The Median Method

Emin Karayel

July 1, 2022

Abstract

The median method is an amplification result for randomized approximation algorithms described in [1]. Given an algorithm whose result is in a desired interval with a probability larger than $\frac{1}{2}$, it is possible to improve the success probability, by running the algorithm multiple times independently and using the median. In contrast to using the mean, the amplification of the success probability grows exponentially with the number of independent runs.

This entry contains a formalization of the underlying theorem: Given a sequence of n independent random variables, which are in a desired interval with a probability $\frac{1}{2} + \alpha$. Then their median will be in the desired interval with a probability of $1 - \exp(-2\alpha^2 n)$. In particular, the success probability approaches 1 exponentially with the number of variables.

In addition to that, this entry also contains a proof that order-statistics of Borel-measurable random variables are themselves measurable and that generalized intervals in linearly ordered Borel-spaces are measurable.

Contents

1 Intervals are Borel measurable

theory *Median*
  imports *HOL−Probability.Hoeffding HOL−Library.Multiset*
begin

This section contains a proof that intervals are Borel measurable, where an interval is defined as a convex subset of linearly ordered space, more precisely, a set is an interval, if for each triple of points $x < y < z$: If x and z are in the set so is y. This includes ordinary intervals like $\{a..b\}$, $\{a<..<b\}$ but also for example $\{x::rat.\ x * x < (2::rat)\}$ which cannot be expressed in the standard notation.

In the *HOL−Analysis.Borel-Space* there are proofs for the measurability of each specific type of interval, but those unfortunately do not help if we want to express the result about the median bound for arbitrary types of intervals.

```
definition interval :: ('a :: linorder) set ⇒ bool where
  interval I = (∀ x y z. x ∈ I ⟶ z ∈ I ⟶ x ≤ y ⟶ y ≤ z ⟶ y ∈ I)

definition up-ray :: ('a :: linorder) set ⇒ bool where
  up-ray I = (∀ x y. x ∈ I ⟶ x ≤ y ⟶ y ∈ I)

lemma up-ray-borel:
  assumes up-ray (I :: (('a :: linorder-topology) set))
  shows I ∈ borel
proof (cases closed I)
  case True
  then show ?thesis using borel-closed by blast
next
  case False
  hence b:¬ closed I by blast

  have open I
  proof (rule Topological-Spaces.openI)
    fix x
    assume c:x ∈ I
    show ∃ T. open T ∧ x ∈ T ∧ T ⊆ I
    proof (cases ∃ y. y < x ∧ y ∈ I)
      case True
      then obtain y where a:y < x ∧ y ∈ I by blast
      have open {y<..} by simp
      moreover have x ∈ {y<..} using a by simp
      moreover have {y<..} ⊆ I
        apply (rule subsetI)
        using a assms(1) apply (simp add: up-ray-def)
        by (metis less-le-not-le)
      ultimately show ?thesis by blast
    next
      case False
      hence I ⊆ {x..} using linorder-not-less by auto
      moreover have {x..} ⊆ I
        using c assms(1)  apply (simp add: up-ray-def)
        by blast
      ultimately have I = {x..}
        by (rule order-antisym)
```

moreover have *closed* $\{x..\}$ **by** *simp*
ultimately have *False* **using** *b* **by** *auto*
then show *?thesis* **by** *simp*
qed
qed
then show *?thesis* **by** *simp*
qed

definition *down-ray* :: $('a :: linorder)\ set \Rightarrow bool$ **where**
down-ray $I = (\forall\, x\ y.\ y \in I \longrightarrow x \leq y \longrightarrow x \in I)$

lemma *down-ray-borel*:
assumes *down-ray* $(I :: (('a :: linorder\text{-}topology)\ set))$
shows $I \in$ *borel*
proof $-$
have *up-ray* $(-I)$ **using** *assms*
by (*simp add*: *up-ray-def down-ray-def*, *blast*)
hence $(-I) \in$ *borel* **using** *up-ray-borel* **by** *blast*
thus $I \in$ *borel*
by (*metis borel-comp double-complement*)
qed

