4 treeList n summary m deg mi ma)
  hence degprop:deg ≥ 2
    by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  then obtain deg' where deg = Suc (Suc deg')
    using add-2-eq-Suc le-Suc-ex by blast
  show ?case
  proof(cases x > ma)
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    hence x ≤ ma by simp
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    show ?thesis
    proof(cases ?h< length treeList)
      case True
      hence hprop: ?h< length treeList by simp
      let  ?minlow = vebt-mint (treeList ! ?h)
      show ?thesis
      proof(cases ?minlow ≠ None ∧ (Some ?l >o  ?minlow))
        case True
       hence T_pred' (Node (Some (mi, ma)) deg treeList summary) x =  1+  T_pred' (treeList ! ?h) ?l
          using ‹deg = Suc (Suc -)› False hprop by simp
        moreover have treeList  ! ?h ∈ set treeList using hprop by simp
        moreover hence T_pred' (treeList ! ?h) ?l ≤ 1 + height (treeList ! ?h) using 4(1) by simp
        ultimately have T_pred' (Node (Some (mi, ma)) deg treeList summary) x ≤  1+  1+ height
(treeList ! ?h) by simp
        then show ?thesis
          by (smt (verit) Suc-le-mono ‹T_pred' (Node (Some (mi, ma)) deg treeList summary) x = 1 +
T_pred' (treeList ! high x (deg div 2)) (low x (deg div 2))› ‹T_pred' (treeList ! high x (deg div 2)) (low x
(deg div 2)) ≤ 1 + height (treeList ! high x (deg div 2))› ‹treeList ! high x (deg div 2) ∈ set treeList›
height-compose-child le-trans plus-1-eq-Suc)
      next
        case False
```

```
      hence T_pred' (Node (Some (mi, ma)) deg treeList summary) x =  1+  T_pred' summary ?h
        using hprop ‹x ≤ ma›
        by (cases vebt-pred summary ?h) (auto simp add: ‹deg = Suc (Suc -)› Let-def)
       hence T_pred' (Node (Some (mi, ma)) deg treeList summary) x ≤ 1 + 1 +height summary
using 4(2)[of ?h] by simp
      then show ?thesis by(simp add: le-trans)
    qed
  next
    case False
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  qed
 qed
next
 case (5 treeList n summary m deg mi ma)
 then obtain n' :: nat where n = Suc n'
   by (metis add.right-neutral add-Suc-right nth-mem old.nat.exhaust valid-0-not)
 then have deg = Suc (Suc (Suc (n' + n')))
   using 5 by linarith
 then have degprop:deg ≥ 2
   by linarith
 then show ?case
 proof(cases x > ma)
   case True
   then show ?thesis
     by (simp add: ‹deg = Suc (Suc -)›)
 next
   case False
   hence x ≤ ma by simp
   let ?l = low x (deg div 2)
   let ?h = high x (deg div 2)
   show ?thesis
   proof(cases ?h< length treeList)
     case True
     hence hprop: ?h< length treeList by simp
     let  ?minlow = vebt-mint (treeList ! ?h)
     show ?thesis
     proof(cases ?minlow ≠ None ∧ (Some ?l >_o  ?minlow))
       case True
      hence T_pred' (Node (Some (mi, ma)) deg treeList summary) x =  1+  T_pred' (treeList ! ?h) ?l
         using hprop False
         by (simp add: ‹deg = Suc (Suc -)› Let-def)
       moreover have treeList  ! ?h ∈ set treeList using hprop by simp
       moreover hence T_pred' (treeList ! ?h) ?l ≤ 1 + height (treeList ! ?h) using 5(1) by simp
       ultimately have T_pred' (Node (Some (mi, ma)) deg treeList summary) x ≤  1+  1+ height
(treeList ! ?h) by simp
      then show ?thesis
       by (smt (verit) Suc-le-mono ‹T_pred' (Node (Some (mi, ma)) deg treeList summary) x = 1 +
T_pred' (treeList ! high x (deg div 2)) (low x (deg div 2))› ‹T_pred' (treeList ! high x (deg div 2)) (low x
```

(*deg div 2*)) ≤ *1* + *height* (*treeList* ! *high x* (*deg div 2*))› ‹*treeList* ! *high x* (*deg div 2*) ∈ *set treeList*› *height-compose-child le-trans plus-1-eq-Suc*)
next
case *False*
hence T_{pred}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *1*+ T_{pred}' *summary ?h*
using *hprop* ‹*x* ≤ *ma*›
by (*cases vebt-pred summary ?h*)
(*auto simp add*: ‹*deg* = *Suc* (*Suc* -)›)
hence T_{pred}' (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *1* + *1* +*height summary*
using *5*(*2*)[*of ?h*] **by** *simp*
then show *?thesis* **by**(*simp add*: *le-trans*)
qed
next
case *False*
then show *?thesis*
by (*simp add*: ‹*deg* = *Suc* (*Suc* -)›)
qed
qed
qed *simp*+

theorem *pred-bound-size-univ'*: *invar-vebt t n* ⟹ *u* = *2*^*n* ⟹ T_{pred}' *t x* ≤ *2* + *lb* (*lb u*)
using *pred-bound-height'*[*of t n x*] *height-double-log-univ-size*[*of u n t*] **by** *simp*

end
end

theory *VEBT-DeleteBounds* **imports** *VEBT-Bounds VEBT-Delete VEBT-DeleteCorrectness*
begin

11.6 Running Time Bounds for Deletion

context begin
interpretation *VEBT-internal* **.**

fun T_{delete}::*VEBT* ⇒ *nat* ⇒ *nat* **where**
T_{delete} (*Leaf a b*) *0* = *1*|
T_{delete} (*Leaf a b*) (*Suc 0*) = *1*|
T_{delete} (*Leaf a b*) (*Suc* (*Suc n*)) = *1*|
T_{delete} (*Node None deg treeList summary*) - = *1*|
T_{delete} (*Node* (*Some* (*mi*, *ma*)) *0 treeList summary*) *x* = *1*|
T_{delete} (*Node* (*Some* (*mi*, *ma*)) (*Suc 0*) *treeList summary*) *x* =*1* |
T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *3* + (
if (*x* < *mi* ∨ *x* > *ma*) *then 1*
else 3 + (*if* (*x* = *mi* ∧ *x* = *ma*) *then 3*
else 13 + (*if x* = *mi then* T_{mint} *summary* + T_{mint} (*treeList* ! *the* (*vebt-mint summary*))+
7 else 1)+
(*if x* = *mi then 1 else 1*) +
(*let xn* = (*if x* = *mi*
then the (*vebt-mint summary*) ∗ *2*^(*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the*

```
(vebt-mint summary)))
                        else x);
                  minn = (if x = mi then xn else mi);
                  l = low xn (deg div 2);
                  h = high xn (deg div 2) in
                   if h < length treeList
                     then( 4 + T_delete (treeList ! h) l +(
                       let newnode = vebt-delete (treeList ! h) l;
                           newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                          if minNull newnode
                          then( 1 + T_delete summary h + (
                            let sn = vebt-delete summary h in
                           2+ (if xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                                1 + (if maxs = None
                                                                   then 1
                                                                   else 8+ T_maxt (newlist ! the maxs)
                                                                ) )
                                                          else 1)
                          ))else
                           2 + (if xn = ma then 6+ T_maxt (newlist ! h)   else  1)
                     )))else  1  )))
```

end

context *VEBT-internal* **begin**

lemma *tdeletemimi*:
  assumes $deg \geq 2$
  shows T_{delete} (*Node* (*Some* (*mi*, *mi*)) *deg treeList summary*) $x \leq 9$
proof −
  obtain deg' :: *nat* **where** $deg = Suc\ (Suc\ deg')$
    using *add-2-eq-Suc assms le-Suc-ex* **by** *blast*

  then show *?thesis*
    by *simp*
qed

lemma *minNull-delete-time-bound*: *invar-vebt t n* $\Longrightarrow$ *minNull* (*vebt-delete t x*) $\Longrightarrow$ T_{delete} *t x* ≤ 9
proof(*induction t n rule*: *invar-vebt.induct*)
  case (*1 a b*)
  then consider $x = 0 \mid x = Suc\ 0 \mid x'$ **where** $x = Suc\ (Suc\ x')$
    by (*metis nat.exhaust*)
  then show *?case*
    by *cases simp-all*
next
  case (*2 treeList n summary m deg*)
  then show *?case* **by** *simp*
next

```
  case (3 treeList n summary m deg)
  then show ?case by simp
next
  case (4 treeList n summary m deg mi ma)
  hence deg ≥ 2
    by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  show ?case
  proof (cases mi = ma)
    case True
    then show ?thesis
      using ‹2 ≤ deg› tdeletemimi by metis
  next
    case False
    obtain mi' ma' treeList' summary' where
      vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
        Node (Some (mi', ma')) deg treeList' summary' (is ?LHS = ?RHS mi' ma' treeList' summary')
    proof atomize-elim
      show ∃ mi' ma' treeList' summary'. ?LHS = ?RHS mi' ma' treeList' summary'
      proof (cases x < mi ∨ x > ma)
        case True
        show ?thesis
          by (simp add: delt-out-of-range[OF True ‹2 ≤ deg›])
      next
        case False
        then have mi ≤ x ∧ x ≤ ma
          by linarith
        then show ?thesis
          by (simp
              add: del-in-range[of mi x ma deg, OF ‹mi ≤ x ∧ x ≤ ma› ‹mi ≠ ma› ‹2 ≤ deg›] Let-def
              split: if-split)
      qed
    qed
    then have ¬ minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)
      by simp
    then have False
      using ‹minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)›
      by contradiction
    then show ?thesis ..
  qed
next
  case (5 treeList n summary m deg mi ma)
  hence deg ≥ 2
    by (metis Suc-1 add-mono-thms-linordered-semiring(1) le-add1 plus-1-eq-Suc set-n-deg-not-0)
  show ?case
  proof (cases mi = ma)
    case True
    then show ?thesis
      using ‹2 ≤ deg› tdeletemimi by metis
  next
```

```
    case False
    obtain mi' ma' treeList' summary' where
      vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
        Node (Some (mi', ma')) deg treeList' summary' (is ?LHS = ?RHS mi' ma' treeList' summary')
    proof atomize-elim
      show ∃ mi' ma' treeList' summary'. ?LHS = ?RHS mi' ma' treeList' summary'
      proof (cases x < mi ∨ x > ma)
        case True
        show ?thesis
          by (simp add: delt-out-of-range[OF True ‹2 ≤ deg›])
      next
        case False
        then have mi ≤ x ∧ x ≤ ma
          by linarith
        then show ?thesis
          by (simp
              add: del-in-range[of mi x ma deg, OF ‹mi ≤ x ∧ x ≤ ma› ‹mi ≠ ma› ‹2 ≤ deg›] Let-def
              split: if-split)
      qed
    qed
    then have ¬ minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)
      by simp
    then have False
      using ‹minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)›
      by contradiction
    then show ?thesis ..
  qed
qed

lemma delete-bound-height: invar-vebt t n ⟹ T_delete t x ≤ (1+ height t)*70
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then consider x = 0 | x = Suc 0 | x' where x = Suc (Suc x')
    by (metis nat.exhaust)
  then show ?case
    by cases simp-all
next
  case (2 treeList n summary m deg)
  then show ?case by simp
next
  case (3 treeList n summary m deg)
  then show ?case by simp
next
  case (4 treeList n summary m deg mi ma)
  hence deggy: deg ≥ 2
    by (metis add-self-div-2 deg-not-0 div-greater-zero-iff)
  then obtain deg' :: nat where deg = Suc (Suc deg')
    using add-2-eq-Suc le-Suc-ex by blast
  then show ?case
```

```
 proof(cases  (x < mi ∨ x > ma))
  case True
  hence  T_delete (Node (Some (mi, ma)) deg treeList summary) x = 4
   by (simp add: ‹deg = Suc (Suc deg')›)
  then show ?thesis using  T_delete.simps(7)[of mi ma deg−2 treeList summary x] by auto
 next
  case False
  hence mi ≤ x ∧ x ≤ ma by simp
  hence 0: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 3+3 + (if (x = mi ∧ x =
ma) then 3
          else 13 + ( if x = mi then T_mint summary + T_mint (treeList ! the (vebt-mint summary))+
7 else 1 )+
                    (if x = mi then 1 else 1)  +
               (  let xn = (if x = mi
                         then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                         else x);
                minn = (if x = mi then xn else mi);
                l = low xn (deg div 2);
                h = high xn (deg div 2) in
                 if h < length treeList
                    then( 4 + T_delete (treeList ! h) l  +(
                      let newnode = vebt-delete (treeList ! h) l;
                          newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                      if minNull newnode
                      then(  1 +  T_delete summary h + (
                        let sn = vebt-delete summary h in
                       2+ (if xn  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                          1 + (if maxs = None
                                                            then 1
                                                            else 8+  T_maxt (newlist ! the maxs)
                                                           ) )
                                                  else 1)
                      ))else
                       2 + (if xn = ma then 6+  T_maxt (newlist ! h)   else  1)
                  )))else  1  ))
   by (simp add: ‹deg = Suc (Suc deg')›)
  then show ?thesis
  proof(cases  (x = mi ∧ x = ma))
   case True
   hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 9 using 0 by simp
   then show ?thesis by simp
  next
   case False
   hence 1: T_delete (Node (Some (mi, ma)) deg treeList summary) x =3+3 +13+
           (if x = mi then T_mint summary + T_mint (treeList ! the (vebt-mint summary))+ 7 else
1 )+
            (if x = mi then 1 else 1)  +
             (let xn = (if x = mi
```

```
                    then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                    else x);
              minn = (if x = mi then xn else mi);
              l = low xn (deg div 2);
              h = high xn (deg div 2) in
               if h < length treeList
                  then( 4 + T_delete (treeList ! h) l +(
                    let newnode = vebt-delete (treeList ! h) l;
                        newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                      if minNull newnode
                      then( 1 + T_delete summary h + (
                        let sn = vebt-delete summary h in
                       2+ (if xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+ T_maxt (newlist ! the maxs)
                                                   ) )
                                             else 1)
                      ))else
                       2 + (if xn = ma then 6+ T_maxt (newlist ! h)  else 1)
                  )))else 1 ) using 0
     by (simp add: False)
   then show ?thesis
   proof(cases x = mi)
     case True
    let ?xn = the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint
summary)))
    have 2: T_delete (Node (Some (mi, ma)) deg treeList summary) x =3+3 +13+ T_mint summary
+
                 T_mint (treeList ! the (vebt-mint summary))+ 7+1 +
              (let l = low ?xn (deg div 2);
              h = high ?xn (deg div 2) in
               if h < length treeList
                  then( 4 + T_delete (treeList ! h) l +(
                    let newnode = vebt-delete (treeList ! h) l;
                        newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                      if minNull newnode
                      then( 1 + T_delete summary h + (
                        let sn = vebt-delete summary h in
                       2+ (if ?xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+ T_maxt (newlist ! the maxs)
                                                   ) ) else 1) ))else
                       2 + (if ?xn = ma then 6+ T_maxt (newlist ! h)  else 1)
                  )))else 1 )
      using 1
      unfolding True Let-def
```

```
    by simp
  let ?l = low ?xn (deg div 2)
  let ?h = high ?xn (deg div 2)
 have 3: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 3+3 +13+ T_mint summary
+
          T_mint (treeList ! the (vebt-mint summary))+ 7+1 +
        (if ?h < length treeList
          then( 4 + T_delete (treeList ! ?h) ?l +(
            let newnode = vebt-delete (treeList ! ?h) ?l;
                newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
               if minNull newnode
               then( 1 + T_delete summary ?h + (
                 let sn = vebt-delete summary ?h in
                2+ (if ?xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                          1 + (if maxs = None
                                            then 1
                                            else 8+ T_maxt (newlist ! the maxs)
                                          ) ) else 1) ))else
                2 + (if ?xn = ma then 6+ T_maxt (newlist ! ?h) else 1)
        )))else 1 )
  using 2 by meson
 then show ?thesis
 proof(cases ?h < length treeList)
  case True
  hence ?h < length treeList by simp
  let ?newnode = vebt-delete (treeList ! ?h) ?l
  let ?newlist = treeList[?h:= ?newnode]
  have invar-vebt (treeList ! ?h) n
    using 4.IH(1) True nth-mem by blast
  hence invar-vebt ?newnode n
    using delete-pres-valid by blast
  have 4: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 37 + T_delete (treeList !
?h) ?l +(
               let newnode = vebt-delete (treeList ! ?h) ?l;
                   newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
                   if minNull newnode
                  then( 1 + T_delete summary ?h + (
                    let sn = vebt-delete summary ?h in
                   2+ (if ?xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                          1 + (if maxs = None
                                            then 1
                                            else 8+ T_maxt (newlist ! the maxs)
                                          ) ) else 1) ))else
                   2 + (if ?xn = ma then 6+ T_maxt (newlist ! ?h) else 1)
        ))
  using 3 mint-bound[of treeList ! the (vebt-mint summary)] mint-bound[of summary]
  by simp
 hence 5: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 38 + T_delete (treeList
! ?h) ?l +(
```

$T_{minNull}$ *?newnode* + (
if minNull ?newnode
then(1 + T_{delete} *summary ?h* + (
let sn = vebt-delete summary ?h in
2+ (*if ?xn = ma then 1* + T_{maxt} *sn* + (*let maxs = vebt-maxt sn in*
1 + (*if maxs = None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)))*else*
2 + (*if ?xn = ma then 6*+ T_{maxt} (*?newlist* ! *?h*) *else 1*)
))
unfolding *Let-def* **by** *linarith*
then show *?thesis*
proof(*cases minNull ?newnode*)
case *True*
hence *6*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + T_{delete} (*treeList* ! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
let sn = vebt-delete summary ?h in
2+ (*if ?xn = ma then 1* + T_{maxt} *sn* + (*let maxs = vebt-maxt sn in*
1 + (*if maxs = None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)) **using** *5 minNull-bound*[*of ?newnode*] **by** *presburger*
have *7*: T_{delete} (*treeList* ! *?h*) *?l* ≤ *9* **using** *True*
minNull-delete-time-bound[*of treeList* ! *?h*]
using ‹*invar-vebt* (*treeList* ! *high* (*the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*deg div 2*)) *n*› **by** *blast*
let *?sn = vebt-delete summary ?h*
have *8*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + T_{delete} (*treeList* ! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
2+ (*if ?xn = ma then 1* + T_{maxt} *?sn* + (*let maxs = vebt-maxt ?sn in*
1 + (*if maxs = None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*))
by (*meson 6*)
hence *9*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + *9* + *1* + T_{delete} *summary ?h* + *2*+
(*if ?xn = ma then 1* + T_{maxt} *?sn* + (*let maxs = vebt-maxt ?sn in*
1 + (*if maxs = None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*) **using** *6 7* **by** *force*
hence *10*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *51* + T_{delete} *summary ?h* +
(*if ?xn = ma then 1* + T_{maxt} *?sn* + (*let maxs = vebt-maxt ?sn in*

$1 + (\mathit{if}\ \mathit{maxs} = \mathit{None}$
$\mathit{then}\ 1$
$\mathit{else}\ 8 + \ T_{maxt}\ (\mathit{?newlist}\ !\ \mathit{the}\ \mathit{maxs})$
$)\ )\ \mathit{else}\ 1)$ **by** *simp*

then show *?thesis*
proof(*cases ?xn = ma*)
case *True*
hence *10*: $T_{delete}\ (\mathit{Node}\ (\mathit{Some}\ (\mathit{mi},\ \mathit{ma}))\ \mathit{deg}\ \mathit{treeList}\ \mathit{summary})\ x \leq 51 + \ T_{delete}\ \mathit{summary}\ \mathit{?h} +$
$1 + T_{maxt}\ \mathit{?sn} + (\mathit{let}\ \mathit{maxs} = \mathit{vebt\text{-}maxt}\ \mathit{?sn}\ \mathit{in}$
$1 + (\mathit{if}\ \mathit{maxs} = \mathit{None}$
$\mathit{then}\ 1$
$\mathit{else}\ 8 + \ T_{maxt}\ (\mathit{?newlist}\ !\ \mathit{the}\ \mathit{maxs})$
)) **using** *10*
by *presburger*
also have $\ldots \leq 55 + \ T_{delete}\ \mathit{summary}\ \mathit{?h} +$
$(\mathit{let}\ \mathit{maxs} = \mathit{vebt\text{-}maxt}\ \mathit{?sn}\ \mathit{in}$
$1 + (\mathit{if}\ \mathit{maxs} = \mathit{None}$
$\mathit{then}\ 1$
$\mathit{else}\ 8 + \ T_{maxt}\ (\mathit{?newlist}\ !\ \mathit{the}\ \mathit{maxs})$
)) **using** *maxt-bound*[*of ?sn*] **by** *force*
also have $\ldots \leq 55 + T_{delete}\ \mathit{summary}\ \mathit{?h} + 1 + 8 + T_{maxt}\ (\mathit{?newlist}\ !\ \mathit{the}\ (\mathit{vebt\text{-}maxt}\ \mathit{?sn}))$
by (*cases vebt-maxt ?sn*; *fastforce*)
also have $\ldots \leq 67 + T_{delete}\ \mathit{summary}\ \mathit{?h}$
using *maxt-bound*[*of ?newlist* ! *the* (*vebt-maxt ?sn*)] **by** *linarith*
also have $\ldots \leq 67 + (1 + \mathit{height}\ \mathit{summary}) * 70$
using *4.IH*(*2*) **by** *simp*
also have $\ldots \leq 67 + (\mathit{height}\ (\mathit{Node}\ (\mathit{Some}\ (\mathit{mi},\ \mathit{ma}))\ \mathit{deg}\ \mathit{treeList}\ \mathit{summary})) * 70$
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*
finally show *?thesis*
by *simp*
next
case *False*
hence *11*: $T_{delete}\ (\mathit{Node}\ (\mathit{Some}\ (\mathit{mi},\ \mathit{ma}))\ \mathit{deg}\ \mathit{treeList}\ \mathit{summary})\ x \leq 52 + T_{delete}\ \mathit{summary}\ \mathit{?h}$
using *10* **by** *simp*
also have $\ldots \leq 52 + \ (1 + \mathit{height}\ \mathit{summary}) * 70$
using *4.IH*(*2*) **by** *simp*
also have $\ldots \leq 52 + (\mathit{height}\ (\mathit{Node}\ (\mathit{Some}\ (\mathit{mi},\ \mathit{ma}))\ \mathit{deg}\ \mathit{treeList}\ \mathit{summary})) * 70$
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*
finally show *?thesis*
by *simp*
qed
next
case *False*
hence *6*: $T_{delete}\ (\mathit{Node}\ (\mathit{Some}\ (\mathit{mi},\ \mathit{ma}))\ \mathit{deg}\ \mathit{treeList}\ \mathit{summary})\ x \leq 38 + T_{delete}\ (\mathit{treeList}\ !\ \mathit{?h})\ \mathit{?l}\ +($
$T_{minNull}\ \mathit{?newnode} + \ 2 + (\mathit{if}\ \mathit{?xn} = \mathit{ma}\ \mathit{then}\ 6 + \ T_{maxt}\ (\mathit{?newlist}\ !\ \mathit{?h})\ \ \mathit{else}$

1)) **using** *5* **by** *simp*
moreover have *invar-vebt* (*?newlist* ! *?h*) *n*
by (*simp add*: *True* ‹*invar-vebt* (*vebt-delete* (*treeList* ! *high* (*the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*deg div 2*)) (*low* (*the* (*vebt-mint summary*) ∗ *2* ^(*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*deg div 2*))) *n*›)
ultimately have T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
43 + T_{delete} (*treeList* ! *?h*) *?l* + (*if ?xn* = *ma then 6*+ T_{maxt} (*?newlist* ! *?h*) *else 1*)
using *minNull-bound*[*of ?newnode*] **by** *linarith*
moreover have (*if ?xn* = *ma then 6*+ T_{maxt} (*?newlist* ! *?h*) *else 1*) ≤ *9*
apply(*cases ?xn* = *ma*) **using** *maxt-bound*[*of* (*?newlist* ! *?h*)] **by** *simp*+
ultimately have T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *55* + T_{delete} (*treeList* ! *?h*) *?l* **by** *force*
also have ... ≤ *55* + (*1* + *height* (*treeList* ! *?h*)) ∗ *70*
using *4*.*IH*(*1*) *True* **by** *simp*
also have ... ≤ *55* + (*height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *70*
using *height-compose-child*[*of treeList* ! *?h* *treeList Some* (*mi*, *ma*) *deg summary*]
by (*meson True mult-le-mono1 nat-add-left-cancel-le nth-mem*)
finally show *?thesis*
by *simp*
qed
next
case *False*
hence T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* = *3*+*3* +*13*+ T_{mint} *summary* +
T_{mint} (*treeList* ! *the* (*vebt-mint summary*))+ *7*+*1* +*1* **using** *3* **by** *simp*
hence T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *34* **using**
mint-bound[*of treeList* ! *the* (*vebt-mint summary*)]
mint-bound[*of summary*] **by** *simp*
then show *?thesis* **by** *simp*
qed
next
case *False*
let *?l* = *low x* (*deg div 2*)
let *?h* = *high x* (*deg div 2*)
have *2*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* =*3*+*3* +*13*+*1* +*1* +
(*let l* = *low x* (*deg div 2*);
h = *high x* (*deg div 2*) *in*
if h < *length treeList*
then(*4* + T_{delete} (*treeList* ! *h*) *l* +(
let newnode = *vebt-delete* (*treeList* ! *h*) *l*;
newlist = *treeList*[*h*:= *newnode*]*in 1* + $T_{minNull}$ *newnode* + (
if minNull newnode
then(*1* + T_{delete} *summary h* + (
let sn = *vebt-delete summary h in*
2+ (*if x* = *ma then 1* + T_{maxt} *sn* + (*let maxs* = *vebt-maxt sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*newlist* ! *the maxs*)
)) *else 1*)))*else*

```
                    2 + (if x = ma then 6+  T_maxt (newlist ! h)   else  1)
                ))) else  1  )
    using 1  False by simp
  hence 3: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+(
          if ?h < length treeList
            then( 4 + T_delete (treeList ! ?h) ?l  +(
              let newnode = vebt-delete (treeList ! ?h) ?l;
                newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
              if minNull newnode
              then(  1 +  T_delete summary ?h + (
                let sn = vebt-delete summary ?h in
                2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+  T_maxt (newlist ! the maxs)
                                                  ) ) else 1)  ))else
                2 + (if x = ma then 6+  T_maxt (newlist ! ?h)   else  1)
            ))) else  1  )
    by (simp add: Let-def)
  then show ?thesis
  proof(cases ?h < length treeList)
    case True
    hence ?h < length treeList by simp
    let  ?newnode = vebt-delete (treeList ! ?h) ?l
    let ?newlist = treeList[?h:= ?newnode]
    have invar-vebt  (treeList ! ?h) n
      using 4.IH(1) True nth-mem by blast
    hence invar-vebt ?newnode n
      using delete-pres-valid by blast
     hence 4: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+ 4 + T_delete
(treeList ! ?h) ?l
            + 1 + T_minNull ?newnode + (
                if minNull ?newnode
                then(  1 +  T_delete summary ?h + (
                  let sn = vebt-delete summary ?h in
                  2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+  T_maxt (?newlist ! the maxs)
                                                  ) ) else 1)  ))else
                2 + (if x = ma then 6+  T_maxt (?newlist ! ?h)   else  1))
      using 3   mint-bound[of treeList ! the (vebt-mint summary)] mint-bound[of summary]
      by (smt (verit) True add.assoc)
    hence 5: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 26 + T_delete (treeList
! ?h) ?l +
                  T_minNull ?newnode + (
                    if minNull ?newnode
                    then(  1 +  T_delete summary ?h + (
                      let sn = vebt-delete summary ?h in
```

$2+$ (*if* x $=$ ma *then* $1 + T_{maxt}$ sn + (*let* $maxs$ = *vebt-maxt* sn *in*
1 + (*if* $maxs$ = *None*
then 1
else $8+$ T_{maxt} (*?newlist* ! *the* $maxs$)
)) *else* 1)))*else*
2 + (*if* x = ma *then* $6+$ T_{maxt} (*?newlist* ! *?h*) *else* 1)) **by** *force*

then show *?thesis*

proof(*cases minNull ?newnode*)

case *True*

hence *6*: T_{delete} (*Node* (*Some* (mi, ma)) *deg treeList summary*) $x \leq 29 + T_{delete}$ (*treeList* ! *?h*) *?l*
$+ 1 +$ T_{delete} *summary ?h* + (
let sn = *vebt-delete summary ?h in*
$2+$ (*if* x $=$ ma *then* $1 + T_{maxt}$ sn + (*let* $maxs$ = *vebt-maxt* sn *in*
1 + (*if* $maxs$ = *None*
then 1
else $8+$ T_{maxt} (*?newlist* ! *the* $maxs$)
)) *else* 1)) **using** *5 minNull-bound*[*of ?newnode*] **by** *force*

have *7*: T_{delete} (*treeList* ! *?h*) *?l* $\leq$ *9* **using** *True*
minNull-delete-time-bound[*of treeList* ! *?h*]
using ‹*invar-vebt* (*treeList* ! *high* x (*deg div* 2)) n› **by** *blast*

let *?sn* = *vebt-delete summary ?h*

have *8*: T_{delete} (*Node* (*Some* (mi, ma)) *deg treeList summary*) $x \leq 29 + T_{delete}$ (*treeList* ! *?h*) *?l*
$+ 1 +$ T_{delete} *summary ?h* + (
$2+$ (*if* x $=$ ma *then* $1 + T_{maxt}$ *?sn* + (*let* $maxs$ = *vebt-maxt ?sn in*
1 + (*if* $maxs$ = *None*
then 1
else $8+$ T_{maxt} (*?newlist* ! *the* $maxs$)
)) *else* 1))

by (*meson 6*)

hence *9*: T_{delete} (*Node* (*Some* (mi, ma)) *deg treeList summary*) $x \leq 29 + 9 + 1 +$ T_{delete} *summary ?h* + $2+$
(*if* x $=$ ma *then* $1 + T_{maxt}$ *?sn* + (*let* $maxs$ = *vebt-maxt ?sn in*
1 + (*if* $maxs$ = *None*
then 1
else $8+$ T_{maxt} (*?newlist* ! *the* $maxs$)
)) *else* 1)

using *6 7* **by** *force*

hence *10*: T_{delete} (*Node* (*Some* (mi, ma)) *deg treeList summary*) $x \leq 41 +$ T_{delete} *summary ?h* +
(*if* x $=$ ma *then* $1 + T_{maxt}$ *?sn* + (*let* $maxs$ = *vebt-maxt ?sn in*
1 + (*if* $maxs$ = *None*
then 1
else $8+$ T_{maxt} (*?newlist* ! *the* $maxs$)
)) *else* 1)

by *simp*

then show *?thesis*

```
          proof(cases x = ma)
            case True
              hence 10: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 41 + T_delete
summary ?h +
                                    1 + T_maxt ?sn + (let maxs = vebt-maxt ?sn in
                                                        1 + (if maxs = None
                                                          then 1
                                                          else 8+ T_maxt (?newlist ! the maxs)
                                                        ))
              using 10 by presburger
            also have ... ≤ 45 + T_delete summary ?h +
                                    (let maxs = vebt-maxt ?sn in
                                                        1 + (if maxs = None
                                                          then 1
                                                          else 8+ T_maxt (?newlist ! the maxs)
                                                        ))
              using maxt-bound[of ?sn] by force
            also have ... ≤ 45 + T_delete summary ?h + 1 + 8 + T_maxt (?newlist ! the (vebt-maxt
?sn))
              by (cases vebt-maxt ?sn; fastforce)
            also have ... ≤ 57 + T_delete summary ?h
              using maxt-bound[of ?newlist ! the (vebt-maxt ?sn)] by linarith
            also have ... ≤ 57 + (1+ height summary) * 70
              using 4.IH(2) by simp
            also have ... ≤ 57 + (height (Node (Some (mi, ma)) deg treeList summary)) * 70
              using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
            finally show ?thesis
              by simp
          next
            case False
              hence 11: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 42 + T_delete
summary ?h
              using 10 by simp
              hence T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 42 + (1+ height
summary)*70
              by (meson 4.IH(2) le-trans nat-add-left-cancel-le)
            hence T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤
          42 + (height (Node (Some (mi, ma)) deg treeList summary) )*70 using height-compose-summary[of
summary Some (mi, ma) deg treeList ]
              by (meson add-mono-thms-linordered-semiring(2) le-refl mult-le-mono order-trans)
            then show ?thesis by simp
          qed
        next
          case False
          hence 6: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 26 + T_delete (treeList
! ?h) ?l +(
                        T_minNull ?newnode + 2 + (if x = ma then 6+ T_maxt (?newlist ! ?h) else
1)) using 5 by simp
          moreover have invar-vebt (?newlist ! ?h) n
```

