Fast selection algorithms

Milano Christmas Workshop

Zahari Kassabov

December 20, 2016

The problem

Given a list of elements $\{l_1 ... l_n\}$from some totally ordered set (can define $l_i \leq l_j$) , find the element at a given position.

Usual formulation

Given the list $l^{l} = \{l^{l}_{1} ... l^{l}_{n}\}$ and the index i:

Return $l = \{l_1 \dots l_n\}$, a permutation of l^{l}, such that:

- l_i is the ith element.
- $l_j \geq l_i \quad \forall j > i$
- $l_j \leq l_i \quad \forall j > i$

“Partition l around i“.

Usual formulation

Given the list $l' = \{l'_1 ... l'_n\}$ and the index i:

Return $l = \{l_1 \dots l_n\}$, a permutation of l', such that:

- l_i is the ith element.
- $l_j \geq l_i \quad \forall j > i$
- $l_j \leq l_i \quad \forall j > i$

"Partition l around i".

We want to do it fast!

Applications

Partial sort Give me my ten latest friends on Facebook. (Select + sort).

Descriptive/robust statistics 3rd quartile of this data?

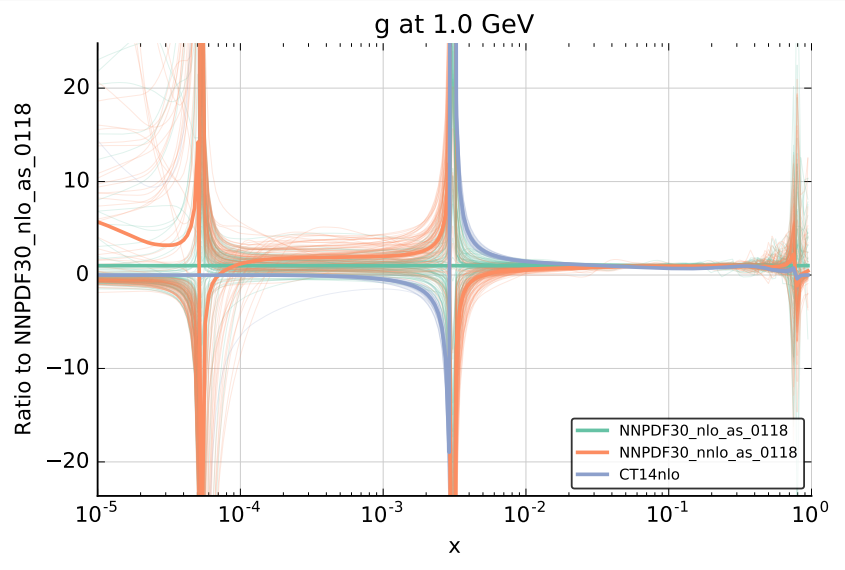
Applications

Partial sort Give me my ten latest friends on Facebook. (Select + sort).

Descriptive/robust statistics 3rd quartile of this data?

Machine learning Quantile regression, Quantile clustering.

Visualization Make sure 90 percent of the points are in the plot.



Problem where 1% optimizations are well worth it.

Obvious solution (baseline)

Sort the array, return ith element.

- Worst case and average complexity $n \log n$
- Usually a large constant in front.

Obvious solution (baseline)

Sort the array, return ith element.

- Worst case and average complexity $n \log n$
- Usually a large constant in front.

But problem can be solved in $\mathcal{O}(n)$.

Usual solution: QUICKSELECT

Given a subroutine

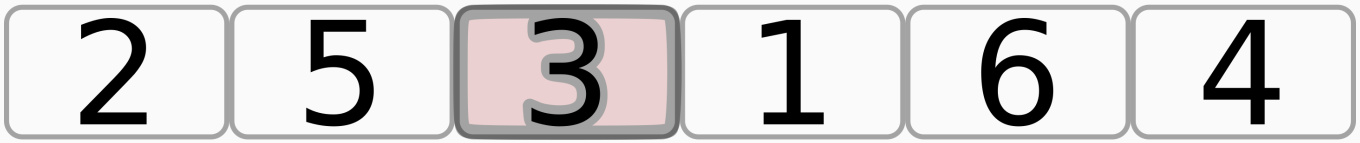
```
PARTITION
INPUTS: (ARRAY a), (INDEX i)
EFFECT: "a partitioned around some index p"
RETURN INDEX p
{
   ...
}
```

We define:

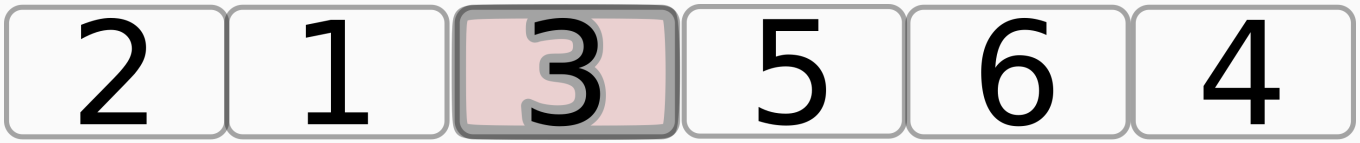
```
QUICKSELECT
INPUTS: (ARRAY a), (INDEX i)
EFFECT: "a partitioned around i"
{
   WHILE TRUE
       pivot := PARTITION(a,i)
       IF pivot = i RETURN
       IF pivot > i
          a = a[:pivot]
       ELSE
          i := i-pivot-1
          a := a[pivot+1:]


}
```

Example *i*=4

Example *i*=4



Example i=4



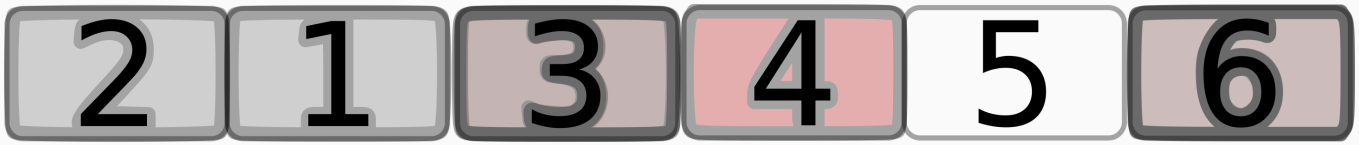
Example *i*=4

Example *i*=4

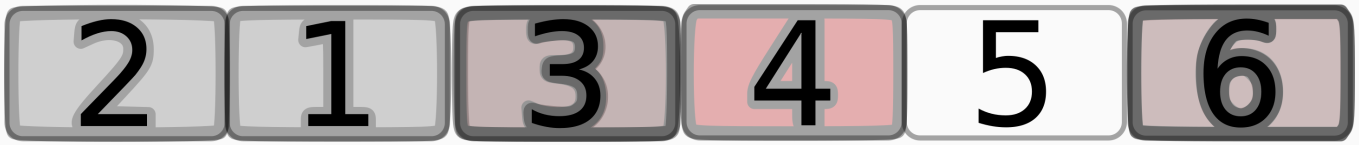
Example *i*=4

Example *i*=4

Example *i*=4



Example *i*=4



Done!

