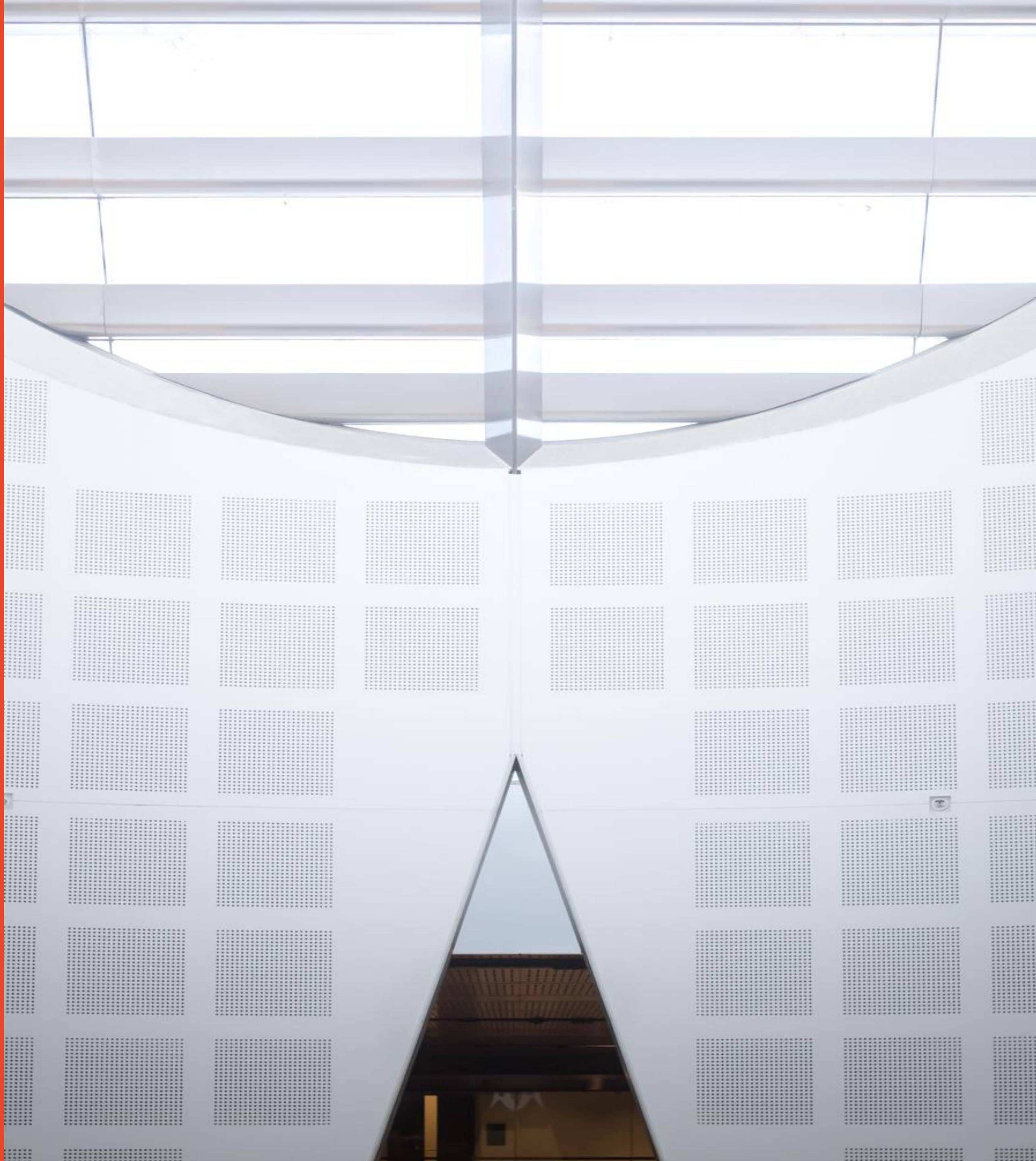
COMPx270: Randomised and Advanced Algorithms
Lecture 2: Concentration bounds, and tricks

Clément Canonne

School of Computer Science

THE UNIVERSITY OF SYDNEY



A question 🚌

You're waiting for the bus, but don't have the schedule (or a smartphone). The person next to you tells you that the bus comes on average every 5 minutes.