Main result of this section:

lemma *interval-borel*:
assumes *interval* $(I :: (('a :: linorder\text{-}topology)\ set))$
shows $I \in$ *borel*
proof (*cases* $I = \{\}$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain *x* **where** $a{:}x \in I$ **by** *blast*
have $\bigwedge y\ z.\ y \in I \cup \{x..\} \Longrightarrow y \leq z \Longrightarrow z \in I \cup \{x..\}$
by (*metis assms a interval-def IntE UnE Un-Int-eq*(*1*) *Un-Int-eq*(*2*) *atLeast-iff nle-le order.trans*)
hence *up-ray* $(I \cup \{x..\})$
using *up-ray-def* **by** *blast*
hence $b{:}I \cup \{x..\} \in$ *borel*
using *up-ray-borel* **by** *blast*

have $\bigwedge y\ z.\ y \in I \cup \{..x\} \Longrightarrow z \leq y \Longrightarrow z \in I \cup \{..x\}$
by (*metis assms a interval-def UnE UnI1 UnI2 atMost-iff dual-order.trans linorder-le-cases*)
hence *down-ray* $(I \cup \{..x\})$
using *down-ray-def* **by** *blast*
hence $c{:}I \cup \{..x\} \in$ *borel*
using *down-ray-borel* **by** *blast*

have $I = (I \cup \{x..\}) \cap (I \cup \{..x\})$

using *a* **by** *fastforce*

then show *?thesis* **using** *b c*
by (*metis sets.Int*)
qed

2 Order statistics are Borel measurable

This section contains a proof that order statistics of Borel measurable random variables are themselves Borel measurable.

The proof relies on the existence of branch-free comparison-sort algorithms. Given a sequence length these algorithms perform compare-swap operations on predefined pairs of positions. In particular the result of a comparison does not affect future operations. An example for a branch-free comparison sort algorithm is shell-sort and also bubble-sort without the early exit.

The advantage of using such a comparison-sort algorithm is that it can be lifted to work on random variables, where the result of a comparison-swap operation on two random variables X and Y can be represented as the expressions $\lambda\omega.\ min\ (X\ \omega)\ (Y\ \omega)$ and $\lambda\omega.\ max\ (X\ \omega)\ (Y\ \omega)$.

Because taking the point-wise minimum (resp. maximum) of two random variables is still Borel measurable, and because the entire sorting operation can be represented using such compare-swap operations, we can show that all order statistics are Borel measuable.

fun *sort-primitive* **where**
sort-primitive i j f k = (*if k* = *i then min* (*f i*) (*f j*) *else* (*if k* = *j then max* (*f i*) (*f j*) *else f k*))

fun *sort-map* **where**
sort-map f n = *fold id* [*sort-primitive j i. i* <− [*0*..<*n*], *j* <− [*0*..<*i*]] *f*

lemma *sort-map-ind*:
sort-map f (*Suc n*) = *fold id* [*sort-primitive j n. j* <− [*0*..<*n*]] (*sort-map f n*)
by *simp*

lemma *sort-map-strict-mono*:
fixes f :: *nat* ⇒ *'b* :: *linorder*
shows $j < n \Longrightarrow i < j \Longrightarrow$ *sort-map f n i* ≤ *sort-map f n j*
proof (*induction n arbitrary*: *i j*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
define *g* **where** *g* = (*λk. fold id* [*sort-primitive j n. j* <− [*0*..<*k*]] (*sort-map f n*))
define *k* **where** *k* = *n*
have *a*:(∀ *i j. j* < *n* ⟶ *i* < *j* ⟶ *g k i* ≤ *g k j*) ∧ (∀ *l. l* < *k* ⟶ *g k l* ≤ *g k n*)

```
  proof (induction k)
    case 0
    then show ?case using Suc by (simp add:g-def del:sort-map.simps)
  next
    case (Suc k)
    have g (Suc k) = sort-primitive k n (g k)
      by (simp add:g-def)
    then show ?case using Suc
      apply (cases g k k ≤ g k n)
       apply (simp add:min-def max-def)
       using less-antisym apply blast
      apply (cases g k n ≤ g k k)
       apply (simp add:min-def max-def)
       apply (metis less-antisym max.coboundedI2 max.orderE)
      by simp
  qed

  hence ⋀i j. j < Suc n ⟹ i < j ⟹ g n i ≤ g n j
    apply (simp add:k-def) using less-antisym by blast
  moreover have sort-map f (Suc n) = g n
    by (simp add:sort-map-ind g-def del:sort-map.simps)
  ultimately show ?case
    apply (simp del:sort-map.simps)
    using Suc by blast
qed

lemma sort-map-mono:
  fixes f :: nat ⇒ 'b :: linorder
  shows j < n ⟹ i ≤ j ⟹ sort-map f n i ≤ sort-map f n j
  by (metis sort-map-strict-mono eq-iff le-imp-less-or-eq)

lemma sort-map-perm:
  fixes f :: nat ⇒ 'b :: linorder
  shows image-mset (sort-map f n) (mset [0..<n]) = image-mset f (mset [0..<n])
proof −
  define is-swap where is-swap = (λ(ts :: ((nat ⇒ 'b) ⇒ nat ⇒ 'b)). ∃ i < n. ∃ j
< n. ts = sort-primitive i j)
  define t :: ((nat ⇒ 'b) ⇒ nat ⇒ 'b) list
    where t = [sort-primitive j i. i <− [0..<n], j <− [0..<i]]

  have a: ⋀x f. is-swap x ⟹ image-mset (x f) (mset-set {0..<n}) = image-mset
f (mset-set {0..<n})
  proof −
    fix x
    fix f :: nat ⇒ 'b :: linorder
    assume is-swap x
    then obtain i j where x-def: x = sort-primitive i j and i-bound: i < n and
j-bound:j < n
      using is-swap-def by blast
```