Selecting the pivot

Selecting the pivot (the function PARTITION) has a crucial impact on the performance of Quickselect.

Selecting the pivot

Selecting the pivot (the function PARTITION) has a crucial impact on the performance of Quickselect.

- Random pivot $\mathcal{O}(n)$ on average.

Selecting the pivot

Selecting the pivot (the function PARTITION) has a crucial impact on the performance of Quickselect.

- Random pivot $\mathcal{O}(n)$ on average.
- Bad pivot $\mathcal{O}(n^2)$.

Selecting the pivot

Selecting the pivot (the function PARTITION) has a crucial impact on the performance of Quickselect.

- Random pivot $\mathcal{O}(n)$ on average.
- Bad pivot $\mathcal{O}(n^2)$.
  - It's *easy* to have systesplmatically bad pivots for certain kinds of data.

Selecting the pivot

Selecting the pivot (the function PARTITION) has a crucial impact on the performance of Quickselect.

- Random pivot $\mathcal{O}(n)$ on average.
  - Sometimes we want to be deterministic.
- Bad pivot $\mathcal{O}(n^2)$.
  - It's *easy* to have systesplmatically bad pivots for certain kinds of data.

Background on pivot selection

We use the Hoare Partition as building block:

```
PARTITION
INPUTS: (ARRAY a), (ELEMENR a[i])
EFFECT: "a partitioned around a[i] which is now a[p]"
RETURN INDEX p
{
   ...
}
```

Background on pivot selection

We use the Hoare Partition as building block:

```
PARTITION
INPUTS: (ARRAY a), (ELEMENR a[i])
EFFECT: "a partitioned around a[i] which is now a[p]"
RETURN INDEX p
{
   ...
}
```

- Hoare Partition runs in $\mathcal{O}(n)$.
- QuickSelect runs in $\mathcal{O}(n)$ if:
  - PARTITION runs in $\mathcal{O}(n)$.
  - Each partition eliminates at least a fixed fraction f of the array.

Background on pivot selection

We use the Hoare Partition as building block:

```
PARTITION
INPUTS: (ARRAY a), (ELEMENR a[i])
EFFECT: "a partitioned around a[i] which is now a[p]"
RETURN INDEX p
{
   ...
}
```

- Hoare Partition runs in $\mathcal{O}(n)$.
- QuickSelect runs in $\mathcal{O}(n)$ if:
  - PARTITION runs in $\mathcal{O}(n)$.
  - Each partition eliminates at least a fixed fraction f of the array.

$$T(n) \leq T(n(1-f)) + kn$$

Background on pivot selection

We use the Hoare Partition as building block:

```
PARTITION
INPUTS: (ARRAY a), (ELEMENR a[i])
EFFECT: "a partitioned around a[i] which is now a[p]"
RETURN INDEX p
{
   ...
}
```

- Hoare Partition runs in $\mathcal{O}(n)$.
- QuickSelect runs in $\mathcal{O}(n)$ if:
  - PARTITION runs in $\mathcal{O}(n)$.
  - Each partition eliminates at least a fixed fraction f of the array.

$$wn \leq wn(1-f) + kn$$

Background on pivot selection

We use the Hoare Partition as building block:

```
PARTITION
INPUTS: (ARRAY a), (ELEMENR a[i])
EFFECT: "a partitioned around a[i] which is now a[p]"
RETURN INDEX p
{
   ...
}
```

- Hoare Partition runs in $\mathcal{O}(n)$.
- QuickSelect runs in $\mathcal{O}(n)$ if:
  - PARTITION runs in $\mathcal{O}(n)$.
  - Each partition eliminates at least a fixed fraction f of the array.

$$w \leq \frac{k}{f}$$

State of art and history

- Quickslect (Hoare, 1961).

State of art and history

- Quickslect (Hoare, 1961).
- Median of medians, BFRS (Blum et al, 1971) Guaranteed linearity, but slow in practice.

State of art and history

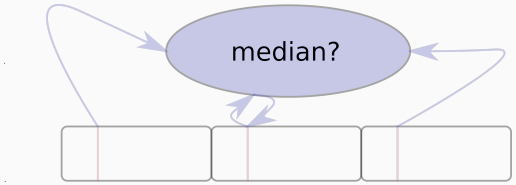
- Quickslect (Hoare, 1961).
- Median of medians, BFRS (Blum et al, 1971) Guaranteed linearity, but slow in practice.
- Various random implementations e.g. “median of 3 randomized”.

State of art and history

- Quickslect (Hoare, 1961).
- Median of medians, BFRS (Blum et al, 1971) Guaranteed linearity, but slow in practice.
- Various random implementations e.g. “median of 3 randomized”.
- QuickSelectAdaptive (Alexandrescu, 2016) Claims to be the fastest.

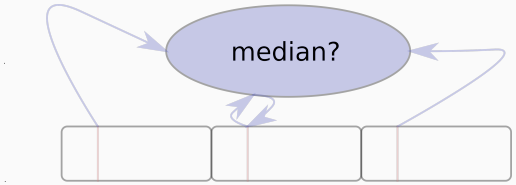
QuickSelectAdaptive (pivot selection for median)

1. Divide the list in three thirds.
2. Take on element from each third and put the median in the center third.



QuickSelectAdaptive (pivot selection for median)

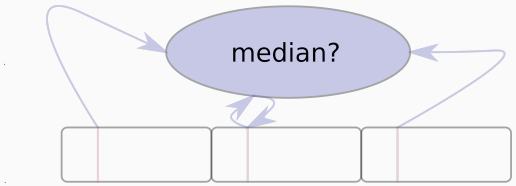
1. Divide the list in three thirds.
2. Take on element from each third and put the median in the center third.