```
          by (simp add: True ‹invar-vebt (vebt-delete (treeList ! high x (deg div 2)) (low x (deg div
2))) n›)
        ultimately have T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤
            29 + T_delete (treeList ! ?h) ?l + (if x = ma then 6+ T_maxt (?newlist ! ?h) else 1)
          using minNull-bound[of ?newnode] by linarith
        moreover have (if x = ma then 6+ T_maxt (?newlist ! ?h) else 1) ≤ 9
          apply(cases x = ma) using maxt-bound[of (?newlist ! ?h) ] by simp+
        ultimately have T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤
                  38 + T_delete (treeList ! ?h) ?l by force
        also have ... ≤ 38 + (1 + height (treeList ! ?h)) * 70
          using 4.IH(1) True by simp
        also have ... ≤ 38 + (height (Node (Some (mi, ma)) deg treeList summary)) * 70
          using height-compose-child[of treeList ! ?h treeList Some (mi, ma) deg summary]
          by (meson True add-le-cancel-left mult-le-mono1 nth-mem)
        finally show ?thesis
          by presburger
      qed
    next
      case False
      hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+1 using 3 by simp
      hence T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 22 using
          mint-bound[of treeList ! the (vebt-mint summary)]
          mint-bound[of summary] by simp
      then show ?thesis by simp
    qed
  qed
 qed
qed
next
 case (5 treeList n summary m deg mi ma)
 hence deggy: deg ≥ 2
  by (metis Suc-1 add-mono-thms-linordered-semiring(1) le-add1 plus-1-eq-Suc set-n-deg-not-0)
 then obtain deg' where deg = Suc (Suc deg')
  using add-2-eq-Suc le-Suc-ex by blast
 then show ?case
 proof(cases (x < mi ∨ x > ma))
  case True
  hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 4
   by (simp add: ‹deg = Suc (Suc deg')›)
  then show ?thesis using T_delete.simps(7)[of mi ma deg−2 treeList summary x] by auto
 next
  case False
  hence mi ≤ x ∧ x ≤ ma by simp
  hence 0: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 3+3 + (if (x = mi ∧ x =
ma) then 3
        else 13 + ( if x = mi then T_mint summary + T_mint (treeList ! the (vebt-mint summary))+
7 else 1 )+
                  (if x = mi then 1 else 1) +
            ( let xn = (if x = mi
```

```
                    then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                    else x);
              minn = (if x = mi then xn else mi);
              l = low xn (deg div 2);
              h = high xn (deg div 2) in
               if h < length treeList
                  then( 4 + T_delete (treeList ! h) l +(
                    let newnode = vebt-delete (treeList ! h) l;
                        newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                    if minNull newnode
                    then( 1 + T_delete summary h + (
                      let sn = vebt-delete summary h in
                     2+ (if xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+ T_maxt (newlist ! the maxs)
                                                     ) )
                                              else 1)
                    ))else
                     2 + (if xn = ma then 6+ T_maxt (newlist ! h)  else 1)
                )))else 1 ))
  by (simp add: ‹deg = Suc (Suc deg')›)
 then show ?thesis
 proof(cases (x = mi ∧ x = ma))
  case True
  hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 9 using 0 by simp
  then show ?thesis by simp
 next
  case False
  hence 1: T_delete (Node (Some (mi, ma)) deg treeList summary) x =3+3 +13+
          ( if x = mi then T_mint summary + T_mint (treeList ! the (vebt-mint summary))+ 7 else
1 )+
                  (if x = mi then 1 else 1) +
            ( let xn = (if x = mi
                    then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                    else x);
              minn = (if x = mi then xn else mi);
              l = low xn (deg div 2);
              h = high xn (deg div 2) in
               if h < length treeList
                  then( 4 + T_delete (treeList ! h) l +(
                    let newnode = vebt-delete (treeList ! h) l;
                        newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                    if minNull newnode
                    then( 1 + T_delete summary h + (
                      let sn = vebt-delete summary h in
                     2+ (if xn = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
```

```
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+  T_maxt (newlist ! the maxs)
                                                    ) )
                                               else 1)
                          ))else
                            2 + (if xn = ma then 6+  T_maxt (newlist ! h)   else  1)
                      )))else  1  ) using 0
      by (simp add: False)
    then show ?thesis
    proof(cases x = mi)
      case True
      let ?xn =  the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint
summary)))
     have 2: T_delete (Node (Some (mi, ma)) deg treeList summary) x =3+3 +13+ T_mint summary
+
                  T_mint (treeList ! the (vebt-mint summary))+ 7+1 +
                (let  l = low ?xn (deg div 2);
                h = high ?xn (deg div 2) in
                 if h < length treeList
                   then( 4 + T_delete (treeList ! h) l  +(
                     let newnode = vebt-delete (treeList ! h) l;
                         newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                       if minNull newnode
                       then(  1 +  T_delete summary h + (
                         let sn = vebt-delete summary h in
                        2+ (if ?xn  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+  T_maxt (newlist ! the maxs)
                                                    ) ) else 1)  ))else
                        2 + (if ?xn = ma then 6+  T_maxt (newlist ! h)   else  1)
                    )))else  1  )
        using 1
        unfolding True Let-def
        by force
      let ?l = low ?xn (deg div 2)
      let ?h = high ?xn (deg div 2)
     have 3: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 3+3 +13+ T_mint summary
+
                  T_mint (treeList ! the (vebt-mint summary))+ 7+1 +
                (if ?h < length treeList
                   then( 4 + T_delete (treeList ! ?h) ?l  +(
                     let newnode = vebt-delete (treeList ! ?h) ?l;
                         newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
                       if minNull newnode
                       then(  1 +  T_delete summary ?h + (
                         let sn = vebt-delete summary ?h in
                        2+ (if ?xn  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
```

```
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+  T_maxt (newlist ! the maxs)
                                                    ) ) else 1)  ))else
                          2 + (if ?xn = ma then 6+  T_maxt (newlist ! ?h)   else  1)
                    )))else  1  )
      using  2  by meson
    then show ?thesis
    proof(cases ?h < length treeList)
      case True
      hence ?h < length treeList by simp
      let  ?newnode = vebt-delete (treeList ! ?h) ?l
      let ?newlist = treeList[?h:= ?newnode]
      have invar-vebt  (treeList ! ?h) n
        using 5.IH(1) True nth-mem by blast
      hence invar-vebt ?newnode n
        using delete-pres-valid by blast
      have 4: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 37 + T_delete (treeList !
?h) ?l  +(
                      let newnode = vebt-delete (treeList ! ?h) ?l;
                        newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
                        if minNull newnode
                       then(  1 +  T_delete summary ?h + (
                          let sn = vebt-delete summary ?h in
                        2+ (if ?xn  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+  T_maxt (newlist ! the maxs)
                                                    ) ) else 1)  ))else
                          2 + (if ?xn = ma then 6+  T_maxt (newlist ! ?h)   else  1)
                    ))
      using 3   mint-bound[of treeList ! the (vebt-mint summary)] mint-bound[of summary]
      by simp
    hence 5: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 38 + T_delete (treeList
! ?h) ?l  +(
                       T_minNull ?newnode + (
                        if minNull ?newnode
                        then(  1 +  T_delete summary ?h + (
                          let sn = vebt-delete summary ?h in
                        2+ (if ?xn  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                    1 + (if maxs = None
                                                      then 1
                                                      else 8+  T_maxt (?newlist ! the maxs)
                                                    ) ) else 1)  ))else
                          2 + (if ?xn = ma then 6+  T_maxt (?newlist ! ?h)   else  1)
                    ))
      unfolding Let-def
      by linarith
    then show ?thesis
```

proof(*cases minNull ?newnode*)
case *True*
hence *6*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + T_{delete} (*treeList* ! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
let sn = *vebt-delete summary ?h in*
2+ (*if ?xn* = *ma then 1* + T_{maxt} *sn* + (*let maxs* = *vebt-maxt sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)) **using** *5 minNull-bound*[*of ?newnode*] **by** *presburger*
have *7*: T_{delete} (*treeList* ! *?h*) *?l* ≤ *9* **using** *True*
minNull-delete-time-bound[*of treeList* ! *?h*]
using ‹*invar-vebt* (*treeList* ! *high* (*the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*deg div 2*)) *n*› **by** *blast*
let *?sn* = *vebt-delete summary ?h*
have *8*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + T_{delete} (*treeList* ! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
2+ (*if ?xn* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*))
by (*meson 6*)
hence *9*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *39* + *9* + *1* + T_{delete} *summary ?h* + *2*+
(*if ?xn* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)
using *6 7* **by** *force*
hence *10*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *51* + T_{delete} *summary ?h* +
(*if ?xn* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)
by *simp*
then show *?thesis*
proof(*cases ?xn* = *ma*)
case *True*
hence *10*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *51* + T_{delete} *summary ?h* +
1 + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*

then 1
else 8+ T_{maxt} (?newlist ! the maxs)
))
using *10* **by** *presburger*
also have *11*: ... ≤ *55* + T_{delete} *summary ?h* +
(*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (?newlist ! the maxs)
))
using *maxt-bound*[*of ?sn*] **by** *force*
also have *12*: ... ≤ *55* + T_{delete} *summary ?h* + *1* + *8* + T_{maxt} (*?newlist ! the* (*vebt-maxt ?sn*))
by (*cases vebt-maxt ?sn*; *fastforce*)
also have ... ≤ *67* + T_{delete} *summary ?h*
using *maxt-bound*[*of ?newlist ! the* (*vebt-maxt ?sn*)] **by** *linarith*
also have ... ≤ *67* + (*1* + *height summary*) ∗ *70*
using *5.IH*(*2*) **by** *simp*
also have ... ≤ *67* + (*height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *70*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*
finally show *?thesis*
by *simp*
next
case *False*
hence *11*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *52* + T_{delete} *summary ?h*
using *10* **by** *simp*
also have ... ≤ *52* + (*1*+ *height summary*) ∗ *70*
using *5.IH*(*2*) **by** *simp*
also have ... ≤ *52* + (*height* (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*)) ∗ *70*
using *height-compose-summary*[*of summary Some* (*mi*, *ma*) *deg treeList*] **by** *simp*
finally show *?thesis*
by *simp*
qed
next
case *False*
hence *6*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *38* + T_{delete} (*treeList ! ?h*) *?l* +(
$T_{minNull}$ *?newnode* + *2* + (*if ?xn* = *ma then 6+ T_{maxt}* (*?newlist ! ?h*) *else 1*)) **using** *5* **by** *simp*
moreover have *invar-vebt* (*?newlist ! ?h*) *n*
by (*simp add*: *True* ‹*invar-vebt* (*vebt-delete* (*treeList ! high* (*the* (*vebt-mint summary*) ∗ *2* ^ (*deg div 2*) + *the* (*vebt-mint* (*treeList ! the* (*vebt-mint summary*)))) (*deg div 2*)) (*low* (*the* (*vebt-mint summary*) ∗ *2* ^(*deg div 2*) + *the* (*vebt-mint* (*treeList ! the* (*vebt-mint summary*)))) (*deg div 2*))) *n*›)
ultimately have T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤
43 + T_{delete} (*treeList ! ?h*) *?l* + (*if ?xn* = *ma then 6+ T_{maxt}* (*?newlist ! ?h*) *else 1*)
using *minNull-bound*[*of ?newnode*] **by** *linarith*
moreover have (*if ?xn* = *ma then 6+ T_{maxt}* (*?newlist ! ?h*) *else 1*) ≤ *9*
apply(*cases ?xn* = *ma*) **using** *maxt-bound*[*of* (*?newlist ! ?h*)] **by** *simp+*

```
        ultimately have  T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 55 + T_delete
(treeList ! ?h) ?l by force
        also have ... ≤ 55 + (1 + height (treeList ! ?h)) * 70
          using 5.IH(1) True by simp
        also have ... ≤ 55 + (height (Node (Some (mi, ma)) deg treeList summary)) * 70
          using height-compose-child[of treeList ! ?h treeList Some (mi, ma) deg summary]
          by (meson True mult-le-mono1 nat-add-left-cancel-le nth-mem)
        finally show ?thesis
          by simp
      qed
    next
      case False
    hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 3+3 +13+ T_mint summary
+
                T_mint (treeList ! the (vebt-mint summary))+ 7+1 +1 using 3 by simp
      also have ... ≤ 34
        using mint-bound[of treeList ! the (vebt-mint summary)] mint-bound[of summary]
        by linarith
      finally show ?thesis
        by simp
    qed
  next
    case False
    let ?l = low x (deg div 2)
    let ?h = high x (deg div 2)
    have 2: T_delete (Node (Some (mi, ma)) deg treeList summary) x =3+3 +13+1 +1 +
            (let  l = low x (deg div 2);
            h = high x (deg div 2) in
             if h < length treeList
               then( 4 + T_delete (treeList ! h) l  +(
                 let newnode = vebt-delete (treeList ! h) l;
                     newlist = treeList[h:= newnode]in 1 + T_minNull newnode + (
                   if minNull newnode
                   then(  1 +  T_delete summary h + (
                     let sn = vebt-delete summary h in
                     2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                 1 + (if maxs = None
                                                   then 1
                                                   else 8+  T_maxt (newlist ! the maxs)
                                                 ) ) else 1)  ))else
                     2 + (if x = ma then 6+  T_maxt (newlist ! h)   else  1)
                 )))else  1  ) using 1  False by simp
    hence 3: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+(
            if ?h < length treeList
               then( 4 + T_delete (treeList ! ?h) ?l  +(
                 let newnode = vebt-delete (treeList ! ?h) ?l;
                     newlist = treeList[?h:= newnode]in 1 + T_minNull newnode + (
                   if minNull newnode
                   then(  1 +  T_delete summary ?h + (
```

```
                    let sn = vebt-delete summary ?h in
                   2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+  T_maxt (newlist ! the maxs)
                                                  ) ) else 1)  ))else
                   2 + (if x = ma then 6+  T_maxt (newlist ! ?h)   else  1)
               )))else  1  )
      by (simp add: Let-def)
    then show ?thesis
    proof(cases ?h < length treeList)
      case True
      hence ?h < length treeList by simp
      let  ?newnode = vebt-delete (treeList ! ?h) ?l
      let ?newlist = treeList[?h:= ?newnode]
      have invar-vebt  (treeList ! ?h) n
        using 5.IH(1) True nth-mem by blast
      hence invar-vebt ?newnode n
        using delete-pres-valid by blast
       hence 4: T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+ 4 + T_delete
(treeList ! ?h) ?l
             + 1 + T_minNull ?newnode + (
                  if minNull ?newnode
                  then(  1 +  T_delete summary ?h + (
                    let sn = vebt-delete summary ?h in
                   2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                  1 + (if maxs = None
                                                    then 1
                                                    else 8+  T_maxt (?newlist ! the maxs)
                                                  ) ) else 1)  ))else
                        2 + (if x = ma then 6+  T_maxt (?newlist ! ?h)   else  1)) using 3
mint-bound[of treeList ! the (vebt-mint summary)]
        mint-bound[of summary] by (smt (verit) True add.assoc)
      hence 5: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 26 + T_delete (treeList
! ?h) ?l +
                    T_minNull ?newnode + (
                      if minNull ?newnode
                      then(  1 +  T_delete summary ?h + (
                        let sn = vebt-delete summary ?h in
                       2+ (if x  = ma then 1 + T_maxt sn + (let maxs = vebt-maxt sn in
                                                      1 + (if maxs = None
                                                        then 1
                                                        else 8+  T_maxt (?newlist ! the maxs)
                                                      ) ) else 1)  ))else
                       2 + (if x = ma then 6+  T_maxt (?newlist ! ?h)   else  1)) by force
      then show ?thesis
      proof(cases minNull ?newnode )
        case True
       hence 6: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 29 + T_delete (treeList
```

! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
let sn = *vebt-delete summary ?h in*
2+ (*if x* = *ma then 1* + T_{maxt} *sn* + (*let maxs* = *vebt-maxt sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*))
using *5 minNull-bound*[*of ?newnode*] **by** *force*
have *7*: T_{delete} (*treeList* ! *?h*) *?l* ≤ *9* **using** *True*
minNull-delete-time-bound[*of treeList* ! *?h*]
using ‹*invar-vebt* (*treeList* ! *high x* (*deg div 2*)) *n*› **by** *blast*
let *?sn* = *vebt-delete summary ?h*
have *8*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *29* + T_{delete} (*treeList* ! *?h*) *?l*
+ *1* + T_{delete} *summary ?h* + (
2+ (*if x* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*))
by (*meson 6*)
hence *9*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *29* + *9* + *1* + T_{delete} *summary ?h* + *2*+
(*if x* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)
using *6 7* **by** *force*
hence *10*: T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *41* + T_{delete} *summary ?h* +
(*if x* = *ma then 1* + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
)) *else 1*)
by *simp*
then show *?thesis*
proof(*cases x* = *ma*)
case *True*
hence T_{delete} (*Node* (*Some* (*mi*, *ma*)) *deg treeList summary*) *x* ≤ *41* + T_{delete} *summary ?h* +
1 + T_{maxt} *?sn* + (*let maxs* = *vebt-maxt ?sn in*
1 + (*if maxs* = *None*
then 1
else 8+ T_{maxt} (*?newlist* ! *the maxs*)
))
using *10* **by** *presburger*

```
      also have ... ≤ 45 +  T_delete summary ?h +
                            (let maxs = vebt-maxt ?sn in
                                                  1 + (if maxs = None
                                                     then 1
                                                     else 8+  T_maxt (?newlist ! the maxs)
                                                  ))
        using maxt-bound[of ?sn] by linarith
      also have ... ≤ 45 + T_delete summary ?h + 1 + 8 + T_maxt (?newlist ! the (vebt-maxt
?sn))
        by (cases  vebt-maxt ?sn; fastforce)
      also have ... ≤ 57 +  T_delete summary ?h
        using maxt-bound[of ?newlist ! the (vebt-maxt ?sn)] by linarith
      also have ... ≤ 57 +  (1+ height summary)∗70
        using 5.IH(2) by simp
      also have ... ≤ 57 +  (height  (Node (Some (mi, ma)) deg treeList summary) )∗70
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis by presburger
    next
      case False
     hence T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 42 + T_delete summary
?h
        using 10 by simp
      also have ... ≤ 42 + (1 + height summary) ∗ 70
        using 5.IH(2) by simp
      also have ... ≤ 42 + (height (Node (Some (mi, ma)) deg treeList summary)) ∗ 70
        using height-compose-summary[of summary Some (mi, ma) deg treeList] by simp
      finally show ?thesis
        by simp
    qed
  next
    case False
   hence 6: T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 26 + T_delete (treeList
! ?h) ?l  +(
                    T_minNull ?newnode +  2 + (if x = ma then 6+  T_maxt (?newlist ! ?h)  else
1)) using 5 by simp
    moreover have invar-vebt (?newlist ! ?h) n
      by (simp add: True ‹invar-vebt (vebt-delete (treeList ! high x (deg div 2)) (low x (deg div
2))) n›)
    ultimately have T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤
          29 + T_delete (treeList ! ?h) ?l + (if x = ma then 6+  T_maxt (?newlist ! ?h)  else 1)
      using minNull-bound[of ?newnode] by linarith
    moreover have  (if x = ma then 6+  T_maxt (?newlist ! ?h)  else 1) ≤ 9
      apply(cases x = ma) using maxt-bound[of  (?newlist ! ?h) ] by simp+
    ultimately have  T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤
                  38 + T_delete (treeList ! ?h) ?l by force
    also have ... ≤ 38 + (1 + height (treeList ! ?h))∗70
      by (meson 5.IH(1) True le-trans nat-add-left-cancel-le nth-mem)
    also have ... ≤ 38 + (height  (Node (Some (mi, ma)) deg treeList summary))∗70
      using height-compose-child[of treeList ! ?h  treeList Some (mi, ma) deg summary]
```

```
          by (meson True mult-le-mono1 nat-add-left-cancel-le nth-mem)
        finally show ?thesis
          by presburger
      qed
    next
      case False
      hence T_delete (Node (Some (mi, ma)) deg treeList summary) x = 21+1 using 3 by simp
      hence T_delete (Node (Some (mi, ma)) deg treeList summary) x ≤ 22 using
        mint-bound[of treeList ! the (vebt-mint summary)]
        mint-bound[of summary] by simp
      then show ?thesis by simp
    qed
  qed
 qed
qed

theorem delete-bound-size-univ: invar-vebt t n ⟹ u = 2^n ⟹ T_delete t x ≤ 140 + 70 * lb (lb u)
  using delete-bound-height[of t n x] height-double-log-univ-size[of u n t] by simp

fun T_delete'::VEBT ⇒ nat ⇒ nat where
  T_delete' (Leaf a b) 0 = 1|
  T_delete' (Leaf a b) (Suc 0) = 1|
  T_delete' (Leaf a b) (Suc (Suc n)) = 1|
  T_delete' (Node None deg treeList summary) - = 1|
  T_delete' (Node (Some (mi, ma)) 0 treeList summary) x = 1|
  T_delete' (Node (Some (mi, ma)) (Suc 0) treeList summary) x =1 |
  T_delete' (Node (Some (mi, ma)) deg treeList summary) x =  (
            if (x < mi ∨ x > ma) then 1
            else if (x = mi ∧ x = ma) then 1
            else  (  let xn = (if x = mi
                        then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the (vebt-mint summary)))
                        else x);
                  minn = (if x = mi then xn else mi);
                  l = low xn (deg div 2);
                  h = high xn (deg div 2) in
                   if h < length treeList
                     then(  T_delete' (treeList ! h) l  +(
                       let newnode = vebt-delete (treeList ! h) l;
                           newlist = treeList[h:= newnode]in
                       if minNull newnode
                       then  T_delete' summary h
                       else 1
                     ))else  1  ))


lemma tdeletemimi':deg ≥ 2 ⟹ T_delete' (Node (Some (mi, mi)) deg treeList summary) x ≤ 1
```

```
  using T_delete'.simps(7)[of mi mi deg−2 treeList summary x]
  apply(cases x ≠ mi)
  apply (metis add-2-eq-Suc' le-add-diff-inverse2 le-eq-less-or-eq linorder-neqE-nat)
  by (metis add-2-eq-Suc' eq-imp-le le-add-diff-inverse2)

lemma minNull-delete-time-bound': invar-vebt t n ⟹ minNull (vebt-delete t x) ⟹  T_delete' t x ≤
1
proof(induction t n rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
    by (metis T_delete'.simps(1) T_delete'.simps(2) T_delete'.simps(3) vebt-buildup.cases order-refl)
next
  case (4 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
   by (metis One-nat-def VEBT-internal.set-n-deg-not-0 gr0-implies-Suc linorder-not-less not-less-eq-eq)
  then have deg = Suc (Suc (n' + n'))
    unfolding ‹deg = n + m› ‹m = n› by linarith
  show ?case
  proof (cases (x < mi ∨ x > ma))
    case True
    then show ?thesis
      using 4.prems ‹deg = Suc (Suc -)› delt-out-of-range by force
  next
    case False
    hence x ≤ ma ∧ x ≥ mi by simp
    show ?thesis
    proof (cases (x = mi ∧ x = ma))
      case True
      then show ?thesis
        using ‹deg = Suc (Suc -)› tdeletemimi' by force
    next
      case False
      hence ¬ (x = mi ∧ x = ma) by simp
      then obtain mi' ma' treeList' summary' where
        vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
            Node (Some (mi', ma')) deg treeList' summary'
        by atomize-elim (simp add: ‹deg = Suc (Suc -)› Let-def)
      then have ¬ minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)
        by simp
      then have False
        using ‹minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)›
        by contradiction
      then show ?thesis ..
    qed
  qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
   by (metis One-nat-def VEBT-internal.set-n-deg-not-0 gr0-implies-Suc linorder-not-less not-less-eq-eq)
```

```
  then have deg = Suc (Suc (Suc (n' + n')))
    unfolding ‹deg = n + m› ‹m = Suc n› by linarith
  show ?case
  proof (cases (x < mi ∨ x > ma))
    case True
    then show ?thesis
      using 5.prems ‹deg = Suc (Suc -)› delt-out-of-range by force
  next
    case False
    hence mi ≤ x andx ≤ ma by simp-all
    then show ?thesis
    proof (cases (x = mi ∧ x = ma))
      case True
      then show ?thesis
        using ‹deg = Suc (Suc -)› tdeletemimi' by force
    next
      case False
      hence ¬ (x = mi ∧ x = ma) by simp
      then obtain mi' ma' treeList' summary' where
        vebt-delete (Node (Some (mi, ma)) deg treeList summary) x =
          Node (Some (mi', ma')) deg treeList' summary'
        by atomize-elim (simp add: ‹deg = Suc (Suc -)› Let-def)
      then have ¬ minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)
        by simp
      then have False
        using ‹minNull (vebt-delete (Node (Some (mi, ma)) deg treeList summary) x)›
        by contradiction
      then show ?thesis ..
    qed
  qed
qed simp+


lemma delete-bound-height': invar-vebt t n ⟹ T_delete' t x ≤ 1+ height t
proof(induction t n arbitrary: x rule: invar-vebt.induct)
  case (1 a b)
  then show ?case
    apply(cases x ≤ 0)
    apply simp
    apply(cases x = 1)
    apply simp
    using T_delete'.simps(3)[of a b x−2] height.simps(1)[of a b]
    by (metis One-nat-def T_delete'.simps(3) vebt-buildup.cases lessI less-Suc-eq-le plus-1-eq-Suc)
next
  case (4 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis deg-not-0 gr0-implies-Suc)
  then have deg = Suc (Suc (n' + n'))
    using ‹deg = n + m› ‹m = n› by presburger
```

```
  show ?case
  proof (cases (x < mi ∨ x > ma))
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    hence miama:mi ≤ x ∧ x ≤ ma by simp
    then show ?thesis
    proof(cases x = mi ∧ x = ma)
      case True
      then show ?thesis
        by (simp add: ‹deg = Suc (Suc -)›)
    next
      case False
      let ?xn = (if x = mi
                        then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                        else x)
      let ?minn = (if x = mi then ?xn else mi)
      let ?l = low ?xn (deg div 2)
      let ?h = high ?xn (deg div 2)
      have 0:T_delete' (Node (Some (mi, ma)) deg treeList summary) x = (if ?h < length treeList
                  then(  T_delete' (treeList ! ?h) ?l  +(
                     let newnode = vebt-delete (treeList ! ?h) ?l;
                         newlist = treeList[?h:= newnode]in
                     if minNull newnode
                     then  T_delete' summary ?h
                     else 1
                  ))else  1)
        using False miama
        by (simp add: ‹deg = Suc (Suc -)› Let-def)
      show ?thesis
      proof (cases ?h < length treeList)
        case True
        let ?newnode = vebt-delete (treeList ! ?h) ?l
        let ?newlist = treeList[?h:= ?newnode]
        have 1:T_delete' (Node (Some (mi, ma)) deg treeList summary) x =
                          T_delete' (treeList ! ?h) ?l  +(
                     if minNull ?newnode
                     then  T_delete' summary ?h
                     else 1  ) using 0 True by simp
        then show ?thesis
        proof(cases minNull ?newnode)
          case True
          hence  T_delete' (treeList ! ?h) ?l ≤ 1
            by (metis 0 1 4.IH(1) minNull-delete-time-bound' nat-le-iff-add nth-mem)
          hence T_delete' (Node (Some (mi, ma)) deg treeList summary) x ≤ 1 + T_delete' summary ?h
            using 1 True by auto
```

```
        also have ... ≤ 1+1+ height summary
          using 4(2)[of ?h] by simp
        also have ... ≤ 1 + height (Node (Some (mi, ma)) deg treeList summary)
          by simp
        finally show ?thesis .
      next
        case False
        hence T_delete' (Node (Some (mi, ma)) deg treeList summary) x =
                          1+  T_delete' (treeList ! ?h) ?l    using 1 by simp
        also have ... ≤ 1 + 1 + height (treeList ! ?h)
          using 4.IH(1) True by force
        also have ... ≤ 1 + height (Node (Some (mi, ma)) deg treeList summary)
          by (simp add: True)
        finally show ?thesis .
      qed
    qed (simp add : 0)
  qed
qed
next
  case (5 treeList n summary m deg mi ma)
  then obtain n' :: nat where n = Suc n'
    by (metis Suc-0-div-numeral(1) Suc-eq-plus1-left add.commute div-by-1 list-decode.cases nth-mem
      numerals(1) valid-0-not)
  then have deg = Suc (Suc (Suc (n' + n')))
    using ‹deg = n + m› ‹m = Suc n› by presburger
  show ?case
  proof(cases (x < mi ∨ x > ma))
    case True
    then show ?thesis
      by (simp add: ‹deg = Suc (Suc -)›)
  next
    case False
    hence miama:mi ≤ x ∧ x ≤ ma by simp
    then show ?thesis
    proof(cases x = mi ∧ x = ma)
      case True
      then show ?thesis
        by (simp add: ‹deg = Suc (Suc -)›)
    next
      case False
      let ?xn = (if x = mi
                        then the (vebt-mint summary) * 2^(deg div 2) + the (vebt-mint (treeList ! the
(vebt-mint summary)))
                        else x)
      let ?minn = (if x = mi then ?xn else mi)
      let ?l = low ?xn (deg div 2)
      let ?h = high ?xn (deg div 2)
      have 0:T_delete' (Node (Some (mi, ma)) deg treeList summary) x = (if ?h < length treeList
                  then(  T_delete' (treeList ! ?h) ?l  +(
```

