15-251

Great Theoretical Ideas in Computer Science

Randomized Algorithms

March 24th, 2015

So far

Formalization of computation/algorithm

Computability / Uncomputability

Computational Complexity

- How to analyze it
- Some neat algorithms

Identifying intractable problems. NP-completeness.

Dealing with intractable problems: Approximation algs.

Next

Randomized algs.

More mathematical tools:

- number theory
- linear algebra
- fields, polynomials

Other important TCS concepts:

- cryptography
- Markov chains
- quantum computation
- communication complexity
- CS perspective on proofs

Randomness and the universe

Does the universe have true randomness?

Newtonian physics suggests that the universe evolves deterministically.

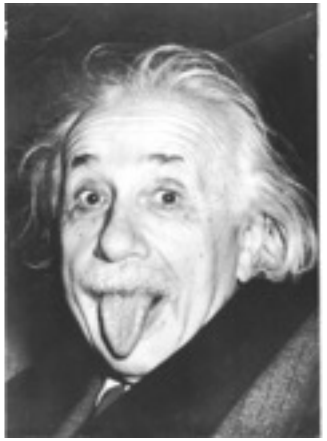
Quantum physics says otherwise.

Randomness and the universe

Does the universe have true randomness?

God does not play dice with the world.

- *Albert Einstein*



Einstein, don't tell God what to do.

- *Niels Bohr*

Randomness and the universe

Does the universe have true randomness?

Even if it doesn't, we can still model our uncertainty about things using probability.

Randomness is an essential component in modeling and analyzing nature.

It also plays a key role in computer science.

Randomness in computer science

Cryptography

Can't achieve unpredictability without randomness.

Simulating real-world events

Statistics via sampling

e.g. *election polls*

Learning theory

Data is generated by some probability distribution.

Coding Theory

Encode data to be able to deal with random noise.

Randomness in computer science

Randomized models for deterministic objects

e.g. *the www graph*

Quantum computing

Randomness in inherent in quantum mechanics.

Speeding up algorithms

...

Randomness and algorithms

How can randomness be used in computation?

Where can randomness come into the picture?

Given some algorithm that solves a problem…

- What if the input is chosen randomly?
- What if the algorithm can make random choices?

Randomness and algorithms

How can randomness be used in computation?

Where can randomness come into the picture?

Given some algorithm that solves a problem…

- What if the input is chosen randomly?
- What if the algorithm can make random choices?

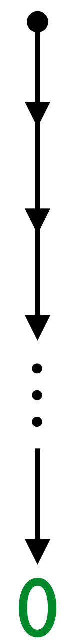
Randomness and algorithms

Let's allow the algorithm to flip a coin when it wants.
It can make decisions based on the outcomes of the flips.

We call such an algorithm a randomized algorithm.

Comparing with a deterministic algorithm:

In a deterministic algorithm, for a fixed input, computational steps are determined:



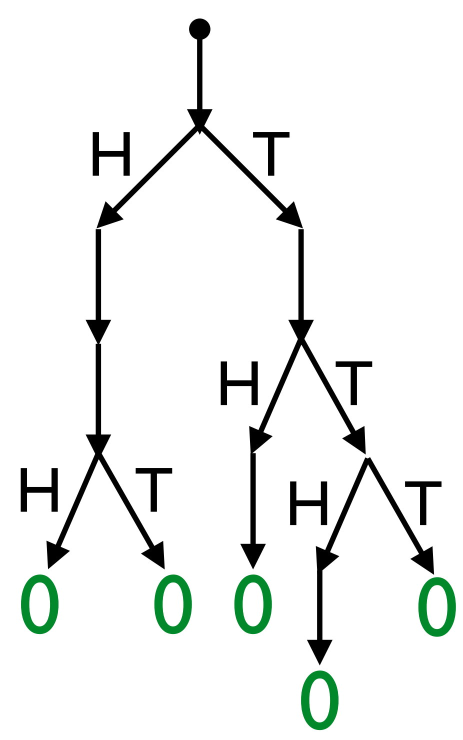
Randomness and algorithms

Let's allow the algorithm to flip a coin when it wants.
It can make decisions based on the outcomes of the flips.

We call such an algorithm a randomized algorithm.

Comparing with a deterministic algorithm:

In a randomized algorithm, for a fixed input:



Randomness and algorithms

Let's allow the algorithm to flip a coin when it wants.
It can make decisions based on the outcomes of the flips.

We call such an algorithm a randomized algorithm.

Why should we expect a randomized algorithm to be potentially useful?

Think about the power of population sampling.

Randomness and algorithms

An algorithm has 2 important parameters:

- correctness (or how correct it is)
- complexity (say with respect to running time)

If we ask our randomized algorithm to be

- always correct,
- always run in time O(T(n)),

then we have a deterministic alg. with time compl. O(T(n))

(take your randomized alg. and assume you always get Heads)

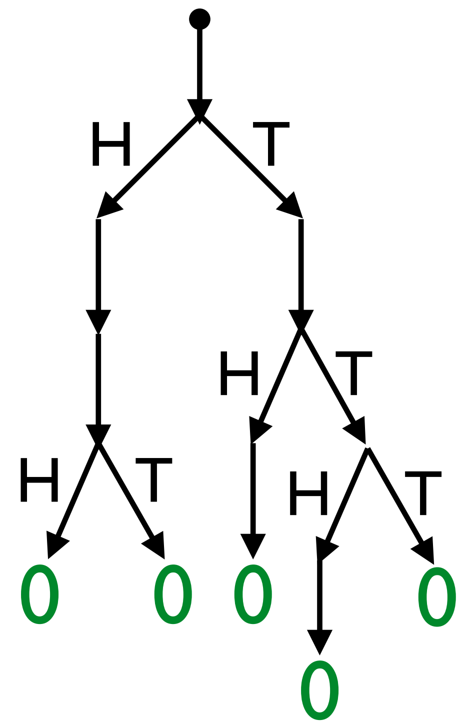
So for a randomized algorithm to be interesting:

- it is not correct all the time, **or**
- it doesn't always run in time O(T(n))

Randomness and algorithms

So for a randomized algorithm to be interesting:

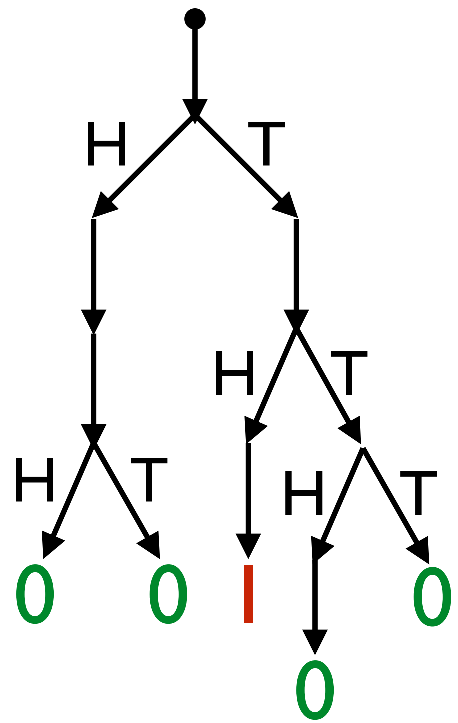
- it is not correct all the time, **or**
- it doesn't always run in time O(T(n))



Randomness and algorithms

So for a randomized algorithm to be interesting:

- it is not correct all the time, **or**
- it doesn't always run in time O(T(n))



Error probability: probability of red

Running time: length of the longest path

Types of randomized algorithms

2 Types:

- Monte Carlo algorithms

  > For every input, there is a certain probability of error.

  > There is a worst-case running time guarantee.

- Las Vegas algorithms

  > For every input, gives the correct answer.
  (worst-case correctness guarantee)

  > For every input, there is a certain probability that the running time is larger than desired or expected.

Example of a Monte Carlo Algorithm:
Min Cut

Example of a Las Vegas Algorithm:
Quicksort

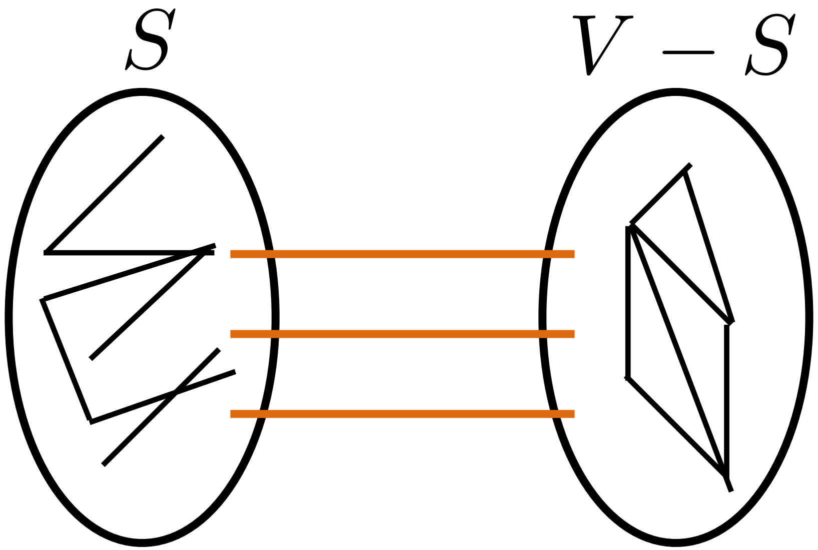
Example of a Monte Carlo Algorithm: Min Cut

Gambles with correctness.
Doesn't gamble with resources.