3. Repeat steps 1 and 2 for the center third.

QuickSelectAdaptive (pivot selection for median)

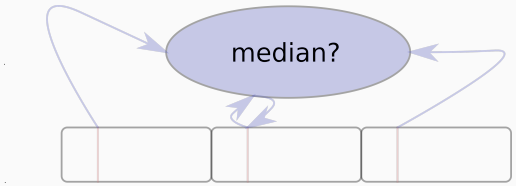
1. Divide the list in three thirds.
2. Take on element from each third and put the median in the center third.



3. Repeat steps 1 and 2 for the center third.
4. Call QuickSelect on the centermost ninth and for median.
5. Partition the whole array around the center median.

QuickSelectAdaptive (pivot selection for median)

1. Divide the list in three thirds.
2. Take on element from each third and put the median in the center third.



3. Repeat steps 1 and 2 for the center third.
4. Call QuickSelect on the centermost ninth and for median.
5. Partition the whole array around the center median.

We eliminate at least 2/9 of the array.

“Adaptions” for off-median

- Call inner quickselect for the same quantile as the one we are trying to find.

“Adaptions” for off-median

- Call inner quickselect for the same quantile as the one we are trying to find.
- Different group divisions:
  - Original was median of 3, median of 3.
  - Lower median of 4, median of 3 (for $i/n < 7/16$).
  - Lower median of 4, minimum of 3. (for $i/n < 1/12$).
  - (converse for bigger than median)

Optimization

Competitive performances are achieved by removing the first partition in 3. Just take the center.

Fall back to the full algorithm in a bad pivot is found.

Now we eliminate 1/9 of the array for median in the worst case.

QuickSelectAdaptive as starting point

- Deterministic approach, similar to “Median of medians”.

QuickSelectAdaptive as starting point

- Deterministic approach, similar to “Median of medians”.
- Improves the layout, and the speed of the pivot selection.

QuickSelectAdaptive as starting point

- Deterministic approach, similar to “Median of medians”.
- Improves the layout, and the speed of the pivot selection.
- Can provide guaranteed linearity.

QuickSelectAdaptive as starting point

- Deterministic approach, similar to “Median of medians”.
- Improves the layout, and the speed of the pivot selection.
- Can provide guaranteed linearity.
- Lots of special cases.

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?
- Sacrifice worst case guarantees in favor of average case run time + a fallback?

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?
- Sacrifice worst case guarantees in favor of average case run time + a fallback?
  - We want to eliminate $\max(i, n - i)$ elements at each pass.

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?
- Sacrifice worst case guarantees in favor of average case run time + a fallback?
  - We want to eliminate $\max(i, n - i)$ elements at each pass.
- Can we have a simpler algorithm?

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?
- Sacrifice worst case guarantees in favor of average case run time + a fallback?
  - We want to eliminate $\max(i, n - i)$ elements at each pass.
- Can we have a simpler algorithm?
- Can we eliminate the inner quickselect?

Design considerations

- Can we improve the performance for common cases (e.g. i.i.d data)?
- Sacrifice worst case guarantees in favor of average case run time + a fallback?
  - We want to eliminate $\max(i, n - i)$ elements at each pass.
- Can we have a simpler algorithm?
- Can we eliminate the inner quickselect?
- Cheap pivots work good in practice.

So let’s begin!

- Assume i.i.d data.

So let’s begin!

- Assume i.i.d data.
  - Because we only care about order, everything is isomorphic to the uniform distribution.

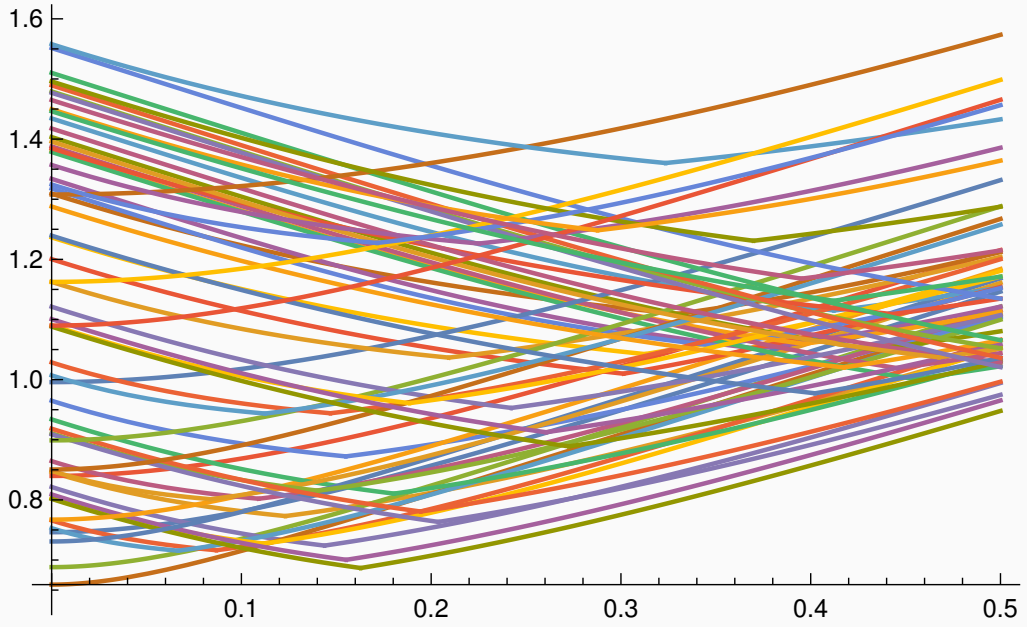
So let’s begin!

- Assume i.i.d data.
  - Because we only care about order, everything is isomorphic to the uniform distribution.
- QuickSelect adaptive is based on iterative splits (i.e “median of 3 and the median of 3”). Can we optimize the splits for a given index?

So let’s begin!

- Assume i.i.d data.
  - Because we only care about order, everything is isomorphic to the uniform distribution.
- QuickSelect adaptive is based on iterative splits (i.e “median of 3 and the median of 3”). Can we optimize the splits for a given index?
  - Generalize like ((4,2),(3,2)) means: “Divide in groups of 4, take the elements in the second quantile, then divide that in groups of 3 and take the median”

Optimizing splits



- Minima are of the form ((a,1), (b,b)), i.e. minimum and maximum.
- Minima and maxima are much easier to compute than medians, and scale well with the number of elements.