define *inv* **where** $inv = mset\text{-}set\ \{k.\ k < n \wedge k \neq i \wedge k \neq j\}$
have $b{:}\{0..{<}n\} = \{k.\ k < n \wedge k \neq i \wedge k \neq j\} \cup \{i,j\}$
apply (*rule order-antisym*, *rule subsetI*, *simp*, *blast*, *rule subsetI*, *simp*)
using *i-bound j-bound* **by** *meson*
have $c{:}\bigwedge k.\ k \in\#\ inv \Longrightarrow (x\ f)\ k = f\ k$
by (*simp add*:*x-def inv-def*)
have $image\text{-}mset\ (x\ f)\ inv = image\text{-}mset\ f\ inv$
apply (*rule multiset-eqI*)
using *c multiset.map-cong0* **by** *force*
moreover have $image\text{-}mset\ (x\ f)\ (mset\text{-}set\ \{i,j\}) = image\text{-}mset\ f\ (mset\text{-}set\ \{i,j\})$
apply (*cases* $i = j$)
by (*simp add*:*x-def max-def min-def*)+
moreover have $mset\text{-}set\ \{0..{<}n\} = inv + mset\text{-}set\ \{i,j\}$
by (*simp only*:*inv-def b*, *rule mset-set-Union*, *simp*, *simp*, *simp*)
ultimately show $image\text{-}mset\ (x\ f)\ (mset\text{-}set\ \{0..{<}n\}) = image\text{-}mset\ f\ (mset\text{-}set\ \{0..{<}n\})$
by *simp*
qed

have $(\forall x \in set\ t.\ is\text{-}swap\ x) \Longrightarrow image\text{-}mset\ (fold\ id\ t\ f)\ (mset\ [0..{<}n]) = image\text{-}mset\ f\ (mset\ [0..{<}n])$
by (*induction t arbitrary*:*f*, *simp*, *simp add*:*a*)
moreover have $\bigwedge x.\ x \in set\ t \Longrightarrow is\text{-}swap\ x$
apply (*simp add*:*t-def is-swap-def*)
by (*meson atLeastLessThan-iff imageE less-imp-le less-le-trans*)
ultimately have $image\text{-}mset\ (fold\ id\ t\ f)\ (mset\ [0..{<}n]) = image\text{-}mset\ f\ (mset\ [0..{<}n])$ **by** *blast*
then show *?thesis* **by** (*simp add*:*t-def*)
qed

lemma *list-eq-iff*:
assumes $mset\ xs = mset\ ys$
assumes *sorted xs*
assumes *sorted ys*
shows $xs = ys$
using *assms properties-for-sort* **by** *blast*

lemma *sort-map-eq-sort*:
fixes $f :: nat \Rightarrow ('b :: linorder)$
shows $map\ (sort\text{-}map\ f\ n)\ [0..{<}n] = sort\ (map\ f\ [0..{<}n])$ (**is** $?A = ?B$)
proof −
have $mset\ ?A = mset\ ?B$
using *sort-map-perm*[**where** $f{=}f$ **and** $n{=}n$]
by (*simp del*:*sort-map.simps*)
moreover have *sorted ?B*
by *simp*
moreover have *sorted ?A*
apply (*subst sorted-wrt-iff-nth-less*)

apply (*simp del*:*sort-map.simps*)
by (*metis sort-map-mono nat-less-le*)
ultimately show *?A* = *?B*
using *list-eq-iff* **by** *blast*
qed