Cut Problems

Max Cut Problem (Ryan's favorite problem):

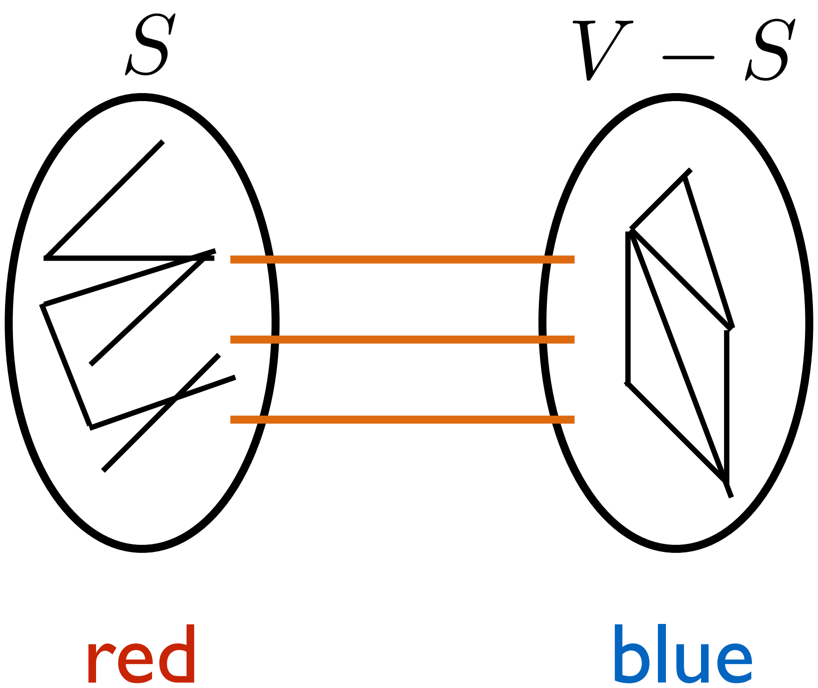
Given a graph $G = (V, E)$,
find a non-empty subset $S \subset V$ such that
number of edges from S to $V - S$ is maximized.



Cut Problems

Max Cut Problem (Ryan's favorite problem):

Given a graph $G = (V, E)$,
color the vertices red and blue so that the number of edges with two colors (e = {u,v}) is maximized.



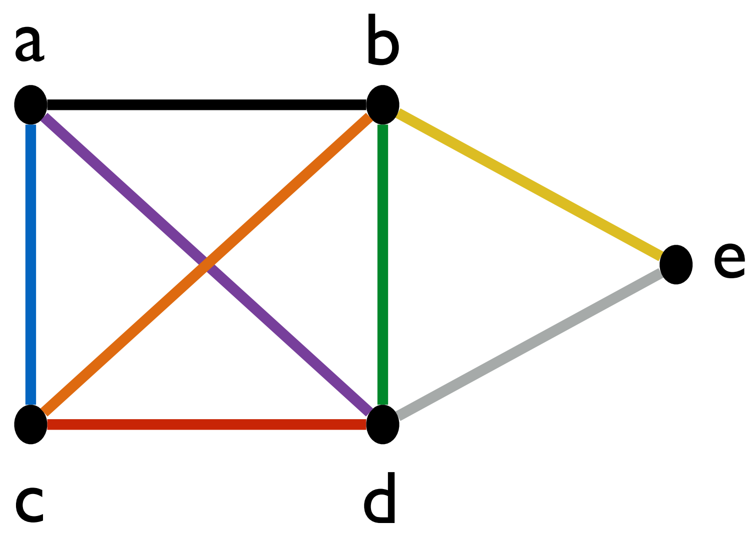
Cut Problems

Min Cut Problem (my favorite problem):

Given a graph $G = (V, E)$,
find a non-empty subset $S \subset V$ such that
number of edges from S to $V - S$ is *minimized*.

Let's see a super simple randomized algorithm for it.

Contraction algorithm for min cut



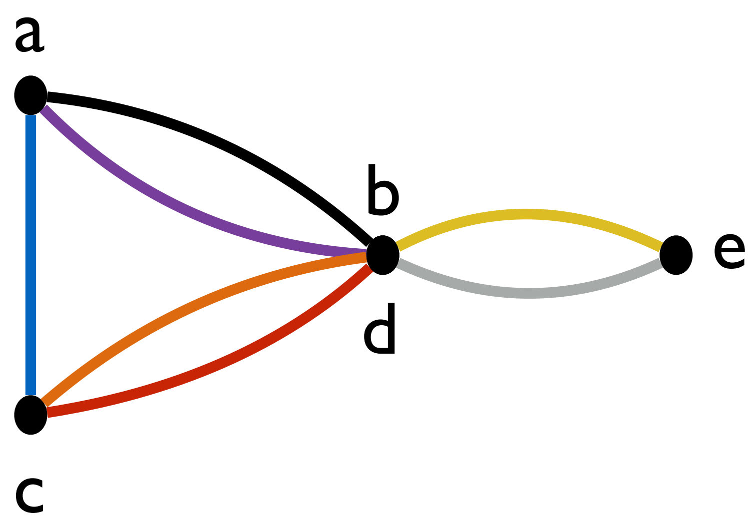
Select an edge randomly:

Size of min-cut: 2

Green edge selected.

Contract that edge.

Contraction algorithm for min cut



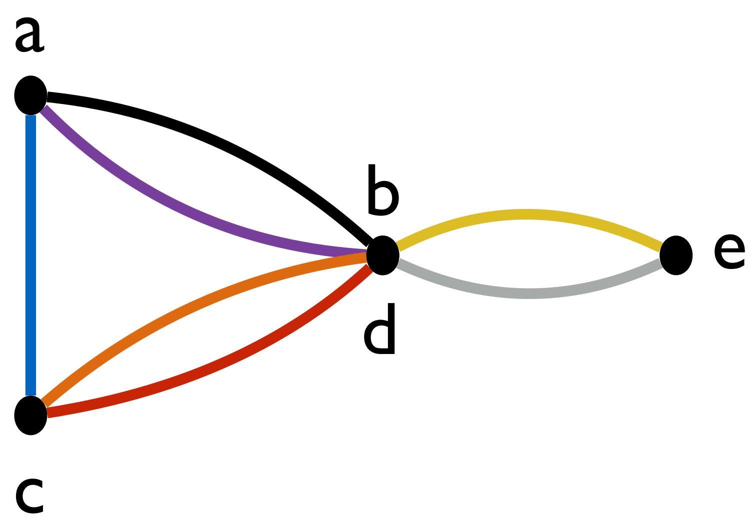
Select an edge randomly:

Size of min-cut: 2

Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

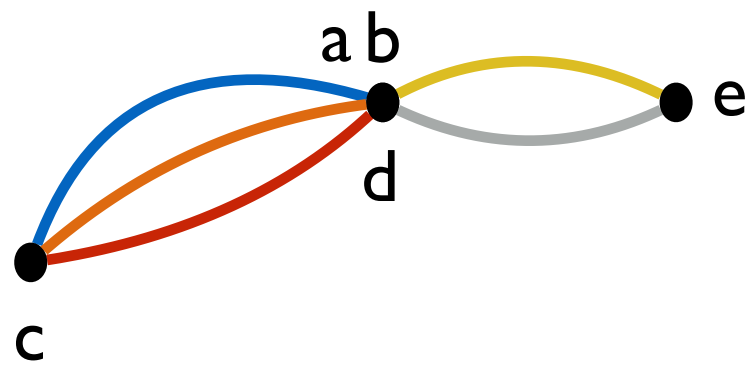


Select an edge randomly: *Size of min-cut: 2*

Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

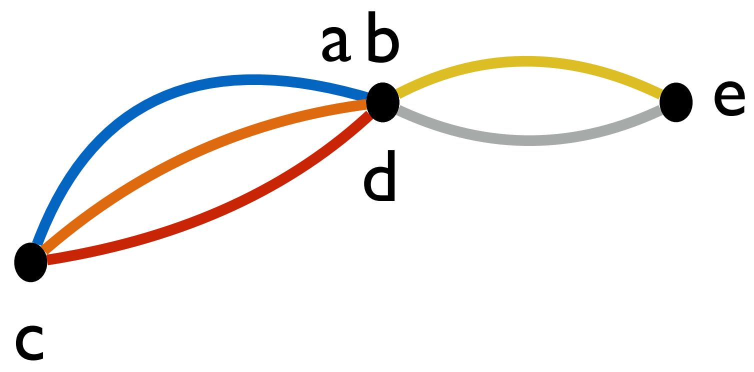


Select an edge randomly: *Size of min-cut: 2*

Purple edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



Select an edge randomly:

Size of min-cut: 2

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Select an edge randomly: *Size of min-cut: 2*

Blue edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

Select an edge randomly:

Size of min-cut: 2

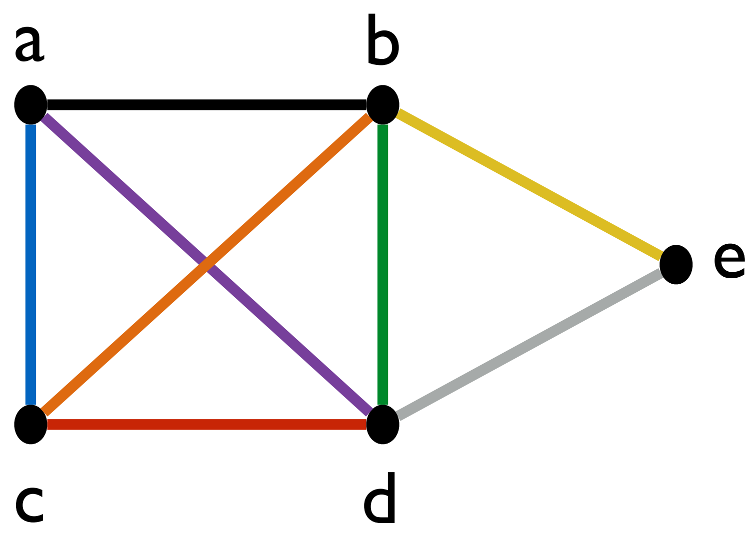
Blue edge selected.