If you decide to wait up to 20 minutes before giving up and walking, what are the chances you will get a bus?

A question 🎰

You're given a Las Vegas algorithm A, but don't know anything about the details. The algo designer tells you that its expected running time is at most T.

If you decide to wait up to 4T steps before stopping A and returning "🤷‍♀️", what are the chances the algorithm will have terminated?

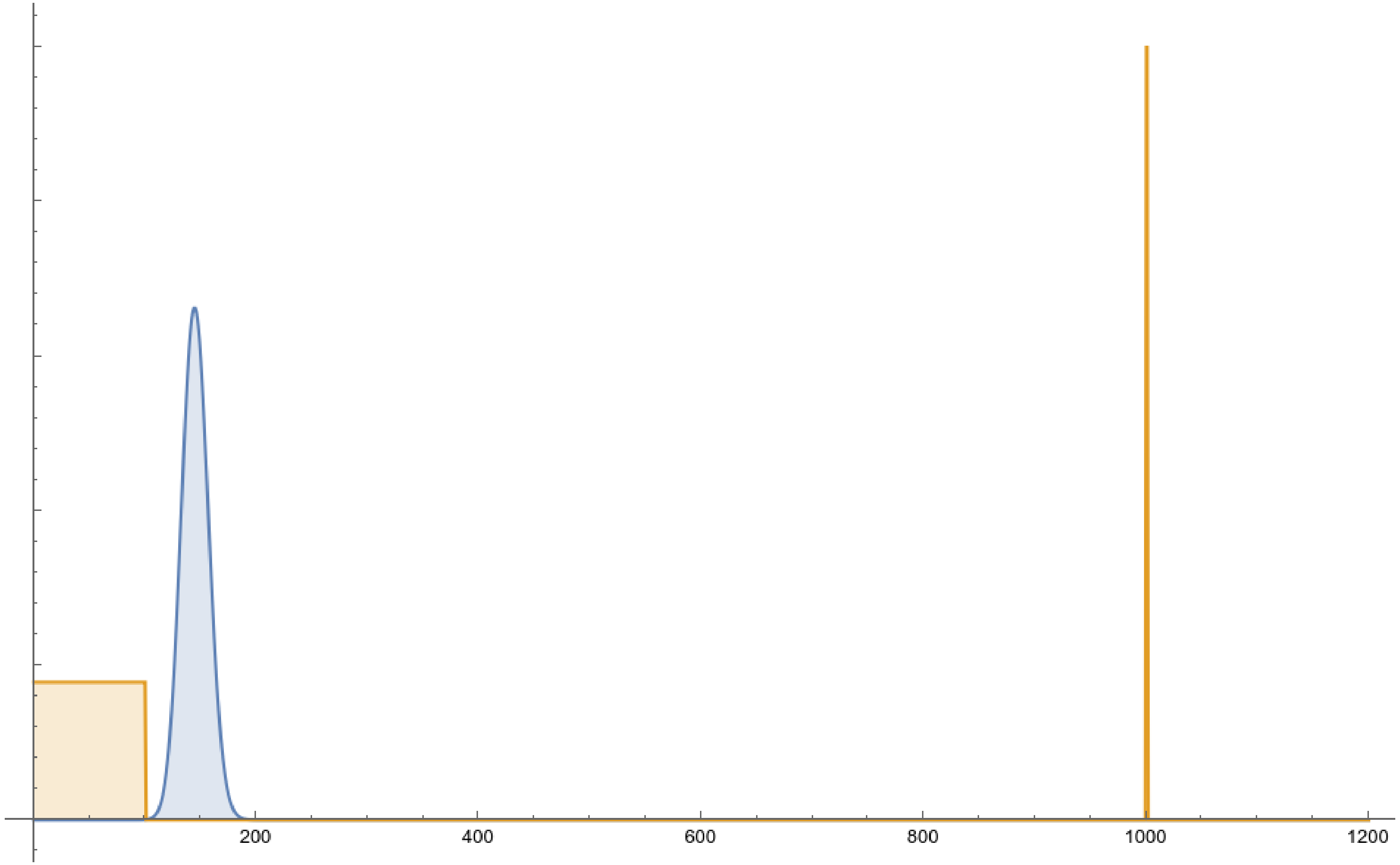
Why do we care?