Example: median3

Quite ugly!

```
void median3(mreal *x, mint *a, mint *b, mint *c){
    if(ari(x, *a) <= ari(x, *b)){ //123 312 132
        if(ari(x, *c) < ari(x,*a)){ //312
            SWAP(*a, *b);
            SWAP(*a, *c);
        } else{ //123 132
            if (ari(x, *c) < ari(x,*b)){ //132
                SWAP(*b, *c);
            }//123
        }
    } else{ //213 321 231
        if(ari(x, *a) <= ari(x, *c)){ //213
            SWAP(*a,*b);
        }else{//321 231
            if(ari(x, *b) < ari(x, *c)){//231
                SWAP(*a, *c);
                SWAP(*b, *a);
            }else{//321
                SWAP(*a,*c);
            }
        }
    }
}
```

A new idea

- Take groups of a, compute the minimum, the maximum of the n/a.

A new idea

- Take groups of a, compute the minimum, the maximum of the n/a.
- No inner quickselect.

A new idea

- Take groups of a, compute the minimum, the maximum of the n/a.
- No inner quickselect.
- Can we find a as a function of i?

A new idea

- Take groups of a, compute the minimum, the maximum of the n/a.
- No inner quickselect.
- Can we find a as a function of i?
  - Yes! If we assume i.i.d. data.

Beta distributions

Given n random variables sampled with uniform distribution, the j-th element has the distribution:

$$\text{PDF[OrderDistribution[\{}\mathit{UniformDistribution}[], n\}\text{ , j], x]=}$$

$$\frac{x^{j-1}(1-x)^{n-j}}{B(j,-j+n+1)}$$

Beta distributions

Given n random variables sampled with uniform distribution, the j-th element has the distribution:

$$\text{PDF[OrderDistribution[\{}\mathit{UniformDistribution}[], n\}\text{ , j], x]=}$$

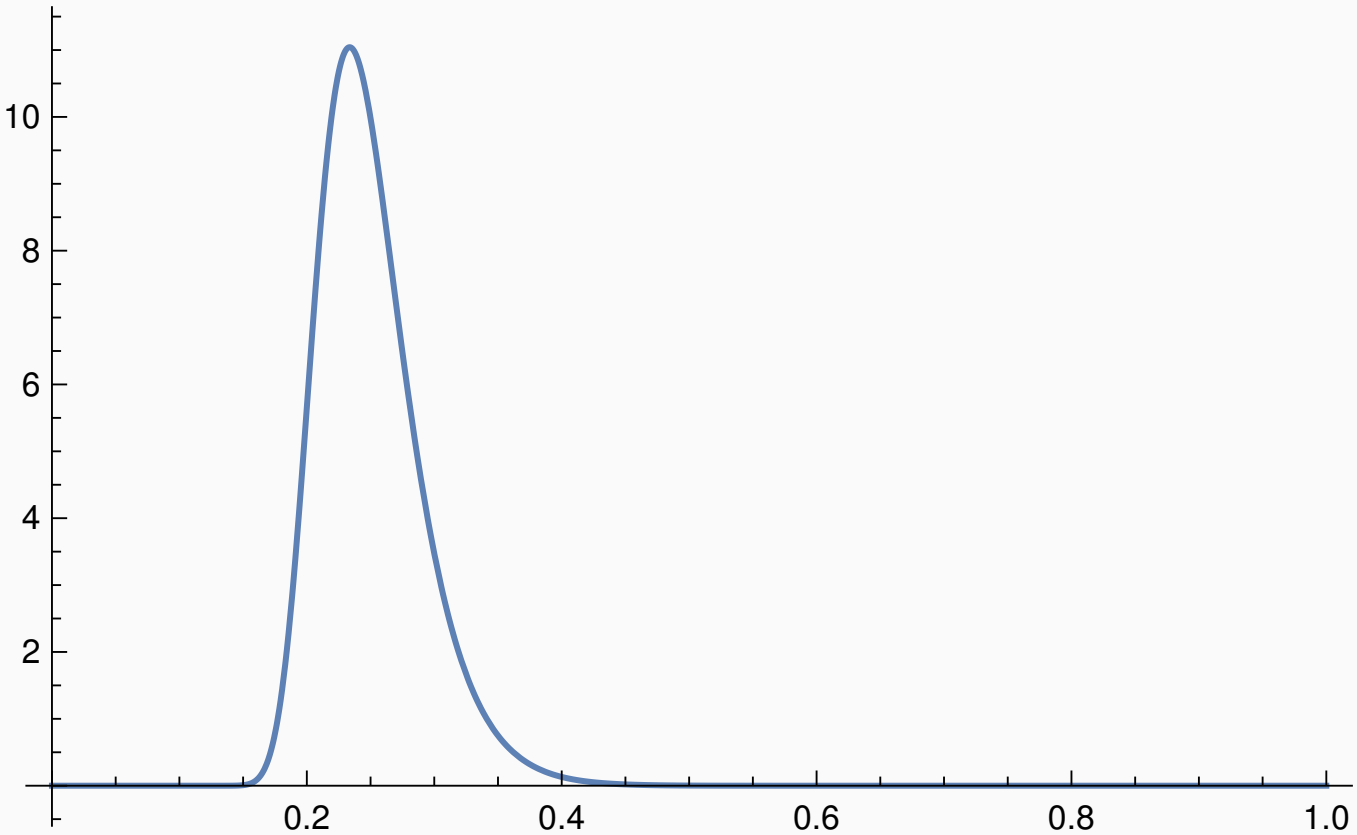
$$\frac{x^{j-1}(1-x)^{n-j}}{B(j,-j+n+1)}$$

The maximum of the minima of a has the distribution:

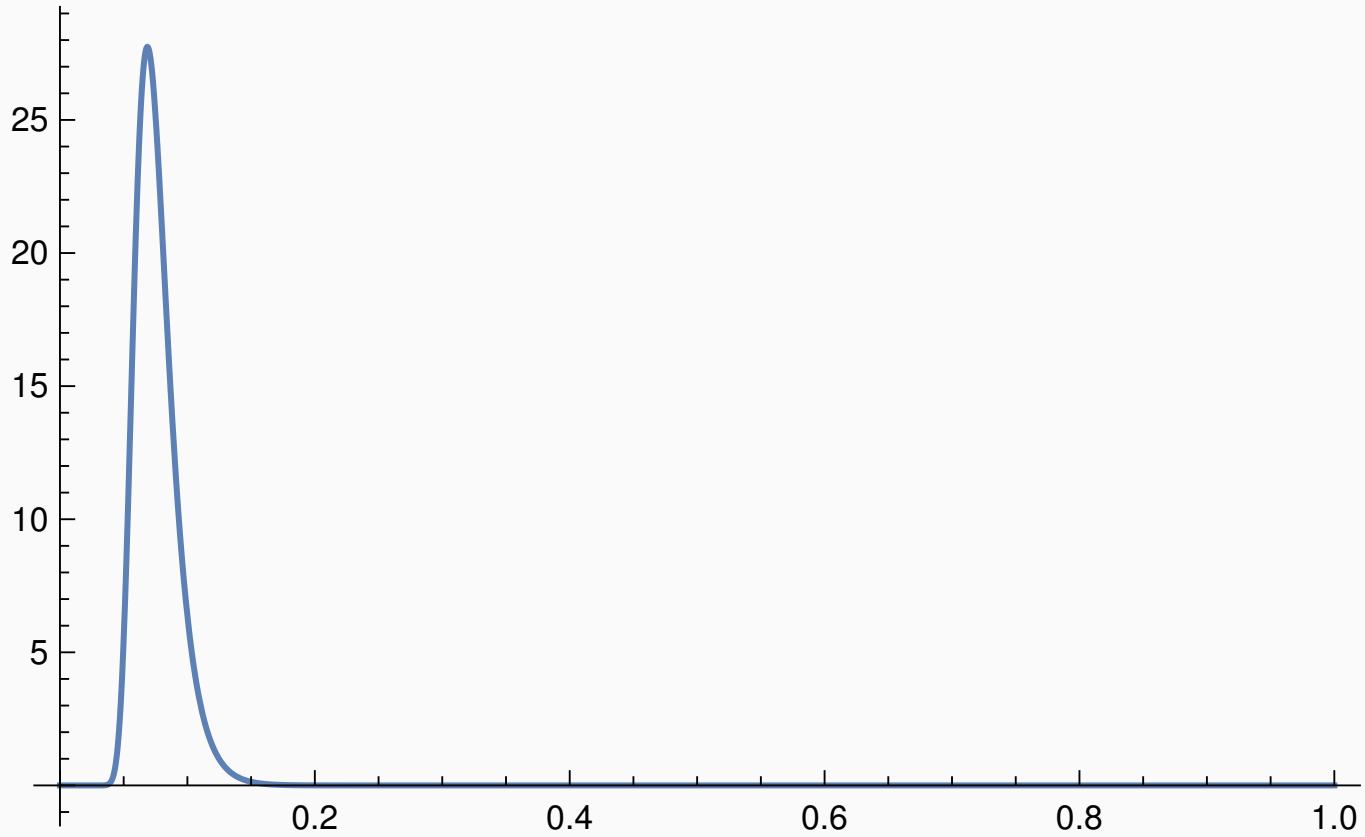
$$\text{PDF[OrderDistribution}\left[\left\{\text{BetaDistribution}[1,a],\frac{n}{a}\right\},\frac{n}{a}\right],x]=$$