Contract that edge. (delete self loops)

When two vertices remain, you have your cut:

{a, b, c, d} {e} size: 2

Contraction algorithm for min cut



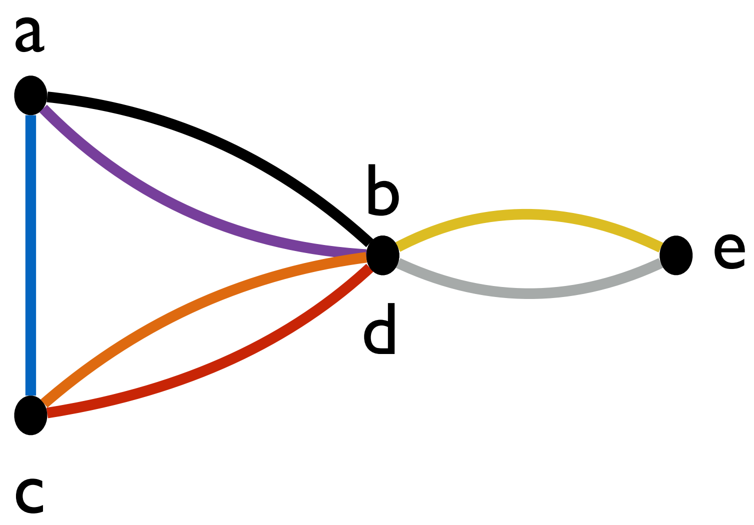
Select an edge randomly:

Size of min-cut: 2

Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



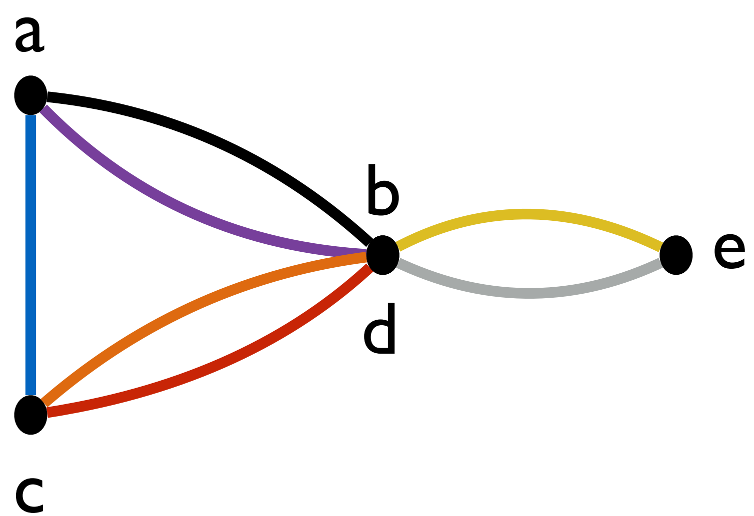
Select an edge randomly:

Size of min-cut: 2

Green edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



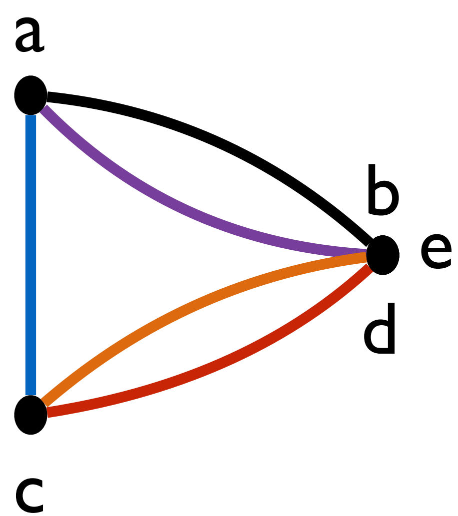
Select an edge randomly:

Size of min-cut: 2

Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



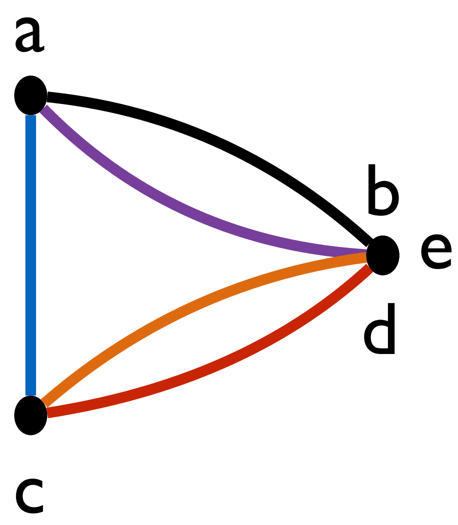
Select an edge randomly:

Size of min-cut: 2

Yellow edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



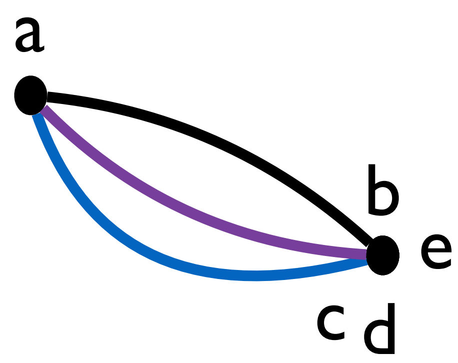
Select an edge randomly:

Size of min-cut: 2

Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut

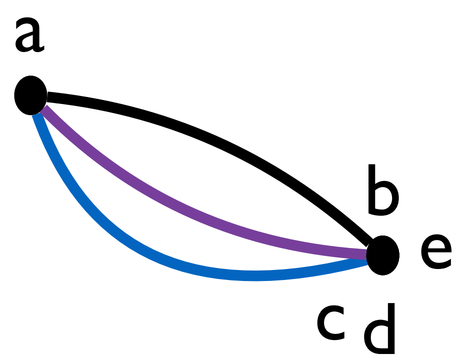


Select an edge randomly: *Size of min-cut: 2*

Red edge selected.

Contract that edge. (delete self loops)

Contraction algorithm for min cut



Select an edge randomly:

Size of min-cut: 2

Red edge selected.

Contract that edge. (delete self loops)

When two vertices remain, you have your cut:

{a} {b,c,d,e} size: 3

Contraction algorithm for min cut

Theorem:

Let $G = (V, E)$ be a graph with *n* vertices.
Fix some min cut in the graph. The probability that the contraction algorithm will output this cut is $2/n(n-1)$.

Should we be impressed?

- The algorithm runs in polynomial time.

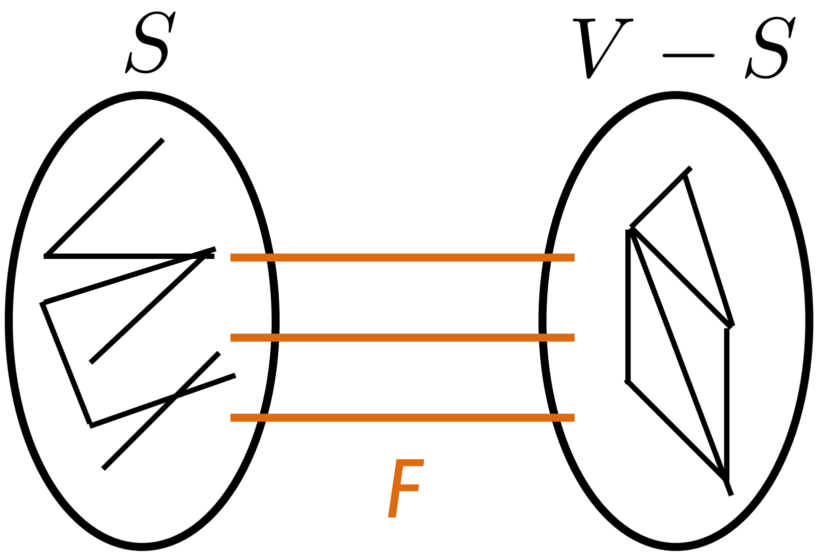
- There are exponentially many cuts. ($\approx 2^n$)

- There is a way to boost the probability of success to

$$1 - \frac{1}{e^n}$$ (and still remain in polynomial time)

Contraction algorithm for min cut

Proof of Theorem:



Fix some minimum cut.