Often we will obtain or give guarantees about expectations:

- Expected running time of an algorithm
- Expected quality of the output
- Expected amount of resources used

This is useful, but often not enough.

Why do we care?



Probability mass functions of two random variables with the same expectation, but **very** different behaviour

Why do we care?

Often we will obtain or give guarantees about expectations:

- Expected running time of an algorithm
- Expected quality of the output
- Expected amount of resources used

We also want to argue about concentration: "usually not too far from the expectation"

Some concentration tools (among many) 🧰

Some concentration tools: Markov's inequality

Suppose you only know two things about a random variable X:

1. $X \geq 0$
2. $\mathbb{E}[X]$ (or an upper bound on it)

Then, for every $t > 0$,

$$\Pr[\, X \geq t \,] \leq \mathbb{E}[X]/t$$

This is Markov's Inequality: "you can't be 10 times your expectation more than 10 percent of the time." (*if* you're non-negative)

Some concentration tools: Markov's inequality

Proof.

Application: from Las Vegas to Monte Carlo algorithms

Suppose I have a Las Vegas algorithm A for a task with expected running time (at most) T. Then I have a Monte Carlo algorithm A' with worst-case running time O(T) and failure probability 1%.

Application: from Las Vegas to Monte Carlo algorithms

Suppose I have a Las Vegas algorithm A for a task with expected running time (at most) T. Then I have a Monte Carlo algorithm A' with worst-case running time O(T) and failure probability 1%.

Idea: run A for up to 100T steps, abort and output anything if you go over.

Application: from Las Vegas to Monte Carlo algorithms

Suppose I have a Las Vegas algorithm A for a task with expected running time (at most) T. Then I have a Monte Carlo algorithm A' with worst-case running time O(T) and failure probability 1%.

Idea: run A for up to 100T steps, abort and output anything if you go over.

Only issue: to get error probability δ, running time becomes $O(T/\delta)$.

Some concentration tools: Chebyshev's inequality

Suppose you know two things about a random variable X:

1. $\mathbb{E}[X]$
2. Var[X] (or an upper bound on it)

Then, for every $t > 0$,

$$\Pr[\ |X-\mathbb{E}[X]| \geq t\ ] \leq \mathrm{Var}[X]/t^2$$

This is Chebyshev's Inequality: "you can't deviate from your expectation by more than 10 standard deviations more than 1 percent of the time."

Some concentration tools: Chebyshev's inequality

Proof.

Application: from Las Vegas to Monte Carlo algorithms

Suppose I have a Las Vegas algorithm A for a task with expected running time (at most) T and variance σ^2. Then I have a Monte Carlo algorithm A' with worst-case running time $O(T)$ and failure probability 1%.

Idea: run A for up to $T+10\sigma$ steps, abort and output anything if you go over.

Better? To get error probability δ, running time becomes $T+O(\sigma/\sqrt{\delta})$.

Some concentration tools: Chernoff/Hoeffding bounds

Suppose you know two things about a random variable X:

1. It can be written as the sum of many independent r.v.s $X_1,..,X_n$
2. Each X_t is bounded in [0,1]

Then, for every γ in (0,1],

$$\Pr[\ |X - \mathbb{E}[X]| > \gamma\mathbb{E}[X]\ ] \leq 2 \exp(-\gamma^2\mathbb{E}[X]/3)$$

This is the Chernoff Bound: "you can't deviate from your expectation by more than a relative amount except with exponentially small probability."