$$\frac{n(1-x)^{a-1}\left(1-(1-x)^{a}\right)^{\frac{n}{a}-1}}{aB(1,a)}$$

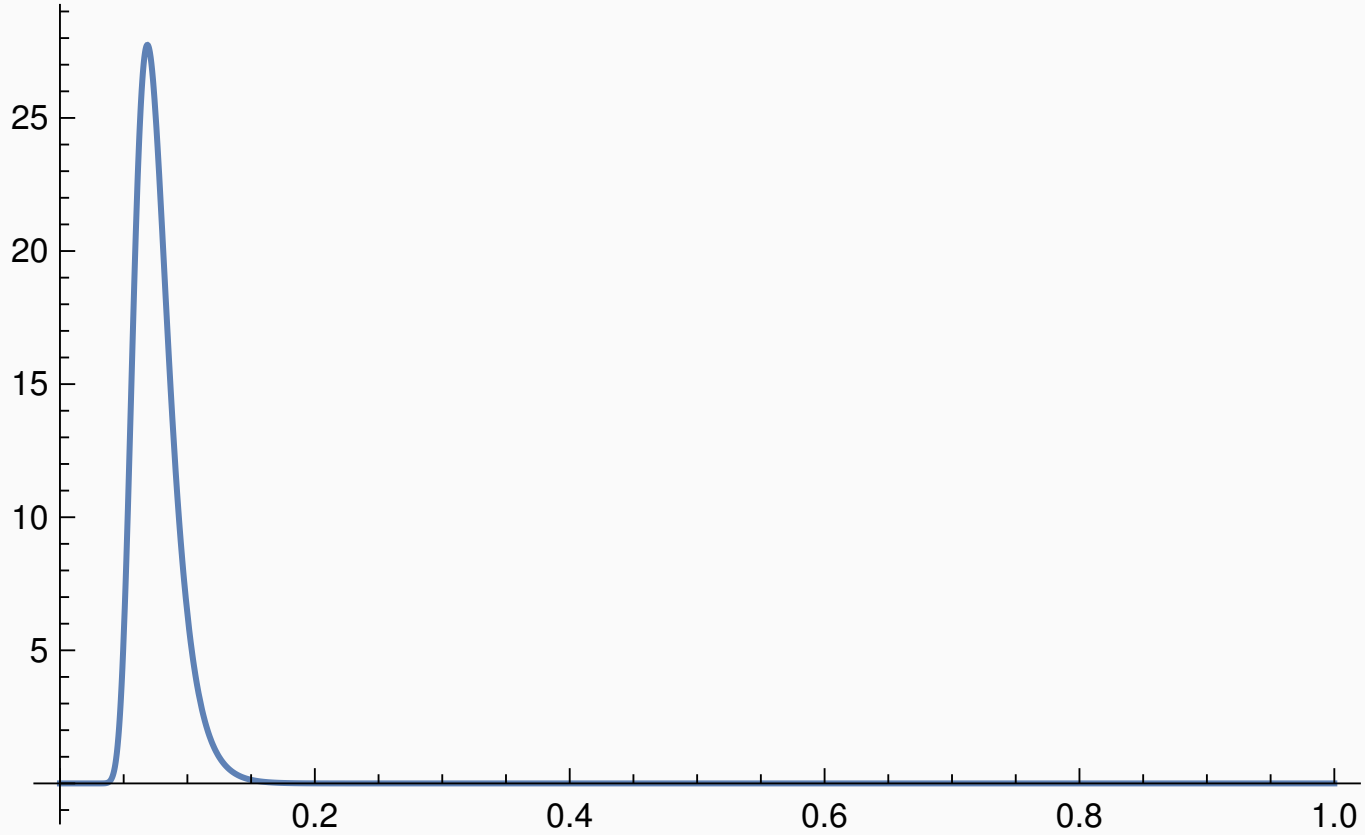
Example a=23 n=10000



Example a=70 n=10000



Example a=70 n=10000



Important: Have to get closer to the center than the extreme.

Optimizing a

We optimize the peak of the p.d.f as a function of $q = i/n$.

$$\frac{dpdf(x)}{x}|_{x=q} = 0 \Rightarrow$$

$$n(1-q)^{a-1}\left(1-(1-q)^{a}\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^{a}-a+1\right) = 0$$

Optimizing a

We optimize the peak of the p.d.f as a function of $q = i/n$.

$$\frac{dpdf(x)}{x}|_{x=q} = 0 \Rightarrow$$

$$n(1-q)^{a-1}\left(1-(1-q)^a\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^a - a + 1\right) = 0$$

Mathematica helpfully finds a solution (after some mangling):

$$a \to \frac{\log(1-x) - \text{ProductLog}((n-1)(x-1)\log(1-x))}{\log(1-x)}$$

But this is too expensive in a high performance code.

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^{a}\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^{a}-a+1\right)=0$$

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^{a}\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^{a}-a+1\right)=0$$

so:

$$(n-1)(1-q)^{a}-a+1=0$$

$$a=\frac{\log(a-1)-\log(n-1)}{\log(1-q)}$$

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^{a}\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^{a}-a+1\right)=0$$

so:

$$(n-1)(1-q)^{a}-a+1=0$$

$$a^{(t)}=\frac{\log(a^{(t-1)}-1)-\log(n-1)}{\log(1-q)}$$

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^{a}\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^{a}-a+1\right)=0$$

so:

$$(n-1)(1-q)^{a}-a+1=0$$

$$a^{(t)}=\frac{\log(a^{(t-1)}-1)-\log(n-1)}{\log(1-q)}$$

- Set $a^{(0)}=1/q$ (from $q=1/n\rightarrow a=n$).

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^a\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^a-a+1\right)=0$$

so:

$$(n-1)(1-q)^a-a+1=0$$

$$a^{(t)}=\frac{\log(a^{(t-1)}-1)-\log(n-1)}{\log(1-q)}$$

- Set $a^{(0)}=1/q$ (from $q=1/n\rightarrow a=n$).
- $a^{(2)}$ is well defined for all values in the useful range.

An iterative approximation

We have:

$$n(1-q)^{a-1}\left(1-(1-q)^a\right)^{\frac{n}{a}-1}\left((n-1)(1-q)^a-a+1\right)=0$$

so:

$$(n-1)(1-q)^a-a+1=0$$

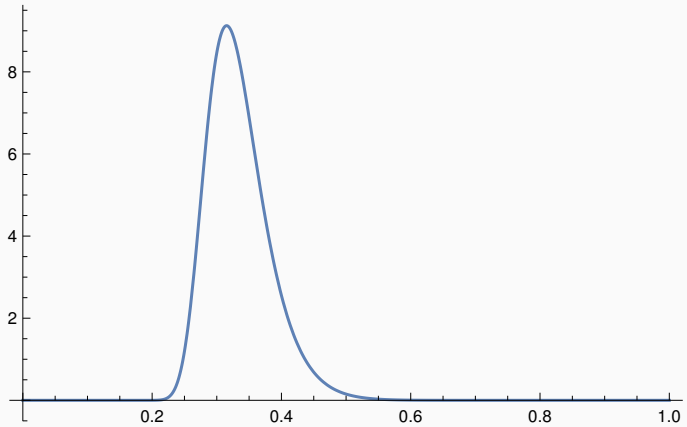
$$a^{(t)}=\frac{\log(a^{(t-1)}-1)-\log(n-1)}{\log(1-q)}$$

- Set $a^{(0)}=1/q$ (from $q=1/n\to a=n$).
- $a^{(2)}$ is well defined for all values in the useful range.
- $a^{(2)}\lesssim a$: Gives values closer to the center!