$|F| = k$

$|V| = n$

$|E| = m$

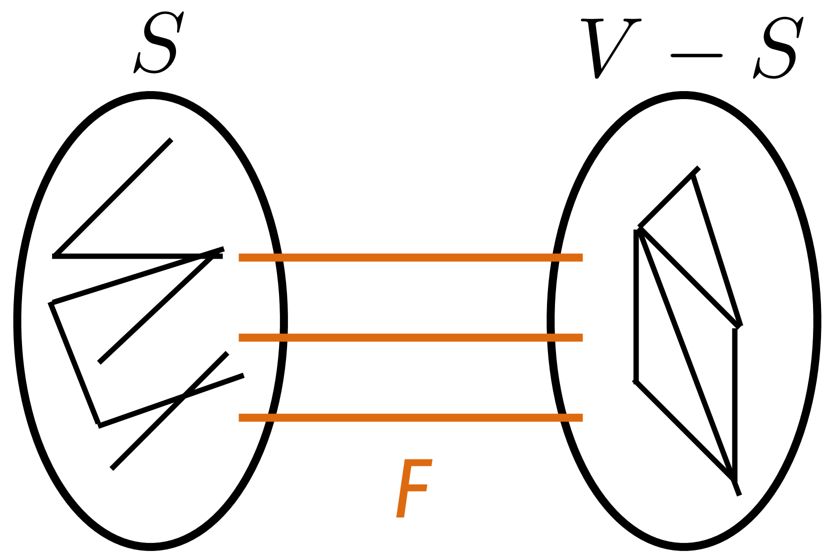
When does the algorithm make an error?
(How can it not end up with the above cut?)

What if the algorithm picks an edge in F to contract?
Then it cannot output F.

What if it never picks an edge in F to contract?
Then it will output F.

Contraction algorithm for min cut

<u>**Proof of Theorem:**</u>



Pr[alg. outputs *F*] =

Pr[alg. never contracts an edge in *F*]

<u>**Goal**</u>: Show this probability is at least 2 / *n*(*n*-1).

Let E_i = an edge in *F* is contracted in iteration *i*.

How many iterations are there? *n*-2

Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$]

Contraction algorithm for min cut

Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

$$= \Pr[\overline{E_1}] \cdot \Pr[\overline{E_2} | \overline{E_1}] \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots$$

$$\Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]$$

$$\Pr[E_1] = \frac{k}{m}$$

We actually want this in terms of *n* and not *m*.

Contraction algorithm for min cut

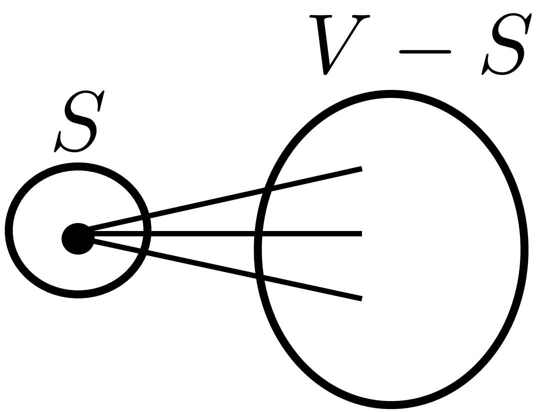
Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

Observation: $\forall v \in V : \deg(v) \geq k$

(if not, min-cut has less than *k* edges)



Recall: $\sum_{v \in V} \deg(v) = 2m \implies 2m \geq kn$

$$\Pr[E_1] = \frac{k}{m} \leq \frac{2}{n} \implies \Pr[\overline{E_1}] \geq \left(1 - \frac{2}{n}\right)$$

Contraction algorithm for min cut

Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$
$$\geq \left(1 - \frac{2}{n}\right) \cdot \Pr[\overline{E_2} | \overline{E_1}] \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots$$
$$\Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]$$

$$\Pr[\overline{E_2} | \overline{E_1}] = 1 - \Pr[E_2 | \overline{E_1}] = 1 - \frac{k}{\#\text{ remaining edges}}$$

again, want to write in terms of *k* and *n*

Contraction algorithm for min cut

Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

$$\Pr[\overline{E_2}|\overline{E_1}] = 1 - \Pr[E_2|\overline{E_1}] = 1 - \frac{k}{\#\text{ remaining edges}}$$

again, want to write in terms of *k* and *n*

Observation: At every point in the algorithm

$$\forall v \in V : \deg(v) \geq k$$

(if not, min-cut has less than *k* edges)

After one contraction: $2m' \geq k(n-1)$

$$\#\text{ remaining edges} \geq k(n-1)/2$$

Contraction algorithm for min cut

Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

$$\Pr[\overline{E_2} | \overline{E_1}] = 1 - \Pr[E_2 | \overline{E_1}] = 1 - \frac{k}{\text{\# remaining edges}}$$

$$\geq 1 - \frac{k}{k(n-1)/2} = 1 - \frac{2}{n-1}$$

Contraction algorithm for min cut

Proof of Theorem:

Let E_i = an edge in *F* is contracted in iteration *i*.

Goal: Pr[$\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}$] is at least 2 / *n*(*n*-1).

$$\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}]$$

$$\geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots$$

$$\Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{n-(n-4)}\right)\left(1 - \frac{2}{n-(n-3)}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) = 2/n(n-1) \qquad \square$$

Contraction algorithm for min cut

Theorem:

Let $G = (V, E)$ be a graph with *n* vertices.
Fix some min cut in the graph. The probability that the contraction algorithm will output this cut is $2/n(n-1)$.

Should we be impressed?

- The algorithm runs in polynomial time.

- There are exponentially many cuts. ($\approx 2^n$)

- There is a way to boost the probability of success to

$$1 - \frac{1}{e^n}$$ (and still remain in polynomial time)

Contraction algorithm for min cut

Theorem:

Let $G = (V, E)$ be a graph with *n* vertices.
Fix some min cut in the graph. The probability that the contraction algorithm will output this cut is $2/n(n-1)$.

Should we be impressed?

- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\approx 2^n$)
- There is a way to boost the probability of success to

$$1 - \frac{1}{e^n}$$ (and still remain in polynomial time)

Boosting success by repeated trials

Run the algorithm *t* times using fresh random bits.

Output the smallest cut among the ones you find.

What is the relation between *t* and success probability?

Again, fix some minimum cut.

Let A_i = in the i'th repetition, we <u>don't</u> find this min cut.

$$\Pr[\text{fail to find this cut}] = \Pr[A_1 \cap A_2 \cap \cdots \cap A_t]$$

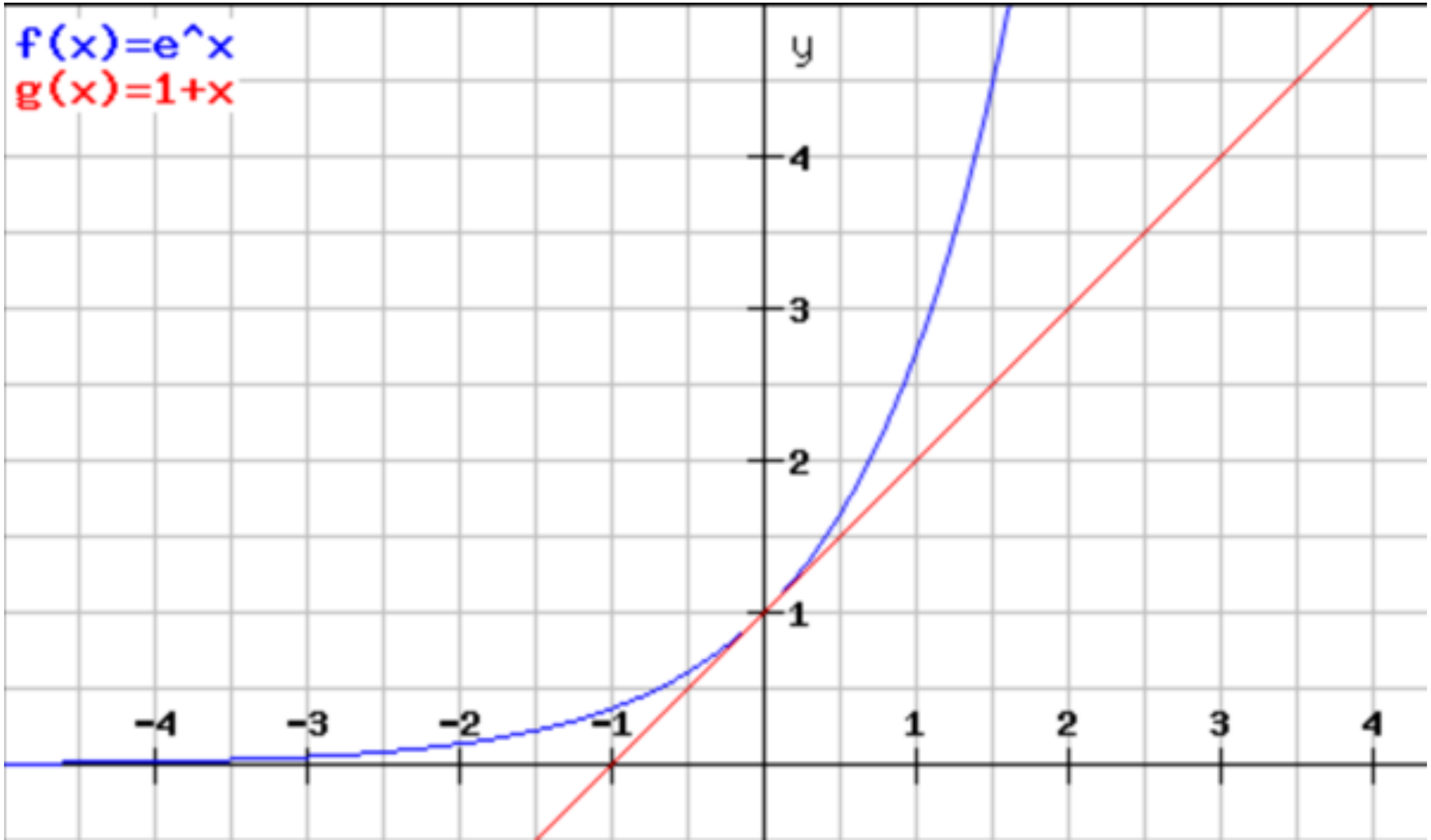
$$= \Pr[A_1]\Pr[A_2]\cdots\Pr[A_t] = \Pr[A_1]^t$$

$$\leq \left(1 - \frac{2}{n(n-1)}\right)^t \leq \left(1 - \frac{1}{n^2}\right)^t$$