lemma *order-statistics-measurable-aux*:
fixes X :: *nat* ⇒ *'a* ⇒ (*'b* :: {*linorder-topology*, *second-countable-topology*})
assumes $n \geq 1$
assumes $j < n$
assumes $\bigwedge i.\ i < n \Longrightarrow X\ i \in$ *measurable M borel*
shows (λx. (*sort-map* ($\lambda i.\ X\ i\ x$) n) j) ∈ *measurable M borel*
proof −
have *n-ge-0*:$n > 0$ **using** *assms* **by** *simp*
define *is-swap* **where** *is-swap* = (λ(*ts* :: ((*nat* ⇒ *'b*) ⇒ *nat* ⇒ *'b*)). $\exists i < n.\ \exists j$ $< n$. *ts* = *sort-primitive i j*)
define *t* :: ((*nat* ⇒ *'b*) ⇒ *nat* ⇒ *'b*) *list*
where *t* = [*sort-primitive j i*. *i* <− [*0*..<*n*], *j* <− [*0*..<*i*]]

define *meas-ptw* :: (*nat* ⇒ *'a* ⇒ *'b*) ⇒ *bool*
where *meas-ptw* = (λf. ($\forall k.\ k < n \longrightarrow f\ k \in$ *borel-measurable M*))

have *ind-step*:
$\bigwedge x$ (*g* :: *nat* ⇒ *'a* ⇒ *'b*). *meas-ptw g* ⟹ *is-swap x* ⟹ *meas-ptw* ($\lambda k\ \omega.\ x\ (\lambda i.$ $g\ i\ \omega)\ k$)
proof −
fix *x g*
assume *meas-ptw g*
hence *a*:$\bigwedge k.\ k < n \Longrightarrow g\ k \in$ *borel-measurable M* **by** (*simp add*:*meas-ptw-def*)
assume *is-swap x*
then obtain *i j* **where** *x-def*:*x*=*sort-primitive i j* **and** *i-le*:$i < n$ **and** *j-le*:$j <$ n
by (*simp add*:*is-swap-def*, *blast*)
have $\bigwedge k.\ k < n \Longrightarrow (\lambda\omega.\ x\ (\lambda i.\ g\ i\ \omega)\ k) \in$ *borel-measurable M*
proof −
fix *k*
assume $k < n$
thus ($\lambda\omega.\ x\ (\lambda i.\ g\ i\ \omega)\ k$) ∈ *borel-measurable M*
apply (*simp add*:*x-def*)
apply (*cases* $k = i$, *simp*)
using *a i-le j-le borel-measurable-min* **apply** *blast*
apply (*cases* $k = j$, *simp*)
using *a i-le j-le borel-measurable-max* **apply** *blast*
using *a* **by** *simp*
qed
thus *meas-ptw* ($\lambda k\ \omega.\ x\ (\lambda i.\ g\ i\ \omega)\ k$)
by (*simp add*:*meas-ptw-def*)
qed

have $(\forall x \in set\ t.\ is\text{-}swap\ x) \Longrightarrow meas\text{-}ptw\ (\lambda\ k\ \omega.\ (fold\ id\ t\ (\lambda k.\ X\ k\ \omega))\ k)$
proof (*induction t rule*:*rev-induct*)
case *Nil*
then show *?case* **using** *assms* **by** (*simp add*:*meas-ptw-def*)
next
case (*snoc x xs*)
have *a*:$meas\text{-}ptw\ (\lambda k\ \omega.\ fold\ (\lambda a.\ a)\ xs\ (\lambda k.\ X\ k\ \omega)\ k)$ **using** *snoc* **by** *simp*
have *b*:*is-swap x* **using** *snoc* **by** *simp*
show *?case* **using** *ind-step*[*OF a b*] **by** *simp*
qed
moreover have $\bigwedge x.\ x \in set\ t \Longrightarrow is\text{-}swap\ x$
apply (*simp add*:*t-def is-swap-def*)
by (*meson atLeastLessThan-iff imageE less-imp-le less-le-trans*)
ultimately show *?thesis* **using** *assms*
by (*simp add*:*t-def*[*symmetric*] *meas-ptw-def*)
qed

Main results of this section:

lemma *order-statistics-measurable*:
fixes $X :: nat \Rightarrow 'a \Rightarrow ('b :: \{linorder\text{-}topology,\ second\text{-}countable\text{-}topology\})$
assumes $n \geq 1$
assumes $j < n$
assumes $\bigwedge i.\ i < n \Longrightarrow X\ i \in measurable\ M\ borel$
shows $(\lambda x.\ (sort\ (map\ (\lambda i.\ X\ i\ x)\ [0..<n]))\ !\ j) \in measurable\ M\ borel$
apply (*subst sort-map-eq-sort*[*symmetric*])
using *assms* **by** (*simp add*:*order-statistics-measurable-aux del*:*sort-map.simps*)

definition *median* **where**
$median\ n\ f = \ sort\ (map\ f\ [0..<n])\ !\ (n\ div\ 2)$

lemma *median-measurable*:
fixes $X :: nat \Rightarrow 'a \Rightarrow ('b :: \{linorder\text{-}topology,\ second\text{-}countable\text{-}topology\})$
assumes $n \geq 1$
assumes $\bigwedge i.\ i < n \Longrightarrow X\ i \in measurable\ M\ borel$
shows $(\lambda x.\ median\ n\ (\lambda i.\ X\ i\ x)) \in measurable\ M\ borel$
apply (*simp add*:*median-def*)
apply (*rule order-statistics-measurable*[*OF assms*(*1*) *- assms*(*2*)])
using *assms*(*1*) **by** *force+*