```
                    let newnode = vebt-delete (treeList ! ?h) ?l;
                        newlist = treeList[?h:= newnode]in
                       if minNull newnode
                       then  T_delete' summary ?h
                       else 1
                  ))else  1)
      using False miama
      by (simp add: ‹deg = Suc (Suc -)› Let-def)
    show ?thesis
    proof (cases ?h < length treeList)
      case True
      let ?newnode = vebt-delete (treeList ! ?h) ?l
      let ?newlist = treeList[?h:= ?newnode]
      have 1:T_delete' (Node (Some (mi, ma)) deg treeList summary) x =
                         T_delete' (treeList ! ?h) ?l  +(
                       if minNull ?newnode
                       then  T_delete' summary ?h
                       else 1  ) using 0 True by simp
      then show ?thesis
      proof(cases minNull ?newnode)
        case True
        hence  T_delete' (treeList ! ?h) ?l ≤ 1
          by (metis 0 1 5.IH(1) minNull-delete-time-bound' nat-le-iff-add nth-mem)
        hence T_delete' (Node (Some (mi, ma)) deg treeList summary) x ≤ 1+ T_delete' summary ?h
          using 1 True by auto
        also have ... ≤ 1 + 1 + height summary
          using 5(2)[of ?h] by simp
        also have ... ≤ 1 + height (Node (Some (mi, ma)) deg treeList summary)
          by simp
        finally show ?thesis .
      next
        case False
        hence T_delete' (Node (Some (mi, ma)) deg treeList summary) x =
                          1+  T_delete' (treeList ! ?h) ?l    using 1 by simp
        also have ... ≤ 1 + 1+ height (treeList ! ?h)
          using 5.IH(1) True by auto
        also have ... ≤ 1 + height (Node (Some (mi, ma)) deg treeList summary)
          by (simp add: True)
        finally show ?thesis .
      qed
    qed (simp add : 0)
  qed
 qed
qed simp+

theorem delete-bound-size-univ': invar-vebt t n ⟹ u =  2^n ⟹   T_delete'  t x ≤ 2 +  lb (lb u)
 using delete-bound-height'[of t n x] height-double-log-univ-size[of u n t] by simp

end
```

end

theory *VEBT-Space* **imports** *VEBT-Definitions Complex-Main*
begin

12 Space Complexity and *buildup* Time Consumption

12.1 Space Comlexity of valid van Emde Boas Trees

Space Complexity is linear in relation to universe sizes

context *VEBT-internal* **begin**

fun *space*:: *VEBT* ⇒ *nat* **where**
space (*Leaf a b*) = *3*|
space (*Node info deg treeList summary*) = *5* + *space summary* + *length treeList* + *foldr* (λ *a b*. *a*+*b*) (*map space treeList*) *0*

fun *space'*:: *VEBT* ⇒ *nat* **where**
space' (*Leaf a b*) = *4*|
space' (*Node info deg treeList summary*) = *6* + *space' summary* + *foldr* (λ *a b*. *a*+*b*) (*map space' treeList*) *0*

Count in reals

fun *cnt*:: *VEBT* ⇒ *real* **where**
cnt (*Leaf a b*) = *1*|
cnt (*Node info deg treeList summary*) = *1* + *cnt summary* + *foldr* (λ *a b*. *a*+*b*) (*map cnt treeList*) *0*

12.2 Auxiliary Lemmas for List Summation

lemma *list-every-elemnt-bound-sum-bound*:∀ *x* ∈ *set xs*. *f x* ≤ *bound* ⟹ *foldr* (λ *a b*. *a*+*b*) (*map f xs*) *i* ≤ *length xs* ∗ *bound* + *i*
by(*induction xs*) *auto*

lemma *list-every-elemnt-bound-sum-bound-real*:∀ *x* ∈ *set* (*xs*::'*a list*). (*f*::'*a*⇒*real*) *x* ≤ (*bound*::*real*) ⟹ *foldr* (λ *a b*. *a*+*b*) (*map f xs*) *i* ≤ *real*(*length xs*) ∗ *bound* + *i*
apply(*induction xs*) **apply** *simp*
apply (*simp add*: *algebra-simps*)
done

lemma *foldr-one*: *d* ≤ *foldr* (+) *ys* (*d*::*nat*)
by (*induction ys*) *auto*

lemma *foldr-zero*: ∀ *i* < *length xs*. *xs* ! *i* > *0* ⟹
foldr (λ *a b*. *a*+*b*) *xs* (*d*::*nat*) − *d* ≥ *length xs*
proof(*induction xs*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a xs*)

hence $\forall\, i{<}length\ xs.\ 0 < xs\ !\ i$
by *auto*
hence $length\ xs \leq foldr\ (+)\ xs\ d - d$ **using** *Cons.IH* **by** *simp*
have $a \geq 1$
by (*metis gr0-conv-Suc length-Cons less-one local.Cons*(2) *not-gr0 not-less nth-Cons-0*)
then show *?case*
by (*metis Nat.add-diff-assoc* ‹$length\ xs \leq foldr\ (+)\ xs\ d - d$› *add-mono-thms-linordered-semiring*(1) *foldr.simps*(2) *foldr-one length-Cons o-apply plus-1-eq-Suc*)
qed

lemma *foldr-mono*: $length\ xs = length\ ys \Longrightarrow \forall\ i < length\ xs.\ xs\ !\ i < ys\ !\ i \Longrightarrow c \leq d \Longrightarrow$
$foldr\ (\lambda\ a\ b.\ a{+}b)\ xs\ c\ + length\ ys \leq foldr\ (\lambda\ a\ b.\ a{+}b)\ ys\ (d{::}nat)$
proof(*induction xs arbitrary*: *d c ys*)
case *Nil*
then show *?case* **using** *length-0-conv list.size*(3) *foldr-one* **by** *simp*
next
case (*Cons a xs*)
then obtain *y ys1* **where** $ys = y\ \#ys1$
by (*metis Suc-leI Suc-le-length-iff nth-equalityI*)
hence *0*:$length\ xs = length\ ys1$
using *Cons.prems*(1) **by** *force*
hence *1*:$\forall\, i{<}length\ xs.\ xs\ !\ i < ys1\ !\ i$ **using** *Cons.prems*(2)
using ‹$ys = y\ \#\ ys1$› **by** *force*
hence *3*: $\forall\, i{<}length\ ys1.\ ys1\ !\ i > 0$
by (*metis 0 less-nat-zero-code neq0-conv*)
have $foldr\ (+)\ (a\ \#\ xs)c = a + foldr\ (+)\ xs\ (c)$ **by** *simp*
have $foldr\ (+)\ (ys)\ d = y + foldr\ (+)\ ys1\ (d)$
by (*simp add*: ‹$ys = y\ \#\ ys1$›)
have *2*:$a < y$ **using** *Cons.prems*(2) ‹$ys = y\ \#\ ys1$›
by (*metis length-Cons nth-Cons-0 zero-less-Suc*)
have *4*:$foldr\ (+)\ xs\ c \leq foldr\ (+)\ ys1\ d - length\ ys1$
using *Cons.IH*[*of ys1 c d*] *0 1 Cons.prems*(3) **by** *simp*
have $foldr\ (+)\ ys1\ d \geq length\ ys1$ **using** *foldr-zero*[*of ys1 d*] *3* **by** *simp*
hence $a + foldr\ (+)\ xs\ c < y + foldr\ (+)\ ys1\ d - length\ ys1$ **using** *2 foldr-zero*[*of ys1 d*] *4* **by** *simp*
then show *?case*
using ‹$ys = y\ \#\ ys1$› **by** *auto*
qed

lemma *two-realpow-ge-two* :$(n{::}real) \geq 1 \Longrightarrow (2{::}real)\hat{}n \geq 2$
by (*metis less-one not-less of-nat-1 of-nat-le-iff of-nat-numeral power-increasing power-one-right zero-neq-numeral*)

lemma *foldr0*: $foldr\ (+)\ xs\ (c{+}d) = foldr\ (+)\ xs\ (d{::}real) + c$
by(*induction xs*) *auto*

lemma *f-g-map-foldr-bound*: $(\forall\ x \in set\ xs.\ f\ x \leq c * g\ x)$
$\Longrightarrow foldr\ (\lambda\ a\ b.\ a{+}b)\ (map\ f\ xs)\ d \leq c * foldr\ (\lambda\ a\ b.\ a{+}b)\ (map\ g\ xs)\ (0{::}real) + d$
by(*induction xs*) (*auto simp add*: *algebra-simps*)

lemma *real-nat-list*: *real* (*foldr* (+) (*map f xs*) (*c*::*nat*))
= *foldr* (+) (*map* (λ *x*. *real*(*f x*))*xs*) *c*
by(*induction xs arbitrary*: *c*) *auto*

12.3 Actual Space Reasoning

lemma *space-space'*: *space' t* > *space t*
proof(*induction t*)
case (*Node info deg treeList summary*)
hence ∀ *i* < *length treeList* . (*map space treeList*)!*i* < (*map space' treeList*)!*i*
by *simp*
hence *0*:*foldr* (+) (*map space treeList*) *0* + *length treeList* ≤ *foldr* (+) (*map space' treeList*) *0*
using *foldr-mono*[*of* (*map space treeList*) (*map space' treeList*) *0 0*] **by** *simp*
have *1*:*space summary* < *space' summary* **using** *Node* **by** *simp*
hence *foldr* (+) (*map space treeList*) *0* + *length treeList* + *space summary* ≤
foldr (+) (*map space' treeList*) *0* + *space' summary* **using** *0* **by** *simp*
then show *?case* **using** *space'.simps*(*2*)[*of info deg treeList summary*]
space.simps(*2*)[*of info deg treeList summary*] **by** *simp*
qed *simp*

lemma *cnt-bound*:
defines *c* ≡ *1.5*
shows *invar-vebt t n* ⟹ *cnt t* ≤ *2*∗((*2*^*n* − *c*)::*real*)
proof(*induction t n rule*: *invar-vebt.induct*)
case (*2 treeList n summary m deg*)
hence ∀ *t*∈*set treeList*. (*cnt t*) ≤ *2* ∗ (*2* ^ *n* − *c*) **by** *simp*
hence *foldr* (λ *a b*. *a*+*b*) (*map cnt treeList*) *0* ≤ *2*^*n*∗*2* ∗ ((*2*^*n* − *c*)::*real*)
using *list-every-elemnt-bound-sum-bound-real*[*of treeList cnt 2*∗((*2*^*n* − *c*)::*real*) *0*] *2*
by (*auto simp add*: *algebra-simps*)
hence *cnt* (*Node None deg treeList summary*) ≤ *2*∗(*2*^*n*+*1*)∗(*2*^*n*−*c*) + *1* **using** *2*
by(*auto simp add*: *algebra-simps*)
hence *cnt* (*Node None deg treeList summary*) ≤ *2*∗(*2*^(*n*+*n*) + (*1*−*c*)∗*2*^*n* − *c* + *1*/*2*)
by(*auto simp add*: *algebra-simps power-add*)
moreover have *2*∗(*2*^(*n*+*n*) + (*1*−*c*)∗*2*^*n* − *c* + *1*/*2*) ≤ *2*∗(*2*^(*n*+*n*) + −*0.5*∗*1* − *1.5* + *1*/*2*)
by(*auto simp add*: *algebra-simps two-realpow-ge-one c-def*)
moreover hence *2*∗(*2*^(*n*+*n*) + (*1*−*c*)∗*2*^*n* − *c* + *1*/*2*) ≤ *2*∗(*2*^(*n*+*n*) − *1.5*)
by(*auto simp add*: *algebra-simps power-add*)
ultimately have *cnt* (*Node None deg treeList summary*) ≤ *2*∗(*2*^(*n*+*n*) − *1.5*) **by** *simp*
then show *?case* **using** *c-def 2*(*5*) *2*(*6*) **by** *simp*
next
case (*3 treeList n summary m deg*)
hence ∀ *t*∈*set treeList*. (*cnt t*) ≤ *2* ∗ (*2* ^ *n* − *c*) **by** *simp*
hence *foldr* (λ *a b*. *a*+*b*) (*map cnt treeList*) *0* ≤ *2*^(*n*+*1*)∗*2* ∗ ((*2*^*n* − *c*)::*real*)
using *list-every-elemnt-bound-sum-bound-real*[*of treeList cnt 2*∗((*2*^*n* − *c*)::*real*) *0*] *3*
by (*auto simp add*: *algebra-simps*)
moreover
hence *cnt* (*Node None deg treeList summary*) ≤ *2*∗(*2*^*n*∗*2*^*m* − *c*∗ *2*^(*m*) + *2*^(*m*) − *c* + *1*/*2*)
using *3*

by (*auto simp add*: *algebra-simps powr-add*)
moreover have *2*(2^n*2^m − c* 2^(m) + 2^(m) − c + 1/2) = 2*(2^(n+m) + (1−c)* 2^(m) − c + 1/2)*
by (*auto simp add*: *algebra-simps power-add*)
moreover have *2*(2^(n+m) + (1−c)* 2^(m) − c + 1/2) ≤ 2*(2^(n+m) + −0.5*1 − 1.5 + 1/2)*
by(*auto simp add*: *algebra-simps two-realpow-ge-one c-def*)
moreover hence *2*(2^(n+m) + (1−c)*2^m − c + 1/2) ≤ 2*(2^(n+m) − 1.5)*
by(*auto simp add*: *algebra-simps power-add*)
ultimately have *cnt (Node None deg treeList summary) ≤ 2*(2^(n+m) − 1.5)* **by** *simp*
then show *?case* **using** *c-def 3(5) 3(6)* **by** *simp*
next
case (*4 treeList n summary m deg mi ma*)
hence *∀ t∈set treeList. (cnt t) ≤ 2 * (2 ^ n − c)* **by** *simp*
hence *foldr (λ a b. a+b) (map cnt treeList) 0 ≤ 2^n*2 * ((2^n − c)::real)*
using *list-every-elemnt-bound-sum-bound-real*[*of treeList cnt 2*((2^n − c)::real) 0*] *4*
by (*auto simp add*: *algebra-simps*)
hence *cnt (Node (Some (mi, ma)) deg treeList summary) ≤ 2*(2^n+1)*(2^n−c) + 1* **using** *4*
by(*auto simp add*: *algebra-simps*)
hence *cnt (Node None deg treeList summary) ≤ 2*(2^(n+n) + (1−c)*2^n − c + 1/2)*
by(*auto simp add*: *algebra-simps power-add*)
moreover have *2*(2^(n+n) + (1−c)*2^n − c + 1/2) ≤ 2*(2^(n+n) + −0.5*1 − 1.5 + 1/2)*
by(*auto simp add*: *algebra-simps two-realpow-ge-one c-def*)
moreover hence *2*(2^(n+n) + (1−c)*2^n − c + 1/2) ≤ 2*(2^(n+n) − 1.5)*
by(*auto simp add*: *algebra-simps power-add*)
ultimately have *cnt (Node None deg treeList summary) ≤ 2*(2^(n+n) − 1.5)* **by** *simp*
then show *?case* **using** *c-def 4* **by** *simp*
next
case (*5 treeList n summary m deg mi ma*)
hence *∀ t∈set treeList. (cnt t) ≤ 2 * (2 ^ n − c)* **by** *simp*
hence *foldr (λ a b. a+b) (map cnt treeList) 0 ≤ 2^(n+1)*2 * ((2^n − c)::real)*
using *list-every-elemnt-bound-sum-bound-real*[*of treeList cnt 2*((2^n − c)::real) 0*] *5*
by (*auto simp add*: *algebra-simps*)
moreover
hence *cnt (Node (Some (mi, ma)) deg treeList summary) ≤ 2*(2^n*2^m − c* 2^(m) + 2^(m) − c + 1/2)*
using *5*
by (*auto simp add*: *algebra-simps powr-add*)
moreover have *2*(2^n*2^m − c* 2^(m) + 2^(m) − c + 1/2) = 2*(2^(n+m) + (1−c)* 2^(m) − c + 1/2)*
by (*auto simp add*: *algebra-simps power-add*)
moreover have *2*(2^(n+m) + (1−c)* 2^(m) − c + 1/2) ≤ 2*(2^(n+m) + −0.5*1 − 1.5 + 1/2)*
by(*auto simp add*: *algebra-simps two-realpow-ge-one c-def*)
moreover hence *2*(2^(n+m) + (1−c)*2^m − c + 1/2) ≤ 2*(2^(n+m) − 1.5)*
by(*auto simp add*: *algebra-simps power-add*)
ultimately have *cnt (Node None deg treeList summary) ≤ 2*(2^(n+m) − 1.5)* **by** *simp*
then show *?case* **using** *c-def 5* **by** *simp*
qed (*simp add*: *cnt.simps c-def*)

theorem *cnt-bound'*: *invar-vebt t n* $\Longrightarrow$ *cnt t* $\leq$ *2* $*$ *(2* $\hat{}$ *n* $-$ *1)*
using *cnt-bound* **by** *fastforce*

lemma *space-cnt*: *space' t* $\leq$ *6*$*$*cnt t*
proof(*induction t*)
case (*Node info deg treeList summary*)
hence $\forall$ *t*$\in$*set treeList. space' t* $\leq$ *6* $*$ *cnt t* **by** *blast*
hence *foldr* (λ *a b. a+b*) (*map space' treeList*) *0* $\leq$
6 $*$*foldr* (λ *a b. a+b*) (*map cnt treeList*) *0*
using *f-g-map-foldr-bound*[*of treeList space' 6 cnt 0*]
by(*auto simp add*: *algebra-simps real-nat-list*)
then show *?case*
using *Node.IH*(*2*) **by** *force*
qed *simp*

lemma *space-2-pow-bound*: **assumes** *invar-vebt t n* **shows** *real* (*space' t*) $\leq$ *12* $*$ (*2*$\hat{}$*n* $-$*1*)
by (*smt* (*verit*, *best*) *assms cnt-bound' space-cnt*)

lemma *space'-bound*:
assumes *invar-vebt t n u* = *2*$\hat{}$*n*
shows *space' t* $\leq$ *12* $*$ *u*
proof $-$
have *real* (*space' t*) $\leq$ *real* (*12* $*$ *u*)
using *assms* **using** *space-2-pow-bound*[*of t n*] **by** *fastforce*
then have *space' t* $\leq$ *12* $*$ *2* $\hat{}$ *n*
using *assms of-nat-le-iff* **by** *blast*
then show *?thesis*
using *assms* **by** *auto*
qed

Main Theorem

theorem *space-bound*:
assumes *invar-vebt t n u* = *2*$\hat{}$*n*
shows *space t* $\leq$ *12* $*$ *u*
by (*metis assms dual-order.trans less-imp-le-nat space'-bound space-space'*)

12.4 Complexity of Generation Time

Space complexity is closely related to tree generation time complexity

Time approximation for replicate function. $T_{replicate}\ n\ t\ x$ denotes runnig time of the n-times replication of x into a list. t models runtime for generation of a single x.

fun $T_{buildup}$::*nat* $\Rightarrow$ *nat* **where**
$T_{buildup}$ *0* = *3*|
$T_{buildup}$ (*Suc 0*) = *3*|
$T_{buildup}$ *n* = (*if even n then 1* + (*let half* = *n div 2 in*
9 + $T_{buildup}$ *half* + (*2*$\hat{}$*half*) $*$ ($T_{buildup}$ *half* + *1*))
else (*let half* = *n div 2 in*

```
                11 + T_buildup (Suc half) +  (2^(Suc half))* (T_buildup half + 1 )))

fun T_build::nat ⇒ nat where
T_build 0 = 4|
T_build (Suc 0) = 4|
T_build  n = (if even n then 1 + (let half = n div 2 in
               10 + T_build half +  (2^half) * (T_build half))
             else (let half = n div 2 in
                  12 + T_build (Suc half) +  (2^(Suc half))* (T_build half)))

lemma buildup-build-time: T_buildup  n < T_build  n
proof(induction n rule: T_buildup.induct)
  case (3 va)
  then show ?case
  proof(cases even (Suc (Suc va)))
    case True

    have T_buildup (Suc (Suc va)) = 10 + T_buildup (Suc (Suc va) div 2) +
      2 ^ (Suc (Suc va) div 2) * (T_buildup (Suc (Suc va) div 2) + 1)
      unfolding T_buildup.simps T_build.simps if-P[OF True] Let-def
      by presburger

    also have ... < 11 + T_build (Suc (Suc va) div 2) +
      2 ^ (Suc (Suc va) div 2) * T_build (Suc (Suc va) div 2)
      using 3.IH True
     by (metis Suc-eq-plus1 Suc-leI add-less-le-mono add-less-mono less-add-one nat-mult-le-cancel-disj
         numeral-plus-one semiring-norm(5))

    also have ... = T_build (Suc (Suc va))
      unfolding T_buildup.simps T_build.simps if-P[OF True] Let-def
      by presburger

    finally show ?thesis .
  next
    case False
    hence *: (let half = Suc (Suc va) div 2
          in 11 + T_buildup (Suc half) + 2 ^ Suc half * (T_buildup half + 1))
           <  (let half = Suc (Suc va) div 2
            in 12 + T_build (Suc half) + 2 ^ Suc half * T_build half)
      unfolding Let-def
    proof−
      assume odd (Suc (Suc va))
    have  11 + T_buildup (Suc (Suc (Suc va) div 2))
          < 12 + T_build (Suc (Suc (Suc va) div 2))
      using 3.IH(3) False add-less-mono by presburger
    moreover have  2 ^ Suc (Suc (Suc va) div 2) * (T_buildup (Suc (Suc va) div 2) + 1)
                 ≤ 2 ^ Suc (Suc (Suc va) div 2) * T_build (Suc (Suc va) div 2)
      by (metis 3.IH(4) False Suc-leI add.commute mult-le-mono2 plus-1-eq-Suc)
    ultimately show  11 + T_buildup (Suc (Suc (Suc va) div 2)) +
```

$2 \hat{} Suc\ (Suc\ (Suc\ va)\ div\ 2) * (T_{buildup}\ (Suc\ (Suc\ va)\ div\ 2) + 1)$
$< 12 + T_{build}\ (Suc\ (Suc\ (Suc\ va)\ div\ 2)) +$
$2 \hat{} Suc\ (Suc\ (Suc\ va)\ div\ 2) * T_{build}\ (Suc\ (Suc\ va)\ div\ 2)$
using *add-mono-thms-linordered-field*(3) **by** *blast*
qed
show *?thesis* **apply**(*subst* $T_{buildup}$.*simps*)
apply(*subst* T_{build}.*simps*)
using *False* ∗
by *simp*
qed
qed *simp*+

lemma *listsum-bound*: $(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x \geq (0::real)) \Longrightarrow$
foldr (+) (*map f xs*) *y* ≥ *y*
apply(*induction xs arbitrary*: *y*)
apply *simp*
apply(*subst list.map*(2))
apply(*subst foldr.simps*)
apply (*simp add*: *add-increasing*)
done

lemma *cnt-non-neg*: *cnt t* ≥ 0
by (*induction t*) (*simp add*: *VEBT-internal.listsum-bound*)+

lemma *foldr-same*: $(\bigwedge x\ y.\ x \in set\ (xs::real\ list) \Longrightarrow y \in set\ xs \Longrightarrow x = y) \Longrightarrow$
$(\bigwedge x\ .\ (x::real) \in set\ xs \Longrightarrow x = (y::real)) \Longrightarrow$
foldr $(\lambda\ (a::real)\ (b::real).\ a+b)$ *xs* 0 = *real* (*length xs*) ∗ *y*
apply(*induction xs*)
apply *simp*
apply(*subst foldr.simps*)
unfolding *comp-def*
proof −
fix *a* :: *real* **and** *xsa* :: *real list*
assume *a1*: $[\![\bigwedge x\ y.\ [\![x \in set\ xsa;\ y \in set\ xsa]\!] \Longrightarrow x = y;\ \bigwedge x.\ x \in set\ xsa \Longrightarrow x = y]\!] \Longrightarrow$ *foldr* (+) *xsa* 0 = *real* (*length xsa*) ∗ *y*
assume $\bigwedge x\ y.\ [\![x \in set\ (a\ \#\ xsa);\ y \in set\ (a\ \#\ xsa)]\!] \Longrightarrow x = y$
assume *a2*: $\bigwedge x.\ x \in set\ (a\ \#\ xsa) \Longrightarrow x = y$
then have *f3*: *a* = *y*
by *simp*
then have *a* ∗ *real* (*length xsa*) = *foldr* (+) *xsa* 0
using *a2 a1* **by** (*metis* (*no-types*) *list.set-intros*(2) *mult.commute*)
then show *a* + *foldr* (+) *xsa* 0 = *real* (*length* (*a* # *xsa*)) ∗ *y*
using *f3* **by** (*simp add*: *distrib-left mult.commute*)
qed

lemma *foldr-same-int*: $(\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow x = y) \Longrightarrow$
$(\bigwedge x\ .\ x \in set\ xs \Longrightarrow x = y) \Longrightarrow$
foldr (+) *xs* 0 = (*length xs*) ∗ *y*

```
  apply(induction xs)
    apply simp
    apply(subst foldr.simps)
    apply fastforce
  done

lemma t-build-cnt: T_build n ≤ cnt (vebt-buildup n) * 13
proof(induction n rule: T_build.induct)
  case 1
  then show ?case by simp
next
  case 2
  then show ?case by simp
next
  case (3 va)
  then show ?case
  proof(cases even (Suc (Suc va)))
    case True
    hence *: T_build (Suc (Suc va)) = 11+
          T_build (Suc (Suc va) div 2) +
         2 ^(Suc (Suc va) div 2) * (T_build (Suc (Suc va) div 2))
      apply(subst T_build.simps)
      by simp
    have real (T_build (Suc (va div 2))) ≤ 13 * cnt (vebt-buildup (Suc (va div 2)))
      using 3.IH(1) True by force
    moreover hence 1: 2 ^(Suc (Suc va) div 2)* (T_build (Suc (Suc va) div 2)) ≤
                  2 ^(Suc (Suc va) div 2) * ((cnt (vebt-buildup (Suc (Suc va) div 2)))*13)
       using ordered-semiring-class.mult-mono[of (T_build (Suc (Suc va) div 2)) ((cnt (vebt-buildup
(Suc (Suc va) div 2)))*13)
           2 ^(Suc (Suc va) div 2) 2 ^(Suc (Suc va) div 2)] by simp
    ultimately have T_build (Suc (Suc va) div 2) +
        2 ^(Suc (Suc va) div 2) * (T_build (Suc (Suc va) div 2)) ≤
           cnt (vebt-buildup (Suc (Suc va) div 2))*13 +
         2 ^(Suc (Suc va) div 2) * ((cnt (vebt-buildup (Suc (Suc va) div 2)))*13)
          by (smt (verit) 3.IH(1) True of-nat-add)
        have 10: (foldr (+)
              (replicate (l) ((cnt (vebt-buildup (Suc (Suc va) div 2)))
               )) 0) =
                l * ((cnt (vebt-buildup (Suc (Suc va) div 2)))) for l
          using foldr-same[of (replicate l (cnt (vebt-buildup (Suc (Suc va) div 2))))
                        cnt (vebt-buildup (Suc (Suc va) div 2)) ]
            length-replicate by simp
        have cnt (vebt-buildup (Suc (Suc va) div 2))*13 +
         2 ^(Suc (Suc va) div 2) * ((cnt (vebt-buildup (Suc (Suc va) div 2)))*13) + 11≤
         13* cnt (vebt-buildup (Suc (Suc va)))
         apply(subst vebt-buildup.simps)
      using True apply simp
      apply(subst sym[OF foldr-replicate])
    proof−
```

assume *even va*
have *2∗ (2 ^(va div 2) ∗ cnt (vebt-buildup (Suc (va div 2)))) =*
foldr (+) (replicate (2 ∗ 2 ^(va div 2)) (cnt (vebt-buildup (Suc (va div 2))))) 0
apply(*rule sym*)
using *10 div2-Suc-Suc*[*of va*] **by** *simp*
then show *26 ∗ (2 ^(va div 2) ∗ cnt (vebt-buildup (Suc (va div 2))))*
≤ 2 + 13 ∗ foldr (+) (replicate (2 ∗ 2 ^(va div 2)) (cnt (vebt-buildup (Suc (va div 2))))) 0
by *simp*
qed
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *∗ 1 3.IH*(*1*) *True numeral-Bit1 numeral-plus-numeral numeral-plus-one of-nat-add of-nat-numeral semiring-norm*(*2*))
next
case *False*
have *12 + T_{build} (Suc (Suc (va div 2))) + 2 ^Suc (Suc (va div 2)) ∗ T_{build} (Suc (va div 2))*
≤ cnt (Node None (Suc (Suc va)) (replicate (2 ^Suc (Suc (va div 2))) (vebt-buildup (Suc (va div 2))))
(vebt-buildup (Suc (Suc (va div 2))))) ∗ 13
apply(*subst cnt.simps*)
proof−
have *10*: (*foldr (+)*
(replicate (l) ((cnt (vebt-buildup (Suc (Suc va) div 2)))
)) 0) =
l ∗ ((cnt (vebt-buildup (Suc (Suc va) div 2))) **for** *l*
using *foldr-same*[*of* (*replicate l* (*cnt* (*vebt-buildup* (*Suc* (*Suc va*) *div 2*))))
cnt (*vebt-buildup* (*Suc* (*Suc va*) *div 2*))]
length-replicate **by** *simp*
hence *map-cnt*: *foldr (+) (map cnt (replicate (2 ^Suc (Suc (va div 2))) (vebt-buildup (Suc (va div 2))))) 0 =*
2 ^Suc (Suc (va div 2)) ∗ cnt (vebt-buildup (Suc (va div 2))) **by** *simp*
have T_{build} *(Suc (Suc (va div 2))) ≤ 13 ∗ cnt (vebt-buildup (Suc (Suc (va div 2))))*
using *3.IH*(*3*) *False* **by** *force*
moreover have T_{build} *(Suc (va div 2)) ≤ 13 ∗ cnt*(*vebt-buildup (Suc (va div 2))*)
using *3.IH*(*4*) *False* **by** *force*
moreover have *add-double-trans*: (*a*::*real*) *≤ b ⟹ c ≤ d ⟹*
i ≥ 0⟹ a + c∗i ≤ b + d∗i **for** *a b c d i*
using *mult-right-mono* **by** *fastforce*
ultimately have *real*(T_{build} *(Suc (Suc (va div 2)))) + 2 ^Suc (Suc (va div 2)) ∗ real*(T_{build} *(Suc (va div 2))) ≤*
13 ∗ cnt (vebt-buildup (Suc (Suc (va div 2)))) +
2 ^Suc (Suc (va div 2)) ∗ (13 ∗ cnt(*vebt-buildup (Suc (va div 2))*))
by (*meson add-mono-thms-linordered-semiring*(*1*) *mult-mono of-nat-0-le-iff order-refl zero-le-numeral zero-le-power*)
hence *11*:(*12 + T_{build} (Suc (Suc (va div 2))) + 2 ^Suc (Suc (va div 2)) ∗ T_{build} (Suc (va div 2))) ≤*
12 + 13 ∗ cnt (vebt-buildup (Suc (Suc (va div 2)))) +
2 ^Suc (Suc (va div 2)) ∗ 13 ∗ cnt(*vebt-buildup (Suc (va div 2))*)
using *algebra-simps* **by** *simp*
show *(12 + T_{build} (Suc (Suc (va div 2))) +*

```
    2 ^ Suc (Suc (va div 2)) * T_build (Suc (va div 2)))
  ≤ (1 + cnt (vebt-buildup (Suc (Suc (va div 2)))) +
     foldr (+) (map cnt (replicate (2 ^ Suc (Suc (va div 2))) (vebt-buildup (Suc (va div 2))))) 0) *
13
       apply(subst map-cnt)
       using 11 algebra-simps by simp
     qed
  then show ?thesis
    apply(subst vebt-buildup.simps)
    apply(subst T_build.simps)
    using False by force
qed
qed

lemma t-buildup-cnt: T_buildup n ≤ cnt (vebt-buildup n) * 13
  apply(rule order.trans[where b = real(T_build n)])
  apply(rule order.strict-implies-order)
  apply (simp add: VEBT-internal.buildup-build-time)
  apply(rule t-build-cnt)
 done

lemma count-buildup: cnt (vebt-buildup n) ≤ 2 * 2^n
 by (smt (verit, ccfv-threshold) VEBT-internal.cnt-bound' add.right-neutral add-less-mono buildup-gives-valid
cnt.simps(1) even-Suc lessI odd-pos one-le-power plus-1-eq-Suc vebt-buildup.elims)

lemma count-buildup': cnt (vebt-buildup n) ≤ 2 * (2::nat)^n
   by (simp add: VEBT-internal.count-buildup)

theorem vebt-buildup-bound: u = 2^n ⟹ T_buildup n ≤ 26 * u
  using count-buildup'[of n] t-buildup-cnt[of n] by linarith
```

Count in natural numbers

```
fun cnt'::: VEBT ⇒ nat where
cnt' (Leaf a b) = 1|
cnt' (Node info deg treeList summary) = 1 + cnt' summary + foldr (λ a b. a+b) (map cnt' treeList)
0

lemma cnt-cnt-eq:cnt t = cnt' t
proof(induction t)
  case (Node x1 x2 x3 t)
  then show ?case
    by simp (smt (verit, best) map-eq-conv of-nat-0 real-nat-list)
qed auto

end
end
```

13 Functional Interface

theory *VEBT-Intf-Functional*
imports *Main*
VEBT-Definitions VEBT-Space
VEBT-Uniqueness
VEBT-Member
VEBT-Insert VEBT-InsertCorrectness
VEBT-MinMax
VEBT-Pred VEBT-Succ
VEBT-Bounds
VEBT-Delete VEBT-DeleteCorrectness VEBT-DeleteBounds

begin

13.1 Code Generation Setup

13.1.1 Code Equations

Code generator seems to not support patterns and nat code target

```
context begin
  interpretation VEBT-internal .

lemma vebt-member-code [code]:
  vebt-member (Leaf a b) x = (if x = 0 then a else if x=1 then b else False)
  vebt-member (Node None t r e) x = False
  vebt-member (Node (Some (mi, ma)) deg treeList summary) x =
  (if deg = 0 ∨ deg = Suc 0 then False else (
    if x = mi then True else
    if x = ma then True else
    if x < mi then False else
   if x > ma then False else
    (let
          h = high x (deg div 2);
          l = low x (deg div 2) in
       (if h < length treeList
          then vebt-member (treeList ! h) l
          else False))))
     apply simp
    apply simp
  proof(goal-cases)
    case 1
    consider deg = 0| deg = Suc 0
      | n  where deg = Suc (Suc n)
      by (meson vebt-buildup.cases)
    then show ?case apply(cases)
      by simp-all
  qed
```

```
lemma vebt-insert-code [code]:
  vebt-insert (Leaf a b) x = (if x=0 then Leaf True b else if x=1 then Leaf a True else Leaf a b)
  vebt-insert (Node info deg treeList summary) x = (
    if deg ≤ 1 then
      (Node info deg treeList summary)
    else ( case info of
      None ⇒  (Node (Some (x,x)) deg  treeList summary)
    | Some mima ⇒ ( case mima of (mi, ma) ⇒ (
        let
          xn = (if x < mi then mi else x);
          minn = (if x< mi then x else mi);
          l= low xn (deg div 2); h = high xn (deg div 2)
        in (
          if h < length treeList ∧ ¬ (x = mi ∨ x = ma) then
            Node (Some (minn, max xn ma))
                 deg
                 (treeList[h:= vebt-insert (treeList ! h) l])
                 (if minNull (treeList ! h) then  vebt-insert summary h else summary)
          else Node (Some (mi, ma)) deg treeList summary)
    ))))
    apply simp
   apply simp
proof(goal-cases)
  case 1
  consider deg = 0| deg = Suc 0
    | n  where deg = Suc (Suc n)
    by (meson vebt-buildup.cases)
  then show ?case apply(cases)
      apply simp+
    apply(cases info)
     apply simp+
    apply(cases the info)
    apply simp
    by meson

qed

lemma vebt-succ-code [code]:
  vebt-succ (Leaf a b) x = (if b∧ x = 0 then Some 1 else None)
  vebt-succ (Node info  deg treeList summary) x = (if deg ≤ 1 then None else
  (case info of None ⇒ None |
  (Some mima) ⇒ (case mima of (mi, ma) ⇒ (
                if x < mi then (Some mi)
                else (let l = low x (deg div 2); h = high x (deg div 2) in(
                     if h < length treeList then
                            let maxlow = vebt-maxt (treeList ! h) in
                            (if maxlow ≠ None ∧ (Some l <o  maxlow) then
                                          Some (2^(deg div 2)) *o Some h +o vebt-succ (treeList ! h) l
                             else let sc = vebt-succ summary h in
```

```
                    if sc = None then None
                    else Some (2^(deg div 2)) *o sc +o vebt-mint (treeList ! the sc) )

              else None))))))
  apply (cases (Leaf a b,x) rule: vebt-succ.cases; simp)
  apply (cases (Node info  deg treeList summary,x) rule: vebt-succ.cases; simp add: Let-def)
  done

lemma vebt-pred-code[code]:
  vebt-pred (Leaf a b) x = (if x = 0 then None else if x = 1 then
                              (if a then Some 0 else None) else
                           (if b then Some 1 else if a then Some 0 else None)) and
  vebt-pred (Node info deg treeList summary) x =(if deg ≤ 1 then None else (
  case info of None ⇒ None |
  (Some mima) ⇒ (case mima of (mi, ma) ⇒ (
                    if x > ma then Some ma
                    else (let l = low x (deg div 2); h = high x (deg div 2) in
                    if h < length treeList then
                        let minlow = vebt-mint (treeList ! h) in
                        (if minlow ≠ None ∧ (Some l >o  minlow) then
                                    Some (2^(deg div 2)) *o Some h +o vebt-pred (treeList ! h) l
                         else let pr = vebt-pred summary h in
                         if pr = None then (if x > mi then Some mi else None)
                         else Some (2^(deg div 2)) *o pr +o vebt-maxt (treeList ! the pr) )
                  else None)))))
  apply (cases (Leaf a b,x) rule: vebt-pred.cases; simp)
  apply (cases (Node info  deg treeList summary,x) rule: vebt-pred.cases; simp add: Let-def)
  done


lemma vebt-delete-code [code]:
  vebt-delete (Leaf a b) x = (if x = 0 then Leaf False b else if x = 1 then Leaf a False else Leaf a b)
  vebt-delete (Node info deg treeList summary) x = (
    case info of
      None ⇒ (Node info deg treeList summary)
    | Some mima ⇒ (
        if deg ≤ 1 then (Node info deg treeList summary)
        else (case mima of (mi, ma) ⇒ (
          if (x < mi ∨ x > ma) then (Node (Some (mi, ma)) deg treeList summary)
          else if (x = mi ∧ x = ma) then (Node None deg treeList summary)
          else let
            xn = (if x = mi then the (vebt-mint summary) * 2^(deg div 2)
                                + the (vebt-mint (treeList ! the (vebt-mint summary)))
                  else x);
            minn = (if x = mi then xn else mi);
            l = low xn (deg div 2);
            h = high xn (deg div 2)
          in
            if h < length treeList then let
```