Boosting success by repeated trials

$$\Pr[\text{error}] \leq \left(1 - \frac{1}{n^2}\right)^t$$

Extremely useful inequality: $\forall x \in \mathbb{R}: \; 1 + x \leq e^x$



Boosting success by repeated trials

$$\Pr[\text{error}] \leq \left(1 - \frac{1}{n^2}\right)^t$$

Extremely useful inequality: $\forall x \in \mathbb{R}: \; 1 + x \leq e^x$

Take $x = -1/n^2$

$$\Pr[\text{error}] \leq \left(e^{-1/n^2}\right)^t \quad = e^{-t/n^2}$$

$$t = n^2 \qquad \Longrightarrow \Pr[\text{error}] \leq 1/e$$

$$t = cn^2 \qquad \Longrightarrow \Pr[\text{error}] \leq 1/e^c$$

$$t = n^2 \ln n \qquad \Longrightarrow \Pr[\text{error}] \leq 1/n$$

$$t = n^3 \qquad \Longrightarrow \Pr[\text{error}] \leq 1/e^n$$

Boosting success by repeated trials

We can always boost the success probability of Monte Carlo algorithms via repeated trials.

Conclusion for min cut

We have a polynomial time algorithm that solves the min cut problem with probability $1 - 1/e^n$.

↓

Theoretically, not equal to 1.
Practically, equal to 1.

Example of a Las Vegas Algorithm: Quicksort

Doesn't gamble with correctness.
Gambles with resources.

Quicksort Algorithm

4	8	2	7	99	5	0

On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S

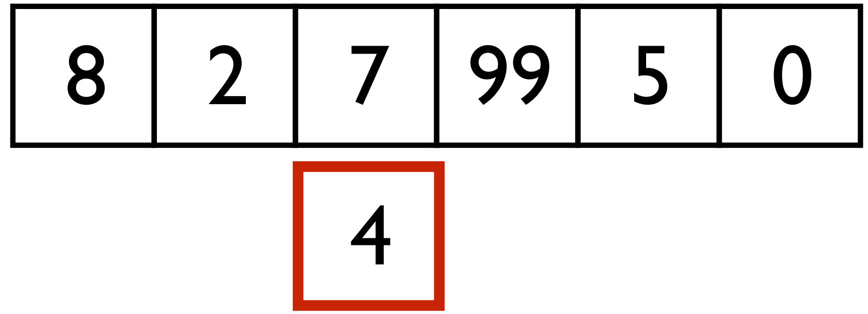
Quicksort Algorithm

4	8	2	7	99	5	0

On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m

Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m

Quicksort Algorithm

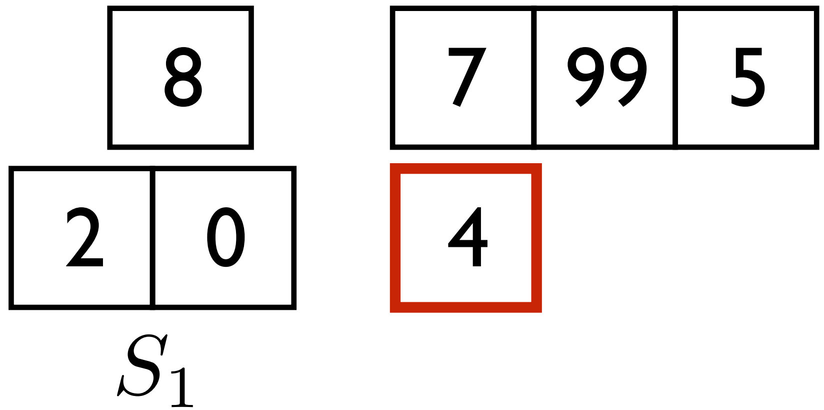
8	2	7	99	5	0

4

On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$

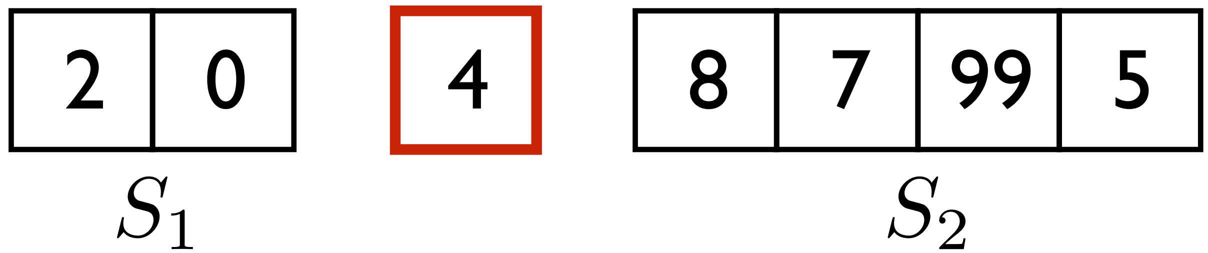
Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$

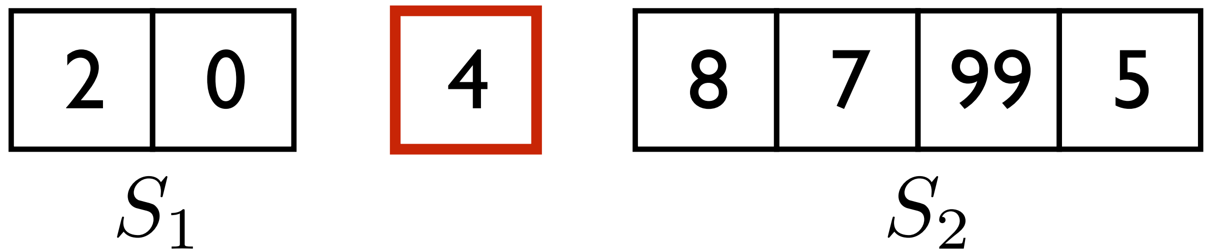
Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$

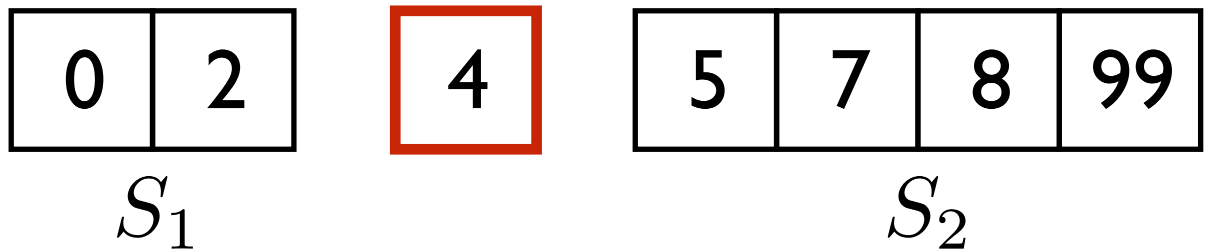
Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
- Recursively sort S_1 and S_2 .

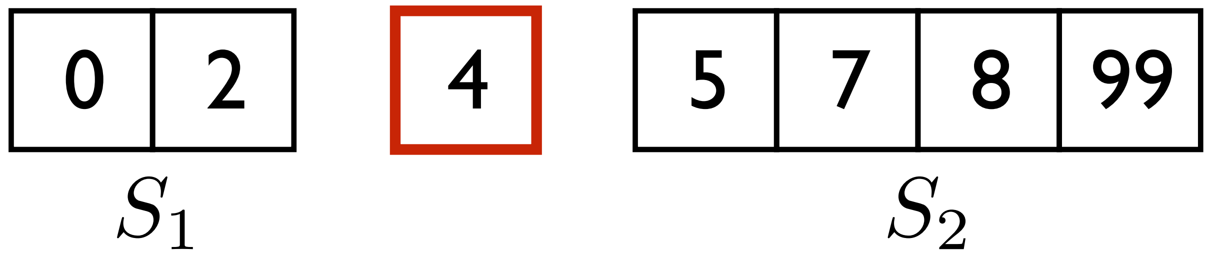
Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
- Recursively sort S_1 and S_2 .

Quicksort Algorithm



On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return S
- Pick uniformly at random a "pivot" x_m
- Compare x_m to all other x's
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
- Recursively sort S_1 and S_2 .
- Return $[S_1, x_m, S_2]$

Quicksort Algorithm Analysis

This is a Las Vegas algorithm:

- always gives the correct answer
- running time can vary depending on our luck

Quicksort Algorithm Analysis

Worst case scenario:

Suppose we always end up picking the first element as the pivot.

For an input like

7	6	5	4	3	2	1

how many comparisons would we make?

$$(n-1)+(n-2)+\cdots+2+1 = \frac{n(n-1)}{2} = \Omega(n^2)$$

Recursive relation for the number of comparisons:

$$T(n) = T(n-1) + (n-1)$$

Quicksort Algorithm Analysis

Best case scenario:

What is the best choice of pivot?

No matter which pivot you choose,
you'll make |S|-1 comparisons before the recursive calls.