3 The Median Method

This section contains the proof for the probability that the median of independent random variables will be in an interval with high probability if the individual variables are in the same interval with probability larger than $\frac{1}{2}$. The proof starts with the elementary observation that the median of a seqeuence with n elements is in an interval I if at least half of them are in I. This works because after sorting the sequence the elements that will be in the interval must necessarily form a consecutive subsequence, if its length is larger than $\frac{n}{2}$ the median must be in it.

The remainder follows the proof in [1, §2.1] using the Hoeffding inequality to estimate the probability that at least half of the sequence elements will be in the interval I.

lemma *interval-rule*:
  assumes *interval I*
  assumes $a \leq x$ $x \leq b$
  assumes $a \in I$
  assumes $b \in I$
  shows $x \in I$
  using *assms(1)* **apply** (*simp add:interval-def*)
  using *assms* **by** *blast*

lemma *sorted-int*:
  assumes *interval I*
  assumes *sorted xs*
  assumes $k <$ *length xs* $i \leq j$ $j \leq k$
  assumes *xs* ! $i \in I$ *xs* ! $k \in I$
  shows *xs* ! $j \in I$
  apply (*rule interval-rule*[**where** *a=xs* ! *i* **and** *b=xs* ! *k*])
  using *assms* **by** (*simp add: sorted-nth-mono*)+

lemma *mid-in-interval*:
  assumes *2∗length* (*filter* (λ*x*. $x \in I$) *xs*) > *length xs*
  assumes *interval I*
  assumes *sorted xs*
  shows *xs* ! (*length xs div 2*) $\in I$
proof −
  have *length* (*filter* (λ*x*. $x \in I$) *xs*) > *0* **using** *assms(1)* **by** *linarith*
  then obtain *v* **where** *v-1*: *v* < *length xs* **and** *v-2*: *xs* ! $v \in I$
    by (*metis filter-False in-set-conv-nth length-greater-0-conv*)

  define *J* **where** *J* = {*k*. *k* < *length xs* ∧ *xs* ! $k \in I$}

  have *card-J-min*: *2∗card J* > *length xs*
    using *assms(1)* **by** (*simp add:J-def length-filter-conv-card*)

  consider
    (*a*) *xs* ! (*length xs div 2*) $\in I$ |
    (*b*) *xs* ! (*length xs div 2*) $\notin I$ ∧ *v* > (*length xs div 2*) |
    (*c*) *xs* ! (*length xs div 2*) $\notin I$ ∧ *v* < (*length xs div 2*)
    by (*metis linorder-cases v-2*)
  thus *?thesis*
  proof (*cases*)
    case *a*
    then show *?thesis* **by** *simp*
  next

case *b*
have *p*:⋀*k*. *k* ≤ *length xs div 2* ⟹ *xs* ! *k* ∉ *I*
using *b v-2 sorted-int*[*OF assms*(*2*) *assms*(*3*) *v-1*, **where** *j*=*length xs div 2*]
apply *simp* **by** *blast*
have *card J* ≤ *card* {*Suc* (*length xs div 2*)..<*length xs*}
apply (*rule card-mono*, *simp*)
apply (*rule subsetI*, *simp add*:*J-def not-less-eq-eq*[*symmetric*])
using *p* **by** *metis*
hence *card J* ≤ *length xs* − (*Suc* (*length xs div 2*))
using *card-atLeastLessThan* **by** *metis*
hence *length xs* ≤ *2*∗(*length xs* − (*Suc* (*length xs div 2*)))
using *card-J-min* **by** *linarith*
hence *False*
apply (*simp add*:*nat-distrib*)
apply (*subst* (*asm*) *le-diff-conv2*) **using** *b v-1* **apply** *linarith*
by *simp*
then show *?thesis* **by** *simp*
next
case *c*
have *p*:⋀*k*. *k* ≥ *length xs div 2* ⟹ *k* < *length xs* ⟹ *xs* ! *k* ∉ *I*
using *c v-1 v-2 sorted-int*[*OF assms*(*2*) *assms*(*3*), **where** *i* =*v* **and** *j*=*length xs div 2*] **apply** *simp* **by** *blast*
have *card J* ≤ *card* {*0*..<(*length xs div 2*)}
apply (*rule card-mono*, *simp*)
apply (*rule subsetI*, *simp add*:*J-def not-less-eq-eq*[*symmetric*])
using *p linorder-le-less-linear* **by** *blast*
hence *card J* ≤ (*length xs div 2*)
using *card-atLeastLessThan* **by** *simp*
then show *?thesis* **using** *card-J-min* **by** *linarith*
qed
qed