```
            newnode = vebt-delete (treeList ! h) l;
            newlist = treeList[h:= newnode]
          in
            if minNull newnode then let
              sn = vebt-delete summary h;
              maxn =
                if xn = ma then let
                  maxs = vebt-maxt sn
                  in
                    if maxs = None then minn
                    else 2^(deg div 2) * the maxs + the (vebt-maxt (newlist ! the maxs))
                else ma
              in (Node (Some (minn, maxn)) deg newlist sn)
            else let
              maxn = (if xn = ma then h * 2^(deg div 2) + the( vebt-maxt (newlist ! h))
                                 else ma)
            in (Node (Some (minn, maxn)) deg newlist summary)
          else (Node (Some (mi, ma)) deg treeList summary)
    ))))
  apply (cases (Leaf a b,x) rule: vebt-delete.cases; simp)
  apply (cases (Node info  deg treeList summary,x) rule: vebt-delete.cases; simp add: Let-def)
  done
end

lemmas [code] =
  VEBT-internal.high-def VEBT-internal.low-def VEBT-internal.minNull.simps
  VEBT-internal.less.simps VEBT-internal.mul-def VEBT-internal.add-def
  VEBT-internal.option-comp-shift.simps VEBT-internal.option-shift.simps

export-code
  vebt-buildup
  vebt-insert
  vebt-member
  vebt-maxt
  vebt-mint
  vebt-pred
  vebt-succ
  vebt-delete
  checking SML
```

13.2 Correctness Lemmas

```
named-theorems vebt-simps ‹Simplifier rules for VEBT functional interface›

locale vebt-inst =
  fixes n :: nat
begin

  interpretation VEBT-internal .
```

13.2.1 Space Bound

theorem *vebt-space-linear-bound*:
fixes t
defines $u \equiv 2\hat{}n$
shows *invar-vebt* $t\ n \Longrightarrow$ *space* $t \leq 12{*}u$
by (*simp add*: *space-bound u-def*)

13.2.2 Buildup

lemma *invar-vebt-buildup*[*vebt-simps*]: *invar-vebt* (*vebt-buildup* n) $n \longleftrightarrow n{>}0$
by (*auto simp add*: *buildup-gives-valid deg-not-0*)

lemma *set-vebt-buildup*[*vebt-simps*]: *set-vebt* (*vebt-buildup* i) $= \{\}$
by (*metis VEBT-internal.buildup-gives-empty VEBT-internal.buildup-gives-valid VEBT-internal.set-vebt-set-vebt'-valid neq0-conv invar-vebt.intros*(*1*) *vebt-buildup.simps*(*1*))

lemma *time-vebt-buildup*: $u = 2\hat{}n \Longrightarrow T_{buildup}\ \ n \leq 26 * u$
using *vebt-buildup-bound* **by** *simp*

13.2.3 Equality

lemma *set-vebt-equal*[*vebt-simps*]: *invar-vebt* $t_1\ n \Longrightarrow$ *invar-vebt* $t_2\ n \Longrightarrow t_1{=}t_2 \longleftrightarrow$ *set-vebt* $t_1 =$ *set-vebt* t_2
by (*auto simp*: *unique-tree*)

13.2.4 Member

lemma *set-vebt-member*[*vebt-simps*]: *invar-vebt* $t\ n \Longrightarrow$ *vebt-member* $t\ x \longleftrightarrow x{\in}$*set-vebt* t
by (*rule member-correct*)

theorem *time-vebt-member*: *invar-vebt* $t\ n \Longrightarrow u = \ 2\hat{}n \Longrightarrow\ \ T_{member}\ t\ x \leq 30 + 15 * lb\ (lb\ u)$
using *member-bound-size-univ* **by** *auto*

13.2.5 Insert

theorem *invar-vebt-insert*[*vebt-simps*]: *invar-vebt* $t\ n \Longrightarrow x{<}\ 2\hat{}n \Longrightarrow$ *invar-vebt* (*vebt-insert* $t\ x$) n
by (*simp add*: *valid-pres-insert*)

theorem *set-vebt-insert*[*vebt-simps*]: *invar-vebt* $t\ n\ \ \Longrightarrow x < 2\hat{}n\ \ \Longrightarrow$ *set-vebt* (*vebt-insert* $t\ x$) $=$ *set-vebt* $t \cup \{x\}$
by (*meson insert-correct*[*symmetric*])

theorem *time-vebt-insert*: *invar-vebt* $t\ n \Longrightarrow u = \ 2\hat{}n \Longrightarrow\ \ \ T_{insert}\ t\ x \leq 46 + 23 * lb\ (lb\ u)$
by (*meson insert-bound-size-univ*)

13.2.6 Maximum

theorem *set-vebt-maxt*: *invar-vebt* $t\ n \Longrightarrow$ *vebt-maxt* $t =$ *Some* $x \longleftrightarrow$ *max-in-set* (*set-vebt* t) x
by (*metis maxt-sound maxt-corr set-vebt-set-vebt'-valid*)

theorem *set-vebt-maxt'*: *invar-vebt t n* $\Longrightarrow$ *vebt-maxt t = Some x* $\longleftrightarrow$ (*x*$\in$*set-vebt t* $\wedge$ ($\forall$ *y*$\in$*set-vebt t. x*$\geq$*y*))
using *set-vebt-maxt* **unfolding** *max-in-set-def* **by** *blast*

lemma *set-vebt-maxt''*[*vebt-simps*]:
invar-vebt t n $\Longrightarrow$ *vebt-maxt t* = (*if set-vebt t* = {} *then None else Some* (*Max* (*set-vebt t*)))
by (*metis Max-ge Max-in VEBT-internal.set-vebt-finite VEBT-internal.set-vebt-set-vebt'-valid empty-iff option.exhaust set-vebt-maxt'*)

lemma *time-vebt-maxt*: T_{maxt} *t* $\leq$ *3*
by (*simp add*: *maxt-bound*)

13.2.7 Minimum

theorem *set-vebt-mint*[*vebt-simps*]: *invar-vebt t n* $\Longrightarrow$ *vebt-mint t = Some x* $\longleftrightarrow$ *min-in-set* (*set-vebt t*) *x*
by (*metis VEBT-internal.mint-corr VEBT-internal.mint-sound VEBT-internal.set-vebt-set-vebt'-valid*)

theorem *set-vebt-mint'*: *invar-vebt t n* $\Longrightarrow$ *vebt-mint t = Some x* $\longleftrightarrow$ (*x*$\in$*set-vebt t* $\wedge$ ($\forall$ *y*$\in$*set-vebt t. x*$\leq$*y*))
using *set-vebt-mint* **unfolding** *min-in-set-def* **by** *blast*

lemma *set-vebt-mint''*[*vebt-simps*]:
invar-vebt t n $\Longrightarrow$ *vebt-mint t* = (*if set-vebt t* = {} *then None else Some* (*Min* (*set-vebt t*)))
by (*metis Min-in Min-le VEBT-internal.set-vebt-finite VEBT-internal.set-vebt-set-vebt'-valid empty-iff option.exhaust set-vebt-mint'*)

lemma *time-vebt-mint*: T_{mint} *t* $\leq$ *3*
by (*simp add*: *mint-bound*)

13.3 Emptiness determination

A tree is empty if and only if its minimum is None

lemma *vebt-minNull-mint*: *minNull t* $\longleftrightarrow$ *vebt-mint t = None*
by (*meson VEBT-internal.minNullmin VEBT-internal.minminNull*)

lemma *set-vebt-minNull*: *invar-vebt t n* $\Longrightarrow$ *minNull t* $\longleftrightarrow$ *set-vebt t* = {}
by (*metis VEBT-internal.minNullmin VEBT-internal.minminNull VEBT-internal.mint-corr-help-empty VEBT-internal.set-vebt-set-vebt'-valid vebt-inst.set-vebt-mint''*)

lemma *time-vebt-minNull*: $T_{minNull}$ *t* $\leq$ *1*
using *minNull-bound* **by** *auto*

13.3.1 Successor

theorem *set-vebt-succ*: *invar-vebt t n* $\Longrightarrow$ *vebt-succ t x = Some sx* $\longleftrightarrow$ *is-succ-in-set* (*set-vebt t*) *x sx*
by (*simp add*: *succ-corr set-vebt-set-vebt'-valid*)

lemma *set-vebt-succ'*[*vebt-simps*]: *invar-vebt t n* $\Longrightarrow$ *vebt-succ t x* = (*if* $\exists$ *y* $\in$ *set-vebt t*. *y* > *x then Some* (*LEAST y* $\in$ *set-vebt t*. *y* > *x*) *else None*)
apply (*clarsimp*;*safe*)
subgoal
apply(*clarsimp simp add*: *succ-correct is-succ-in-set-def Least-le*)
by (*metis* (*no-types*, *lifting*) *LeastI-ex*)
subgoal by (*meson succ-correct is-succ-in-set-def option.exhaust-sel*)
done

theorem *time-vebt-succ*:
fixes *t* **defines** *u* $\equiv$ *2^n*
shows *invar-vebt t n* $\Longrightarrow$ T_{succ} *t x* $\leq$ *54* + *27* $*$ *lb* (*lb u*)
using *succ-bound-size-univ* **unfolding** *u-def* **by** *presburger*

13.3.2 Predecessor

theorem *set-vebt-pred*: *invar-vebt t n* $\Longrightarrow$ *vebt-pred t x* = *Some px* $\longleftrightarrow$ *is-pred-in-set* (*set-vebt t*) *x px*
by (*simp add*: *pred-corr set-vebt-set-vebt'-valid*)

theorem *set-vebt-pred'*[*vebt-simps*]: *invar-vebt t n* $\Longrightarrow$
vebt-pred t x = (*if* $\exists$ *y* $\in$ *set-vebt t*. *y* < *x then Some* (*GREATEST y*. *y* $\in$ *set-vebt t* $\wedge$ *y* < *x*) *else None*)
apply (*clarsimp simp*: *member-correct pred-empty pred-correct is-pred-in-set-def*)
by (*metis* (*no-types*, *lifting*) *GreatestI-nat Greatest-le-nat less-imp-le*)

theorem *time-vebt-pred*: **fixes** *t* **defines** *u* $\equiv$ *2^n*
shows *invar-vebt t n* $\Longrightarrow$ T_{pred} *t x* $\leq$ *58* + *29* $*$ *lb* (*lb u*)
unfolding *u-def* **by** (*meson pred-bound-size-univ*)

13.3.3 Delete

theorem *invar-vebt-delete*[*vebt-simps*]: *invar-vebt t n* $\Longrightarrow$ *invar-vebt* (*vebt-delete t x*) *n*
by (*simp add*: *delete-pres-valid*)

theorem *set-vebt-delete*[*vebt-simps*]: *invar-vebt t n* $\Longrightarrow$ *set-vebt* (*vebt-delete t x*) = *set-vebt t* $-$ {*x*}
by (*metis delete-correct invar-vebt-delete set-vebt-set-vebt'-valid*)

theorem *time-vebt-delete*: **fixes** *t* **defines** *u* $\equiv$ *2^n*
shows *invar-vebt t n* $\Longrightarrow$ T_{delete} *t x* $\leq$ *140* + *70* $*$ *lb* (*lb u*)
unfolding *u-def* **by** (*meson delete-bound-size-univ*)

end

13.4 Interface Usage Example

experiment
begin

definition *test n xs ys* $\equiv$ *let*

```
    t = vebt-buildup n;
    t = foldl vebt-insert t (0#xs);

    f = (λx. if vebt-member t x then x else the (vebt-pred t x))
  in
    map f ys


  context fixes n :: nat begin
    interpretation vebt-inst n .

    lemmas [simp] = vebt-simps

    lemma [simp]:
      assumes invar-vebt t n ∀ x∈set xs. x<2^n
      shows invar-vebt (foldl vebt-insert t xs) n
      using assms apply (induction xs arbitrary: t)
      by auto

    lemma [simp]:
      assumes invar-vebt t n ∀ x∈set xs. x<2^n
      shows set-vebt (foldl vebt-insert t xs) = set-vebt t ∪ set xs
      using assms
      apply (induction xs arbitrary: t)
      apply auto
      done

    lemma ⟦∀ x∈set xs. x<2^n; n>0⟧ ⟹ test n xs ys = map (λy. (GREATEST y'. y'∈insert 0 (set
xs) ∧ y'≤y)) ys
      unfolding test-def
      apply (auto simp add: Let-def)
      subgoal by (metis (mono-tags, lifting) Greatest-equality le-zero-eq)
      subgoal by (metis (no-types, lifting) Greatest-equality order-refl)
      subgoal by (metis less-le)
      done

  end

end




end

theory VEBT-List-Assn
imports
  Separation-Logic-Imperative-HOL/Sep-Main
  HOL−Library.Rewrite
```

begin

13.5 Lists

fun *list-assn* :: (*'a* ⇒ *'c* ⇒ *assn*) ⇒ *'a list* ⇒ *'c list* ⇒ *assn* **where**
  list-assn P [] [] = *emp*
| *list-assn P* (*a*#*as*) (*c*#*cs*) = *P a c* ∗ *list-assn P as cs*
| *list-assn* - - - = *false*

lemma *list-assn-aux-simps*[*simp*]:
  list-assn P [] *l'* = (↑(*l'*=[]))
  list-assn P l [] = (↑(*l*=[]))
   apply (*cases l'*)
    apply *simp*
   apply *simp*
  apply (*cases l*)
   apply *simp*
  apply *simp*
  done

lemma *list-assn-aux-append*[*simp*]:
  length l1=*length l1'* ⟹
    list-assn P (*l1*@*l2*) (*l1'*@*l2'*)
    = *list-assn P l1 l1'* ∗ *list-assn P l2 l2'*
  apply (*induct rule*: *list-induct2*)
   apply *simp*
  apply (*simp add*: *star-assoc*)
  done

lemma *list-assn-aux-ineq-len*: *length l* ≠ *length li* ⟹ *list-assn A l li* = *false*
proof (*induction l arbitrary*: *li*)
  case (*Cons x l li*) **thus** *?case* **by** (*cases li*; *auto*)
qed *simp*

lemma *list-assn-aux-append2*[*simp*]:
  assumes *length l2*=*length l2'*
  shows *list-assn P* (*l1*@*l2*) (*l1'*@*l2'*)
    = *list-assn P l1 l1'* ∗ *list-assn P l2 l2'*
  apply (*cases length l1* = *length l1'*)
   apply (*erule list-assn-aux-append*)
  apply (*simp add*: *list-assn-aux-ineq-len assms*)
  done

lemma *list-assn-simps*[*simp*]:
  (*list-assn P*) [] [] = *emp*
  (*list-assn P*) (*a*#*as*) (*c*#*cs*) = *P a c* ∗ (*list-assn P*) *as cs*
  (*list-assn P*) (*a*#*as*) [] = *false*
  (*list-assn P*) [] (*c*#*cs*) = *false*

```
    apply simp-all
  done


lemma list-assn-mono:
  ⟦⋀x x'. P x x' ⟹A P' x x'⟧ ⟹ list-assn P l l' ⟹A list-assn P' l l'
  apply (induct P l l' rule: list-assn.induct)
  by (auto intro: ent-star-mono)

lemma list-assn-cong[fundef-cong]:
  assumes xs=xs' xsi=xsi'
  assumes ⋀x xi. x∈set xs' ⟹ xi∈set xsi' ⟹ A x xi = A' x xi
  shows list-assn A xs xsi = list-assn A' xs' xsi'
  using assms
  apply (induct A≡A xs' xsi' arbitrary: xs xsi rule: list-assn.induct)
    apply simp-all
  done

term prod-list


definition listI-assn I A xs xsi ≡
    ↑(length xsi=length xs ∧ I⊆{0..<length xs})
  ∗ Finite-Set.fold (λi a. a ∗ A (xs!i) (xsi!i)) 1 I


lemmas comp-fun-commute-fold-insert =
  comp-fun-commute-on.fold-insert[where S=UNIV, folded comp-fun-commute-def', simplified]


lemma aux: Finite-Set.fold (λi aa. aa ∗ P ((a # as) ! i) ((c # cs) ! i)) emp {0..<Suc (length as)}
  = P a c ∗ Finite-Set.fold (λi aa. aa ∗ P (as ! i) (cs ! i)) emp {0..<length as}
proof −
  have 1: {0..<Suc (length as)} = insert 0 {1..<Suc (length as)} by auto

  have 2: {Suc 0..<Suc (Suc n)} = insert (Suc n) {Suc 0 ..< Suc n} for n by auto
  have 3: {0..<Suc n} = insert n {0..<n} for n by auto

  have A:
    Finite-Set.fold P emp {Suc 0..<Suc n}
    = Finite-Set.fold Q emp {0..<n}
    if ∀ i x. P (Suc i) x = Q i x
      and comp-fun-commute P
      and comp-fun-commute Q
    for P Q n
    using that
    apply (induction n arbitrary: a)
    subgoal by simp
    thm comp-fun-commute-on.fold-insert
```

```
    apply (simp add: comp-fun-commute-fold-insert)
    apply (subst 2)
    apply (subst 3)
    apply (simp add: comp-fun-commute-fold-insert)
    done
  show ?thesis
    apply (simp add: 1)
    apply (subst comp-fun-commute-fold-insert)
    subgoal
      apply unfold-locales
      apply (auto simp: fun-eq-iff algebra-simps)
      done
    subgoal by simp
    subgoal by simp
    apply simp
    apply (rewrite at ¤ = -*- mult.commute)
    apply (rule arg-cong[where f=λx. P - - ∗ x])
    apply (rule A)
    subgoal by auto
    subgoal
      apply unfold-locales
      apply (auto simp: fun-eq-iff algebra-simps)
      done
    subgoal
      apply unfold-locales
      apply (auto simp: fun-eq-iff algebra-simps)
      done
    done
qed


lemma list-assn-conv-idx: list-assn A xs xsi = listI-assn {0..<length xs} A xs xsi
  apply (induction A xs xsi rule: list-assn.induct)
     apply (auto simp: listI-assn-def aux)
  done

lemma listI-assn-conv: n=length xs ⟹ listI-assn {0..<n} A xs xsi = list-assn A xs xsi
  by (simp add: list-assn-conv-idx)

lemma listI-assn-conv': n=length xs ⟹ listI-assn {0..<n} A xs xsi ∗F = list-assn A xs xsi∗ F
  by (simp add: list-assn-conv-idx)

lemma listI-assn-finite[simp]: ¬finite I ⟹ listI-assn I A xs xsi = false
  using subset-eq-atLeast0-lessThan-finite by (auto simp: listI-assn-def)


find-theorems Finite-Set.fold name: cong

lemma mult-fun-commute: comp-fun-commute (λi (a::assn). a ∗ f i)
```

apply *unfold-locales*
apply (*auto simp*: *fun-eq-iff mult-ac*)
done

lemma *listI-assn-weak-cong*:
assumes *I*: $I{=}I'$ $A{=}A'$ *length xs=length xs' length xsi=length xsi'*
assumes *A*: $\bigwedge i.\ [\![i{\in}I;\ i{<}$*length xs*; *length xs=length xsi* $]\!]$
$\Longrightarrow$ $xs!i = xs'!i \wedge xsi!i = xsi'!i$
shows *listI-assn I A xs xsi = listI-assn I' A' xs' xsi'*
unfolding *listI-assn-def*
apply (*simp add*: *I*)
apply (*cases length xsi' = length xs'* $\wedge$ $I' \subseteq \{0..{<}$*length xs'*$\}$; *simp only*:; *simp*)
apply (*rule Finite-Set.fold-cong*[**where** *S=UNIV*, *folded comp-fun-commute-def'*])
apply (*simp-all add*: *mult-fun-commute*)
subgoal by (*meson subset-eq-atLeast0-lessThan-finite*)
subgoal using *A* **by** (*auto simp*: *fun-eq-iff I*)
done

lemma *listI-assn-cong*:
assumes *I*: $I{=}I'$ *length xs=length xs' length xsi=length xsi'*
assumes *A*: $\bigwedge i.\ [\![i{\in}I;\ i{<}$*length xs*; *length xs=length xsi* $]\!]$
$\Longrightarrow$ $xs!i = xs'!i \wedge xsi!i = xsi'!i$
$\wedge$ $A\ (xs!i)\ (xsi!i) = A'\ (xs'!i)\ (xsi'!i)$
shows *listI-assn I A xs xsi = listI-assn I' A' xs' xsi'*
unfolding *listI-assn-def*
apply (*simp add*: *I*)
apply (*cases length xsi' = length xs'* $\wedge$ $I' \subseteq \{0..{<}$*length xs'*$\}$; *simp only*:; *simp*)
apply (*rule Finite-Set.fold-cong*[**where** *S=UNIV*, *folded comp-fun-commute-def'*])
apply (*simp-all add*: *mult-fun-commute*)
subgoal by (*meson subset-eq-atLeast0-lessThan-finite*)
subgoal using *A* **by** (*fastforce simp*: *fun-eq-iff I*)
done

lemma *listI-assn-insert*: $i{\notin}I \Longrightarrow i{<}$*length xs* $\Longrightarrow$
listI-assn (*insert i I*) *A xs xsi* = *A* (*xs*!*i*) (*xsi*!*i*) $*$ *listI-assn I A xs xsi*
apply (*cases finite I*; *simp?*)
unfolding *listI-assn-def*
apply (*subst comp-fun-commute-fold-insert*)
subgoal
apply *unfold-locales*
apply (*auto simp*: *fun-eq-iff algebra-simps*)
done
subgoal by *simp*
subgoal by *simp*
subgoal by (*auto simp*: *algebra-simps*)
done

lemma *listI-assn-extract*:
assumes $i{\in}I$ $i{<}$*length xs*
shows *listI-assn I A xs xsi* $=$ *A* (*xs*!*i*) (*xsi*!*i*) $*$ *listI-assn* ($I{-}\{i\}$) *A xs xsi*
proof $-$
have *1*: I $=$ *insert i* ($I{-}\{i\}$) **using** *assms* **by** *auto*
show *?thesis*
apply (*subst 1*)
apply (*subst listI-assn-insert*)
using *assms* **by** *auto*
qed

lemma *listI-assn-reinsert*:
assumes $P \Longrightarrow_A$ *A* (*xs*!*i*) (*xsi*!*i*) $*$ *listI-assn* ($I{-}\{i\}$) *A xs xsi* $*$ F
assumes $i{<}$*length xs* $i{\in}I$
assumes *listI-assn I A xs xsi* $*$ $F \Longrightarrow_A Q$
shows $P \Longrightarrow_A Q$
proof $-$
show *?thesis*
apply (*rule ent-trans*[*OF assms*(*1*)])
apply (*subst listI-assn-extract*[*symmetric*])
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
done
qed

lemma *listI-assn-reinsert-upd*:
fixes *xs xsi* :: *- list*
assumes $P \Longrightarrow_A$ *A x xi* $*$ *listI-assn* ($I{-}\{i\}$) *A xs xsi* $*$ F
assumes $i{<}$*length xs* $i{\in}I$
assumes *listI-assn I A* (*xs*[*i*:=*x*]) (*xsi*[*i*:=*xi*]) $*$ $F \Longrightarrow_A Q$
shows $P \Longrightarrow_A Q$
proof (*cases length xs* $=$ *length xsi*)
case *True*
have *1*: *listI-assn* ($I{-}\{i\}$) *A xs xsi* $=$ *listI-assn* ($I{-}\{i\}$) *A* (*xs*[*i*:=*x*]) (*xsi*[*i*:=*xi*])
by (*rule listI-assn-cong*) *auto*

have *2*: *A x xi* $=$ *A* ((*xs*[*i*:=*x*])!*i*) ((*xsi*[*i*:=*xi*])!*i*) **using** ‹*i*<*length xs*› *True* **by** *auto*

from *assms*[*unfolded 1 2*] **show** *?thesis*
apply (*rule-tac listI-assn-reinsert*)
apply *assumption*
apply *simp-all*
done

next
case *False*
with *assms*(*1*) **have** $P \Longrightarrow_A$ *false*

```
    by (simp add: listI-assn-def)
  thus ?thesis using ent-false-iff entailsI by blast
qed


lemma listI-assn-reinsert':
  assumes P ⟹_A A (xs!i) (xsi!i) ∗ listI-assn (I−{i}) A xs xsi ∗ F
  assumes i<length xs i∈I
  assumes <listI-assn I A xs xsi ∗ F>c<Q>
  shows <P>c<Q>
proof −
  show ?thesis
    apply (rule cons-pre-rule[OF assms(1)])
    apply (subst listI-assn-extract[symmetric])
    subgoal by fact
    subgoal by fact
    subgoal by fact
    done
qed

lemma listI-assn-reinsert-upd':
  fixes xs xsi :: - list
  assumes P ⟹_A A x xi ∗ listI-assn (I−{i}) A xs xsi ∗ F
  assumes i<length xs i∈I
  assumes <listI-assn I A (xs[i:=x]) (xsi[i:=xi]) ∗ F> c <Q>
  shows <P> c <Q>
  by (meson assms(1) assms(2) assms(3) assms(4) cons-pre-rule ent-refl listI-assn-reinsert-upd)

lemma subst-not-in:
  assumes i∉I  i<length xs
  shows listI-assn I A (xs[i:=x1]) (xsi[i := x2]) = listI-assn I A xs xsi
  apply (rule listI-assn-cong)
  using assms
  by (auto simp add: nth-list-update')

lemma listI-assn-subst:
  assumes i∉I  i<length xs
  shows listI-assn (insert i I) A (xs[i:=x1]) (xsi[i := x2]) = A x1 x2 ∗ listI-assn I A xs xsi
  by (smt (verit, best) assms(1,2) length-list-update listI-assn-def listI-assn-insert
      nth-list-update-eq pure-false star-false-left star-false-right subst-not-in)

lemma extract-pre-list-assn-lengthD: h ⊨ list-assn A xs xsi ⟹ length xsi = length xs
  by (metis list-assn-aux-ineq-len mod-false)