Some concentration tools: Chernoff/Hoeffding bounds

Suppose you know two things about a random variable X:

1. It can be written as the sum of many independent r.v.s $X_1,..,X_n$
2. Each X_t is bounded in [0,1]

Then, for every γ in (0,1],

$$\Pr[\ |X - \mathbb{E}[X]| > \gamma n\ ] \leq 2\exp(-2\gamma^2 n)$$

This is the Hoeffding Bound: "you can't deviate from your expectation by more than an additive amount except with exponentially small probability."

Some concentration tools: the Chernoff bound

Proof.

Tutorial

(Proof by Markov)

Application: from Las Vegas to Monte Carlo algorithms

Suppose I have a Las Vegas algorithm A for a task with expected running time (at most) T. Then I have a Monte Carlo algorithm A' with worst-case running time O(T) and failure probability δ.

Idea: start with the Markov idea: run A for up to 2T steps, abort if you go over. Now, repeat that k=O(log(1/δ)) times.

Much better! To get error probability δ, running time becomes O(T log(1/δ)).

💡 (Actually guarantees that many of the k runs will be successful, not just one.)

Some concentration tools (among many) 🧰

- Markov: minimal assumptions, often weaker, one-sided, $X \geq 0$
- Chebyshev: needs to bound the variance, often suffices, pairwise independence is enough [we'll get back to that]
- Chernoff: stronger assumptions, much stronger guarantees for large deviations. Requires full independence.
- Hoeffding: same, but slightly different guarantees

There are many others + variations, but this is a good toolbox to design and reason about randomised algorithms.

One last tool 🧰

If $E_1, E_2, \ldots, E_k$ are events, then

$$\Pr[\, E_1 \text{ or } E_2 \text{ or } \ldots \text{ or } E_k \,] \leq \Pr[E_1] + \Pr[E_2] + \ldots + \Pr[E_k]$$

This is the Union Bound: works even if the events have weird, intricate dependencies.

One last tool 🧰

If $E_1, E_2, \ldots, E_k$ are events, then

$$\Pr[\, E_1 \text{ or } E_2 \text{ or } \ldots \text{ or } E_k \,] \leq \Pr[E_1] + \Pr[E_2] + \ldots + \Pr[E_k]$$

This is the Union Bound: works even if the events have weird, intricate dependencies.

Corollary:

$$\Pr[\, \text{none of } E_1, E_2, \ldots, E_k \,] \geq 1 - (\Pr[E_1] + \Pr[E_2] + \ldots + \Pr[E_k])$$

From Monte Carlo to Las Vegas algorithms

We have seen how to convert a Las Vegas to a Monte Carlo algorithm.

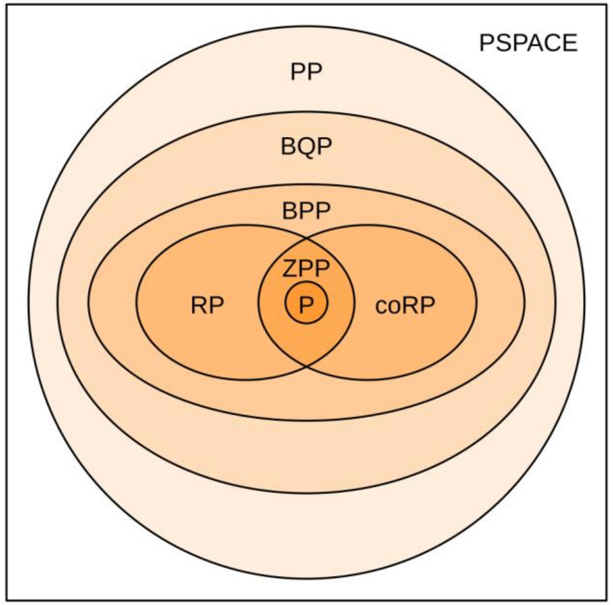
Can we do the opposite?

From Monte Carlo to Las Vegas algorithms

We have seen how to convert a Las Vegas to a Monte Carlo algorithm.

Can we do the opposite?

(No*)



From Monte Carlo to Las Vegas algorithms

We have seen how to convert a Las Vegas to a Monte Carlo algorithm.

Can we do the opposite?

Sometimes. E.g., *if* we can efficiently check the output is good.