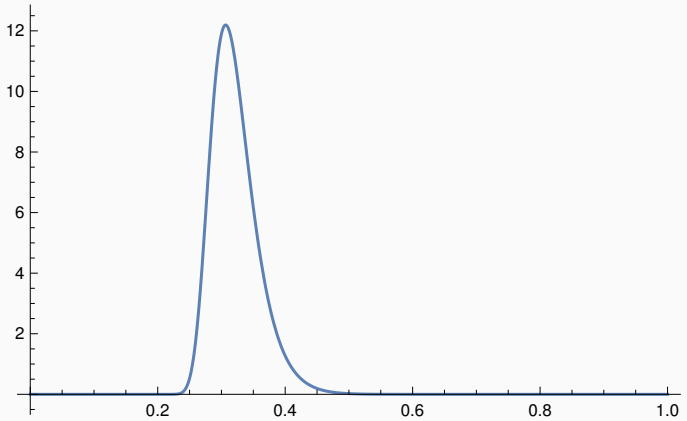
Some values

n	q	a	$a^{(2)}$
10000	0.5	10.1	9.7
10000	0.3	17.9	17.0
10000	0.05	91.7	86.0

Dependence on n

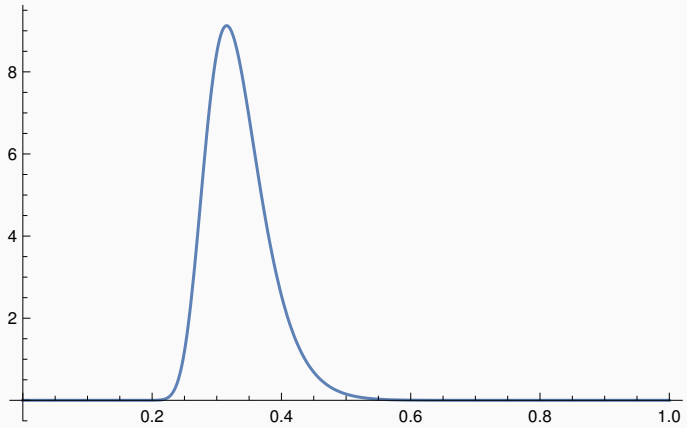


(a) $q = 0.3$, $n = 10000$

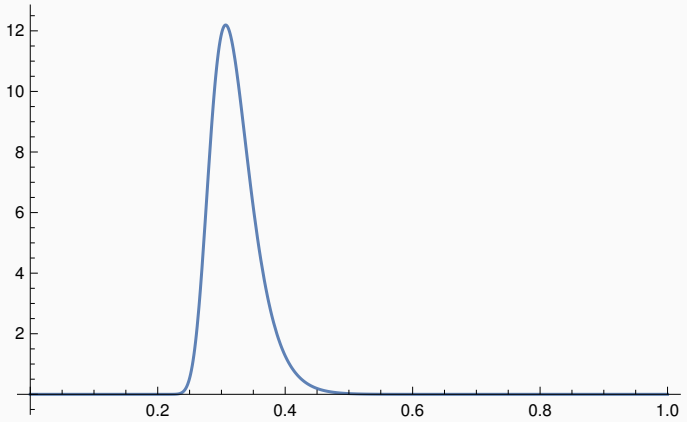


(b) $q = 0.3$, $n = 100000$

Dependence on n



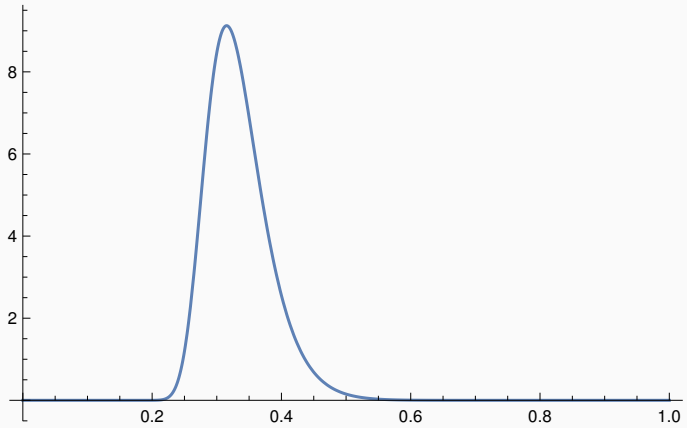
(c) $q = 0.3$, $n = 10000$



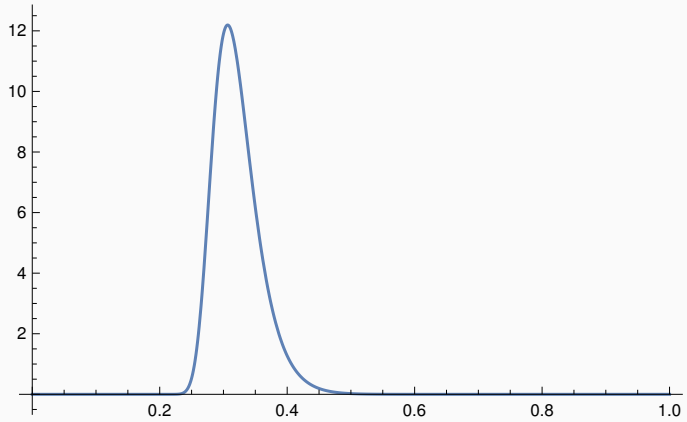
(d) $q = 0.3$, $n = 100000$

- In practice, large values of n increase the cost of the algorithm too much given the small improvement in the pivot.

Dependence on n



(e) $q = 0.3$, $n = 10000$



(f) $q = 0.3$, $n = 100000$

- In practice, large values of n increase the cost of the algorithm too much given the small improvement in the pivot.
- $n = \min(|l|, 30000)$ works better.

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)
- Split l in groups of a and compute the minimum of each group.

Minmax pivot selection

Algorithm:

- Compute round$(a^{(2)}(n, q))$
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Minmax pivot selection

Algorithm:

- Compute round$(a^{(2)}(n, q))$
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Characteristics:

- Very fast pivot selection.

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Characteristics:

- Very fast pivot selection.
- Reasonably accurate for all quantiles.

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Characteristics:

- Very fast pivot selection.
- Reasonably accurate for all quantiles.
- Fails in the right direction.

Minmax pivot selection

Algorithm:

- Compute round($a^{(2)}(n, q)$)
- Split l in groups of a and compute the minimum of each group.
- Pivot is the minimum of the maxima.

Characteristics:

- Very fast pivot selection.
- Reasonably accurate for all quantiles.
- Fails in the right direction.
- Extremely simple: All it needs is a couple of logs, min and max.

Example: median, random input $|l| = 10^8$

```
$ ./qstest
pfrac is 0.500
Pivot is 0.47717 and len is 52279565
pfrac is 0.044
Pivot is 0.49816 and len is 50182145
pfrac is 0.004
Pivot is 0.50064 and len is 248020
pfrac is 0.734
Pivot is 0.49992 and len is 71806
pfrac is 0.083
Pivot is 0.49999 and len is 7409
pfrac is 0.801
Pivot is 0.49998 and len is 1641
pfrac is 0.099
Pivot is 0.49998 and len is 238
pfrac is 0.685
Pivot is 0.49998 and len is 164
Result is: 0.499978
```

Comparison (random input)

$\|l\| = 10^8$	q=0.01	q=0.2	q=0.5
Baeline	259 *ms*	730 *ms*	820 *ms*
MinMax	143 *ms*	408 *ms*	766*ms*

Structured inputs

- I wish this was the end of the talk (and probably you too by now).

Structured inputs

- I wish this was the end of the talk (and probably you too by now).
- But we had to test what happens when the input has some (permutation) structure.