Total number of comparisons:

$$T(n) = T(|S_1|) + T(|S_2|) + (n-1)$$

$$T(n) = T(k) + T(n-k-1) + (n-1)$$

For $k \approx n/2, \quad T(n) = O(n \log n).$

Quicksort Algorithm Analysis

For fun, let's look at the expected number of comparisons.

Let X = number of comparisons

What is $\mathbf{E}[X]$?

How can we bound $\mathbf{E}[X]$?

Indicator r.v.'s + Linearity of expectation

Quicksort: Expected number of comparisons

Indicator r.v.'s + Linearity of expectation

Let X = number of comparisons

We want to write X as a sum of indicator r.v.'s.

$$X = \sum_{i=1}^{k} X_i \; , \qquad X_i = \begin{cases} 1 & \text{if event } E_i \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

Then use linearity of expectation:

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_i X_i\right] = \sum_i \mathbf{E}[X_i] = \sum_i \mathbf{Pr}[E_i]$$

$$\left(\; \mathbf{E}[X_i] = 1 \cdot \mathbf{Pr}[X_i = 1] + 0 \cdot \mathbf{Pr}[X_i = 0] \; \right)$$

Quicksort: Expected number of comparisons

Indicator r.v.'s + Linearity of expectation

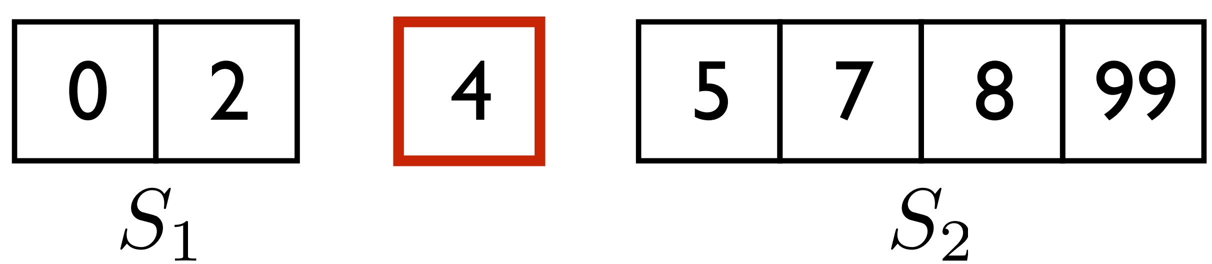
Let X = number of comparisons

We want to write X as a sum of indicator r.v.'s.

Let $X_{ij} =$ # time x_i and x_j get compared.

So: $X = \sum_{1 \le i < j \le n} X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$

How many times do x_i and x_j get compared? 0 or 1



Quicksort: Expected number of comparisons

Indicator r.v.'s + Linearity of expectation

Let X = number of comparisons

We want to write X as a sum of indicator r.v.'s.

Let $X_{ij} =$ # time x_i and x_j get compared.

So: $$X = \sum_{1 \le i < j \le n} X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

$$X_{ij} = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

$$\implies \mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[x_i \text{ and } x_j \text{ are compared}]$$

Quicksort: Expected number of comparisons

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[x_i \text{ and } x_j \text{ are compared}]$$

Let $y_1, y_2, \ldots, y_n$ be the input elements in sorter order.

x_1	x_2	x_3	x_4	x_5	x_6	x_7
4	8	2	7	99	5	0
y_3	y_6	y_2	y_5	y_7	y_4	y_1

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}]$$

Quicksort: Expected number of comparisons

$$\mathbf{E}[X] = \sum_{1 \leq i < j \leq n} \mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}]$$

Claim: $\mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}] = \frac{2}{j-i+1}$

Proof: Define $Y^{ij} = \{y_i, y_{i+1}, \ldots, y_j\}$

Consider the algorithm when a pivot p is being chosen.

Case 1: $p \notin Y^{ij} \implies Y^{ij} \subseteq S_1$ or $Y^{ij} \subseteq S_2$

Case 2: $p \in Y^{ij}$ but $p \neq y_i$ and $p \neq y_j$

$\implies y_i \in S_1$ and $y_j \in S_2$ (y_i and y_j never compared)

Case 3: $p = y_i$ or $p = y_j$ (y_i and y_j are compared)

Quicksort: Expected number of comparisons

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}]$$

<u>**Claim:**</u> $\mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}] = \dfrac{2}{j-i+1}$

<u>**Proof:**</u> Define $Y^{ij} = \{y_i, y_{i+1}, \ldots, y_j\}$

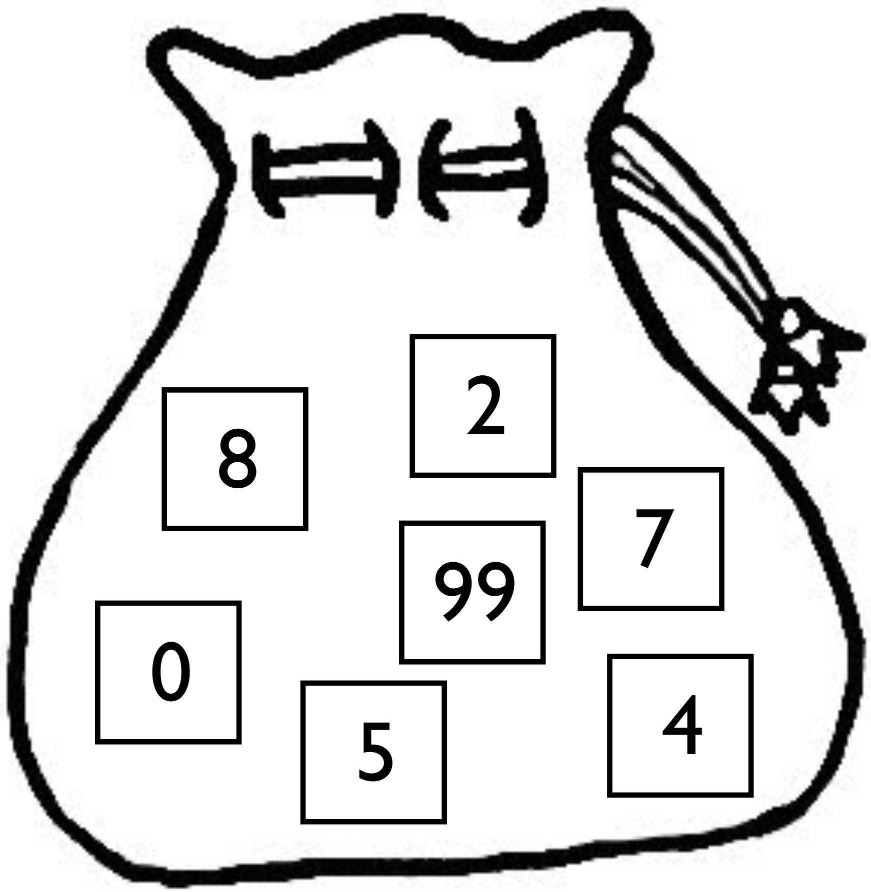
Conclusion: y_i and y_j get compared iff

y_i or y_j is chosen as pivot before $y_{i+1}, y_{i+2}, \ldots, y_{j-1}$

i.e., the first pivot from Y^{ij} was y_i or y_j.

What is the probability of this? $\dfrac{2}{|Y^{ij}|} = \dfrac{2}{j-i+1}$

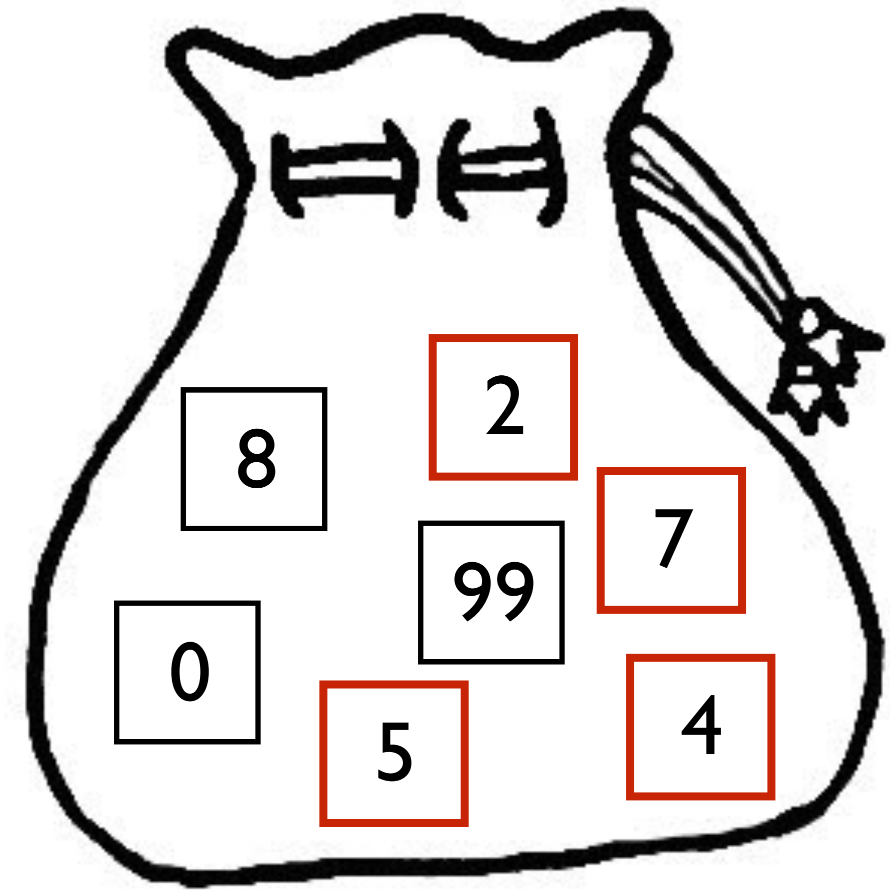
Quicksort: Expected number of comparisons



S

What is the probability 2 and 7 get compared?