From Monte Carlo to Las Vegas algorithms

Theorem. Suppose I have a Monte Carlo algorithm A with worst-case running time T and failure probability $p < 1$, with the following extra guarantee: we can check if A's output is correct in time O(1).

Then there is a *Las Vegas* algorithm A' for the same task with expected running time O(T).

From Monte Carlo to Las Vegas algorithms

Theorem. Suppose I have a Monte Carlo algorithm A with worst-case running time T and failure probability $p < 1$, with the following extra guarantee: we can check if A's output is correct in time $O(1)$.

Then there is a *Las Vegas* algorithm A' for the same task with expected running time $O(T)$.

```
do
    run A on input x, let y be its output
until output y is good
```

From Monte Carlo to Las Vegas algorithms

```
do
    run A on input x, let y be its output
until output y is good
```

Proof.

Recap so far

We have seen how to convert Las Vegas to Monte Carlo, and (sometimes) the other way around.

We have seen some tools (concentration inequalities and union bound) that *surely* will prove useful. ("Chekhov's algorithmic gun")

But Monte Carlo algorithms were only required to succeed with some lame probability, like 2/3. What if we want 99.9999999%?

From Monte Carlo to (better) Monte Carlo

Theorem. Suppose I have a Monte Carlo algorithm A *for a decision problem*, with worst-case running time T and failure probability 1/3.

Then there is a Monte Carlo algorithm A' for the same task with worst-case running time $O(T \log(1/\delta))$ and failure probability δ.

```
for 1 ≤ t ≤ k
      run A on input x, let y_t in {0,1} be its output
return majority(y_1,...,y_k)
```

From Monte Carlo to (better) Monte Carlo

```
for 1 ≤ t ≤ k
    run A on input x, let y_t in {0,1} be its output
return majority(y_1,...,y_k)
```

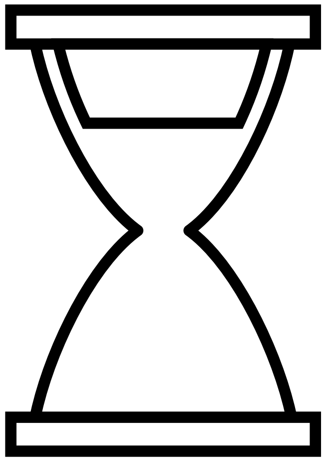
Proof.

From Monte Carlo to (better) Monte Carlo

This was for decision problems. The majority trick can be generalised, for instance,* to real-valued outputs instead of binary: this is the median trick, where you amplify success probability by taking the median of the k outputs.

You'll see the details in the tutorial. Think about why "median" and not "average"!

*Under some conditions



Now, an algorithm

Problem: Given an unsorted array A of n distinct* integers, find the median.

Now, an algorithm

Problem: Given an unsorted array A of n distinct* integers, find the median.

Solution: Quick Selection (Median of Medians algorithm).

Beautiful* divide-and-conquer deterministic algorithm running in time O(n).

Now, an algorithm

Problem: Given an unsorted array A of n distinct* integers, find the median.

Solution: Quick Selection (Median of Medians algorithm).

Beautiful* divide-and-conquer deterministic algorithm running in time O(n).

*In theory

Now, a randomised algorithm

Problem: Given an unsorted array A of n distinct* integers, find the median.

Solution: Randomised Median

Beautiful randomised Monte Carlo algorithm running in time O(n).

Now, a randomised algorithm

Problem: Given an unsorted array A of n distinct* integers, find the median.

Solution: Randomised Median

Beautiful randomised Monte Carlo algorithm running in time O(n).