method unwrap-idx for i ::nat =
  (rewrite in <⌑>-<-> list-assn-conv-idx),
  (rewrite in <⌑>-<-> listI-assn-extract[where i=i]),
  (simp split: if-splits; fail),
  (simp split: if-splits; fail)
```

method *wrap-idx* **uses** *R* =
(*rule R*),
frame-inference,
(*simp split*: *if-splits*; *fail*),
(*simp split*: *if-splits*; *fail*),
(*subst listI-assn-conv*, (*simp*; *fail*))

method *extract-pre-pure* **uses** *dest* =
(*rule hoare-triple-preI* | *drule asm-rl*[*of* -⊨-]),
(*determ* ‹*elim mod-starE dest*[*elim-format*]›)*?*,
((*determ* ‹*thin-tac* - ⊨ -›)+)*?*,
(*simp* (*no-asm*) *only*: *triv-forall-equality*)*?*

lemma *rule-at-index*:
assumes
1:*P* ⟹$_A$ *list-assn A xs xsi* ∗ *F* **and**
2[*simp*]:*i* < *length xs* **and**
3:<*A* (*xs* ! *i*) (*xsi* ! *i*) ∗
listI-assn ({*0*..<*length xs*}− {*i*}) *A xs xsi* ∗ *F*> *c* <*Q'*> **and**
4: ⋀ *r*. *Q' r* ⟹$_A$ *A* (*xs* ! *i*) (*xsi* ! *i*) ∗
listI-assn ({*0*..<*length xs*}− {*i*}) *A xs xsi*∗ *F' r*
shows
<*P*>*c* <λ *r*. *list-assn A xs xsi* ∗ *F' r*>
apply(*rule cons-pre-rule*[*OF 1*])
apply(*unwrap-idx i*)
apply(*rule cons-post-rule*)
apply(*rule 3*)
apply(*rule ent-trans*[*OF 4*])
apply(*wrap-idx R*: *listI-assn-reinsert-upd*)
apply *simp*
done

end

theory *VEBT-BuildupMemImp*
imports
VEBT-List-Assn
VEBT-Space
Deriving.Derive
VEBT-Member VEBT-Insert
HOL−Library.Countable
Time-Reasoning/Time-Reasoning VEBT-DeleteBounds
begin

14 Imperative van Emde Boas Trees

datatype *VEBTi* = *Nodei* (*nat*∗*nat*) *option nat VEBTi array VEBTi* | *Leafi bool bool*

derive *countable VEBTi*
instance *VEBTi* :: *heap* **by** *standard*

14.1 Assertions on van Emde Boas Trees

fun *vebt-assn-raw* :: *VEBT* ⇒ *VEBTi* ⇒ *assn* **where**
vebt-assn-raw (*Leaf a b*) (*Leafi ai bi*) = ↑(*ai*=*a* ∧ *bi*=*b*)
| *vebt-assn-raw* (*Node mmo deg tree-list summary*) (*Nodei mmoi degi tree-array summaryi*) = (
↑(*mmoi*=*mmo* ∧ *degi*=*deg*)
∗ *vebt-assn-raw summary summaryi*
∗ (∃$_A$ *tree-is*. *tree-array* ↦$_a$ *tree-is* ∗ *list-assn vebt-assn-raw tree-list tree-is*)
)
| *vebt-assn-raw* - - = *false*

lemmas [*simp del*] = *vebt-assn-raw.simps*

context *VEBT-internal* **begin**

lemmas [*simp*] = *vebt-assn-raw.simps*

lemma *TBOUND-VEBT-case*[*TBOUND*]: **assumes** ⋀ *a b*. *ti* = *Leafi a b* ⟹ *TBOUND* (*f a b*) (*bnd a b*)
⋀ *info deg treeArray summary* . *ti* = *Nodei info deg treeArray summary* ⟹
TBOUND (*f′ info deg treeArray summary*) (*bnd′ info deg treeArray summary*)

shows *TBOUND* (*case ti of Leafi a b* ⇒ *f a b* |
Nodei info deg treeArray summary ⇒ *f′ info deg treeArray summary*)
(*case ti of Leafi a b* ⇒ *bnd a b* |
Nodei info deg treeArray summary ⇒ *bnd′ info deg treeArray summary*)
using *assms*
apply(*cases ti*)
apply *auto*
done

Some Lemmas

lemma *length-corresp*:(∃$_A$ *tree-is*. *tree-array* ↦$_a$ *tree-is*) = *true* ⟹ *return* (*length tree-is*) = *Array-Time.len tree-array*
proof−
assume (∃$_A$ *tree-is*. *tree-array* ↦$_a$ *tree-is*) = *true*
then obtain *tree-is* **where** *tree-array* ↦$_a$ *tree-is* = *true*
by (*metis mod-h-bot-iff*(*2*) *mod-h-bot-iff*(*4*) *mod-h-bot-iff*(*8*))
then show *?thesis*
by (*metis assn-basic-inequalities*(*5*) *merge-true-star snga-same-false*)
qed

lemma *heaphelp*:**assumes** *h* ⊨
xa ↦$_a$ *tree-is* ∗ *list-assn vebt-assn-raw treeList tree-is* ∗

vebt-assn-raw summary xb $*\uparrow$*(None = None* $\wedge$ *n = n)**
$\uparrow$ *(xc = Nodei None n xa xb)*
shows *h* $\models$ *vebt-assn-raw (Node None n treeList summary) xc*
proof−
have *h* $\models$ *vebt-assn-raw (Node None n treeList summary) (Nodei None n xa xb)*
using *vebt-assn-raw.simps(2)[of None n treeList summary None n xa xb]* **apply** *simp*
by *(metis assms mod-pure-star-dist star-aci(2))*
then show *?thesis*
using *assms* **by** *auto*
qed

lemma *assnle*: *list-assn vebt-assn-raw treeList tree-is* $*$ *(x13* $\mapsto_a$ *tree-is* $*$ *vebt-assn-raw summary x14)* $\Longrightarrow_A$
vebt-assn-raw summary x14 $*$ *x13* $\mapsto_a$ *tree-is* $*$ *list-assn vebt-assn-raw treeList tree-is*
using *star-aci(2)* **by** *auto*

lemma *ext*: *y < length treeList* $\Longrightarrow$*x13* $\mapsto_a$ *tree-is* $*$ *(vebt-assn-raw summary x14* $*$
(vebt-assn-raw (treeList ! y) (tree-is ! y) $*$ *listI-assn ({0..<length treeList} − {y}) vebt-assn-raw treeList tree-is))*
$\Longrightarrow_A$ *(x13* $\mapsto_a$ *tree-is* $*$ *vebt-assn-raw summary x14* $*$
(listI-assn ({0..<length treeList} − {y}) vebt-assn-raw treeList tree-is) $*$*vebt-assn-raw (treeList ! y) (tree-is ! y)*
by *(metis assn-aci(9) ent-refl star-aci(2))*

lemma *txe:y < length treeList* $\Longrightarrow$ *vebt-assn-raw (treeList ! y) (tree-is ! y)* $*$ *x13* $\mapsto_a$ *tree-is* $*$ *vebt-assn-raw summary x14* $*$
listI-assn ({0..<length treeList} − {y}) vebt-assn-raw treeList tree-is $\Longrightarrow_A$
vebt-assn-raw summary x14 $*$ *x13* $\mapsto_a$ *tree-is* $*$ *list-assn vebt-assn-raw treeList tree-is*
by *(smt (verit, ccfv-SIG) assn-aci(9) assn-times-comm assnle atLeastLessThan-iff less-nat-zero-code listI-assn-extract list-assn-conv-idx not-less)*

lemma *recomp*: *i < length treeList* $\Longrightarrow$ *vebt-assn-raw (treeList ! i) (tree-is ! i)* $*$
listI-assn ({0..<length treeList} − {i}) vebt-assn-raw treeList tree-is $*$
x13 $\mapsto_a$ *tree-is* $*$
vebt-assn-raw summary x14 $\Longrightarrow_A$
vebt-assn-raw summary x14 $*$ *x13* $\mapsto_a$ *tree-is* $*$ *list-assn vebt-assn-raw treeList tree-is*
by *(smt (verit, ccfv-SIG) ab-semigroup-mult-class.mult.commute ab-semigroup-mult-class.mult.left-commute atLeastLessThan-iff ent-refl listI-assn-extract list-assn-conv-idx zero-le)*

lemma *repack*: *i < length treeList* $\Longrightarrow$
vebt-assn-raw (treeList ! i) (tree-is ! i) $*$
Rest $*$
(x13 $\mapsto_a$ *tree-is* $*$ *vebt-assn-raw summary x14* $*$
listI-assn ({0..<length treeList} − {i}) vebt-assn-raw treeList tree-is)
$\Longrightarrow_A$ *Rest* $*$ *vebt-assn-raw summary x14* $*$ *x13* $\mapsto_a$ *tree-is* $*$ *list-assn vebt-assn-raw treeList tree-is*
by *(smt (verit, del-insts) assn-times-assoc atLeastLessThan-iff entails-def leI less-nat-zero-code listI-assn-extract list-assn-conv-idx star-aci(2))*

lemma *big-assn-simp*: $h < length\ treeList \Longrightarrow$
$vebt\text{-}assn\text{-}raw\ (vebt\text{-}delete(treeList\ !\ h)\ l)\ x\ *$
$\uparrow (xaa = vebt\text{-}mint\ (vebt\text{-}delete(treeList\ !\ h)\ l)) *$
$(\ x13 \mapsto_a (tree\text{-}is\ [h := x]) *$
$vebt\text{-}assn\text{-}raw\ summary\ x14\ *$
$listI\text{-}assn\ (\{0..<length\ treeList\} - \{h\})\ vebt\text{-}assn\text{-}raw\ treeList\ tree\text{-}is) \Longrightarrow_A$
$x13 \mapsto_a tree\text{-}is[h:=x]\ *\ vebt\text{-}assn\text{-}raw\ summary\ x14\ *\ \uparrow (xaa = vebt\text{-}mint\ (vebt\text{-}delete(treeList\ !\ h)\ l)) *$
$list\text{-}assn\ vebt\text{-}assn\text{-}raw\ (treeList[h:= (vebt\text{-}delete(treeList\ !\ h)\ l)])\ (tree\text{-}is[h:= x])$
by (*smt* (*verit*) *Diff-iff ab-semigroup-mult-class.mult.left-commute assn-aci*(10) *atLeastLessThan-iff ent-refl*
insertCI insert-Diff-single insert-absorb leI length-list-update less-nat-zero-code listI-assn-subst list-assn-conv-idx)

lemma *tcd*: $i < length\ treeList \Longrightarrow length\ treeList = length\ treeList' \Longrightarrow$
$vebt\text{-}assn\text{-}raw\ y\ x\ *\ x13 \mapsto_a tree\text{-}is[i:= x]\ *\ vebt\text{-}assn\text{-}raw\ summary\ x14\ *\ listI\text{-}assn\ (\{0..<length\ treeList\} - \{i\})\ vebt\text{-}assn\text{-}raw\ (treeList[i :=y])\ (tree\text{-}is[i := x])$
$\Longrightarrow_A x13 \mapsto_a tree\text{-}is[i:= x]\ *\ vebt\text{-}assn\text{-}raw\ summary\ x14\ *\ list\text{-}assn\ vebt\text{-}assn\text{-}raw\ (treeList[i :=y])\ (tree\text{-}is[i := x])$
by (*smt* (*verit*, *ccfv-threshold*) *Diff-iff ab-semigroup-mult-class.mult.commute insert-iff length-list-update listI-assn-insert listI-assn-subst mult.assoc recomp subst-not-in*)

lemma *big-assn-simp'*: $h < length\ treeList \Longrightarrow xaa = vebt\text{-}delete\ (treeList\ !\ h) l \Longrightarrow$
$vebt\text{-}assn\text{-}raw\ xaa\ x * \uparrow (xb = vebt\text{-}mint\ xaa) *$
$(x13 \mapsto_a tree\text{-}is[h := x] * vebt\text{-}assn\text{-}raw\ summary\ x14\ *$
$listI\text{-}assn\ (\{0..<length\ treeList\} - \{h\})\ vebt\text{-}assn\text{-}raw\ treeList\ tree\text{-}is) \Longrightarrow_A$
$(x13 \mapsto_a tree\text{-}is[h:= x] * vebt\text{-}assn\text{-}raw\ summary\ x14\ *\ \uparrow (xb = vebt\text{-}mint\ xaa) *$
$list\text{-}assn\ vebt\text{-}assn\text{-}raw\ (treeList[h:= xaa])\ \ (tree\text{-}is[h:= x]))$
by (*metis big-assn-simp*)

lemma *refines-case-VEBTi*[*refines-rule*]: **assumes** $ti = ti' \bigwedge a\ b.\ refines\ (f1\ a\ b)\ (f1'\ a\ b) \bigwedge info\ deg\ treeArray\ summary\ .\ refines\ (f2\ info\ deg\ treeArray\ summary)\ (f2'\ info\ deg\ treeArray\ summary)$
shows *refines* (*case ti of Leafi a b* ⇒ *f1 a b* |
Nodei info deg treeArray summary ⇒ *f2 info deg treeArray summary*)
(*case ti' of Leafi a b*⇒ *f1' a b* |
Nodei info deg treeArray summary ⇒ *f2' info deg treeArray summary*)
using *assms* **apply** (*cases ti'*) **apply** *simp-all*
done

14.2 High and low Bitsequences Definition

definition *highi*::*nat* ⇒ *nat* ⇒ *nat Heap* **where**
highi x n == *return* (*x div* (2^*n*))

definition *lowi*::*nat* ⇒ *nat* ⇒ *nat Heap* **where**
lowi x n == *return* (*x mod* (2^*n*))

lemma *highi-h*: <*emp*> *highi x n* <λ *r*. ↑(*r* = *high x n*)>
by (*simp add*: *high-def highi-def return-cons-rule*)

lemma *highi-hT*: <*emp*> *highi x n* <λ *r*. ↑(*r* = *high x n*)>*T*[*1*]
by (*metis cons-post-rule entails-def highi-def highi-h httI order-refl time-return*)

lemma *lowi-h*: <*emp*> *lowi x n* <λ *r*. ↑(*r* = *low x n*)>
by (*simp add*: *low-def lowi-def return-cons-rule*)

lemma *lowi-hT*: <*emp*> *lowi x n* <λ *r*. ↑(*r* = *low x n*)>*T*[*1*]
by (*metis httI lowi-def lowi-h order-refl time-return*)

15 Imperative Implementation of *vebt − buildup*

fun *replicatei*::*nat* ⇒'*a Heap* ⇒ ('*a list*) *Heap* **where**
replicatei 0 x = *return* []|
replicatei (*Suc n*) *x* = *do*{ *y* <− *x*;
ys <− *replicatei n x*;
return (*y*#*ys*) }

lemma *time-replicate*: ⟦⋀*h*. *time x h* ≤ *c* ⟧ ⟹ *time* (*replicatei n x*) *h* ≤ (*1*+(*1*+*c*)∗*n*)
apply (*induction n arbitrary*: *h*)
apply (*simp add*: *time-simp algebra-simps*)
apply (*auto simp*: *time-simp fails-simp algebra-simps*)
by (*metis add-le-mono group-cancel.add2 nat-arith.suc1*)

lemma *TBOUND-replicate*: ⟦*TBOUND x c*⟧ ⟹ *TBOUND* (*replicatei n x*) (*1*+(*1*+*c*)∗*n*)
by (*meson TBOUND-def time-replicate*)

lemma *refines-replicate*[*refines-rule*]:
refines f f' ⟹ *refines* (*replicatei n f*) (*replicatei n f'*)
apply (*induction n*)
apply *simp-all*
apply *refines*
done

fun *vebt-buildupi'*::*nat* ⇒ *VEBTi Heap* **where**
vebt-buildupi' 0 = *return* (*Leafi False False*)|
vebt-buildupi' (*Suc 0*) = *return* (*Leafi False False*)|
vebt-buildupi' n = (*if even n then* (*let half* = *n div 2 in do*{
treeList <− *replicatei* (*2^half*) (*vebt-buildupi' half*);
assert' (*length treeList* = *2^half*);
trees <− *Array-Time.of-list treeList*;
summary <− (*vebt-buildupi' half*);
return (*Nodei None n trees summary*)})
else (*let half* = *n div 2 in do*{
treeList <− *replicatei* (*2^*(*Suc half*)) (*vebt-buildupi' half*);
assert' (*length treeList* = *2^Suc half*);
trees <− *Array-Time.of-list treeList*;

```
                    summary <- (vebt-buildupi' (Suc half));
                    return (Nodei None n trees summary)} ))

end

context begin
  interpretation VEBT-internal .

fun vebt-buildupi::nat ⇒ VEBTi Heap where
  vebt-buildupi 0 = return (Leafi False False)|
  vebt-buildupi (Suc 0) = return (Leafi False False)|
  vebt-buildupi n =  (if even n then (let half = n div 2 in do{
                    treeList <- replicatei  (2^half) (vebt-buildupi half);
                    trees <- Array-Time.of-list  treeList;
                    summary <- (vebt-buildupi half);
                    return (Nodei None n  trees summary)})
            else (let half = n div 2 in  do{
                    treeList <-    replicatei (2^(Suc half)) (vebt-buildupi half);
                    trees <- Array-Time.of-list  treeList;
                    summary <- (vebt-buildupi (Suc half));
                    return (Nodei None n trees summary)} ))

end

context VEBT-internal begin

lemma vebt-buildupi-refines: refines (vebt-buildupi n) (vebt-buildupi' n)
  apply (induction n rule: vebt-buildupi.induct)
   apply (subst vebt-buildupi.simps; subst vebt-buildupi'.simps; refines)+
  done

fun T-vebt-buildupi where
  T-vebt-buildupi 0 = Suc 0
| T-vebt-buildupi (Suc 0) = Suc 0
| T-vebt-buildupi (Suc (Suc n)) = (
    if even n then
      Suc (Suc (Suc (T-vebt-buildupi (Suc (n div 2)) +
                    (4 * 2 ^ (n div 2) + 2 * (T-vebt-buildupi (Suc (n div 2)) * 2 ^ (n div 2))))))
    else
        Suc (Suc (Suc (T-vebt-buildupi (Suc (Suc (n div 2))) +
                    (8 * 2 ^ (n div 2) + 4 * (T-vebt-buildupi (Suc (n div 2)) * 2 ^ (n div 2)))))))

lemma TBOUND-vebt-buildupi:
  defines foo ≡ T-vebt-buildupi
  shows TBOUND (vebt-buildupi' n) (foo n)
  supply [simp del] = vebt-buildupi'.simps
  supply [TBOUND] = TBOUND-replicate
  apply (induction n rule: vebt-buildupi.induct)
  apply (subst vebt-buildupi'.simps)
```

apply (*rule TBOUND-mono*)
apply (*TBOUND-step*)
apply(*rule asm-rl*[*of* - ≤ -])
apply *defer-le*
apply (*subst vebt-buildupi'.simps*)
apply (*rule TBOUND-mono*)
apply (*TBOUND-step*)
apply(*rule asm-rl*[*of* - ≤ -])
apply *defer-le*
apply (*subst vebt-buildupi'.simps*)
apply (*rule TBOUND-mono*)
apply *TBOUND-step+*
apply(*rule asm-rl*[*of* - ≤ -])
apply *defer-le*
apply (*all* ‹((*determ* ‹*thin-tac* ‹*TBOUND* - -››)+)*?* ›)
apply (*simp-all add*: *foo-def*)
done

lemma *T-vebt-buildupi*: *time* (*vebt-buildupi'* *n*) *h* ≤ *T-vebt-buildupi n*
using *TBOUND-vebt-buildupi*[*THEN TBOUNDD*] **.**

lemma *repli-cons-repl*: <*Q*> *x* <λ *r*. *Q*∗ *A y r* > ⟹ <*Q*> *replicatei n x* <λ *r*. *Q*∗*list-assn A* (*replicate n y*) *r* >
proof(*induction n arbitrary*: *Q*)
case (*Suc n*)
then show *?case*
apply (*sep-auto heap*: *Suc.IH*(*1*))
subgoal
using *frame-rule-left*[*of Q x* λ*r*. *Q* ∗ *A y r A y* -, *unfolded mult.left-assoc*]
by (*metis mult.commute*)
subgoal
by *sep-auto*
done
qed *sep-auto*

corollary *repli-emp*: <*emp*> *x* <λ *r*. *A y r* > ⟹ <*emp*> *replicatei n x* <λ *r*. *list-assn A* (*replicate n y*) *r* >
apply(*rule cons-post-rule*)
apply(*rule repli-cons-repl*[**where** *Q* = *emp*])
apply *sep-auto+*
done

lemma *builupi'corr*: <*emp*> *vebt-buildupi' n* <λ *r*. *vebt-assn-raw* (*vebt-buildup n*) *r*>
proof(*induction n rule*: *vebt-buildup.induct*)
case (*3 n*)
then show *?case*
proof(*cases even* (*Suc* (*Suc n*)))
case *True*

```
  then show ?thesis
    apply( simp add: vebt-buildupi'.simps(2))
    apply(rule bind-rule)
    apply(sep-auto heap: repli-cons-repl)
    apply(rule 3.IH(1))
    apply simp+
    apply sep-auto
    apply (extract-pre-pure dest: extract-pre-list-assn-lengthD; simp)
    apply (sep-auto heap: 3.IH(1))
    done
 next
  case False
  hence 11: <xa ↦a x * list-assn vebt-assn-raw (replicate (4 * 2 ^ (n div 2)) (vebt-buildup (Suc (n
div 2)))) x>
     vebt-buildupi' (Suc (Suc (Suc n) div 2)) <λ r. xa ↦a x * list-assn vebt-assn-raw (replicate (4 *
2 ^ (n div 2)) (vebt-buildup (Suc (n div 2)))) x *
       vebt-assn-raw ( vebt-buildup (Suc (Suc (Suc n) div 2))) r> for xa x
  proof −
    show ?thesis
      by (metis (no-types) 3.IH(4) False frame-rule-left mult.right-neutral)
  qed
   hence vebt-buildupi' (Suc (Suc n)) = do{ treeList <− replicatei (2^(Suc ((Suc (Suc n)) div
2))) (vebt-buildupi' ((Suc (Suc n)) div 2));
                        assert' (length treeList = (2^(Suc ((Suc (Suc n)) div 2))));
                        trees <− Array-Time.of-list treeList;
                         summary <− (vebt-buildupi' (Suc ((Suc (Suc n)) div 2)));
                        return (Nodei None (Suc (Suc n)) trees summary)}
    using vebt-buildupi'.simps(3)[of n] Let-def False
    by auto
 moreover have <emp> do{treeList <− replicatei (2^(Suc ((Suc (Suc n)) div 2))) (vebt-buildupi'
((Suc (Suc n)) div 2));
                        assert' (length treeList = (2^(Suc ((Suc (Suc n)) div 2))));
                        trees <− Array-Time.of-list treeList;
                         summary <− (vebt-buildupi' (Suc ((Suc (Suc n)) div 2)));
                       return (Nodei None (Suc (Suc n)) trees summary)} <vebt-assn-raw (vebt-buildup
(Suc (Suc n)))>
    apply(rule bind-rule)
    apply(sep-auto heap: repli-cons-repl)
    apply(rule 3.IH(3))
    using False apply simp
    using False apply simp
    apply(rule assert'-bind-rule)
    apply (extract-pre-pure dest: extract-pre-list-assn-lengthD; simp)
    apply(rule bind-rule)
    apply sep-auto
    apply(rule bind-rule)
    apply (rule 11)
    apply vcg
    proof−
```

```
      fix x xa xb xc
       show  xa ↦a x ∗ list-assn vebt-assn-raw (replicate (4 ∗ 2 ^ (n div 2)) (vebt-buildup (Suc (n
div 2)))) x ∗
       vebt-assn-raw (vebt-buildup (Suc (Suc (Suc n) div 2))) xb ∗ ↑ (xc = Nodei None (Suc (Suc n))
xa xb) ⟹A vebt-assn-raw (vebt-buildup (Suc (Suc n))) xc
       apply(rule entailsI)
       proof−
        fix h
        assume a:  h ⊨xa ↦a x ∗ list-assn vebt-assn-raw (replicate (4 ∗ 2 ^ (n div 2)) (vebt-buildup
(Suc (n div 2)))) x ∗
          vebt-assn-raw (vebt-buildup (Suc (Suc (Suc n) div 2))) xb ∗ ↑ (xc = Nodei None (Suc (Suc
n)) xa xb)
         then show    h ⊨ vebt-assn-raw (vebt-buildup (Suc (Suc n))) xc
          using heaphelp[of xa x replicate - - - xb Suc (Suc n) xc h, simplified]
          using False by auto
     qed
   qed
   then show ?thesis using calculation
    by presburger
 qed
qed  sep-auto+

lemma htt-vebt-buildupi': < emp> (vebt-buildupi' n) <λ r. vebt-assn-raw (vebt-buildup n) r> T
[T-vebt-buildupi n]
 apply (rule httI-TBOUND)
 apply (rule builupi'corr)
 apply (rule TBOUND-vebt-buildupi)
 done

lemma builupicorr: <emp> vebt-buildupi n <λ r. vebt-assn-raw (vebt-buildup n) r>
 using vebt-buildupi-refines builupi'corr hoare-triple-refines by blast

lemma htt-vebt-buildupi: <emp> (vebt-buildupi n) <λ r. vebt-assn-raw (vebt-buildup n) r> T [T-vebt-buildupi
n]
 apply (rule htt-refine)
 apply (rule htt-vebt-buildupi')
 apply (rule vebt-buildupi-refines)
 done
```

Closed bound for $T - vebt - buildupi$

Amortization

```
lemma T-vebt-buildupi-gq-0: T-vebt-buildupi n > 0
 apply(induction n rule : T-vebt-buildupi.induct)
  apply auto
 done

fun T-vebt-buildupi'::nat ⇒ int where
  T-vebt-buildupi' 0 = 1
| T-vebt-buildupi' (Suc 0) = 1
```

```
| T-vebt-buildupi' (Suc (Suc n)) = (
    if even n then
      3+(T-vebt-buildupi' (Suc (n div 2)) +
           (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2)) * 2 ^(n div 2))))
    else
         3+ (T-vebt-buildupi' (Suc (Suc (n div 2))) +
          (8 * 2 ^(n div 2) + 4 * (T-vebt-buildupi' (Suc (n div 2)) * 2 ^(n div 2)))))

lemma Tbuildupi-buildupi': T-vebt-buildupi n = T-vebt-buildupi' n
  by(induction n rule: T-vebt-buildupi.induct) auto

fun Tb::nat ⇒ int where
  Tb 0 = 3
| Tb (Suc 0) =3
| Tb (Suc (Suc n)) = (
    if even n then
         5+ Tb (Suc (n div 2))  +  (Tb (Suc (n div 2))) * 2 ^(Suc (n div 2))
    else
         5 + Tb (Suc (Suc (n div 2)))  + (Tb (Suc (n div 2))) * 2 ^(Suc (Suc (n div 2))))

lemma Tb-T-vebt-buildupi': T-vebt-buildupi' n ≤ Tb n − 2
proof(induction n rule: T-vebt-buildupi.induct)
  case 1
  then show ?case
    apply(subst Tb.simps)
    apply(subst T-vebt-buildupi'.simps)
    apply simp
    done
next
  case 2
  then show ?case
    apply(subst Tb.simps)
    apply(subst T-vebt-buildupi'.simps)
    apply simp
    done
next
  case (3 n)
  then show ?case
  proof(cases even (Suc (Suc n)))
    case True
    then show ?thesis
      apply(subst Tb.simps)
      apply(subst T-vebt-buildupi'.simps)
      using True apply simp
      thm 3
    proof−
      have 0:T-vebt-buildupi' (Suc (n div 2)) +
       (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2)) * 2 ^(n div 2)))
       ≤ Tb (Suc (n div 2)) − 2  + 2^(Suc (n div 2))*2 +
```

```
          2^(Suc (n div 2)) * (T-vebt-buildupi' (Suc (n div 2)))
    using 3.IH(1) True algebra-simps by simp
  moreover have 1:2^(Suc (n div 2))*2 +
          2^(Suc (n div 2)) * (T-vebt-buildupi' (Suc (n div 2))) =
          2^(Suc (n div 2)) * (T-vebt-buildupi' (Suc (n div 2))) +  2) by algebra
  ultimately have  2:T-vebt-buildupi' (Suc (n div 2)) +
        (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2)))
        ≤ Tb (Suc (n div 2)) − 2  +
          2^(Suc (n div 2)) * (T-vebt-buildupi' (Suc (n div 2)) + 2)by linarith
  hence  3: (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2))
      ≤ 2* 2^(Suc (n div 2))+ 2^(Suc (n div 2))  * ((Tb (Suc (n div 2)) − 2))
    using 3.IH(1) True by simp
  hence  4: (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2))
        ≤ 2^(Suc (n div 2))  * ((Tb (Suc (n div 2)) − 2) +  2)
    using algebra-simps by (smt (verit, del-insts) 1)
  hence  4:   T-vebt-buildupi' (Suc (n div 2)) +
      (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2))
       ≤ Tb (Suc (n div 2)) − (2::int) + 2^(Suc (n div 2)) * (Tb (Suc (n div 2)))
    using 3.IH(1) True by simp
  have 5: (x::int) ≤ (y::int) − (z::int) + a ⟹ z ≥ 0 ⟹ x ≤ y + a for x y z a by simp
  have  T-vebt-buildupi' (Suc (n div 2)) +
        (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2)))
        ≤ Tb (Suc (n div 2)) + 2^(Suc (n div 2)) * (Tb (Suc (n div 2))) using
    5[of  T-vebt-buildupi' (Suc (n div 2)) +
        (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2))
     Tb (Suc (n div 2))   2  Tb (Suc (n div 2)) * (2 * 2 ^(n div 2))]  4 by simp
  then show T-vebt-buildupi' (Suc (n div 2)) +
      (4 * 2 ^(n div 2) + 2 * (T-vebt-buildupi' (Suc (n div 2))) * 2 ^(n div 2)))
       ≤ Tb (Suc (n div 2)) + Tb (Suc (n div 2)) * (2 * 2 ^(n div 2))
    using  power-Suc [of  2 (n div 2)] mult.commute by metis
 qed
next
  case False
  have 3 +
      (T-vebt-buildupi' (Suc (Suc (n div 2))) +
       (8 * 2 ^(n div 2) + 4 * (T-vebt-buildupi' (Suc (n div 2)) * 2 ^(n div 2))))
         ≤  5 + Tb (Suc (Suc (n div 2))) + Tb (Suc (n div 2)) * 2 ^Suc (Suc (n div 2)) − 2
  proof−
    have 0:3 +
      (T-vebt-buildupi' (Suc (Suc (n div 2))) +
       (8 * 2 ^(n div 2) + 4 * (T-vebt-buildupi' (Suc (n div 2)) * 2 ^(n div 2))))
      ≤  1 + Tb (Suc (Suc (n div 2))) + (8 * 2 ^(n div 2) + 4 * (T-vebt-buildupi' (Suc (n div
2)) * 2 ^(n div 2)))
    using 3.IH(3) False by simp
  moreover have   4 * (T-vebt-buildupi' (Suc (n div 2))*2 ^(n div 2)) ≤
       4 * ((Tb (Suc (n div 2)) − 2) * 2 ^(n div 2))
    using 3.IH(4) False algebra-simps by simp
  moreover have 8 * 2 ^(n div 2) + 4 * ((Tb (Suc (n div 2)) − 2) * 2 ^(n div 2)) =
      4* (2 *  2 ^(n div 2) + ((Tb (Suc (n div 2)) − 2) * 2 ^(n div 2))) by simp
```

```
    moreover have 4* (2 *  2 ^(n div 2) + ((Tb (Suc (n div 2)) - 2) * 2 ^(n div 2))) =
        4* ( ((Tb (Suc (n div 2)) - 2) +  2) * 2 ^(n div 2)) by algebra
    moreover hence 4* (2 *  2 ^(n div 2) + ((Tb (Suc (n div 2)) - 2) * 2 ^(n div 2))) =
        4* ((Tb (Suc (n div 2)) ) * 2 ^(n div 2)) by simp
    ultimately have  8 * 2 ^(n div 2) + 4 * (T-vebt-buildupi' (Suc (n div 2))*2 ^(n div 2)) ≤
          4 * (((Tb (Suc (n div 2))-2) + 2 ) * 2 ^(n div 2)) by presburger
    then show ?thesis using 0 by force
  qed
  then show ?thesis
    apply(subst Tb.simps)
    apply(subst T-vebt-buildupi'.simps)
    using False by simp
qed qed

fun Tb'::nat ⇒ nat where
  Tb' 0 = 3
| Tb' (Suc 0) =3
| Tb' (Suc (Suc n)) = (
    if even n then
        5+ Tb' (Suc (n div 2))  +  (Tb' (Suc (n div 2))) * 2 ^(Suc (n div 2))
    else
        5 + Tb' (Suc (Suc (n div 2)))  + (Tb' (Suc (n div 2))) * 2 ^(Suc (Suc (n div 2))))

lemma Tb-Tb': Tb t = Tb' t
  by(induction t rule: Tb.induct) auto

lemma Tb-T-vebt-buildupi: T-vebt-buildupi n ≤ Tb n - 2
  using Tb-T-vebt-buildupi'  Tbuildupi-buildupi' by simp

lemma Tb-T-vebt-buildupi'': T-vebt-buildupi n ≤ Tb' n - 2
  using Tb-T-vebt-buildupi[of n] Tb-Tb' by simp

lemma Tb'-cnt: Tb' n ≤ 5 * cnt' (vebt-buildup n)
proof(induction n rule: vebt-buildup.induct)
  case (3 n)
  then show ?case
  proof(cases even n)
    case True
    have 0: 5 + Tb' (Suc (n div 2)) + Tb' (Suc (n div 2)) * 2 ^ Suc (n div 2)
    ≤ 5 * cnt' ( let half = Suc (Suc n) div 2
                    in Node None (Suc (Suc n)) (replicate (2 ^ half) (vebt-buildup half))
                        (vebt-buildup half))
      unfolding Let-def
      apply(subst cnt'.simps)
    proof-
      have 0:5 * (1 + cnt' (vebt-buildup (Suc (Suc n) div 2)) +
          foldr (+)
           (map cnt' (replicate (2 ^(Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2)))) 0) =
            5 * (1 + cnt' (vebt-buildup (Suc (Suc n) div 2))  + (2 ^(Suc (Suc n) div 2)) * cnt'
```

```
(vebt-buildup (Suc (Suc n) div 2)))
      using map-replicate[of cnt' (2 ^ (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2))]
        foldr-same-int[of replicate (2 ^ (Suc (Suc n) div 2)) (cnt' (vebt-buildup (Suc (Suc n) div 2)))
          (cnt' (vebt-buildup (Suc (Suc n) div 2)))] length-replicate by simp
    have 1: Tb' (Suc (n div 2)) * 2 ^ Suc (n div 2)
        ≤ 5 * (2 ^ (Suc (Suc n) div 2) * cnt' (vebt-buildup (Suc (Suc n) div 2)))
      using True 3.IH(1)[of Suc (Suc n) div 2] by simp
    have 2: Tb' (Suc (n div 2)) ≤ 5 * cnt' (vebt-buildup (Suc (Suc n) div 2))
      using True 3.IH(1)[of Suc (Suc n) div 2] by simp
    have 5 + Tb' (Suc (n div 2)) + Tb' (Suc (n div 2)) * 2 ^ Suc (n div 2) ≤
      5 * (1 + cnt' (vebt-buildup (Suc (Suc n) div 2)) +
        2 ^ (Suc (Suc n) div 2) * cnt' (vebt-buildup (Suc (Suc n) div 2)))
      by (metis 1 2 add-le-mono distrib-left-numeral mult-Suc-right nat-add-left-cancel-le
        plus-1-eq-Suc)
    also have ... = 5 * (1 + cnt' (vebt-buildup (Suc (Suc n) div 2)) +
      foldr (+) (map cnt' (replicate (2 ^ (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2)))) 0)
      by (metis 0)
    finally show 5 + Tb' (Suc (n div 2)) + Tb' (Suc (n div 2)) * 2 ^ Suc (n div 2)
        ≤ 5 * (1 + cnt' (vebt-buildup (Suc (Suc n) div 2)) +
        foldr (+)
         (map cnt' (replicate (2 ^ (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2)))) 0)
      .
  qed
  show ?thesis
    apply (subst vebt-buildup.simps)
    apply(subst Tb'.simps)
    using 0 True apply simp
    done
next
  case False
  have 0: 5 + Tb' (Suc (Suc (n div 2))) + Tb' (Suc (n div 2)) * 2 ^ Suc (Suc (n div 2))
      ≤ 5 * cnt' ( let half = Suc (Suc n) div 2
                in Node None (Suc (Suc n)) (replicate (2 ^ Suc half) (vebt-buildup half))
                   (vebt-buildup (Suc half)))
    unfolding Let-def
    apply(subst cnt'.simps)
  proof−
    have 0:5 * (1 + cnt' (vebt-buildup (Suc (Suc (Suc n) div 2))) +
        foldr (+) (map cnt' (replicate (2 ^ Suc (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div
2))))0)
          = 5 * (1 + cnt' (vebt-buildup (Suc (Suc (Suc n) div 2))) + (2 ^ Suc (Suc (Suc n) div
2)) * cnt' (vebt-buildup (Suc (Suc n) div 2)))
      using map-replicate[of cnt' (2 ^ Suc (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2))]
        foldr-same-int[of replicate (2 ^ Suc (Suc (Suc n) div 2)) (cnt' (vebt-buildup (Suc (Suc n) div
2)))
          (cnt' (vebt-buildup (Suc (Suc n) div 2)))] length-replicate by simp
    have 1: Tb' (Suc (n div 2)) * 2 ^ ((Suc n) div 2)
        ≤ 5 * (2 ^ (Suc (Suc n) div 2) * cnt' (vebt-buildup (Suc (Suc n) div 2)))
      using False 3.IH(3)[of (Suc (Suc n) div 2)] by simp
```