Quicksort: Expected number of comparisons

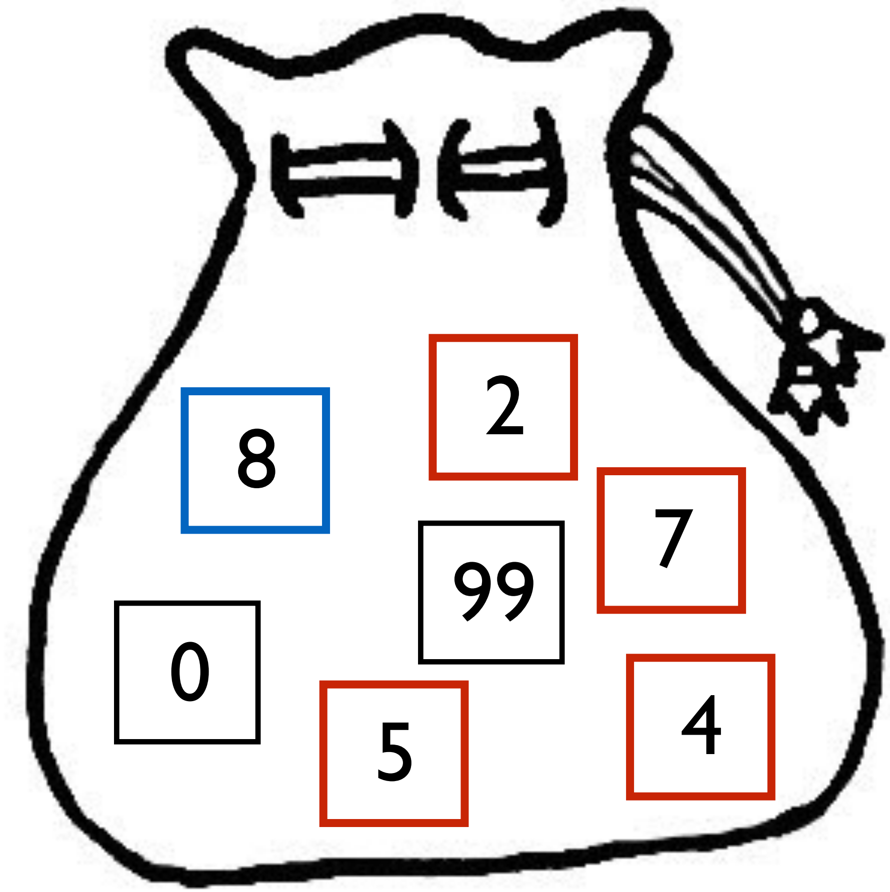


S

What is the probability 2 and 7 get compared?

What is the probability you pick 2 or 7 before 4 or 5?

Quicksort: Expected number of comparisons

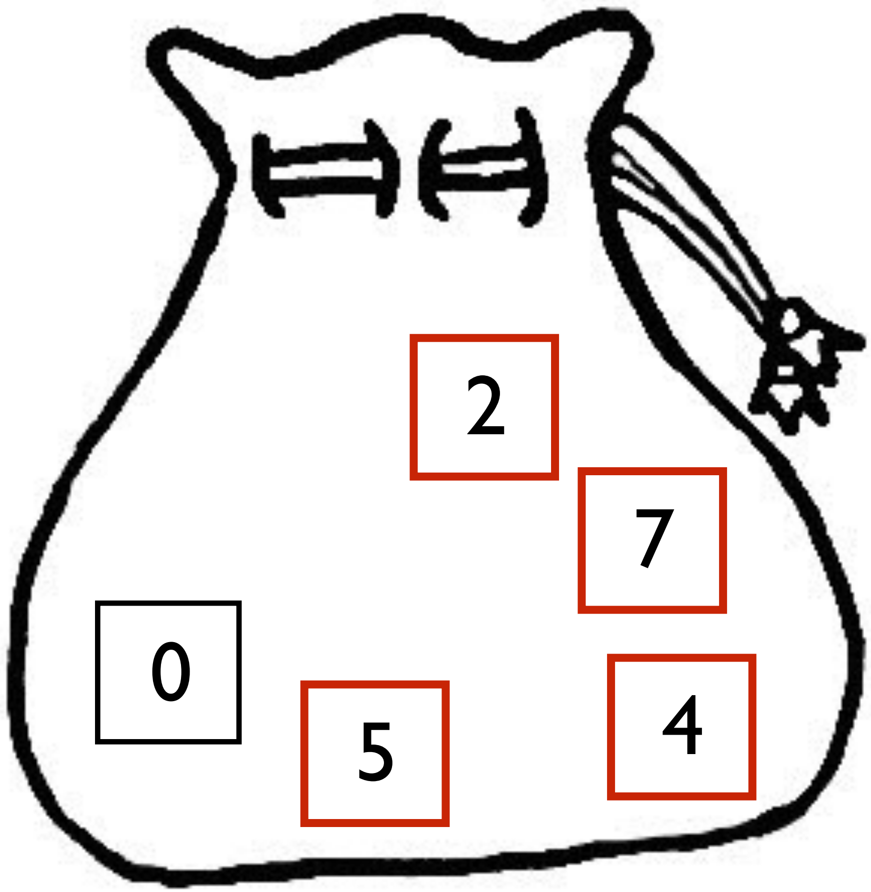


S

What is the probability 2 and 7 get compared?

What is the probability you pick 2 or 7 before 4 or 5?

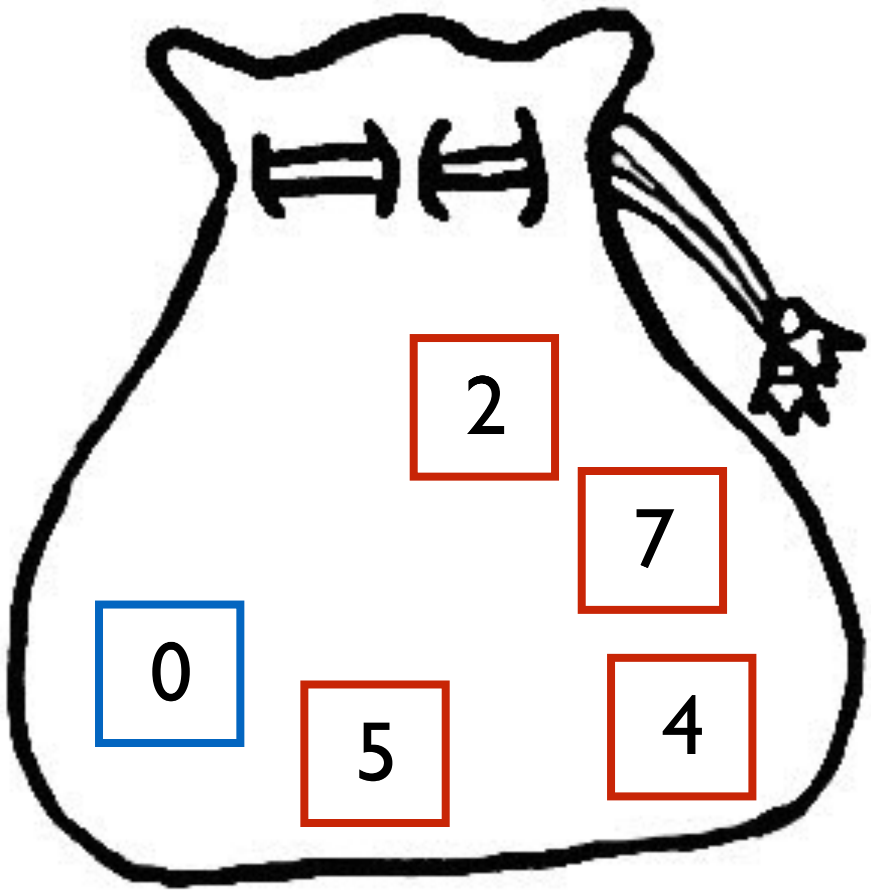
Quicksort: Expected number of comparisons



What is the probability
2 and 7 get compared?

What is the probability
you pick 2 or 7 before 4 or 5?