Introduces the idea of "sampling as a guide"

Randomised Median

Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$

Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)

Sort B

Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B

Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C

Compute the number k of elements of A smaller than $\underline{b}$

Compute the number ℓ of elements of A larger than $\overline{b}$

if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**

  return fail

else if $|C| > \frac{4n\Delta}{m} + 2$ **then**

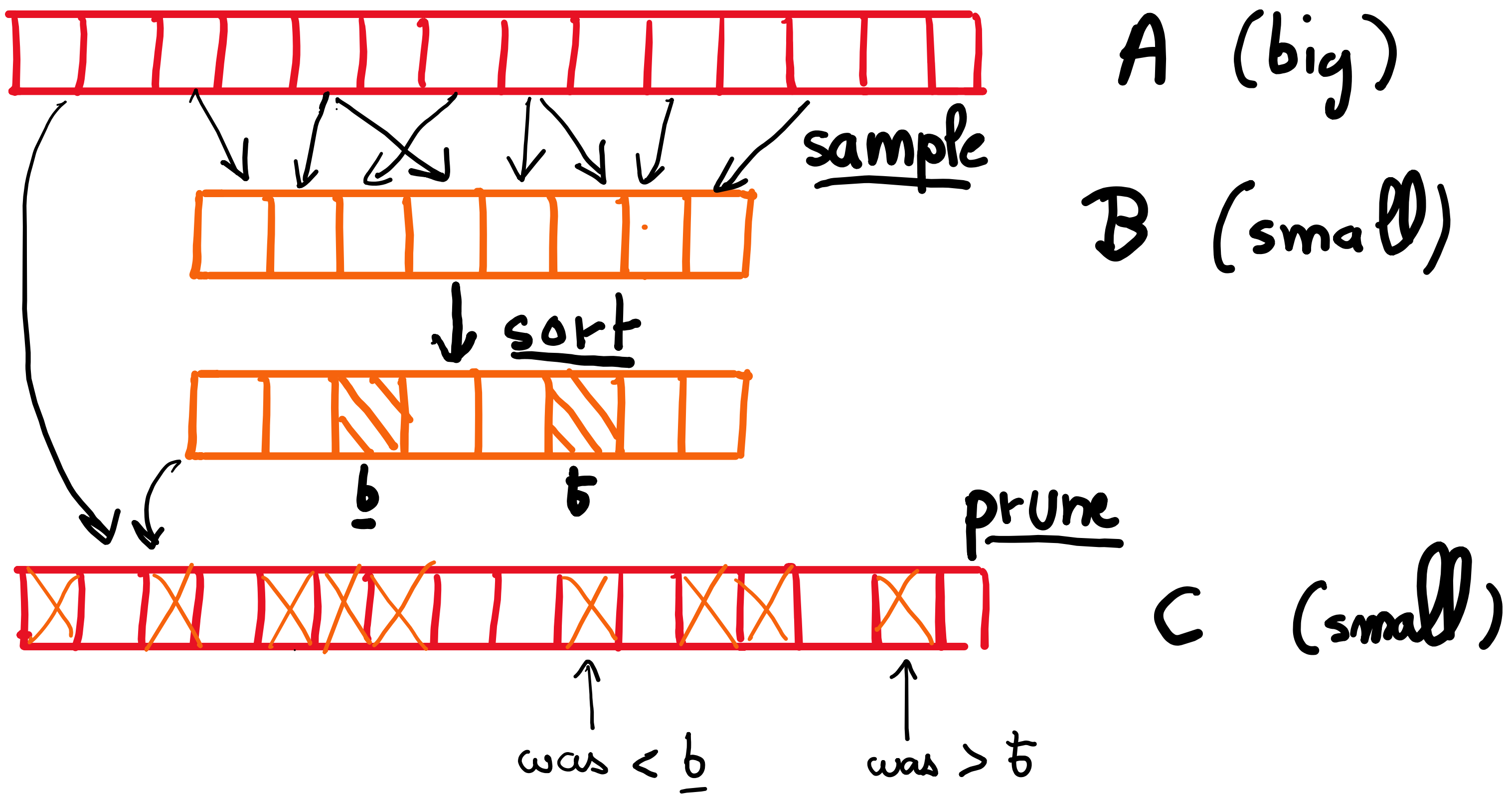
  return fail

else

  Sort C

  return the $(\frac{n+1}{2} - k)$-th element of C.