```
    have 2: Tb' (Suc (Suc (n div 2))) ≤ 5 * cnt' (vebt-buildup (Suc (Suc (Suc n) div 2)))
      using False 3.IH(4)[of (Suc n) div 2] by simp
    have 5 + Tb' (Suc (Suc (n div 2))) + Tb' (Suc (n div 2)) * 2 ^ Suc (Suc (n div 2)) ≤
      5 * (1 + cnt' (vebt-buildup (Suc (Suc (Suc n) div 2)))+
        (2 ^ Suc (Suc (Suc n) div 2)) * cnt' (vebt-buildup (Suc (Suc n) div 2)))
      using 1 2
      by (smt (verit, ccfv-threshold) 3.IH(3) False Suc-1 Suc-n-div-2-gt-zero
          ab-semigroup-mult-class.mult.commute ab-semigroup-mult-class.mult.left-commute
          add-mono-thms-linordered-semiring(1) diff-Suc-1 diff-Suc-Suc distrib-left-numeral
          div-greater-zero-iff even-Suc le-div-geq mult-le-cancel1 mult-numeral-1-right numerals(1)
          zero-less-Suc)
    also have ... = 5 * (1 + cnt' (vebt-buildup (Suc (Suc (Suc n) div 2))) +
      foldr (+) (map cnt' (replicate (2 ^ Suc (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div
2)))) 0)
      by (metis 0)
    finally show 5 + Tb' (Suc (Suc (n div 2))) + Tb' (Suc (n div 2)) * 2 ^ Suc (Suc (n div 2))
        ≤ 5 * (1 + cnt' (vebt-buildup (Suc (Suc (Suc n) div 2))) +
         foldr (+)
        (map cnt' (replicate (2 ^ Suc (Suc (Suc n) div 2)) (vebt-buildup (Suc (Suc n) div 2)))) 0)
      .
  qed
  show ?thesis
    apply (subst vebt-buildup.simps)
    apply(subst Tb'.simps)
    using 0 False apply simp
    done
 qed
qed (subst vebt-buildup.simps cnt'.simps  Tb'.simps , simp )+

lemma T-vebt-buildupi-cnt': T-vebt-buildupi n  ≤ 5 * cnt (vebt-buildup n)
 apply(rule ord-le-eq-trans[where b = real (5 * cnt' (vebt-buildup n))])
  defer
 apply(simp add: cnt-cnt-eq)
 apply(rule of-nat-mono)
 apply(rule order.trans[])
 apply(rule Tb-T-vebt-buildupi'')
 apply(rule order.trans[where b = Tb' n])
 apply simp
 apply(rule Tb'-cnt)
 done

lemma T-vebt-buildupi-univ:
 assumes u =2^n
 shows   T-vebt-buildupi n  ≤10 * u
proof−
 have cnt (vebt-buildup n) ≤ 2 * u
   using count-buildup[of n] assms by simp
 hence real (T-vebt-buildupi n)  ≤ 5 * 2 * u
   using T-vebt-buildupi-cnt'[of n] by simp
```

then show *?thesis* **by** *simp*
qed

lemma *htt-vebt-buildupi'-univ*:
assumes $u = 2\hat{}n$
shows
$< emp >$ (*vebt-buildupi'* n) $<\lambda\ r.$ *vebt-assn-raw* (*vebt-buildup* n) $r>$ $T\ [10 * u]$
apply (*rule httI-TBOUND*)
apply (*rule builupi'corr*)
apply (*rule TBOUND-mono*[**where** $t = T$-*vebt-buildupi* n])
apply (*rule TBOUND-vebt-buildupi*)
using *T-vebt-buildupi-univ*[*of* u n] *assms* **apply** *simp*
done

We obtain the main theorem for *buildupi*

lemma *htt-vebt-buildupi-univ*:
assumes $u = 2\hat{}n$
shows
$< emp >$ (*vebt-buildupi* n) $<\lambda\ r.$ *vebt-assn-raw* (*vebt-buildup* n) $r>$ $T\ [10 * u]$
using *vebt-buildupi-refines*
by (*metis VEBT-internal.htt-vebt-buildupi'-univ assms htt-refine*)

lemma *vebt-buildupi-rule*: $<\uparrow\ (n > 0)>$ *vebt-buildupi* n $<\lambda\ r.$ *vebt-assn-raw* (*vebt-buildup* n) $r >$ $T[10 * 2\hat{}n]$
proof−
have *vebt-buildupi'-rule*: $<\uparrow\ (n > 0)>$ *vebt-buildupi'* n $<\lambda\ r.$ *vebt-assn-raw* (*vebt-buildup* n) $r >$
using *builupicorr*[*of* n]
apply *simp*
using *VEBT-internal.builupi'corr* **by** *blast*
have *vebt-buildupi'-rule-univ*: $<\uparrow\ (n > 0)>$ *vebt-buildupi'* n $<\lambda\ r.$ *vebt-assn-raw* (*vebt-buildup* n) $r > T[10 * 2\hat{}n]$
apply (*rule httI-TBOUND*)
apply(*rule vebt-buildupi'-rule*)
apply(*rule TBOUND-refines*[**where** c = *vebt-buildupi'* n])
apply(*rule TBOUND-mono*[**where** t=*T-vebt-buildupi* n])
apply(*rule TBOUND-vebt-buildupi*)
using *T-vebt-buildupi-univ*[*of* $2\hat{}n$ n]
apply *simp*
apply(*rule refines-refl*)
done
show *?thesis*
using *vebt-buildupi-refines htt-refine vebt-buildupi'-rule-univ* **by** *blast*
qed

lemma *TBOUND-buildupi*: **assumes** $n>0$ **shows** *TBOUND* (*vebt-buildupi* n) $(10 * 2\ \hat{}\ n)$
using *vebt-buildupi-rule*[*of* n] **unfolding** *htt-def TBOUND-def*
apply *auto*
subgoal for h

using *time-return*[*of Leafi False False h*] **by** *simp*
subgoal for *h*
using *time-return*[*of Leafi False False h*] **by** *simp*
done

16 Minimum and Maximum Determination

end

context begin
interpretation *VEBT-internal* **.**

fun *vebt-minti*::*VEBTi* ⇒ *nat option Heap* **where**
vebt-minti (*Leafi a b*) = (*if a then return* (*Some 0*) *else if b then return* (*Some 1*) *else return None*)|
vebt-minti (*Nodei None - - -*) = *return None*|
vebt-minti (*Nodei* (*Some* (*mi*,*ma*)) *- - -*) = *return* (*Some mi*)

fun *vebt-maxti*::*VEBTi* ⇒ *nat option Heap* **where**
vebt-maxti (*Leafi a b*) = (*if b then return* (*Some 1*) *else if a then return* (*Some 0*) *else return None*)|
vebt-maxti (*Nodei None - - -*) = *return None*|
vebt-maxti (*Nodei* (*Some* (*mi*,*ma*)) *- - -*) = *return* (*Some ma*)

end

context *VEBT-internal* **begin**

lemma *vebt-minti-h*:<*vebt-assn-raw t ti*> *vebt-minti ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *vebt-mint t*)>

by (*cases t rule*: *vebt-mint.cases*; *cases ti rule*: *vebt-minti.cases*) (*sep-auto*+)

lemma *vebt-maxti-h*:<*vebt-assn-raw t ti*> *vebt-maxti ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *vebt-maxt t*)>
by (*cases t rule*: *vebt-mint.cases*; *cases ti rule*: *vebt-minti.cases*) (*sep-auto*+)

lemma *TBOUND-vebt-maxti*[*TBOUND*]: *TBOUND* (*vebt-maxti t*) *1*
apply (*induction t rule*: *vebt-maxti.induct*)
apply (*subst vebt-maxti.simps*| *TBOUND-step*)+
done

lemma *TBOUND-vebt-minti*[*TBOUND*]: *TBOUND* (*vebt-minti t*) *1*
apply (*induction t rule*: *vebt-minti.induct*)
apply (*subst vebt-minti.simps*| *TBOUND-step*)+
done

lemma *vebt-minti-hT*:<*vebt-assn-raw t ti*> *vebt-minti ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *vebt-mint t*)>*T*[*1*]
using *TBOUND-vebt-minti httI-TBOUND vebt-minti-h* **by** *blast*

lemma *vebt-maxti-hT*:<*vebt-assn-raw t ti*> *vebt-maxti ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *vebt-maxt*

```
t)>T[1]
  using TBOUND-vebt-maxti httI-TBOUND vebt-maxti-h by blast

lemma vebt-maxtilist:i < length ts ⟹
 <list-assn vebt-assn-raw ts tsi> vebt-maxti (tsi ! i)
     < λ r. ↑(r = vebt-maxt (ts ! i)) ∗list-assn vebt-assn-raw ts tsi>
  apply(unwrap-idx i)
  apply (sep-auto heap: vebt-maxti-h)
  apply(wrap-idx R: listI-assn-reinsert-upd)
  apply sep-auto
  done

lemma vebt-mintilist:i < length ts ⟹
 <list-assn vebt-assn-raw ts tsi> vebt-minti (tsi ! i)
     < λ r. ↑(r = vebt-mint (ts ! i)) ∗list-assn vebt-assn-raw ts tsi>
  apply(unwrap-idx i)
  apply (sep-auto heap: vebt-minti-h)
  apply(wrap-idx R: listI-assn-reinsert-upd)
  apply sep-auto
  done
```

17 Membership Test on imperative van Emde Boas Trees

```
end

context begin
interpretation VEBT-internal .

partial-function (heap-time) vebt-memberi::VEBTi ⇒ nat ⇒ bool Heap where
  vebt-memberi t x =
 (case t of
       (Leafi a b ) ⇒ return (if x = 0 then a else if x=1 then b else False) |
        (Nodei info deg treeList summary ) ⇒ (
              case info of None ⇒ return False |
               (Some (mi, ma)) ⇒ ( if deg ≤ 1 then return False else (
                                   if x = mi then return True else
                                   if x = ma then return True else
                                   if x < mi then return False else
                                   if x > ma then return False else
                                   (do {
                                      h ← highi x (deg div 2);
                                      l ← lowi x (deg div 2);
                                      len ← Array-Time.len treeList;
                                      if h < len then do {
                                      th ← Array-Time.nth treeList h;
                                      vebt-memberi th l
                                      } else return False
                                   }))))))
```

end

context *VEBT-internal* **begin**

partial-function (*heap-time*) *vebt-memberi'*::*VEBT* ⇒*VEBTi* ⇒ *nat* ⇒ *bool Heap* **where**
vebt-memberi' t ti x =
(*case ti of*
(*Leafi a b*) ⇒ *return* (*if x* = *0 then a else if x*=*1 then b else False*) |
(*Nodei info deg treeArray summary*) ⇒ (*do* {*assert'* (*is-Node t*);
case info of None ⇒ *return False* |
(*Some* (*mi*, *ma*)) ⇒ (*if deg* ≤ *1 then return False else* (
if x = *mi then return True else*
if x = *ma then return True else*
if x < *mi then return False else*
if x > *ma then return False else*
(*do* {
let (*info'*,*deg'*,*treeList*,*summary'*) =
(*case t of* (*Node info' deg' treeList summary'*) ⇒
(*info'*, *deg'*, *treeList*, *summary'*));
assert'(*info*= *info'* ∧ *deg* = *deg'*);
h ← *highi x* (*deg div 2*);
l ← *lowi x* (*deg div 2*);
assert'(*l* = *low x* (*deg div 2*) ∧ *h* = *high x* (*deg div 2*));
len ← *Array-Time.len treeArray*;
assert'(*len* = *length treeList*);
if h < *len then do* {
assert'(*h* = *high x* (*deg div 2*) ∧ *h* < *length treeList*);

th ← *Array-Time.nth treeArray h*;
vebt-memberi' (*treeList* ! *h*) *th l* }
else return False
}))))}))

lemma *highsimp*: *return* (*high x n*) = *highi x n*
by (*simp add*: *high-def highi-def*)

lemma *lowsimp*: *return* (*low x n*) = *lowi x n*
by (*simp add*: *low-def lowi-def*)

lemma *TBOUND-highi*[*TBOUND*]: *TBOUND* (*highi x n*) *1*
unfolding *highi-def*
apply *TBOUND-step*
done

lemma *TBOUND-lowi*[*TBOUND*]: *TBOUND* (*lowi x n*) *1*
unfolding *lowi-def*
apply *TBOUND-step*
done

Correctness of $vebt - memberi$

lemma *vebt-memberi'-rf-abstr*: <*vebt-assn-raw t ti*> *vebt-memberi' t ti x* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *vebt-member t x*)>
proof(*induction t x arbitrary*: *ti rule*: *vebt-member.induct*)
case (*1 a b x*)
then show *?case* **apply** (*subst vebt-memberi'.simps*) **by**(*cases ti*; *sep-auto*)
next
case (*2 uu uv uw x*)
then show *?case* **apply** (*subst vebt-memberi'.simps*) **by**(*cases ti*; *sep-auto*)
next
case (*3 v uy uz x*)
then show *?case* **apply** (*subst vebt-memberi'.simps*) **by**(*cases ti*; *sep-auto*)
next
case (*4 v vb vc x*)
then show *?case* **apply** (*subst vebt-memberi'.simps*) **by**(*cases ti*; *sep-auto*)
next
case (*5 mi ma va treeList summary x*)
note *IH*[*sep-heap-rules*] = *5.IH*
show *?case*
apply (*subst vebt-memberi'.simps*) **unfolding** *highi-def lowi-def*
apply (*cases ti*;*sep-auto*)
apply(*simp add*: *low-def*)
apply(*simp add*: *high-def*)
apply *sep-auto*
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
apply(*simp add*: *high-def*)
apply *sep-auto*
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
subgoal
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*=(*x div* (*2* ∗ *2* ^ (*va div 2*)))])
apply *simp*
apply *simp*
apply (*sep-auto simp*: *high-def low-def*)
apply (*rule listI-assn-reinsert*)
apply *frame-inference*
apply *simp*
apply *simp*
apply (*rewrite* **in** ⌑ ⟹$_A$ - *list-assn-conv-idx*[*symmetric*])
apply *sep-auto*
done
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
apply (*sep-auto simp*: *high-def*)
done
qed

lemma *TBOUND-vebt-memberi*:
defines *foo-def*: ⋀ *t x*. *foo t x* ≡ *4* ∗ (*1*+*height t*)
shows *TBOUND* (*vebt-memberi' t ti x*) (*foo t x*)

```
  apply (induction arbitrary: t ti x rule: vebt-memberi'.fixp-induct)
   apply (rule TBOUND-fi'-adm)
   apply (rule TBOUND-empty)
   subgoal for f t ti x
    apply(rule TBOUND-mono)
    apply ( TBOUND-step)+
    unfolding foo-def
    apply (auto split: VEBTi.splits option.splits VEBT.splits)
    apply (meson List.finite-set Max-ge finite-imageI imageI le-max-iff-disj nth-mem)
   done
  done

lemma vebt-memberi-refines: refines (vebt-memberi ti  x) (vebt-memberi' t ti x)
  apply (induction arbitrary: t ti x rule: vebt-memberi'.fixp-induct)
  subgoal using refines-adm[where t = λ arg. vebt-memberi (snd (fst arg)) (snd arg)]
   by simp
  subgoal by simp
  subgoal for f t ti x
   apply(subst vebt-memberi.simps)
   apply refines
   done
  done

lemma htt-vebt-memberi:
  <vebt-assn-raw t ti>vebt-memberi ti x <λ r. vebt-assn-raw t ti ∗ ↑(r = vebt-member t x)>T[ 5 +
5 ∗ height t]
  apply (rule htt-refine[where c = vebt-memberi' t ti x])
   prefer 2
   apply(rule vebt-memberi-refines)
  apply (rule httI-TBOUND)
   apply(rule vebt-memberi'-rf-abstr)
  apply(rule TBOUND-mono)
   apply(rule TBOUND-vebt-memberi)
  apply simp
  done

lemma htt-vebt-memberi-invar-vebt: assumes invar-vebt t n shows
  <vebt-assn-raw t ti> vebt-memberi ti x <λ r. vebt-assn-raw t ti ∗ ↑(r = vebt-member t x)>T[5 +
5 ∗ (nat ⌈lb n ⌉)]
  by (metis assms heigt-uplog-rel htt-vebt-memberi nat-int)
```

17.1 *minNulli*: empty tree?

```
fun minNulli::VEBTi ⇒ bool Heap where
  minNulli (Leafi False False) = return True|
  minNulli (Leafi - - ) = return False|
  minNulli (Nodei None - - -) = return True|
  minNulli (Nodei (Some -) - - -) = return  False
```

lemma *minNulli-rule*[*sep-heap-rules*]: <*vebt-assn-raw t ti*> *minNulli ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *minNull t*)>
by (*cases t rule*: *minNull.cases*; *cases ti rule*: *minNulli.cases*) (*sep-auto*+)

lemma *TBOUND-minNulli*[*TBOUND*]: *TBOUND* (*minNulli t*) *1*
apply (*induction t rule*: *minNulli.induct*)
apply (*subst minNulli.simps* | *TBOUND-step*)+
done

lemma *minNrulli-ruleT*:
<*vebt-assn-raw t ti*> *minNulli ti* <λ*r*. *vebt-assn-raw t ti* ∗ ↑(*r* = *minNull t*)>*T*[*1*]
by (*metis TBOUND-minNulli hoare-triple-def httI-TBOUND minNulli-rule*)

18 Imperative *vebt* − *insert* to van Emde Boas Tree

end

context begin
interpretation *VEBT-internal* **.**

partial-function (*heap-time*) *vebt-inserti*::*VEBTi* ⇒ *nat* ⇒*VEBTi Heap* **where**
vebt-inserti t x = (*case t of*
(*Leafi a b*) ⇒ (*if x=0 then return* (*Leafi True b*) *else if x=1*
then return (*Leafi a True*) *else return* (*Leafi a b*)) |
(*Nodei info deg treeArray summary*) ⇒ (*case info of None* ⇒
if deg ≤ *1 then*
return (*Nodei info deg treeArray summary*)
else
return (*Nodei* (*Some* (*x*,*x*)) *deg treeArray summary*)|
(*Some minma*) ⇒
(*if deg* ≤ *1*
then return (*Nodei info deg treeArray summary*)
else (*do*{
mi <− *return* (*fst minma*);
ma <− *return* (*snd minma*);
xn <− (*if x* < *mi then return mi else return x*);
minn <− (*if x* < *mi then return x else return mi*);
l<− *lowi xn* (*deg div 2*);
h <− *highi xn* (*deg div 2*);
len ← *Array-Time.len treeArray*;
if h < *len* ∧ ¬ (*x* = *mi* ∨ *x* = *ma*) *then do* {
node <− *Array-Time.nth treeArray h*;
empt <− *minNulli node*;
newnode <− *vebt-inserti node l*;
newarray <− *Array-Time.upd h newnode treeArray*;
newsummary<−(*if empt then*
vebt-inserti summary h

```
                                                    else  return summary);
                                                man <− (if xn > ma then return xn else return ma);
                                                    return (Nodei (Some (minn, man)) deg newarray
newsummary)}
                                                  else return  (Nodei (Some (mi,ma)) deg treeArray
summary)
                                          }))))

end

context VEBT-internal begin

partial-function (heap-time) vebt-inserti'::VEBT ⇒VEBTi ⇒ nat ⇒VEBTi Heap where
  vebt-inserti' t ti x = (case ti of
            (Leafi a b) ⇒ (if x=0 then return (Leafi True b) else if x=1
                              then return (Leafi a True) else return (Leafi a b)) |
            (Nodei info deg treeArray summary) ⇒ ( case info of None ⇒
                                                  if deg ≤ 1 then
                                                      return  (Nodei info deg treeArray summary)
                                                  else
                                                          return (Nodei (Some (x,x)) deg treeArray
summary)|
                                              (Some minma) ⇒
                                                   ( if deg ≤ 1
                                              then return  (Nodei info deg treeArray summary)
                                               else  (
                                    do{
                                        assert' (is-Node t);
                                        let (info',deg',treeList,summary') =
                                        (case t of (Node info' deg' treeList summary') ⇒
                                        (info', deg', treeList, summary'));
                                        assert'(info= info' ∧ deg = deg');
                                                  let (mi', ma') = (the info');
                                                  mi <− return (fst minma);
                                                  ma <− return (snd minma);
                                                  xn <− (if x < mi then return mi else return x);
                                                  let xn' =  (if x < mi' then  mi' else x);
                                                  minn <− (if x < mi then return x else return mi);
                                                  let  minn' = (if x < mi' then x else mi');
                                                  l<− low xi xn (deg div 2);
                                                  assert' (l = low xn' (deg' div 2));
                                                  h <− highi xn (deg div 2);
                                                  len ← Array-Time.len treeArray;
                                                  if h < len ∧ ¬ (x = mi ∨ x = ma) then do {
                                                     assert' (h = high xn' (deg' div 2));
                                                     assert'( h < length treeList);
                                                     node <− Array-Time.nth treeArray h;
                                                     empt <− minNulli node;
                                                     assert' (empt = minNull (treeList ! h));
```

```
                                        newnode <− vebt-inserti' (treeList ! h) node l;
                                        newarray <− Array-Time.upd h newnode treeArray;
                                        newsummary<−(if empt then
                                                vebt-inserti' summary' summary h
                                                else  return summary);
                                       man <− (if xn > ma then return xn else return ma);
                                           return (Nodei (Some (minn, man)) deg newarray
newsummary)}
                                             else return  (Nodei (Some (mi,ma)) deg treeArray
summary)
                                    }))))

lemmas listI-assn-wrap-insert = listI-assn-reinsert-upd'[
        where  x=VEBT-Insert.vebt-insert - - and  A=vebt-assn-raw ]

lemma  vebt-inserti'-rf-abstr: <vebt-assn-raw t ti> vebt-inserti' t ti x <λr. vebt-assn-raw ( vebt-insert
t x) r >
proof(induction t x arbitrary: ti rule: vebt-insert.induct)
  case (1 a b x)
  then show ?case by (subst vebt-inserti'.simps)(cases ti; sep-auto)
next
  case (2 info ts s x)
  then show ?case by (subst vebt-inserti'.simps) (cases ti; sep-auto)
next
  case (3 info ts s x)
  then show ?case by(subst vebt-inserti'.simps) (cases ti; sep-auto)
next
  case (4 v treeList summary x)
  then show ?case by (subst vebt-inserti'.simps)(cases ti; sep-auto)
next
  case (5 mi ma va treeList summary x)
  note IH1 = 5.IH(1)[OF refl refl - -]
  note IH2 = 5.IH(2)[OF refl refl refl]
  show ?case
    apply (cases ti)
    subgoal
      supply [split del] = if-split
      apply (subst vebt-inserti'.simps; clarsimp split del: )
      apply (assn-simp; intro normalize-rules)
      apply (extract-pre-pure dest: extract-pre-list-assn-lengthD)
      apply (simp only:  fold-if-return distrib-if-bind heap-monad-laws)
      apply (clarsimp simp: lowi-def highi-def)
      apply (sep-auto simp: lowi-def highi-def)
      apply(simp add: low-def)
      apply (metis fst-conv)
      apply(rule bind-rule)
      apply sep-auto
      apply (simp cong: if-cong)
      apply sep-auto
```

```
      apply(simp add: high-def)
      apply (unwrap-idx ((if x < mi then mi else x) div (2 * 2 ^ (va div 2))))
      apply (sep-auto simp: low-def high-def)
      apply (heap-rule IH1)
      subgoal
        by (simp add: low-def high-def split: if-splits)
      subgoal
        by (simp add: low-def high-def split: if-splits)
      subgoal
        by (simp add: low-def high-def split: if-splits)
      apply (sep-auto simp: low-def high-def)
      apply (heap-rule IH2)
      subgoal
        by (simp add: low-def high-def split: if-splits)
      subgoal
        by (simp add: low-def high-def)
      subgoal
        by (simp add: low-def high-def split: if-splits)
      apply (wrap-idx R: listI-assn-wrap-insert)
      apply (sep-auto simp: low-def high-def Let-def)
      apply (wrap-idx R: listI-assn-wrap-insert)
      apply (sep-auto simp: low-def high-def Let-def)+
     done
   subgoal
     by simp
  done
qed

lemma TBOUND-minNull: minNull t ⟹ TBOUND (vebt-inserti' t ti x ) 1
  apply(subst vebt-inserti'.simps)
  apply(cases t rule: minNull.cases; simp)
  apply TBOUND+
   apply (auto split: VEBTi.splits option.splits)
  done

lemma TBOUND-vebt-inserti:
  defines foo-def: ⋀ t x. foo t x ≡ if minNull t then 1 else 13 * (1+height t)
  shows TBOUND (vebt-inserti' t ti x) (foo t x)
proof−
  have fooNull:minNull t ⟹ foo t x = 1 for t x using foo-def by simp
  have fooElse: foo t x ≤ 13* (1+ height t) for t using foo-def by simp
  show ?thesis
    apply (induction arbitrary: t ti x rule: vebt-inserti'.fixp-induct)
    apply (rule TBOUND-fi'-adm)
    apply (rule TBOUND-empty)
    apply (rule TBOUND-mono)
    apply TBOUND-step+
     apply (simp split!: VEBTi.splits VEBT.split option.splits prod.splits if-split)
     apply(simp-all add: foo-def height-i-max)
```

```
    done
qed

lemma vebt-inserti-refines: refines (vebt-inserti ti  x) (vebt-inserti' t ti x)
  apply (induction arbitrary: t ti x rule: vebt-inserti'.fixp-induct)
  subgoal using refines-adm[where t = λ arg. vebt-inserti (snd (fst arg)) (snd arg)]
    by simp
  subgoal
    by simp
  apply(subst vebt-inserti.simps)
  apply refines
  done

lemma htt-vebt-inserti:
  <vebt-assn-raw t ti> vebt-inserti ti x <λ r.  vebt-assn-raw (vebt-insert t x) r>T[ 13 + 13 * height
t]
  apply (rule htt-refine[where c = vebt-inserti' t ti x])
   prefer 2
   apply(rule vebt-inserti-refines)
    apply (rule httI-TBOUND)
   apply(rule vebt-inserti'-rf-abstr)
    apply(rule TBOUND-mono)
    apply(rule TBOUND-vebt-inserti)
    apply simp
  done

lemma htt-vebt-inserti-invar-vebt: assumes invar-vebt t n shows
  <vebt-assn-raw t ti> vebt-inserti ti x <λ r.  vebt-assn-raw (vebt-insert t x) r>T[13 + 13 * (nat ⌈lb
n ⌉)]
  by (metis assms heigt-uplog-rel htt-vebt-inserti nat-int)

end
end

theory VEBT-SuccPredImperative
  imports VEBT-BuildupMemImp VEBT-Succ VEBT-Pred
begin

context begin
interpretation VEBT-internal .
```

19 Imperative Successor

```
partial-function (heap-time) vebt-succi::VEBTi ⇒ nat ⇒ (nat option) Heap where
  vebt-succi t x = (case t of (Leafi a b) ⇒(if x = 0 then (if b then return (Some 1) else return None)
                                  else return None)|
                  (Nodei info deg treeArray summary) ⇒ (
                   case info of None ⇒ return None |
                  (Some mima) ⇒ ( if deg ≤ 1 then return None else
```

```
                    (if x < fst mima then return (Some (fst mima)) else
                     if x ≥ snd mima then return None else
                       do {
                          l <− lowi x (deg div 2);
                          h <− highi x (deg div 2);
                          aktnode <− Array-Time.nth treeArray h;
                             maxlow <− vebt-maxti aktnode;
                             if (maxlow ≠ None ∧ (Some l <o  maxlow))
                             then do {
                                    succy <− vebt-succi aktnode l;
                                     return ( Some (2^(deg div 2)) *o Some h +o succy)
                                   }
                             else do {
                                    succsum <−  vebt-succi summary h;
                                    if succsum = None then
                                       return None
                                     else
                                       do{
                                       nextnode <− Array-Time.nth treeArray (the succsum);
                                           minnext <− vebt-minti nextnode;
                                       return (Some (2^(deg div 2)) *o succsum +o minnext)
                                         }
                                    }

                      })
)))
```

end

context *VEBT-internal* **begin**

partial-function (*heap-time*) *vebt-succi'*::*VEBT* ⇒ *VEBTi* ⇒ *nat* ⇒ (*nat option*) *Heap* **where**
vebt-succi' t ti x = (*case ti of* (*Leafi a b*) ⇒(*if x* = *0 then* (*if b then return* (*Some 1*) *else return None*)
else return None)|
(*Nodei info deg treeArray summary*) ⇒ *do* { *assert'*(*is-Node t*);
let (*info'*,*deg'*,*treeList*,*summary'*) =
(*case t of Node info' deg' treeList summary'* ⇒ (*info'*,*deg'*,*treeList*,*summary'*));
assert'(*info'*=*info* ∧ *deg'*=*deg* ∧ *is-Node t*);
case info of None ⇒ *return None* |
(*Some mima*) ⇒ (*if deg* ≤ *1 then return None else*
(*if x < fst mima then return* (*Some* (*fst mima*)) *else*
if x ≥ *snd mima then return None else*
do {
l <− *lowi x* (*deg div 2*);
h <− *highi x* (*deg div 2*);

assert'(*l* = *low x* (*deg div 2*));
assert'(*h* = *high x* (*deg div 2*));

```
                                    assert'(h < length treeList);

                                    aktnode <- Array-Time.nth treeArray h;
                                    let aktnode' = treeList!h;

                                        maxlow <- vebt-maxti aktnode;
                                        assert' (maxlow = vebt-maxt aktnode');
                                        if (maxlow ≠ None ∧ (Some l <o  maxlow))
                                        then do {
                                                succy <- vebt-succi' aktnode' aktnode l;
                                                return ( Some (2^(deg div 2)) *o Some h +o succy)
                                               }
                                        else do {
                                               succsum <-  vebt-succi' summary' summary h;
                                          assert'(succsum = None ⟷ vebt-succ summary' h = None);
                                               if succsum = None then do{
                                                   return None}
                                                 else
                                                   do{
                                                   nextnode <- Array-Time.nth treeArray (the succsum);
                                                        minnext <- vebt-minti nextnode;
                                                   return (Some (2^(deg div 2)) *o succsum +o minnext)
                                                     }
                                               }

                                    })
)})

theorem vebt-succi'-rf-abstr:invar-vebt t n ⟹ <vebt-assn-raw t ti> vebt-succi' t ti x <λr. vebt-assn-raw
t ti * ↑(r = vebt-succ t x)>
proof(induction t x arbitrary: ti n rule: vebt-succ.induct)
  case (1 uu b)
  then show ?case by(subst vebt-succi'.simps) (cases ti; sep-auto)
next
  case (2 uv uw n)
  then show ?case by(subst vebt-succi'.simps) (cases ti; sep-auto)
next
  case (3 ux uy uz va)
  then show ?case by(subst vebt-succi'.simps) (cases ti; sep-auto)
next
  case (4 v vc vd ve)
  then show ?case by(subst vebt-succi'.simps) (cases ti; sep-auto)
next
  case (5 v vg vh vi)
  then show ?case by(subst vebt-succi'.simps) (cases ti; sep-auto)
next
  case (6 mi ma va treeList summary x)
  have setprop: t ∈ set treeList ⟹ invar-vebt t (n div 2 ) for t using 6(3)
    by (cases) simp+
```

have *listlength*: *length treeList* = *2^(n − n div 2)* **using** *6(3)*
by (*cases*) *simp+*
have *sumprop*: *invar-vebt summary* (*n − n div 2*) **using** *6(3)*
by (*cases*) *simp+*
have *xprop* [*simp*]: ¬ *ma* ≤ *x* ⟹ *high x* (*Suc* (*va div 2*)) < *length treeList*
by (*smt* (*verit*) *6.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux listlength mi-ma-2-deg not-le-imp-less order.strict-trans ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
hence *xprop'* [*simp*]: ¬ *ma* ≤ *x* ⟹ *x div* (*2* ∗ *2* ^ (*va div 2*)) < *length treeList* **unfolding** *high-def* **by** *simp*
show *?case*
apply (*cases ti*)
prefer *2*
subgoal
apply *simp*
done
subgoal for *x11 x12 x13 x14*
supply [*split del*] = *if-split*
apply (*subst vebt-succi'.simps*; *clarsimp split del*:)
apply (*assn-simp*; *intro normalize-rules*)
apply *simp*
apply(*auto split*: *if-split*)
subgoal
apply *sep-auto*
done
apply *sep-auto*
using *6.prems geqmaxNone*
apply *fastforce*
apply *sep-auto*
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
apply (*sep-auto simp*: *lowi-def low-def heap*: *highi-h*)
apply(*sep-auto heap*: *vebt-maxtilist*)
apply *sep-auto*
apply(*simp add*: *high-def low-def*)
apply (*rewrite* **in** <¤>-<-> *list-assn-conv-idx*)
apply(*rewrite* **in** <¤>-<-> *listI-assn-extract*[**where** *i*=(*x div* (*2* ∗ *2* ^ (*va div 2*)))])
apply (*smt* (*verit*) *6.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 high-bound-aux high-def le0 le-add-diff-inverse listlength mi-ma-2-deg nat-le-linear power-Suc*)
apply (*smt* (*verit*) *6.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 high-bound-aux high-def le-add-diff-inverse listlength mi-ma-2-deg nat-le-linear power-Suc*)
apply(*sep-auto heap*: *6.IH*(*1*))
apply(*simp add*: *low-def*)
apply(*simp add*: *high-def*)
apply *simp+*
apply(*rule setprop*)
apply *simp*
subgoal for *tree-is x*
apply *sep-auto*
apply (*smt* (*verit*) *6.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 high-bound-aux high-def le-add-diff-inverse less-shift listlength low-def mi-ma-2-deg nat-le-linear op-*

tion.distinct(*1*) *power-Suc*)
apply(*rule ent-trans*[**where** *Q*= *vebt-assn-raw summary x14* ∗ (*x13* ↦$_a$ *tree-is*)∗
(*list-assn vebt-assn-raw treeList tree-is*)])
apply (*smt* (*verit*) *assn-aci*(*10*) *atLeastLessThan-iff entails-def leI less-nat-zero-code listI-assn-extract list-assn-conv-idx star-aci*(*2*) *xprop*′)
apply(*rule ent-refl*)
done
apply *simp*
apply *sep-auto*
apply(*sep-auto heap*: *6*.*IH*(*2*))
apply (*simp add*: *high-def low-def*)+
apply (*rule sumprop*)
apply(*sep-auto heap*: *6*.*IH*(*2*))
apply (*simp add*: *high-def low-def*)+
apply (*rule sumprop*)
apply *sep-auto*+
apply(*simp add*: *high-def low-def*)+
using *helpyd listlength sumprop*
apply *presburger*+
apply (*sep-auto heap*: *vebt-mintilist*)
using *helpyd listlength sumprop*
apply *presburger*
using *helpyd listlength sumprop*
apply *presburger*+
apply *sep-auto*
done
done
qed

lemma *TBOUND-vebt-succi*:
defines *foo-def*: ⋀ *t x*. *foo t x* ≡ *7* ∗ (*1*+*height t*)
shows *TBOUND* (*vebt-succi*′ *t ti x*) (*foo t x*)
apply (*induction arbitrary*: *t ti x rule*: *vebt-succi*′.*fixp-induct*)
apply (*rule TBOUND-fi*′*-adm*)
apply (*rule TBOUND-empty*)
apply *TBOUND*
apply(*simp add*: *Let-def split*!: *VEBTi.splits VEBT.splits prod.splits option.splits if-splits*)
apply(*simp-all add*: *foo-def max-idx-list*)
done

lemma *vebt-succi-refines*: *refines* (*vebt-succi ti x*) (*vebt-succi*′ *t ti x*)
apply (*induction arbitrary*: *t ti x rule*: *vebt-succi*′.*fixp-induct*)
subgoal using *refines-adm*[**where** *t* = λ *arg*. *vebt-succi* (*snd* (*fst arg*)) (*snd arg*)]
by *simp*
subgoal by *simp*
subgoal for *f t ti x*
apply(*subst vebt-succi.simps*)
apply *refines*
done

done

lemma *htt-vebt-succi*: **assumes** *invar-vebt t n*
shows $<$*vebt-assn-raw t ti*$>$ *vebt-succi ti x* $<\lambda$ *r. vebt-assn-raw t ti* $*$ $\uparrow(r = vebt\text{-}succ\ t\ x) > T[7 + 7*(nat\ \lceil lb\ n \rceil)]$
apply (*rule htt-refine*[**where** *c* = *vebt-succi' t ti x*])
prefer *2*
apply(*rule vebt-succi-refines*)
apply (*rule httI-TBOUND*)
apply(*rule vebt-succi'-rf-abstr*)
apply(*rule assms*)
apply(*rule TBOUND-mono*)
apply(*rule TBOUND-vebt-succi*)
apply *simp*
apply(*rule Nat.eq-imp-le*)
apply (*metis assms nat-int heigt-uplog-rel*)
done

end

context begin
interpretation *VEBT-internal* **.**

partial-function (*heap-time*) *vebt-predi*::*VEBTi* $\Rightarrow$ *nat* $\Rightarrow$ (*nat option*) *Heap* **where**
vebt-predi t x = (*case t of* (*Leafi a b*) $\Rightarrow$(*if x* $\geq$ *2then* (*if b then return* (*Some 1*) *else if a then return* (*Some 0*) *else return None*)
else if x = 1 then (*if a then return* (*Some 0*) *else return None*) *else return None*)|
(*Nodei info deg treeArray summary*) $\Rightarrow$ (
case info of None $\Rightarrow$ *return None* |
(*Some mima*) $\Rightarrow$ (*if deg* $\leq$ *1 then return None else*
(*if x > snd mima then return* (*Some* (*snd mima*)) *else*
do {
l $<-$ *lowi x* (*deg div 2*);
h $<-$ *highi x* (*deg div 2*);
aktnode $<-$ *Array-Time.nth treeArray h*;
minlow $<-$ *vebt-minti aktnode*;
if (*minlow* $\neq$ *None* $\wedge$ (*Some l* $>_o$ *minlow*))
then do {
predy $<-$ *vebt-predi aktnode l*;
return (*Some* (*2*$\widehat{}$(*deg div 2*)) $*_o$ *Some h* $+_o$ *predy*)
}
else do {
predsum $<-$ *vebt-predi summary h*;
if predsum = None then
if x > fst mima then
return (*Some* (*fst mima*))
else