Structured inputs

- I wish this was the end of the talk (and probably you too by now).
- But we had to test what happens when the input has some (permutation) structure.
- Example: Sorted data

Structured inputs

- I wish this was the end of the talk (and probably you too by now).
- But we had to test what happens when the input has some (permutation) structure.
- Example: Sorted data

Structured inputs

- I wish this was the end of the talk (and probably you too by now).
- But we had to test what happens when the input has some (permutation) structure.
- Example: Sorted data

```
$ ./qstest
pfrac is 0.500
Pivot is 99999000.00000 and len is 99999000
pfrac is 0.500
Pivot is 99998000.00000 and len is 99998000
pfrac is 0.500
Pivot is 99997000.00000 and len is 99997000
pfrac is 0.500
...
```

Structured inputs

- I wish this was the end of the talk (and probably you too by now).
- But we had to test what happens when the input has some (permutation) structure.
- Example: Sorted data

```
$ ./qstest
pfrac is 0.500
Pivot is 99999000.00000 and len is 99999000
pfrac is 0.500
Pivot is 99998000.00000 and len is 99998000
pfrac is 0.500
Pivot is 99997000.00000 and len is 99997000
pfrac is 0.500
...
```

QUADRATIC PERFORMANCE!

Possible Solutions

- Could do fallback.
  - But will not fly if fails for such common input.

Possible Solutions

- Could do fallback.
  - But will not fly if fails for such common input.
- Difficult to change the original idea so that it has e.g. guaranteed linear behavior.

Possible Solutions

- Could do fallback.
  - But will not fly if fails for such common input.
- Difficult to change the original idea so that it has e.g. guaranteed linear behavior.
- Need some way to make the pattern that fails more complicated.

Solution: MinMax3

- Make 3 guesses using MinMax.

Solution: MinMax3

- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).

Solution: MinMax3

- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).
  - 1 “from the far side” (e.g. max and then min for $q < 0.5$).

Solution: MinMax3

- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).
  - 1 “from the far side” (e.g. max and then min for $q < 0.5$).
- Take the median of the 3 guesses as the pivot.

Solution: MinMax3

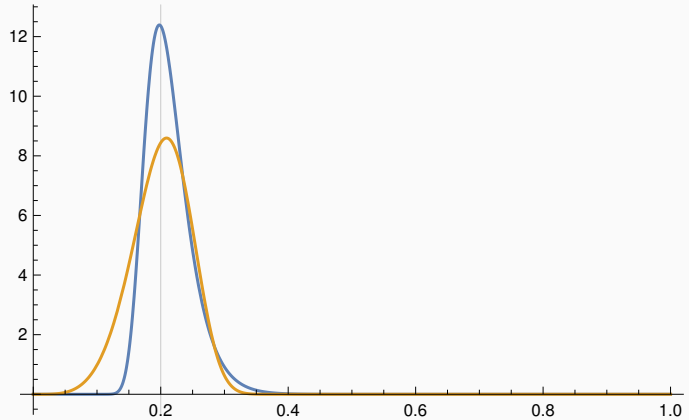
- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).
  - 1 “from the far side” (e.g. max and then min for $q < 0.5$).
- Take the median of the 3 guesses as the pivot.
- Align layout so it works well for common structures.

Solution: MinMax3

- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).
  - 1 “from the far side” (e.g. max and then min for $q < 0.5$).
- Take the median of the 3 guesses as the pivot.
- Align layout so it works well for common structures.
- Play with rounding to get a guess closer to the median.

Solution: MinMax3

- Make 3 guesses using MinMax.
  - 2 “from the closer side” (e.g. min and then max for $q < 0.5$).
  - 1 “from the far side” (e.g. max and then min for $q < 0.5$).
- Take the median of the 3 guesses as the pivot.
- Align layout so it works well for common structures.
- Play with rounding to get a guess closer to the median.



Example (sorted input)

```
$ ./qstest
Seed is 1482243662
pfrac is 0.500
Guess 1 is 999.0000
Guess 2 is 66665666.0000
Guess 3 is 99999000.0000
Pivot is 66665666.00000 and len is 66665666
pfrac is 0.750
Guess 1 is 1665.0000
Guess 2 is 44443300.0000
Guess 3 is 66665190.0000
Pivot is 44443300.00000 and len is 22222365
pfrac is 0.250
Guess 1 is 44796036.0000
Guess 2 is 51851231.0000
Guess 3 is 65431090.0000
Pivot is 51851231.00000 and len is 7407930
pfrac is 0.750
Guess 1 is 44796036.0000
Guess 2 is 49381445.0000
Guess 3 is 51850755.0000
Pivot is 49381445.00000 and len is 2469785
pfrac is 0.250
Guess 1 is 49422609.0000
Guess 2 is 50205206.0000
Guess 3 is 51714021.0000
Pivot is 50205206.00000 and len is 823760
pfrac is 0.751
...
```

Example (random input)

```
pfrac is 0.500
Guess 1 is 0.5711
Guess 2 is 0.4334
Guess 3 is 0.5249
Pivot is 0.52493 and len is 52498111
pfrac is 0.952
Guess 1 is 0.4928
Guess 2 is 0.4936
Guess 3 is 0.4968
Pivot is 0.49358 and len is 3132244
pfrac is 0.202
Guess 1 is 0.5008
Guess 2 is 0.5027
Guess 3 is 0.4959
Pivot is 0.50080 and len is 721230
pfrac is 0.879
Guess 1 is 0.4999
Guess 2 is 0.5000
Guess 3 is 0.4997
Pivot is 0.49988 and len is 90799
pfrac is 0.041
Guess 1 is 0.4999
Guess 2 is 0.4999
Guess 3 is 0.4999
Pivot is 0.49994 and len is 5633
pfrac is 0.657
Guess 1 is 0.4999
Guess 2 is 0.4999
Guess 3 is 0.4999
...
Result is: 0.499920
```

Results (random input)

$\|l\| = 10^8$	q=0.01	q=0.2	q=0.5
Baeline	259 *ms*	730 *ms*	820 *ms*
MinMax	143 *ms*	408 *ms*	766*ms*
MinMax3	145 *ms*	453*ms*	748 *ms*

Conclusions

- The 3 guesses approach doesn't damage performance too much.
- Quadratic behavior is very hard to get (tried with lots inputs).
  - Still check quality and have a fallback algorithm in case we get unlucky.
    - Or somebody tries to hack us.
- MinMax3 is faster than any algorithm I found in the literature for random data.
  - By a large amount for off-median quantiles!

Thank you!