Randomised Median



Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$
Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)
Sort B
Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B
Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C
Compute the number k of elements of A smaller than $\underline{b}$
Compute the number ℓ of elements of A larger than $\overline{b}$
if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**
    return fail
else if $|C| > \frac{4n\Delta}{m} + 2$ **then**
    return fail
else
    Sort C
    return the $(\frac{n+1}{2} - k)$-th element of C.

Randomised Median

Require: array A of n distinct integers
1: Set $\Delta = 4\sqrt{m}$
2: Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)
3: Sort B
4: Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B
5: Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C
6: Compute the number k of elements of A smaller than $\underline{b}$
7: Compute the number ℓ of elements of A larger than $\overline{b}$
8: **if** $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**
9: **return** fail
10: **else if** $|C| > \frac{4n\Delta}{m} + 2$ **then**
11: **return** fail
12: **else**
13: Sort C
14: **return** the $(\frac{n+1}{2} - k)$-th element of C.

Running time:

$O(m \log m) + O(n) + O((n/\sqrt{m}) \log (n/\sqrt{m}))$

Choose m to have $n/\sqrt{m} = m$ (balance first and last terms), get $O(n)$

Randomised Median

Correctness?

Only incorrect when it outputs fail. 3 possible "bad events":

1. Too many elements (in A) smaller than $\underline{b}$ — E_1
2. Too many elements (in A) bigger than $\overline{b}$ — E_2
3. Too many elements in C — E_3

Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$

Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)

Sort B

Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B

Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C

Compute the number k of elements of A smaller than $\underline{b}$

Compute the number ℓ of elements of A larger than $\overline{b}$

if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**

    return fail

else if $|C| > \frac{4n\Delta}{m} + 2$ **then**

    return fail

else

    Sort C

    return the $(\frac{n+1}{2} - k)$-th element of C.

Randomised Median

Correctness?

By the union bound, can bound separately the probability of these 3 bad events.

"By symmetry" the first two events can be bounded the same way.

Require: array A of n distinct integers

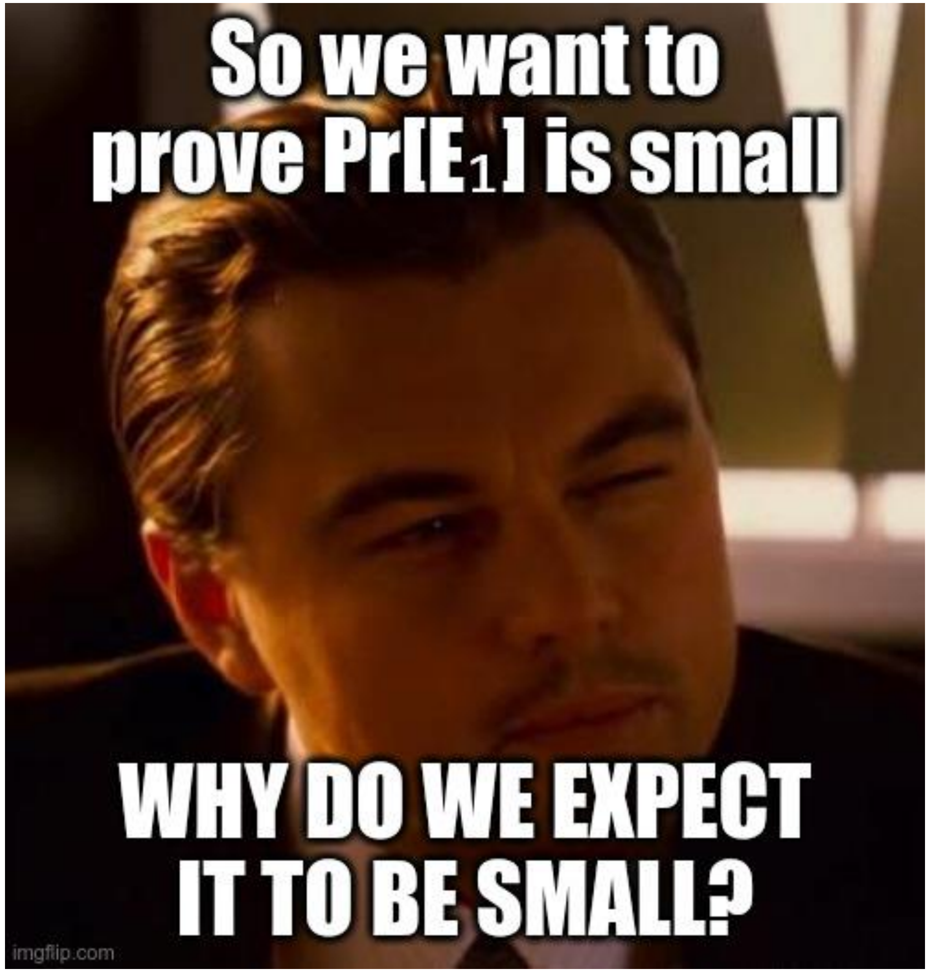
```
Set Δ = 4√m
Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)
Sort B
Let b̲ and b̄ be the (m/2 − Δ)-th and (m/2 + Δ)-th elements of B
Copy every x of A with b̲ ≤ x ≤ b̄ in a new array C
Compute the number k of elements of A smaller than b̲
Compute the number ℓ of elements of A larger than b̄
if k > n/2 or ℓ > n/2 then
    return fail
else if |C| > 4nΔ/m + 2 then
    return fail
else
    Sort C
    return the ((n+1)/2 − k)-th element of C.
```