```
                                    return None
                                else
                                  do{
                                  nextnode <- Array-Time.nth treeArray (the predsum);
                                      maxnext <- vebt-maxti nextnode;
                                  return (Some (2^(deg div 2)) *o predsum +o maxnext)
                                    }
                                }
                        }))))
```

end
context *VEBT-internal* **begin**

20 Imperative Predecessor

partial-function (*heap-time*) *vebt-predi'*::*VEBT* ⇒ *VEBTi* ⇒ *nat* ⇒ (*nat option*) *Heap* **where**
vebt-predi' t ti x = (*case ti of* (*Leafi a b*) ⇒(*if x* ≥ *2then* (*if b then return* (*Some 1*) *else if a then return* (*Some 0*) *else return None*)
else if x = *1 then* (*if a then return* (*Some 0*) *else return None*) *else return None*)|
(*Nodei info deg treeArray summary*) ⇒ (*do* { *assert'*(*is-Node t*);
let (*info'*,*deg'*,*treeList*,*summary'*) =
(*case t of Node info' deg' treeList summary'* ⇒ (*info'*,*deg'*,*treeList*,*summary'*));
assert'(*info'*=*info* ∧ *deg'*=*deg* ∧ *is-Node t*);
case info of None ⇒ *return None* |
(*Some mima*) ⇒ (*if deg* ≤ *1 then return None else*
(*if x* > *snd mima then return* (*Some* (*snd mima*)) *else*
do {
l <- *lowi x* (*deg div 2*);
h <- *highi x* (*deg div 2*);

assert'(*l* = *low x* (*deg div 2*));
assert'(*h* = *high x* (*deg div 2*));
assert'(*h* < *length treeList*);

aktnode <- *Array-Time.nth treeArray h*;
let aktnode' = *treeList*!*h*;
minlow <- *vebt-minti aktnode*;
assert' (*minlow* = *vebt-mint aktnode'*);

if (*minlow* ≠ *None* ∧ (*Some l* >*o* *minlow*))
then do {
predy <- *vebt-predi' aktnode' aktnode l*;
return (*Some* (*2*^(*deg div 2*)) *o Some h* +*o predy*)
}
else do {
predsum <- *vebt-predi' summary' summary h*;
assert'(*predsum* = *None* ⟷ *vebt-pred summary' h* = *None*);
if predsum = *None then*


```
```

```
                                    if x > fst mima then
                                        return (Some (fst mima))
                                        else
                                          return None
                                  else
                                    do{
                                    nextnode <- Array-Time.nth treeArray (the predsum);
                                        maxnext <- vebt-maxti nextnode;
                                    return (Some (2^(deg div 2)) *o predsum +o maxnext)
                                      }
                                }
                        }))}))
```

theorem *vebt-pred'-rf-abstr*:*invar-vebt t n* $\Longrightarrow$ <*vebt-assn-raw t ti*> *vebt-predi' t ti x* <λ*r. vebt-assn-raw t ti* $*$ $\uparrow$(*r = vebt-pred t x*)>
proof(*induction t x arbitrary*: *ti n rule*: *vebt-pred.induct*)
case (*1 uu uv*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*2 a uw*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*3 a b va*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*4 uy uz va vb*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*5 v vd ve vf*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*6 v vh vi vj*)
then show *?case* **by**(*subst vebt-predi'.simps*) (*cases ti*; *sep-auto*)
next
case (*7 mi ma va treeList summary x*)
have *setprop*: *t* $\in$ *set treeList* $\Longrightarrow$ *invar-vebt t* (*n div 2*) **for** *t* **using** *7*(*3*)
by (*cases*) *simp+*
have *listlength*: *length treeList = 2^(n - n div 2)* **using** *7*(*3*)
by (*cases*) *simp+*
have *sumprop*: *invar-vebt summary* (*n - n div 2*) **using** *7*(*3*)
by (*cases*) *simp+*
have *mimapr*: *ma* $\geq$ *mi* **using** *7*(*3*)
by (*cases*) *simp+*
show *?case*
apply (*cases ti*)
prefer *2*
subgoal
apply *simp*
done

subgoal
supply [*split del*] = *if-split*
apply (*subst vebt-predi'.simps*; *clarsimp split del*:)
apply (*assn-simp*; *intro normalize-rules*)
apply *simp*
apply(*cases ma* < *x*)
subgoal
apply *simp*
apply *sep-auto*
done
apply *simp*
apply (*extract-pre-pure dest*: *extract-pre-list-assn-lengthD*)
apply(*sep-auto simp*: *highi-def*)
apply (*sep-auto simp*: *lowi-def*)
apply *sep-auto*
apply(*simp add*: *low-def*)
apply *sep-auto*
apply(*simp add*: *high-def*)
apply *sep-auto*
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)
apply *sep-auto*
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)
apply (*sep-auto heap*: *vebt-mintilist*)
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)
apply *sep-auto*
apply (*rewrite* **in** <¤>-<-> *list-assn-conv-idx*)
apply(*rewrite* **in** <¤>-<-> *listI-assn-extract*[**where** *i*=(*x div* (*2* ∗ *2* ^ (*va div 2*)))])
apply (*smt* (*verit*) *7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le0 le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)

apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)
apply (*sep-auto heap*: *7.IH*(*1*))
apply(*simp add*: *high-def low-def*)+
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)
apply(*rule DEADID.rel-refl*)
apply (*metis greater-shift option.simps*(*3*))
apply(*rule setprop*)
apply(*rule nth-mem*)
apply (*smt* (*verit*) *7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le0 le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)

apply *simp*
subgoal
apply *sep-auto*

apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 greater-shift high-bound-aux high-def leI le-add-diff-inverse listlength low-def mi-ma-2-deg option.distinct*(*1*) *power-Suc*)
apply (*rule recomp*)
apply (*smt* (*verit*) *7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le0 le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)

apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 greater-shift high-bound-aux high-def leI le-add-diff-inverse listlength low-def mi-ma-2-deg option.distinct*(*1*) *power-Suc*)
apply (*rule recomp*)
apply (*smt* (*verit*) *7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le0 le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)

done
apply(*sep-auto heap*: *7.IH*(*2*))
apply(*simp add*: *high-def low-def*)+
apply (*smt* (*verit*) *7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc div-le-dividend high-bound-aux high-def le0 le-add-diff-inverse linorder-neqE-nat listlength mi-ma-2-deg order.strict-trans power-Suc*)

apply(*rule DEADID.rel-refl*)
apply (*simp add*: *low-def*)
apply(*rule sumprop*)
apply *sep-auto*
apply(*sep-auto simp*: *high-def low-def*)+
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 high-bound-aux high-def leI le-add-diff-inverse listlength mi-ma-2-deg power-Suc*)
apply (*simp add*: *high-def low-def*)
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 greater.elims*(*2*) *high-bound-aux high-def leI le-add-diff-inverse listlength mi-ma-2-deg power-Suc*)
apply *sep-auto*
apply(*sep-auto simp*: *high-def low-def*)+
apply *presburger*
apply (*smt* (*verit*) *greater.elims*(*2*) *high-def low-def power-Suc*)
apply (*simp add*: *high-def low-def*)
apply *sep-auto*
subgoal
using *helpypredd listlength sumprop* **apply** *simp*
done
subgoal
using *helpypredd listlength sumprop* **apply** *simp*
done
apply *sep-auto*
apply(*rule cons-pre-rule*)
apply(*sep-auto heap*: *vebt-maxti-h*)
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*=*the* (*vebt-pred summary* (*x div* (*2* ∗ *2* ^ (*va div 2*))))])
apply (*metis atLeastLessThan-iff helpypredd le0 listlength option.sel sumprop*)

apply (*metis helpypredd listlength option.sel sumprop*)
apply (*simp add*: *algebra-simps*)
apply(*rule cons-pre-rule*)
apply(*rule ext*)
using *helpypredd listlength sumprop* **apply** *presburger*
apply(*sep-auto heap*: *vebt-maxti-h*)
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*=*the* (*vebt-pred summary* (*x div* (*2* ∗ *2* ^ (*va div 2*))))])
apply (*metis atLeastLessThan-iff helpypredd le0 listlength option.sel sumprop*)
apply (*metis helpypredd listlength option.sel sumprop*)
apply *simp*
apply(*sep-auto heap*: *vebt-maxti-h*)
apply *sep-auto*
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 high-bound-aux high-def leI le-add-diff-inverse listlength mi-ma-2-deg option.distinct*(*1*) *option.sel power-Suc*)
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 greater.elims*(*2*) *high-bound-aux high-def leI le-add-diff-inverse listlength mi-ma-2-deg option.distinct*(*1*) *option.sel power-Suc*)
apply(*rule txe*)
using *helpypredd listlength sumprop* **apply** *presburger*
apply (*smt* (*verit*) *7.prems deg-deg-n div2-Suc-Suc div-le-dividend dual-order.strict-trans2 greater.elims*(*2*) *high-bound-aux high-def leI le-add-diff-inverse listlength mi-ma-2-deg option.distinct*(*1*) *option.sel power-Suc*)
apply(*rule txe*)
using *helpypredd listlength sumprop* **apply** *presburger*
done
done
qed

lemma *TBOUND-vebt-predi*:
defines *foo-def*: ⋀ *t x*. *foo t x* ≡ *7* ∗ (*1*+*height t*)
shows *TBOUND* (*vebt-predi' t ti x*) (*foo t x*)
apply (*induction arbitrary*: *t ti x rule*: *vebt-predi'.fixp-induct*)
apply (*rule TBOUND-fi'-adm*)
apply (*rule TBOUND-empty*)
apply *TBOUND*
apply (*simp add*: *Let-def split*!: *VEBTi.splits VEBT.splits option.splits prod.splits if-splits*)
apply(*simp-all add*: *foo-def max-idx-list*)
done

lemma *vebt-predi-refines*: *refines* (*vebt-predi ti x*) (*vebt-predi' t ti x*)
apply (*induction arbitrary*: *t ti x rule*: *vebt-predi'.fixp-induct*)
subgoal using *refines-adm*[**where** *t* = λ *arg*. *vebt-predi* (*snd* (*fst arg*)) (*snd arg*)]
by *simp*
subgoal by *simp*
subgoal for *f t ti x*
apply(*subst vebt-predi.simps*)
apply *refines*
done
done

lemma *htt-vebt-predi*: **assumes** *invar-vebt t n*
shows $<$*vebt-assn-raw t ti*$>$ *vebt-predi ti x* $<\lambda$ *r. vebt-assn-raw t ti* $*$ $\uparrow(r = vebt\text{-}pred\ t\ x) > T[\gamma + \gamma * (nat\ \lceil lb\ n \rceil)]$
apply (*rule htt-refine*[**where** $c = vebt\text{-}predi'\ t\ ti\ x$])
prefer *2*
apply(*rule vebt-predi-refines*)
apply (*rule httI-TBOUND*)
apply(*rule vebt-pred'-rf-abstr*)
apply(*rule assms*)
apply(*rule TBOUND-mono*)
apply(*rule TBOUND-vebt-predi*)
apply *simp*
apply(*rule Nat.eq-imp-le*)
apply (*metis assms nat-int heigt-uplog-rel*)
done

end
end

theory *VEBT-DelImperative* **imports** *VEBT-DeleteCorrectness VEBT-SuccPredImperative*
begin

context begin
interpretation *VEBT-internal* **.**

21 Imperative Delete

```
partial-function (heap-time) vebt-deletei::VEBTi ⇒ nat ⇒ VEBTi Heap where
  vebt-deletei t x = (case t of (Leafi a b) ⇒ (if x = 0 then return (Leafi False b) else
                                  if x = 1 then return (Leafi a False) else
                                   return (Leafi a b)) |
                       (Nodei info deg treeArray summary) ⇒ (
                          if deg ≤ 1 then return (Nodei info deg treeArray summary) else
                              case info of None ⇒ return (Nodei info deg treeArray summary)|
                                   (Some mima) ⇒ ( if x < fst mima ∨ x > snd mima then return
(Nodei info deg treeArray summary)
                                        else if fst mima = x ∧ snd mima = x then return (Nodei
None deg treeArray summary)
                                        else do{ xminew <- (if x = fst mima then do {
                                                    firstcluster <- vebt-minti summary;
                                                   firsttree <- Array-Time.nth treeArray (the
firstcluster);
                                                    mintft <- vebt-minti firsttree;
                                                  let xn = (2^(deg div 2) * (the firstcluster) +
                                                        (the mintft) );
                                                  return (xn, xn)
                                                 }
                                                 else return (x, fst mima));
```

```
                                        let xnew = fst xminew;
                                        let minew = snd xminew;
                                        h <− highi xnew (deg div 2);
                                        l <− lowi xnew (deg div 2);
                                        aktnode <− Array-Time.nth treeArray h;
                                        aktnode'<−vebt-deletei aktnode l;
                                        treeArray' <− Array-Time.upd h aktnode' treeArray;
                                        miny <− vebt-minti aktnode';
                                       (if (miny = None) then
                                        do{
                                            summary' <−vebt-deletei summary h;
                                            ma <−  (if xnew = snd mima then
                                            do{
                                                summax <− vebt-maxti summary';
                                                if summax = None then
                                                      return minew
                                                else do{
                                                    maxtree <− Array-Time.nth treeArray' (the
summax);
                                                        mofmtree<− vebt-maxti maxtree;
                                                        return (the summax * 2^(deg div 2) +
                                                                    the mofmtree )
                                                        }
                                                }
                                                else return (snd mima));
                                            return (Nodei (Some (minew, ma)) deg treeArray'
summary')
                                        } else  if xnew = snd mima then
                                          do{
                                             nextree <− Array-Time.nth treeArray' h;
                                             maxnext<− vebt-maxti nextree;
                                             let ma = h * 2^(deg div 2) +
                                                        (the maxnext);
                                            return (Nodei (Some ( minew, ma)) deg treeArray'
summary)
                                             }
                                            else return (Nodei (Some (minew, snd mima)) deg
treeArray' summary) )
                                        }))))
```

end

context *VEBT-internal* **begin**

Some general lemmas

lemma *midextr*:$(P * Q * Q' * R \Longrightarrow_A X) \Longrightarrow (P * R * Q * Q' \Longrightarrow_A X)$
by (*smt* (*verit*, *ccfv-threshold*) *ab-semigroup-mult-class.mult.commute assn-aci*(9) *entails-def mod-frame-fwd*)

lemma *groupy*: $A * B * (C * D) \Longrightarrow_A X \Longrightarrow A * B * C * D \Longrightarrow_A X$

by (*simp add*: *assn-aci*(*9*))

lemma *swappa*: $B* A* C \Longrightarrow_A X \Longrightarrow A *B * C \Longrightarrow_A X$
by (*simp add*: *ab-semigroup-mult-class.mult.commute*)

lemma *mulcomm*: (*i*::*nat*) * (*2* * *2* ^ (*va div 2*)) = (*2* * *2* ^ (*va div 2*)) * *i*
by *simp*

Modified function with ghost variable

partial-function (*heap-time*) *vebt-deletei'*::*VEBT* ⇒ *VEBTi* ⇒ *nat* ⇒ *VEBTi Heap* **where**
vebt-deletei' t ti x = (*case ti of* (*Leafi a b*) ⇒ (*if x = 0 then return* (*Leafi False b*) *else*
if x = 1 then return (*Leafi a False*) *else*
return (*Leafi a b*)) |
(*Nodei info deg treeArray summary*) ⇒ (
do { *assert'*(*is-Node t*);
let (*info'*,*deg'*,*treeList*,*summary'*) =
(*case t of Node info' deg' treeList summary'*
⇒ (*info'*,*deg'*,*treeList*,*summary'*));
assert'(*info'*=*info* ∧ *deg'*=*deg* ∧ *is-Node t*);
if deg ≤ *1 then return* (*Nodei info deg treeArray summary*) *else*
case info of None ⇒ *return* (*Nodei info deg treeArray summary*)|
(*Some mima*) ⇒ (
if x < *fst mima* ∨ *x* > *snd mima then return* (*Nodei info deg treeArray summary*)
else if fst mima = *x* ∧ *snd mima* = *x then return* (*Nodei None deg treeArray summary*)
else do{ *xminew* <− (*if x* = *fst mima then do* {
firstcluster <− *vebt-minti summary*;
firsttree <− *Array-Time.nth* *treeArray* (*the firstcluster*);
mintft <− *vebt-minti firsttree*;
let xn = (*2*^(*deg div 2*) * (*the firstcluster*) +
(*the mintft*));
return (*xn*, *xn*)
}
else return (*x*, *fst mima*));
let xnew = *fst xminew*;
let xn' =
(*if x* = *fst* (*the info'*)
then the (*vebt-mint summary'*) * *2*^(*deg div 2*)
+ *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary'*)))
else x);
assert' (*xnew* = *xn'*);
let minew = *snd xminew*;
assert' (*minew* = (*if x* = *fst* (*the info'*) *then xn'* *else* *fst* (*the info'*)));
h <− *highi xnew* (*deg div 2*);
assert' (*h* = *high xnew* (*deg div 2*));
assert'(*h* < *length treeList*);

```
                                        l <− lowi xnew (deg div 2);
                                        assert'(l = low xnew (deg div 2));
                                        aktnode <− Array-Time.nth treeArray h;
                                        aktnode'<−vebt-deletei' (treeList ! h) aktnode l;
                                        treeArray' <− Array-Time.upd h aktnode' treeArray;
                                      let funnode = vebt-delete (treeList !  h) l;
                                      let treeList' = treeList[h:= funnode];
                                        miny <− vebt-minti aktnode';
                                        assert' (miny = vebt-mint funnode);
                                       (if (miny = None) then
                                        do{
                                            summaryi' <−vebt-deletei' summary' summary h;
                                            ma <−  (if xnew = snd mima then
                                            do{
                                                summax <− vebt-maxti summaryi';
                                                      assert' (summax = vebt-maxt (vebt-delete
summary' h));
                                                if summax = None then
                                                     return minew
                                                else do{
                                                    maxtree <− Array-Time.nth treeArray' (the
summax);
                                                      mofmtree<− vebt-maxti maxtree;
                                                      return (the summax * 2^(deg div 2) +
                                                                  the mofmtree )
                                                      }
                                                }
                                                 else return (snd mima));
                                             return (Nodei (Some (minew, ma)) deg treeArray'
summaryi')
                                        } else  if xnew = snd mima then
                                          do{
                                              nextree <− Array-Time.nth treeArray' h;
                                              maxnext<− vebt-maxti nextree;
                                              assert' (maxnext = vebt-maxt (treeList' ! h));
                                              let ma = h * 2^(deg div 2) +
                                                          (the maxnext);
                                              return (Nodei (Some ( minew, ma)) deg treeArray'
summary)
                                               }
                                              else return (Nodei (Some (minew, snd mima)) deg
treeArray' summary) )
                                        })}))
```

theorem *deleti'-rf-abstr*:
notes *list-update-beyond* [*simp*]
shows *invar-vebt t n* ⟹ <*vebt-assn-raw t ti*> *vebt-deletei' t ti x*< *vebt-assn-raw* (*vebt-delete t x*)>
proof(*induction t x arbitrary*: *ti n rule*: *vebt-delete.induct*)

case (*1 a b*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*2 a b*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*3 a b n*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*4 deg treeList summary uu*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*5 mi ma treeList summary x*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*6 mi ma treeList summary x*)
then show *?case* **by**(*subst vebt-deletei'.simps*) (*cases ti*; *sep-auto*)
next
case (*7 mi ma va treeList summary x*)
have *setprop*: $t \in set\ treeList \Longrightarrow invar\text{-}vebt\ t\ (n\ div\ 2\ )$ **for** t **using** *7*(*3*)
by (*cases*) *simp+*
have *listlength*: $length\ treeList = 2\hat{}(n - n\ div\ 2)$ **using** *7*(*3*)
by (*cases*) *simp+*
have *sumprop*: $invar\text{-}vebt\ summary\ (n - n\ div\ 2)$ **using** *7*(*3*)
by (*cases*) *simp+*
have *mimaxprop*: $mi \leq ma \wedge ma \leq 2\hat{}n$ **using** *7*(*3*)
by *cases simp+*
hence *xbound*: $mi \leq x \Longrightarrow x \leq ma \Longrightarrow high\ x\ (n\ div\ 2) \leq length\ treeList$
using *div-le-mono high-def listlength power-minus-is-div* **by** *auto*
let *?xn* = *the* (*vebt-mint summary*) * 2 ^ (*Suc* (*Suc va*) *div 2*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))
obtain *xnew* **where** *xndef*: *xnew* = *?xn* **by** *simp*
let *?minn* = *?xn*
obtain *minew* **where** *minewdef*: *minew* =*?minn* **by** *simp*
have *highboundn*: $ma \neq mi \Longrightarrow high\ xnew\ (n\ div\ 2) < length\ treeList$
by (*metis 7.prems deg-deg-n div2-Suc-Suc leD mi-ma-2-deg nested-mint power-Suc xndef*)
have *highbound*: $x \leq ma \Longrightarrow high\ x\ (n\ div\ 2) < length\ treeList$
by (*metis* (*no-types*, *opaque-lifting*) *listlength div-le-dividend 7.prems less-imp-diff-less le-add-diff-inverse diff-diff-cancel high-bound-aux*[*of x n div 2 n − n div 2*] *deg-deg-n mi-ma-2-deg*)
let *?aktnode* = (*treeList* !
high (*2 * 2 ^ (va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*)))
obtain *aktnode* **where** *aktnodedef*:*ma* ≠ *mi* ⟹*x*≤ *ma* ⟹*aktnode* = *?aktnode*
by *meson*
let *?newnode* = *vebt-delete ?aktnode* (*low ?xn* (*Suc* (*Suc va*) *div 2*))
obtain *newnode* **where** *newnodedef*:*newnode* = *?newnode* **by** *presburger*
let *?newlist*=*treeList*[*high* (*2 * 2 ^ (va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*)) :=

```
                ?newnode]
  let ?newlist'=treeList[ high x  (Suc (va div 2)) := vebt-delete (treeList ! high x (Suc (va div 2)))
(low x (Suc (va div 2)))]
  show ?case
    apply(cases ti)
     prefer 2
     subgoal
      apply simp
     done
    supply [split del] = if-split
    apply (subst vebt-deletei'.simps; clarsimp split del: )
    apply (assn-simp; intro normalize-rules)
    apply simp
    apply(cases x < mi ∨ ma < x)
    subgoal
      apply simp
      apply sep-auto
      done
    apply simp
    apply(cases mi = x ∧ ma = x)
    subgoal
      apply simp
      apply sep-auto
      done
    apply (extract-pre-pure dest: extract-pre-list-assn-lengthD)
    apply (cases mi = x ∧ ma = x; simp)
    apply(cases x = mi)
    subgoal
      apply simp
      apply sep-auto
      apply(sep-auto heap: vebt-minti-h)
      apply sep-auto
      apply (metis 7.prems listlength mintlistlength option.sel)
      apply sep-auto
      apply (rewrite in <¤>-<-> list-assn-conv-idx)
      apply (rewrite in <¤>-<-> listI-assn-extract[where i=the (vebt-mint summary)])
      apply (metis 7.prems atLeastLessThan-iff le0 listlength mintlistlength option.sel)
      apply (metis 7.prems listlength mintlistlength option.sel)
      apply(sep-auto heap: vebt-minti-h)
      apply(rule cons-pre-rule)
      apply (rule repack)
      apply (metis 7.prems listlength mintlistlength option.sel)
      apply sep-auto
      apply (sep-auto heap: highi-h)
      apply sep-auto
      apply (metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint)
      apply (sep-auto heap: lowi-h)
     apply (metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n div2-Suc-Suc highboundn
mimaxprop power-Suc xndef)
```

apply *sep-auto*
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*= *high* (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*))])
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute atLeastLessThan-iff deg-deg-n leI less-nat-zero-code nested-mint*)
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint*)
apply *sep-auto*
apply(*sep-auto heap*: *7.IH*(*1*))
apply(*simp add*: *algebra-simps*)
apply(*simp add*: *algebra-simps*)
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint*)
apply(*rule setprop*)
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint nth-mem*)
apply *sep-auto*
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint*)
apply(*simp add*: *Let-def*)
apply *sep-auto*
apply(*sep-auto heap*: *vebt-minti-h*)
apply(*rule cons-pre-rule*)
apply(*rule big-assn-simp*[*of high* (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*))
treeList (*low* (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*)))
- - - - *summary*])
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n div2-Suc-Suc highboundn mimaxprop power-Suc xndef*)
apply(*cases vebt-mint* (*vebt-delete* (*treeList* !
high (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*)))
(*low* (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*)))) = *None*)
apply *simp*
subgoal
apply *sep-auto*
apply(*sep-auto heap*: *7.IH*(*2*))
apply(*simp add*: *algebra-simps*)+
subgoal
by (*metis 7.prems add.commute mult.left-commute deg-deg-n div2-Suc-Suc highboundn power-Suc xndef*)
apply(*rule DEADID.rel-refl*)
apply(*rule DEADID.rel-refl*)
apply(*rule minminNull*)
apply (*metis ab-semigroup-mult-class.mult.commute*)
apply(*rule sumprop*)
apply(*rule bind-rule'*[**where** *R*=λ *r*.(*let sn* = *vebt-delete summary* (*high* (*2 * 2 ^(va div 2) * the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*))) *in* (↑(*r* = (
if ?xn = *ma then let maxs* = *vebt-maxt sn in if maxs* = *None then ?xn*

*else 2 ^(Suc (Suc va) div 2) * the maxs + the (vebt-maxt (?newlist ! the maxs)) else ma*)))])

apply(*cases 2 * 2 ^(va div 2) * the (vebt-mint summary) + the (vebt-mint (treeList ! the (vebt-mint summary))) = ma*)

apply *simp+*

apply *sep-auto*

apply(*sep-auto heap*: *vebt-maxti-h*)

apply *sep-auto*

using *delete-pres-valid*[*of summary n − n div 2*
(*high (2 * 2 ^(va div 2) * the (vebt-mint summary) + the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2))*)]
maxt-member[*of vebt-delete summary - n−n div 2*] *member-bound*[*of vebt-delete summary - - n−n div 2*] *listlength sumprop*

apply (*metis both-member-options-equiv-member dele-bmo-cont-corr maxbmo member-bound*)

apply *sep-auto*

apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)

apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*= *the (vebt-maxt (vebt-delete summary (high (2 * 2 ^(va div 2) * the (vebt-mint summary)*
+ the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2)))))])

apply (*metis atLeastLessThan-iff both-member-options-equiv-member dele-bmo-cont-corr le0 length-list-update listlength maxbmo member-bound option.sel sumprop*)

apply (*metis both-member-options-equiv-member dele-bmo-cont-corr length-list-update listlength maxbmo member-bound option.sel sumprop*)

apply(*sep-auto heap*: *vebt-maxti-h*)

apply *sep-auto*

apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)

apply *auto*[*1*]

apply(*cases the(vebt-maxt (vebt-delete summary*
(*high (2 * 2 ^(va div 2) * the (vebt-mint summary) + the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2))))) < length treeList*)

using *txe*

apply (*metis length-list-update option.sel*)

apply (*metis both-member-options-equiv-member dele-bmo-cont-corr listlength maxbmo member-bound option.sel sumprop*)

apply *simp*

apply *sep-auto*

apply (*metis ab-semigroup-mult-class.mult.commute*)

apply *sep-auto*

apply(*simp add*: *Let-def*)

apply(*cases high (the (vebt-mint summary) * (2 * 2 ^(va div 2)) + the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2)) < length treeList*)

apply *simp*

apply(*cases minNull (vebt-delete (treeList !*
*high (the (vebt-mint summary) * (2 * 2 ^(va div 2)) + the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2))*)
(*low (the (vebt-mint summary) * (2 * 2 ^(va div 2)) + the (vebt-mint (treeList ! the (vebt-mint summary)))) (Suc (va div 2))*))

subgoal

apply(*sep-auto simp*: *algebra-simps split*: *if-split*)

done
subgoal
apply(*auto split*: *if-split*)
apply *sep-auto*
prefer *2*
apply(*rule entails-solve-init*)
apply (*tactic ‹Seplogic-Auto.match-frame-tac (resolve-tac @{context} @{thms ent-refl}) @{context} 1›*)
apply *simp*
apply *solve-entails*
apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)
prefer *3*
apply *sep-auto*
apply (*metis ab-semigroup-mult-class.mult.commute minminNull*)+
done
subgoal
apply(*rule entails-solve-init*)
apply *simp*
done
subgoal
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply *sep-auto*
using *7.prems deg-deg-n nested-mint* **apply** *blast*
apply(*rule entails-solve-init*)
apply (*tactic ‹Seplogic-Auto.match-frame-tac (resolve-tac @{context} @{thms ent-refl}) @{context} 1›*)
apply *simp*
apply *solve-entails*
apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)
using *7.prems deg-deg-n nested-mint* **apply** *blast*
apply *sep-auto*
using *7.prems deg-deg-n nested-mint* **apply** *blast*
using *7.prems deg-deg-n nested-mint* **apply** *blast*
apply(*rule entails-solve-init*)
apply (*tactic ‹Seplogic-Auto.match-frame-tac (resolve-tac @{context} @{thms ent-refl}) @{context} 1›*)
apply *simp*
apply *solve-entails*
apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)
using *7.prems deg-deg-n nested-mint* **apply** *blast*
done
subgoal
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply *sep-auto*
apply(*rule entails-solve-init*)
apply (*tactic ‹Seplogic-Auto.match-frame-tac (resolve-tac @{context} @{thms ent-refl}) @{context} 1›*)

apply *simp*
apply *solve-entails*
apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)
apply (*simp add*: *ab-semigroup-mult-class.mult.commute minminNull*)
done
using *7.prems deg-deg-n nested-mint* **apply** *blast*
done
apply(*auto split*: *if-split*)
apply *sep-auto*
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n nested-mint*)
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*= *high* (*2* ∗ *2* ^(*va div 2*) ∗ *the* (*vebt-mint summary*) + *the* (*vebt-mint* (*treeList* ! *the* (*vebt-mint summary*)))) (*Suc* (*va div 2*))])
apply (*metis 7.prems highboundn ab-semigroup-mult-class.mult.commute atLeastLessThan-iff deg-deg-n div2-Suc-Suc leI length-list-update less-nat-zero-code power-Suc xndef*)
apply (*metis 7.prems ab-semigroup-mult-class.mult.commute deg-deg-n length-list-update nested-mint*)
apply(*sep-auto*)
apply(*sep-auto heap*: *vebt-maxti-h*)
apply *sep-auto*
apply (*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply (*metis minNullmin mulcomm option.simps*(*3*))
apply (*metis minNullmin mulcomm option.distinct*(*1*))
subgoal
apply *sep-auto*
apply (*simp add*: *ab-semigroup-mult-class.mult.commute*)
apply(*rule listI-assn-reinsert-upd*[**where** *x* = - ! -])
apply(*rule midextr*)
apply(*rule midextr*)
apply(*rule groupy*)
apply(*rule ent-refl*)
apply (*metis ab-semigroup-mult-class.mult.commute length-list-update*)
apply (*metis ab-semigroup-mult-class.mult.commute atLeastLessThan-iff le0*)
apply(*fr-rot 1*)
apply(*rule swappa*)
apply(*simp add*: *mulcomm*)
apply(*simp add*: *listI-assn-conv*)
apply(*rule ent-refl*)
done
using *7.prems deg-deg-n nested-mint*
apply *blast*
apply *sep-auto*
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply (*simp add*: *minNullmin mulcomm*)
apply (*simp add*: *ab-semigroup-mult-class.mult.commute minNullmin*)
apply *sep-auto*

apply *sep-auto*
using *7.prems deg-deg-n nested-mint* **apply** *blast*
apply(*rule swappa*)
apply(*simp add*: *mulcomm*)
apply(*rule ent-refl*)
done
apply *simp*
apply(*sep-auto heap*: *highi-h*)
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound leI linorder-neqE-nat*)
apply(*sep-auto heap*: *lowi-h*)
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound leI linorder-neqE-nat*)
apply *sep-auto*
apply (*rewrite* **in** <¤>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <¤>-<-> *listI-assn-extract*[**where** *i*= *high x* (*Suc* (*va div 2*))])
apply (*metis 7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc highbound leI le-neq-implies-less less-nat-zero-code*)
apply (*metis 7.prems antisym-conv2 deg-deg-n div2-Suc-Suc highbound not-le-imp-less*)
apply(*sep-auto heap*: *7.IH*(*1*))
apply (*metis 7.prems antisym-conv2 deg-deg-n div2-Suc-Suc highbound not-le-imp-less*)
apply(*rule setprop*)
apply (*metis 7.prems antisym-conv2 deg-deg-n div2-Suc-Suc highbound not-le-imp-less nth-mem*)
apply *sep-auto*
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound not-le-imp-less order.not-eq-order-implies-strict*)
apply(*sep-auto*)
apply(*sep-auto heap*: *vebt-minti-h*)
apply(*rule bind-rule*)
apply(*rule assert'-rule*)
apply (*meson mod-pure-star-dist mod-starE*)
apply(*rule cons-pre-rule*)
apply(*rule big-assn-simp'*[])
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound leI le-neq-implies-less*) **apply** *auto*[*1*]
apply(*cases vebt-mint* (*vebt-delete* (*treeList* ! *high x* (*Suc* (*va div 2*))) (*low x* (*Suc* (*va div 2*)))))
apply *simp*
subgoal
apply *sep-auto*
apply(*sep-auto heap*: *7.IH*(*2*))
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound leI le-neq-implies-less*)
apply *simp+*
apply(*simp add*: *minminNull*)
apply(*rule sumprop*)
apply(*rule bind-rule'*[**where** *R*=λ *r*.(*let sn* = *vebt-delete summary* (*high x* (*Suc* (*va div 2*))) *in*
(↑(*r* = (
if x = *ma then let maxs* = *vebt-maxt sn in if maxs* = *None then mi*
else 2 ^ (*Suc* (*Suc va*) *div 2*) ∗ *the maxs* + *the* (*vebt-maxt* (*?newlist'* ! *the maxs*)) *else ma*))))])
apply(*cases x* = *ma*)
apply *simp*
apply(*sep-auto*)
apply(*sep-auto heap*: *vebt-maxti-h*)

apply(*cases vebt-maxt* (*vebt-delete summary* (*high ma* (*Suc* (*va div 2*)))))
apply *simp*
apply *sep-auto*
apply *simp*
apply *sep-auto*
apply (*metis both-member-options-equiv-member dele-bmo-cont-corr listlength maxbmo member-bound sumprop*)
apply (*rewrite* **in** <⌑>-<-> *list-assn-conv-idx*)
apply (*rewrite* **in** <⌑>-<-> *listI-assn-extract*[**where** *i*=*the* (*vebt-maxt* (*vebt-delete summary* (*high ma* (*Suc* (*va div 2*)))))])
apply (*metis atLeastLessThan-iff both-member-options-equiv-member dele-bmo-cont-corr le0 length-list-update listlength maxbmo member-bound option.sel sumprop*)
apply (*metis both-member-options-equiv-member dele-bmo-cont-corr length-list-update listlength maxbmo member-bound option.sel sumprop*)
apply *sep-auto*
apply(*sep-auto heap*: *vebt-maxti-h*)
apply *sep-auto*
apply (*smt* (*verit*, *ccfv-SIG*) *ab-semigroup-mult-class.mult.commute ab-semigroup-mult-class.mult.left-commute atLeastLessThan-empty-iff2 atLeastLessThan-iff both-member-options-equiv-member deg-deg-n dele-bmo-cont-corr div2-Suc-Suc div-by-Suc-0 div-mult-self1-is-m div-mult-self-is-m empty-iff ent-refl le0 le-neq-implies-less length-list-update listI-assn-extract list-assn-conv-idx listlength maxbmo member-bound mimaxprop numeral-2-eq-2 option.collapse option.distinct*(*1*) *sumprop zero-less-Suc*)
apply *simp*
apply *sep-auto*
apply *sep-auto*
apply(*simp add*: *Let-def*)
apply(*cases high ma* (*Suc* (*va div 2*)) < *length treeList*)
apply *simp*
apply(*simp add*: *Let-def*)
apply(*cases high ma* (*Suc* (*va div 2*)) < *length treeList*)
apply *simp*
apply(*cases minNull* (*vebt-delete* (*treeList* ! *high ma* (*Suc* (*va div 2*))) (*low ma* (*Suc* (*va div 2*)))))
apply *simp*
apply(*simp add*: *Let-def*)
apply *sep-auto*
apply *simp*
apply *sep-auto*
apply (*meson minminNull*)+
apply *simp*
apply(*auto split*: *if-split*)
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound le-refl*)
apply *sep-auto*
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound le-refl*)
apply (*metis 7.prems deg-deg-n div2-Suc-Suc highbound le-refl*)
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)
apply(*simp add*: *Let-def*)
apply(*auto split*: *if-split*)