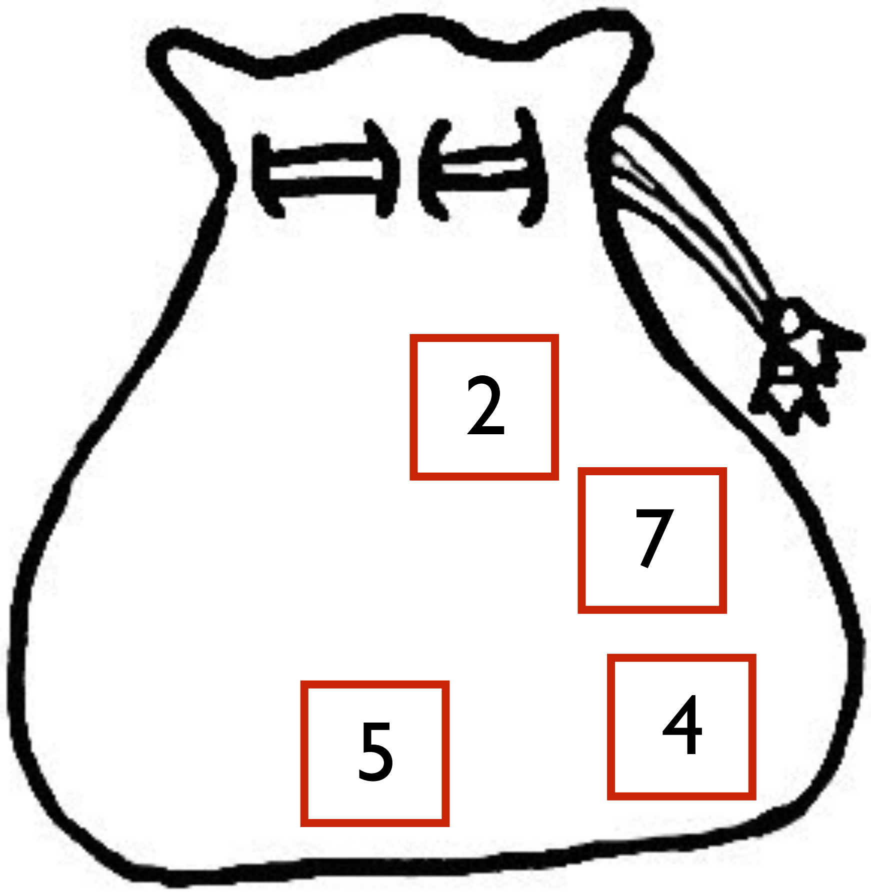
Quicksort: Expected number of comparisons



What is the probability 2 and 7 get compared?

What is the probability you pick 2 or 7 before 4 or 5?

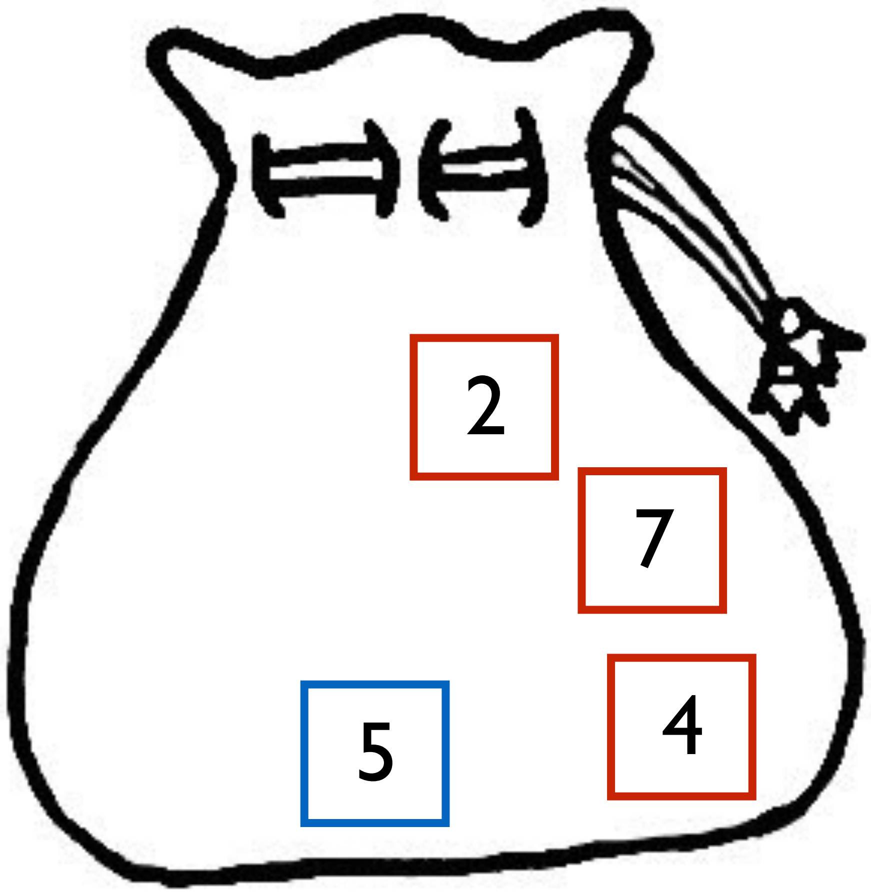
Quicksort: Expected number of comparisons



What is the probability
2 and 7 get compared?

What is the probability
you pick 2 or 7 before 4 or 5?

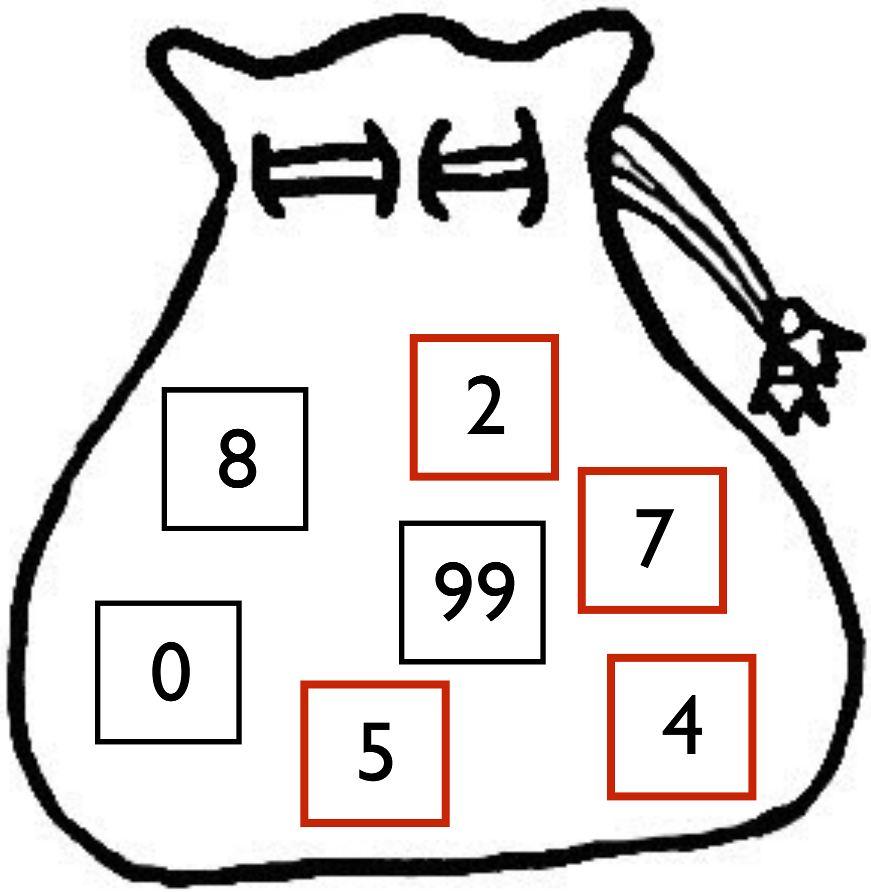
Quicksort: Expected number of comparisons



What is the probability
2 and 7 get compared?

What is the probability
you pick 2 or 7 before 4 or 5?

Quicksort: Expected number of comparisons



What is the probability 2 and 7 get compared?

What is the probability you pick 2 or 7 before 4 or 5?

Let $E_{2,7}$ be this event.

Let R_i be the event that a red element is picked in i'th trial.

$$\mathbf{Pr}[E_{2,7}] = \mathbf{Pr}[E_{2,7}|R_1]\mathbf{Pr}[R_1] + \mathbf{Pr}[E_{2,7}|R_2]\mathbf{Pr}[R_2] + \cdots$$
$$= \mathbf{Pr}[E_{2,7}|R_1](\mathbf{Pr}[R_1] + \mathbf{Pr}[R_2] + \cdots) = 2/4$$

Quicksort: Expected number of comparisons

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}]$$

Claim: $\mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}] = \frac{2}{j-i+1}$

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = 2 \sum_{1 \le i < j \le n} \frac{1}{j-i+1}$$

$$\begin{aligned}
\sum_{1 \le i < j \le n} \frac{1}{j-i+1} &= \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n} && (i = 1) \\
&+ \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} && (i = 2) \\
&+ \cdots && \vdots \\
&+ \frac{1}{2} && (i = n-1)
\end{aligned}$$

Quicksort: Expected number of comparisons

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}]$$

Claim: $\mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}] = \dfrac{2}{j-i+1}$

$$\mathbf{E}[X] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = 2 \sum_{1 \le i < j \le n} \frac{1}{j-i+1}$$

$$\begin{aligned}\sum_{1 \le i < j \le n} \frac{1}{j-i+1} &= \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n} \qquad (\le \ln n) \\ &+ \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} \\ &+ \cdots \\ &+ \frac{1}{2}\end{aligned}$$

So: $\mathbf{E}[X] \le 2n \ln n$
$= O(n \log n)$

Quicksort number of comparisons

From expectation to probability.

We know expected number of comparisons is $\leq 2n \ln n$.

Can we also conclude that with high probability, number of comparisons is $O(n \log n)$?

Yes. And it could be a good homework question…

Conclusion for Quicksort

We have a sorting algorithm that always gives the correct answer, and makes O(n log n) comparisons with high probability.

Final remarks

Randomness adds an interesting dimension to computation.

Often, randomized algorithms are faster and much more elegant than their deterministic counterparts.

There are some problems for which:
- there is a poly-time randomized algorithm,
- we can't find a poly-time deterministic algorithm.

Another million dollar question:

Does every efficient randomized algorithm have an efficient deterministic counterpart?

Is $\mathsf{P} = \mathsf{BPP}$?