$$\Pr[\text{fail}] = \Pr[E_1 \cup E_2 \cup E_3] \leq \Pr[E_1] + \Pr[E_2] + \Pr[E_3]$$

$$= \underbrace{2\Pr[E_1] + \Pr[E_3]}$$

Want to show this is $\leq$ small constant

Randomised Median: why should it work?



Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$

Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)

Sort B

Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B

Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C

Compute the number k of elements of A smaller than $\underline{b}$

Compute the number ℓ of elements of A larger than $\overline{b}$

if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**

    return fail

else if $|C| > \frac{4n\Delta}{m} + 2$ **then**

    return fail

else

    Sort C

    return the $(\frac{n+1}{2} - k)$-th element of C.

Proof.

Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$

Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)

Sort B

Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B

Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C

Compute the number k of elements of A smaller than $\underline{b}$

Compute the number ℓ of elements of A larger than $\overline{b}$

if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**

  return fail

else if $|C| > \frac{4n\Delta}{m} + 2$ **then**

  return fail

else

  Sort C

  return the $(\frac{n+1}{2} - k)$-th element of C.

Proof.

Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$

Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)

Sort B

Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B

Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C

Compute the number k of elements of A smaller than $\underline{b}$

Compute the number ℓ of elements of A larger than $\overline{b}$

if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**

  return fail

else if $|C| > \frac{4n\Delta}{m} + 2$ **then**

  return fail

else

  Sort C

  return the $(\frac{n+1}{2} - k)$-th element of C.

Proof.

Require: array A of n distinct integers

Set $\Delta = 4\sqrt{m}$
Create an array B containing m elements of A chosen independently and uniformly at random (with replacement)
Sort B
Let $\underline{b}$ and $\overline{b}$ be the $(m/2 - \Delta)$-th and $(m/2 + \Delta)$-th elements of B
Copy every x of A with $\underline{b} \leq x \leq \overline{b}$ in a new array C
Compute the number k of elements of A smaller than $\underline{b}$
Compute the number ℓ of elements of A larger than $\overline{b}$
if $k > \frac{n}{2}$ or $\ell > \frac{n}{2}$ **then**
    return fail
else if $|C| > \frac{4n\Delta}{m} + 2$ **then**
    return fail
else
    Sort C
    return the $(\frac{n+1}{2} - k)$-th element of C.

Randomised Median: summary

Problem: Given an unsorted array A of n distinct* integers, find the median.

Solution: Randomised Median is a randomised Monte Carlo algorithm running in time O(n).

+ Probability amplification

This lecture: summary

- Concentration inequalities: Markov (first-moment method), Chebyshev (second-moment method), Chernoff/Hoeffding
- Union bound: your new best friend (with linearity of expectation)
- Probability amplification: majority vote, median trick
- Sampling as a guide or "sketch"