lemma *median-est*:
assumes *interval I*
assumes *2*∗*card* {*k*. *k* < *n* ∧ *f k* ∈ *I*} > *n*
shows *median n f* ∈ *I*
proof −
have *a*:{*k*. *k* < *n* ∧ *f k* ∈ *I*} = {*i*. *i* < *n* ∧ *map f* [*0*..<*n*] ! *i* ∈ *I*}
apply (*rule order-antisym*, *rule subsetI*, *simp*)
by (*rule subsetI*, *simp*, *metis add-0 diff-zero nth-map-upt*)

show *?thesis*
apply (*simp add*:*median-def*)
apply (*rule mid-in-interval*[**where** *I*=*I* **and** *xs*=*sort* (*map f* [*0*..<*n*]), *simplified*])
using *assms a* **apply** (*simp add*:*filter-sort comp-def length-filter-conv-card*)
by (*simp add*:*assms*)
qed

Main results of this section:

```
theorem (in prob-space) median-bound:
  fixes n :: nat
  fixes I :: ('b :: {linorder-topology, second-countable-topology}) set
  assumes interval I
  assumes α > 0
  assumes ε ∈ {0<..<1}
  assumes indep-vars (λ-. borel) X {0..<n}
  assumes n ≥ − ln ε / (2 ∗ α²)
  assumes ⋀i. i < n ⟹ 𝒫(ω in M. X i ω ∈ I) ≥ 1/2+α
  shows 𝒫(ω in M. median n (λi. X i ω) ∈ I) ≥ 1−ε
proof −
  define Y :: nat ⇒ 'a ⇒ real where Y = (λi. indicator I ∘ (X i))

  define t where t = (∑ i = 0..<n. expectation (Y i)) − n/2
  have 0 < −ln ε / (2 ∗ α²)
    apply (rule divide-pos-pos)
     apply (simp, subst ln-less-zero-iff)
    using assms by auto
  also have ... ≤ real n using assms by simp
  finally have real n > 0 by simp
  hence n-ge-1:n ≥ 1 by linarith
  hence n-ge-0:n > 0 by simp

  have ind-comp: ⋀i. indicator I ∘ (X i) = indicator {ω. X i ω ∈ I}
    by (rule ext, simp add:indicator-def comp-def)

  have α ∗ n ≤ (∑ i =0..<n. 1/2 + α) − n/2
    by (simp add:algebra-simps)
  also have ... ≤ (∑ i = 0..<n. expectation (Y i)) − n/2
    apply (rule diff-right-mono, rule sum-mono)
    using assms(6) by (simp add:Y-def ind-comp Collect-conj-eq inf-commute)
  also have ... = t by (simp add:t-def)
  finally have t-ge-a: t ≥ α ∗ n by simp

  have d: 0 ≤ α ∗ n
    apply (rule mult-nonneg-nonneg)
    using assms(2) n-ge-0 by simp+
  also have ... ≤ t using t-ge-a by simp
  finally have t-ge-0: t ≥ 0 by simp

  have (α ∗ n)² ≤ t² using t-ge-a d power-mono by blast
  hence t-ge-a-sq: α² ∗ real n ∗ real n ≤ t²
    by (simp add:algebra-simps power2-eq-square)

  have Y-indep: indep-vars (λ-. borel) Y {0..<n}
    apply (subst Y-def)
    apply (rule indep-vars-compose[where M'=(λ-. borel), OF assms(4)])
    using interval-borel[OF assms(1)] by simp
```

```
  hence b:Hoeffding-ineq M {0..<n} Y (λi. 0) (λi. 1)
   apply (simp add:Hoeffding-ineq-def indep-interval-bounded-random-variables-def)
   by (simp add:prob-space-axioms indep-interval-bounded-random-variables-axioms-def
Y-def Y-indep)

  have c: ⋀ω. (∑ i = 0..<n. Y i ω) > n/2 ⟹ median n (λi. X i ω) ∈ I
  proof −
    fix ω
    assume (∑ i = 0..<n. Y i ω) > n/2
    hence n < 2 ∗ card ({0..<n} ∩ {i. X i ω ∈ I})
      by (simp add:Y-def indicator-def)
    also have ... = 2 ∗ card {i. i < n ∧ X i ω ∈ I}
      apply (simp, rule arg-cong[where f=card])
      by (rule order-antisym, rule subsetI, simp, rule subsetI, simp)
    finally have 2 ∗ card {i. i < n ∧ X i ω ∈ I} > n by simp
    thus median n (λi. X i ω) ∈ I
      using median-est[OF assms(1)] by simp
  qed

  have 1 − ε ≤ 1− exp (− (2 ∗ α^2 ∗ real n))
    apply (simp, subst ln-ge-iff[symmetric])
    using assms(3) apply simp
    using assms(5) apply (subst (asm) pos-divide-le-eq)
     apply (simp add: assms(2) power2-eq-square)
    by (simp add: mult-of-nat-commute)
  also have ... ≤ 1− exp (− (2 ∗ t^2 / real n))
    apply simp
    apply (subst pos-le-divide-eq) using n-ge-0 apply simp
    using t-ge-a-sq by linarith
  also have ... ≤ 1 − 𝒫(ω in M. (∑ i = 0..<n. Y i ω) ≤ n/2)
     using Hoeffding-ineq.Hoeffding-ineq-le[OF b, where ε=t, simplified] n-ge-0
t-ge-0
    by (simp add:t-def)
  also have ... = 𝒫(ω in M. (∑ i = 0..<n. Y i ω) > n/2)
    apply (subst prob-compl[symmetric])
     apply measurable
     using Y-indep apply (simp add:indep-vars-def)
    apply (rule arg-cong2[where f=measure], simp)
   by (rule order-antisym, rule subsetI, simp add:not-le, rule subsetI, simp add:not-le)
  also have ... ≤ 𝒫(ω in M. median n (λi. X i ω) ∈ I)
    apply (rule finite-measure-mono)
     apply (rule subsetI) using c apply simp
    using interval-borel[OF assms(1)] apply measurable
    apply (rule median-measurable[OF n-ge-1])
    using assms(4) by (simp add:indep-vars-def)
  finally show ?thesis by simp
qed
```