```
    apply sep-auto+
    apply (meson minminNull)
    apply sep-auto
    apply (metis 7.prems deg-deg-n div2-Suc-Suc highbound linorder-neqE-nat not-le-imp-less)
   done
  apply simp
  apply(auto split: if-split)
  apply sep-auto
  apply (metis 7.prems deg-deg-n div2-Suc-Suc dual-order.refl highbound)
  apply (rewrite in <⌑>-<-> list-assn-conv-idx)
  apply (rewrite in <⌑>-<-> listI-assn-extract[where i=high ma (Suc (va div 2))])
 apply (metis 7.prems atLeastLessThan-iff deg-deg-n div2-Suc-Suc highbound le0 le-refl length-list-update)
  apply (metis 7.prems deg-deg-n div2-Suc-Suc highbound le-refl length-list-update)
  apply sep-auto
  apply(sep-auto heap: vebt-maxti-h)
  apply sep-auto
  apply(simp add: Let-def)
  apply(auto split: if-split)
  apply(simp add: Let-def)
  apply(auto split: if-split)
  apply(simp add: Let-def)
  subgoal
    apply sep-auto
    apply (metis minNullmin option.distinct)+
   done
  apply sep-auto
  apply(rule ent-trans)
  apply(rule tcd[where treeList'= treeList])
  apply blast+
  apply(rule swappa)
  apply(rule ent-refl)
  apply sep-auto
  apply (metis 7.prems deg-deg-n div2-Suc-Suc highbound le-refl)
  apply (metis 7.prems deg-deg-n div2-Suc-Suc dual-order.refl highbound)
  apply sep-auto
  apply(simp add: Let-def)
  apply(auto split: if-split)
  apply(simp add: Let-def)
  apply(auto split: if-split)
  apply sep-auto
  apply (metis minNullmin option.distinct(1))
  apply sep-auto+
 done
qed

lemma TBOUND-vebt-deletei:
 defines foo-def: ⋀ t x. foo t x ≡ if minNull (vebt-delete t x) then 1 else 20 ∗ (1+height t)
 shows TBOUND (vebt-deletei' t ti x) (foo t x)
proof−
```

have *fooNull*:*minNull* (*vebt-delete t x*) $\Longrightarrow$ *foo t x = 1* **for** *t x* **using** *foo-def* **by** *simp*
have *fooElse*: *foo t x* $\leq$ *20*$*$ (*1*+ *height t*) **for** *t* **using** *foo-def* **by** *simp*
have *succ0foo*: *Suc 0* $\leq$ *foo t x* **for** *t x* **unfolding** *foo-def* **by** *simp*
have *fooNull'*: *vebt-mint* (*vebt-delete t x*) = *None* $\Longrightarrow$ *foo t x = 1* **for** *t x*
by (*simp add*: *fooNull minminNull*)
have *fooNull''*: *vebt-maxt* (*vebt-delete t x*) = *None* $\Longrightarrow$ *foo t x = 1* **for** *t x*
by (*metis fooNull fooNull' vebt-maxt.elims minNull.simps*(*4*) *vebt-mint.simps*(*1*) *option.simps*(*3*))
have *minNotMaxDel*: *x12a* $\geq$ *2* $\Longrightarrow$ *c* $\neq$ *d* $\Longrightarrow$
¬ *minNull* (*vebt-delete* (*Node* (*Some* (*c*, *d*)) *x12a x13 x14*) *y*) **for** *x12a x13 x14 c d y*
apply(*cases* ((*Node* (*Some* (*c*, *d*)) *x12a x13 x14*), *y*) *rule*: *vebt-delete.cases*; *simp*)
apply(*auto simp add*: *Let-def*)
done
have *twentyheight*: *i*< *length x13* $\Longrightarrow$*n* $*$ *height* (*x13* !*i*) $\leq$ *m* + *n* $*$ *max* (*height x14*) (*Max* (*height* ‘ *set x13*)) **for** *i x13 x14 n m*
by (*meson height-i-max mult-le-mono2 trans-le-add2*)
have *summheight*:*n* $*$ *height x14* $\leq$ *m* + *n* $*$ *max* (*height x14*) (*Max* (*height* ‘ *set x13*)) **for** *x14 x13 m n*
apply(*simp add*: *max-def*)
apply (*meson mult-le-mono2 trans-le-add2*)
done
show *?thesis*
apply (*induction arbitrary*: *t ti x rule*: *vebt-deletei'.fixp-induct*)
apply (*rule TBOUND-fi'-adm*)
apply (*rule TBOUND-empty*)
apply *TBOUND*
apply(*simp add*: *Let-def eq-commute*[*of Suc 0 -] succ0foo fooNull' fooNull''*
split!: *VEBT.splits VEBTi.splits option.splits prod.splits*)
apply(*all* ‹(*intro allI impI conjI*) *?*›)
apply(*all* ‹(*clarify*; *simp only*: *succ0foo*; *fail*) *?*›)
apply(*simp-all add*: *foo-def minNotMaxDel twentyheight summheight not-less*)
done
qed

lemma *vebt-deletei-refines*: *refines* (*vebt-deletei ti x*) (*vebt-deletei' t ti x*)
apply (*induction arbitrary*: *t ti x rule*: *vebt-deletei'.fixp-induct*)
subgoal
using *refines-adm*[**where** *t* = λ *arg*. *vebt-deletei* (*snd* (*fst arg*)) (*snd arg*)]
by *simp*
subgoal by *simp*
subgoal for *f t ti x*
apply(*subst vebt-deletei.simps*)
apply *refines*
done
done

lemma *htt-vebt-deletei*: **assumes** *invar-vebt t n*
shows <*vebt-assn-raw t ti*> *vebt-deletei ti x* <λ *r*. *vebt-assn-raw* (*vebt-delete t x*) *r* >*T*[*20* + *20*$*$(*nat* ⌈*lb n*⌉)]
apply (*rule htt-refine*[**where** *c* = *vebt-deletei' t ti x*])

prefer *2*
apply(*rule vebt-deletei-refines*)
apply (*rule httI-TBOUND*)
apply(*rule deleti'-rf-abstr*)
apply(*rule assms*)
apply(*rule TBOUND-mono*)
apply(*rule TBOUND-vebt-deletei*)
apply (*auto simp add*: *if-split*)
apply(*metis assms eq-imp-le heigt-uplog-rel int-eq-iff*)
done

end
end

22 Imperative Interface

theory *VEBT-Intf-Imperative*
imports
VEBT-Definitions
VEBT-Uniqueness
VEBT-Member
VEBT-Insert VEBT-InsertCorrectness
VEBT-MinMax
VEBT-Pred VEBT-Succ
VEBT-Delete VEBT-DeleteCorrectness
VEBT-Bounds
VEBT-DeleteBounds
VEBT-Space
VEBT-Intf-Functional
VEBT-List-Assn
VEBT-BuildupMemImp
VEBT-SuccPredImperative
VEBT-DelImperative
begin

22.1 Code Export

context begin
interpretation *VEBT-internal* **.**

lemmas [*code*] = *replicatei.simps vebt-memberi.simps highi-def lowi-def vebt-inserti.simps minNulli.simps vebt-succi.simps vebt-predi.simps vebt-deletei.simps greater.simps*

end

export-code
vebt-buildupi
vebt-memberi

vebt-inserti
vebt-maxti vebt-minti
vebt-predi vebt-succi
vebt-deletei

checking *SML-imp*

22.2 Interface

definition *vebt-assn*::*nat* ⇒ *nat set* ⇒ *VEBTi* ⇒ *assn* **where**
vebt-assn n s ti ≡ ∃$_A$ *t*. *vebt-assn-raw t ti* ∗ ↑(*s* = *set-vebt t* ∧ *invar-vebt t n*)

22.2.1 Buildup

context begin
interpretation *VEBT-internal* **.**

interpretation *vebt-inst* **for** *n* **.**

lemma *vebt-buildupi-rule-basic*[*sep-heap-rules*]: *n* > *0* ⟹ <*emp*> *vebt-buildupi n* <λ *r*. *vebt-assn n* {} *r* >
unfolding *vebt-assn-def*
apply(*rule post-exI-rule*[**where** *x* = *vebt-buildup n*])
using *builupicorr*[*of n*] *invar-vebt-buildup*[*of n*] *set-vebt-buildup*[*of n*]
apply *simp*
done

lemma *vebt-buildupi-rule*: <↑ (*n* > *0*)> *vebt-buildupi n* <λ *r*. *vebt-assn n* {} *r* > *T*[*10* ∗ *2^n*]
unfolding *vebt-assn-def htt-def*
apply *rule*
apply(*rule post-exI-rule*[**where** *x* = *vebt-buildup n*])
using *vebt-buildupi-rule*[*of n*] *invar-vebt-buildup*[*of n*] *set-vebt-buildup*[*of n*]
unfolding *htt-def*
apply *simp*
using *TBOUND-buildupi*[*of n*] **unfolding** *TBOUND-def*
apply *simp*
done

22.2.2 Member

lemma *vebt-memberi-rule*: <*vebt-assn n s ti*> *vebt-memberi ti x* <λ *r*. *vebt-assn n s ti* ∗ ↑(*r* = (*x* ∈ *s*))>*T*[*5* + *5* ∗ (*nat* ⌈*lb n* ⌉)]
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF htt-vebt-memberi-invar-vebt*])
apply *assumption*
apply *simp*
apply (*sep-auto simp*: *member-correct*)

apply *simp*
done

22.2.3 Insert

lemma *vebt-inserti-rule*: $x < 2\hat{}n \Longrightarrow <$*vebt-assn n s ti*$>$ *vebt-inserti ti x* $<\lambda$ *r. vebt-assn n* $(s \cup \{x\})$ *r* $>T[13 + 13 * (nat \lceil lb\ n \rceil)]$
apply(*sep-auto simp*: *norm-pre-pure-iff-htt*)
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF htt-vebt-inserti-invar-vebt*])
apply *assumption*
apply *simp*
apply *sep-auto*
apply (*auto simp add*: *insert-correct*)
apply (*simp add*: *valid-insert-both-member-options-add set-vebt-def*)
apply (*metis UnCI insert-correct*)
apply (*metis UnE insert-correct singletonD*)
using *valid-pres-insert* **by** *presburger*

22.2.4 Maximum

lemma *vebt-maxti-rule*: $<$*vebt-assn n s ti*$>$ *vebt-maxti ti* $<\lambda$ *r. vebt-assn n s ti* $*$ $\uparrow($ *r* $=$ *Some y* $\longleftrightarrow$ *max-in-set s y*$)>T[1]$
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF vebt-maxti-hT*])
apply(*rule ent-refl*)
apply (*sep-auto simp*: *set-vebt-maxt*)
by *simp*

22.2.5 Minimum

lemma *vebt-minti-rule*: $<$*vebt-assn n s ti*$>$ *vebt-minti ti* $<\lambda$ *r. vebt-assn n s ti* $*$ $\uparrow($ *r* $=$ *Some y* $\longleftrightarrow$ *min-in-set s y*$)>T[1]$
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF vebt-minti-hT*])
apply(*rule ent-refl*)
apply (*sep-auto simp*: *set-vebt-mint*)
by *auto*

22.2.6 Successor

lemma *vebt-succi-rule*: $<$*vebt-assn n s ti*$>$ *vebt-succi ti x* $<\lambda$ *r. vebt-assn n s ti* $*$ $\uparrow($ *r* $=$ *Some y* $\longleftrightarrow$ *is-succ-in-set s x y*$)>T[7 + 7 * (nat \lceil lb\ n \rceil)]$
unfolding *vebt-assn-def*

apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF htt-vebt-succi*])
apply *assumption*
apply *simp*
apply (*sep-auto simp*: *set-vebt-succ*)
apply *simp*
done

22.2.7 Predecessor

lemma *vebt-predi-rule*: <*vebt-assn n s ti*> *vebt-predi ti x* <λ *r*. *vebt-assn n s ti* ∗ ↑(*r* = *Some y* ⟷ *is-pred-in-set s x y*)>*T*[7 + 7 ∗ (*nat* ⌈*lb n* ⌉)]
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF htt-vebt-predi*])
apply *assumption*
apply *simp*
apply (*sep-auto simp*: *set-vebt-pred*)
apply *simp*
done

22.2.8 Delete

lemma *vebt-deletei-rule*: <*vebt-assn n s ti* > *vebt-deletei ti x* <λ *r*. *vebt-assn n* (*s* − {*x*}) *r* >*T*[*20* + *20* ∗ (*nat* ⌈*lb n* ⌉)]
unfolding *vebt-assn-def*
apply(*rule norm-pre-ex-rule-htt*)
apply(*clarsimp simp*: *norm-pre-pure-iff-htt*)
apply(*rule htt-cons-rule*[*OF htt-vebt-deletei*])
apply *assumption*
apply *simp*
apply *sep-auto*
apply (*auto simp add*: *set-vebt-delete invar-vebt-delete*)
done

22.3 Setup of VCG

lemmas *vebt-heap-rules*[*THEN htt-htD*,*sep-heap-rules*] =
vebt-buildupi-rule
vebt-memberi-rule
vebt-inserti-rule
vebt-maxti-rule
vebt-minti-rule
vebt-succi-rule
vebt-predi-rule
vebt-deletei-rule

end

end

23 Interface Usage Example

theory *VEBT-Example*
imports *VEBT-Intf-Imperative VEBT-Example-Setup*
begin

23.1 Test Program

definition *test n xs ys* ≡ *do* {
  t ← *vebt-buildupi n*;
  t ← *mfold* (λ*x s*. *vebt-inserti s x*) (*0*#*xs*) *t*;

  let f = (λ*x*. *if*$_m$ *vebt-memberi t x then return x else the* $_m$ (*vebt-predi t x*));

  mmap f ys
}

23.2 Correctness without Time

The non-time part of our datastructure is fully integrated into sep-auto

lemma *fold-list-rl*[*sep-heap-rules*]: ∀ *x*∈*set xs*. *x*<*2*^*n* ⟹ *hoare-triple*
  (*vebt-assn n s t*)
  (*mfold* (λ*x s*. *vebt-inserti s x*) *xs t*)
  (λ*t*′. *vebt-assn n* (*s* ∪ *set xs*) *t*′)
proof (*induction xs arbitrary*: *s t*)
  case *Nil*
  then show *?case* **by** *sep-auto*
next
  case (*Cons a xs*)

  note *Cons.IH*[*sep-heap-rules*]

  show *?case* **using** *Cons.prems*
    by *sep-auto*

qed

lemma *test-hoare*: ⟦∀ *x*∈*set xs*. *x*<*2*^*n*; *n*>*0*⟧ ⟹
    <*emp*> (*test n xs ys*) <λ*r*. ↑(*r* = *map* (λ*y*. (*GREATEST y*′. *y*′∈*insert 0* (*set xs*) ∧ *y*′≤*y*)) *ys*)
>$_t$
  unfolding *test-def*
    supply *R* = *mmap-pure-aux*[**where** *f*=(λ*y*. (*GREATEST y*′. *y*′∈*insert 0* (*set xs*) ∧ *y*′≤*y*))]
    apply (*sep-auto decon*: *R*)
    subgoal
      by (*metis* (*mono-tags*, *lifting*) *GreatestI-ex-nat zero-le-numeral*)

```
    subgoal
      by (metis (no-types, lifting) Greatest-equality le-eq-less-or-eq)
    subgoal for x xa xb
      apply sep-auto
      subgoal
        apply (auto simp: is-pred-in-set-def)
        subgoal
          by (metis (mono-tags, lifting) GreatestI-nat le-eq-less-or-eq)
        subgoal
          by (metis (mono-tags, lifting) GreatestI-ex-nat bot-nat-0.not-eq-extremum le-eq-less-or-eq)
        subgoal
          by (metis (no-types, lifting) Greatest-le-nat less-imp-le)
        done
      done
    subgoal
      by sep-auto
  done
```

23.3 Time Bound Reasoning

We use some ad-hoc reasoning to also show the time-bound of our test program. A generalization of such methods, or the integration of this entry into existing reasoning frameworks with time is left to future work.

```
lemma insert-time-pure[cond-TBOUND]:a < 2^n ⟹
  §vebt-assn n S ti§ TBOUND (vebt-inserti ti a) (13 + 13 * nat ⌈log 2 (real n)⌉)
  by(rule htt-elim, rule vebt-inserti-rule, simp)

lemma member-time-pure[cond-TBOUND]:§vebt-assn n S ti§ TBOUND (vebt-memberi ti a) (5 + 5
* nat ⌈log 2 (real n)⌉)
  by(rule htt-elim, rule  vebt-memberi-rule)

lemma pred-time-pure[cond-TBOUND]:§vebt-assn n S ti§ TBOUND (vebt-predi ti a) (7 + 7 * nat
⌈log 2 (real n)⌉)
   by(rule htt-elim, rule  vebt-predi-rule)

lemma TBOUND-mfold[cond-TBOUND]:
  (⋀ x. x ∈ set xs ⟹ x < 2^n) ⟹
        § vebt-assn n S ti § TBOUND (mfold (λx s. vebt-inserti s x) xs ti)  (length xs  * (13 + 13 *
nat ⌈log 2 n ⌉) + 1)
  apply(induction xs arbitrary: ti S)
   apply(subst mfold.simps)
   apply(cond-TBOUND, simp)
  apply sep-auto
  subgoal for a xs ti S
    apply(rule cond-TBOUND-mono[where b = (13 + 13 * nat ⌈log 2 (real n)⌉) + (length xs * (13
+ 13 * nat ⌈log 2 (real n)⌉) + 1)])
    apply(rule cond-TBOUND, auto|(rule vebt-heap-rules(3), auto))+
    done
  done
```

lemma *TBOUND-mmap*[*cond-TBOUND*]:
defines *b-def*: *b ys n* ≡ *1* + *length ys* ∗ (*5* + *5* ∗ *nat* ⌈*log 2* (*real n*)⌉ + *9* + *7* ∗ *nat* ⌈*log 2* (*real n*)⌉)
shows § *vebt-assn n S ti* § *TBOUND*
(*mmap* (λ*x*. *if*$_m$ *vebt-memberi ti x then return x*
else vebt-predi ti x ⤜ (λ*x*. *return* (*the x*))) *ys*) (*b ys n*)
apply(*induction ys arbitrary*:)
apply(*subst mmap.simps*)
subgoal
unfolding *b-def*
apply(*rule cond-TBOUND-mono*[**where** *b* = *1*], *rule cond-TBOUND-return*, *simp*)
done
apply *sep-auto*
subgoal for *a ys*
apply(*rule cond-TBOUND-mono*[
where *b* = ((*5* + *5* ∗ *nat* ⌈*log 2* (*real n*)⌉) + *max* *1* ((*7* + *7* ∗ *nat* ⌈*log 2* (*real n*)⌉) + *1*))
+(*b ys n* + *1*)])
apply(*rule cond-TBOUND-bind*[**where** *Q* = λ *r*. *vebt-assn n S ti*])
apply(*rule cond-TBOUND* | *rule mmap-pres* | *sep-auto* | *rule cond-TBOUND-cons*)+
unfolding *b-def*
apply *simp*
done
done

lemma *TBOUND-test*[*cond-TBOUND*]: ⟦∀ *x*∈*set xs*. *x*<*2*^*n*; *n*>*0* ⟧ ⟹
§ ↑ (*n*> *0*) § *TBOUND* (*test n xs ys*) (*10* ∗ *2*^*n* + (
(*length* (*0*#*xs*) ∗ (*13* + *13* ∗ *nat* ⌈*log 2 n* ⌉) + *1*) +
(*1* + *length ys* ∗ (*5* + *5* ∗ *nat* ⌈*log 2* (*real n*)⌉ + *9* + *7* ∗ *nat* ⌈*log 2* (*real n*)⌉)))))
unfolding *test-def*
apply(*cond-TBOUND*| *rule htt-elim*[*OF vebt-buildupi-rule*] | *sep-auto*)+
done

lemma *test-hoare-with-time*: ⟦∀ *x*∈*set xs*. *x*<*2*^*n*; *n*>*0*⟧ ⟹
<*emp*> (*test n xs ys*) <λ*r*. ↑(*r* = *map* (λ*y*. (*GREATEST y*′. *y*′∈*insert 0* (*set xs*) ∧ *y*′≤*y*)) *ys*) ∗ *true* >
T[*10* ∗ *2* ^ *n* +
(*length* (*0* # *xs*) ∗ (*13* + *13* ∗ *nat* ⌈*log 2* (*real n*)⌉) + *1* +
(*1* + *length ys* ∗ (*5* + *5* ∗ *nat* ⌈*log 2* (*real n*)⌉ + *9* + *7* ∗ *nat* ⌈*log 2* (*real n*)⌉)))]
apply(*rule htt-intro*, *rule test-hoare*, *simp*+)
apply(*rule cond-TBOUND-mono*, *rule cond-TBOUND-cons*)
defer
apply(*rule TBOUND-test*, *simp*+)
done

end

24 Conclusion

We have formalized van Emde Boas trees in Isabelle, proving correct a functional and an imperative version, together with space and run-time bounds. This work amends a list [4] of formally verified CLRS algorithms [3].

Closing we sketch some enhancements of van Emde Boas trees in Isabelle. An examination of the data structure points out that there is probably a *join* operation with the semantics *set-vebt* (*vebt-join* s t) = *set-vebt* $s \cup$ *set-vebt* t. We make the restriction of joining only valid trees with equal degree numbers. Obviously, the join of two leaves is trivial. If one tree is empty or singleton, a join is implemented by immediately returning the other tree or performing an insertion before. Otherwise, summary and subtrees are to be joined recursively and afterwards we have to determine minimum and maximum. Certainly, this last step can be complicated, because argument trees may also coincide on minima or maxima.

One may also consider the treatment of associated satellite data. Those are to be stored in ordinary subtrees, whereas the definition of summary trees does not have to be changed. We can transfer this to Isabelle by introducing another data type representing van Emde Boas trees. The adapted *naive-member* and *membermima* still refer to integer keys, but we add an auxiliary function *assoc* such that *assoc* t x y holds iff the key x is associated with the value y. A *both-member-options* is also defined and can be used for specifying a suitable validness invariant. We may show a conjecture like *both-member-options* t $x \longleftrightarrow \exists\, y.$ *assoc* t x y. Besides, valid trees enforce keys to be associated with at most one value. All canonical functions f are shifted to the new type and return a key-value pair (x, y) or the modified tree. Proofs for being x the desired successor etc. are obtained by reuse and adaptation of prior proofs. In addition, modified canonical functions f' may only return the associated values y. We show the proposition $\exists\, x.\ f\ t\ i = (x, y) \longleftrightarrow f'\ t\ i = y$. All writing operations require a reasoning regarding the proper (non-)modification of associations. The modified functions f' are to be exposed to a user later on.

Moreover, we did not consider lazy implementation. Currently, *vebt-buildup* n generates a full van Emde Boas tree of degree n. A *lazy implementation* would construct a subtree only if needed. From this just a constant amount of additional effort per recursive step will arise. Thereby, proven running time bounds of $\mathcal{O}(\log \log u)$ will be preserved. Beside this, a lazy implementation can also be obtained by exporting verified Isabelle code to Haskell, which heavily applies the lazy evaluation technique.

Obviously, a lazy implementation would drastically reduce memory usage. Each insertion allocates $\mathcal{O}(\log \log u)$ space and hence an implementation that does not store empty subtrees gives us memory consumption in $\mathcal{O}(n \cdot \log \log u)$ where n is the number of elements currently stored. Furthermore, one may replace ordinary arrays by *dynamic perfect hashing* [9] allowing treatment of elements in (amortized) constant time and linear space. Unfortunately, a linear memory consumption in $\mathcal{O}(n)$ is achieved at cost of some worst case runtime bounds [10]. By this, $\mathcal{O}(\log \log u)$ is turned to an amortized bound for *vebt-insert* and *vebt-delete*, since the complexity of those functions is indeed affected by the amortization. An implementation of this van Emde Boas tree variant requires verified dynamic perfect hashing and amortization in Isabelle to build on.

We used Imperative HOL due to its support of arrays and type reflexive references that are

necessary for setting up a recursive tree data structure. For generating verified code, however, there also exist other frameworks, e.g. Isabelle LLVM [11] [12]. It supports refinement-based verification of correctness and worst-case time complexities. Additionally, verified programs can be exported to LLVM code, which itself is compiled to executable machine code. Strikingly, code of the introsort algorithm generated by this formalization stayed competitive with the GNU C++ library [12].

References

[1] P. van Emde Boas. "Preserving order in a forest in less than logarithmic time". In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. 1975, pp. 75–84. DOI: 10.1109/SFCS.1975.26.

[2] P. E. Boas, R. Kaas, and E. Zijlstra. "Design and implementation of an efficient priority queue". In: *Mathematical Systems Theory* 10.1 (1976), pp. 99–127. DOI: 10.1007/bf01683268.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844.

[4] T. Nipkow, M. Eberl, and M. P. L. Haslbeck. "Verified Textbook Algorithms. A Biased Survey". In: *ATVA 2020, Automated Technology for Verification and Analysis*. Ed. by D. V. Hung and O. Sokolsky. Vol. 12302. LNCS. Invited paper. Springer, 2020, pp. 25–53.

[5] Tobias Nipkow, Jasmin Blanchette, Manuel Eberl, Alejandro Gómez-Londoño, Peter Lammich, Christian Sternagel, Simon Wimmer, Bohua Zhan. *Functional Algorithms, Verified!* https://functional-algorithms-verified.org/. 2021.

[6] P. Lammich and R. Meis. "A Separation Logic Framework for Imperative HOL". In: *Archive of Formal Proofs* (Nov. 2012). https://isa-afp.org/entries/Separation_Logic_Imperative_HOL.html, Formal proof development. ISSN: 2150-914x.

[7] P. Lammich. "Refinement to Imperative HOL". In: *Journal of Automated Reasoning* 62 (Apr. 2019). DOI: 10.1007/s10817-017-9437-1.

[8] B. Zhan and M. P. L. Haslbeck. *Verifying Asymptotic Time Complexity of Imperative Programs in Isabelle*. 2018. arXiv: 1802.01336 [cs.LO].

[9] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. Tarjan. "Dynamic Perfect Hashing: Upper and Lower Bounds". In: vol. 0. Jan. 1988, pp. 524–531. DOI: 10.1109/SFCS.1988.21968.

[10] M. Straka. "Functional Data Structures and Algorithms". https://ufal.mff.cuni.cz/~straka/theses/doctoral-functional_data_structures_and_algorithms.pdf. PhD thesis. Charles University in Prague, Faculty of Mathematics and Physics, 2013.

[11] P. Lammich. "Generating Veried LLVM from Isabelle/HOL". English. In: *ITP 2019: Interactive Theorem Proving*. Interactive Theorem Proving, ITP 2019 ; Conference date: 08-09-2019 Through 13-09-2019. June 2019.

[12] M. P. L. Haslbeck and P. Lammich. "For a Few Dollars More". In: *Programming Languages and Systems*. Ed. by N. Yoshida. Cham: Springer International Publishing, 2021, pp. 292–319. ISBN: 978-3-030-72019-3.