This is a specialization of the above to closed real intervals.

corollary (**in** *prob-space*) *median-bound-1*:
  assumes $\alpha > 0$
  assumes $\varepsilon \in \{0<..<1\}$
  assumes *indep-vars* (λ-. *borel*) X $\{0..<n\}$
  assumes $n \geq - \ln \varepsilon \; / \; (2 * \alpha^2)$
  assumes $\forall i \in \{0..<n\}.\ \mathcal{P}(\omega\ in\ M.\ X\ i\ \omega \in (\{a..b\} :: real\ set)) \geq 1/2{+}\alpha$
  shows $\mathcal{P}(\omega\ in\ M.\ median\ n\ (\lambda i.\ X\ i\ \omega) \in \{a..b\}) \geq 1{-}\varepsilon$
  apply (*rule median-bound*[*OF* - *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)])
   apply (*simp add*:*interval-def*)
  using *assms*(*5*) **by** *auto*

This is a specialization of the above, where $\alpha = \frac{1}{6}$ and the interval is described using a mid point μ and radius δ. The choice of $\alpha = \frac{1}{6}$ implies a success probability per random variable of $\frac{2}{3}$. It is a commonly chosen success probability for Monte-Carlo algorithms (cf. [2, §4] or [3, §1]).

corollary (**in** *prob-space*) *median-bound-2*:
  fixes μ δ :: *real*
  assumes $\varepsilon \in \{0<..<1\}$
  assumes *indep-vars* (λ-. *borel*) X $\{0..<n\}$
  assumes $n \geq -18 * \ln \varepsilon$
  assumes $\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ in\ M.\ abs\ (X\ i\ \omega - \mu) > \delta) \leq 1/3$
  shows $\mathcal{P}(\omega\ in\ M.\ abs\ (median\ n\ (\lambda i.\ X\ i\ \omega) - \mu) \leq \delta) \geq 1{-}\varepsilon$
proof −
  have $b{:}\bigwedge i.\ i < n \Longrightarrow space\ M - \{\omega \in space\ M.\ X\ i\ \omega \in \{\mu - \delta..\mu + \delta\}\} = \{\omega \in space\ M.\ abs\ (X\ i\ \omega - \mu) > \delta\}$
    apply (*rule order-antisym*, *rule subsetI*, *simp*, *linarith*)
    by (*rule subsetI*, *simp*, *linarith*)

  have $\bigwedge i.\ i < n \Longrightarrow 1 - \mathcal{P}(\omega\ in\ M.\ X\ i\ \omega \in \{\mu{-}\ \delta..\mu{+}\delta\}) \leq 1/3$
    apply (*subst prob-compl*[*symmetric*])
     apply (*measurable*)
     using *assms*(*2*) **apply** (*simp add*:*indep-vars-def*)
    apply (*subst b*, *simp*)
    using *assms*(*4*) **by** *simp*

  hence $a{:}\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ in\ M.\ X\ i\ \omega \in \{\mu{-}\ \delta..\mu{+}\delta\}) \geq 2/3$ **by** *simp*

  have $1{-}\varepsilon \leq \mathcal{P}(\omega\ in\ M.\ median\ n\ (\lambda i.\ X\ i\ \omega) \in \{\mu{-}\delta..\mu{+}\delta\})$
    apply (*rule median-bound-1*[*OF* - *assms*(*1*) *assms*(*2*), **where** $\alpha{=}1/6$], *simp*)
     using *assms*(*3*) **apply** (*simp add*:*power2-eq-square*)
    using *a* **by** *simp*
  also have ... $= \mathcal{P}(\omega\ in\ M.\ abs\ (median\ n\ (\lambda i.\ X\ i\ \omega) - \mu) \leq \delta)$
    apply (*rule arg-cong2*[**where** *f*=*measure*], *simp*)
    apply (*rule order-antisym*, *rule subsetI*, *simp*, *linarith*)
    by (*rule subsetI*, *simp*, *linarith*)
  finally show *?thesis* **by** *simp*
qed

4 Some additional results about the median

lemma *sorted-mono-map*:
  assumes *sorted xs*
  assumes *mono f*
  shows *sorted* (*map f xs*)
  using *assms* **apply** (*simp add*:*sorted-wrt-map*)
  apply (*rule sorted-wrt-mono-rel*[**where** $P=(\leq)$])
  by (*simp add*:*mono-def*, *simp*)

This could be added to *HOL.List*:

lemma *map-sort*:
  assumes *mono f*
  shows *sort* (*map f xs*) = *map f* (*sort xs*)
  apply (*rule properties-for-sort*)
   apply *simp*
  by (*rule sorted-mono-map*, *simp*, *simp add*:*assms*)

lemma *median-cong*:
  assumes $\bigwedge i.\ i < n \Longrightarrow f\ i = g\ i$
  shows *median n f* = *median n g*
  apply (*cases* $n = 0$, *simp add*:*median-def*)
  apply (*simp add*:*median-def*)
  apply (*rule arg-cong2*[**where** *f*=(!)])
   apply (*rule arg-cong*[**where** *f*=*sort*], *rule map-cong*, *simp*, *simp add*:*assms*)
  by *simp*

lemma *median-restrict*:
  median n $(\lambda i \in \{0..<n\}.f\ i)$ = *median n f*
  by (*rule median-cong*, *simp*)

lemma *median-commute-mono*:
  assumes $n > 0$
  assumes *mono g*
  shows *g* (*median n f*) = *median n* $(g \circ f)$
  apply (*simp add*: *median-def del*:*map-map*)
  apply (*subst map-map*[*symmetric*])
  apply (*subst map-sort*[*OF assms*(*2*)])
  apply (*subst nth-map*, *simp*) **using** *assms* **apply** *fastforce*
  by *simp*

lemma *median-rat*:
  assumes $n > 0$
  shows *real-of-rat* (*median n f*) = *median n* $(\lambda i.$ *real-of-rat* (*f i*))
  apply (*subst* (*2*) *comp-def*[**where** *g*=*f*, *symmetric*])
  apply (*rule median-commute-mono*[*OF assms*(*1*)])
  by (*simp add*: *mono-def of-rat-less-eq*)

lemma *median-const*:

assumes $k > 0$
shows *median* k $(\lambda i \in \{0..<k\}.\ a) = a$
proof $-$
  have *b*: *sorted* (*map* (λ-. *a*) $[0..<k]$)
    by (*subst sorted-wrt-map*, *simp*)
  have *a*: *sort* (*map* (λ-. *a*) $[0..<k]$) = *map* (λ-. *a*) $[0..<k]$
    by (*subst sorted-sort-id*[*OF b*], *simp*)
  have *median* k $(\lambda i \in \{0..<k\}.\ a)$ = *median* k (λ-. *a*)
    by (*subst median-restrict*, *simp*)
  also have ... = *a*
    apply (*simp add*:*median-def a*)
    apply (*subst nth-map*)
    using *assms* **by** *simp*+
  finally show *?thesis* **by** *simp*
qed

end

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002.

[3] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, 2010.