Data Structures and Algorithms

Probability in Computer Science

CS 225

Brad Solomon

November 13, 2026





UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Formalize the concept of randomized algorithms

Review fundamentals of probability in computing

Distinguish the three main types of 'random' in computer science

↳ Blad's opinion

Observe some theory (proof) tips and tricks on a variety of contexts

Randomized Algorithms

A **randomized algorithm** is one which uses a source of randomness somewhere in its implementation.

MinHash

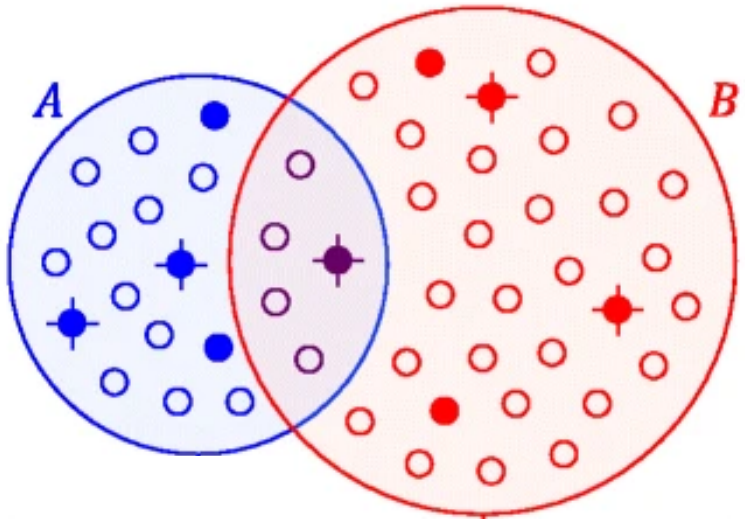
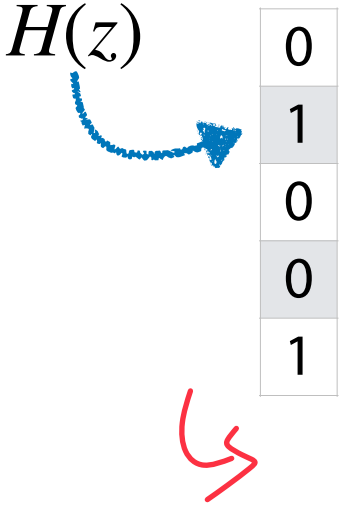


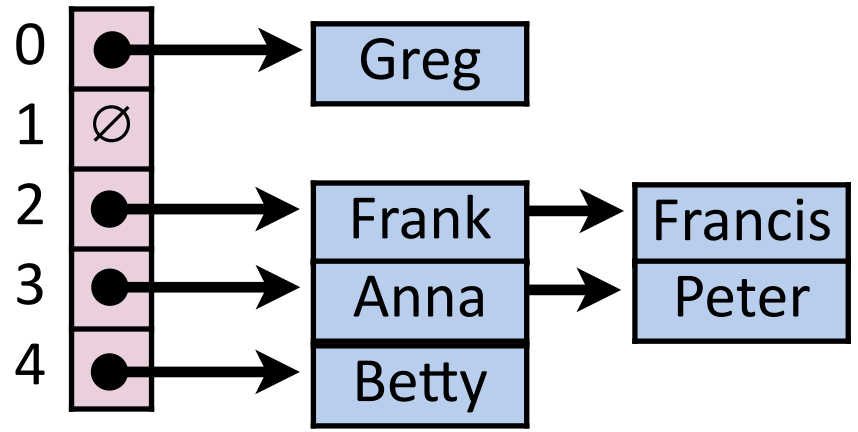
Figure from Ondov et al 2016

Set Similarity

Bloom Filter membership testing



Hash table



count of membership hits

$H(x)$	0	2	1	0	0	4	0	2	0	6
$H(y)$	1	0	2	3	1	0	3	4	0	1
$H(z)$	2	1	0	2	0	1	0	0	7	2

Skip List

A faulty list

Imagine you have a list ADT implementation ***except***…

Every time you called **insert**, it would fail 50% of the time.

1) Have dataset, want to filter out half randomly
   - Is Computational Biology
   - 100x of good things
   - 1-5x of bad things

$(.5)^{100}$

2) Website caching

3) Approximate counting

Quick Primes with Fermat's Primality Test

→ inaccurate but fast!

If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$

$[2, n-1]$

But… ***sometimes*** if n is composite and $a^{n-1} \equiv 1 \pmod{n}$

randomly by choice

Trade off

→ Minor inaccuracy for massive speed gain

$O(1)$

$a = 2$

21,853 non prime #s that equal 1

$25 \cdot 10^9$ #s

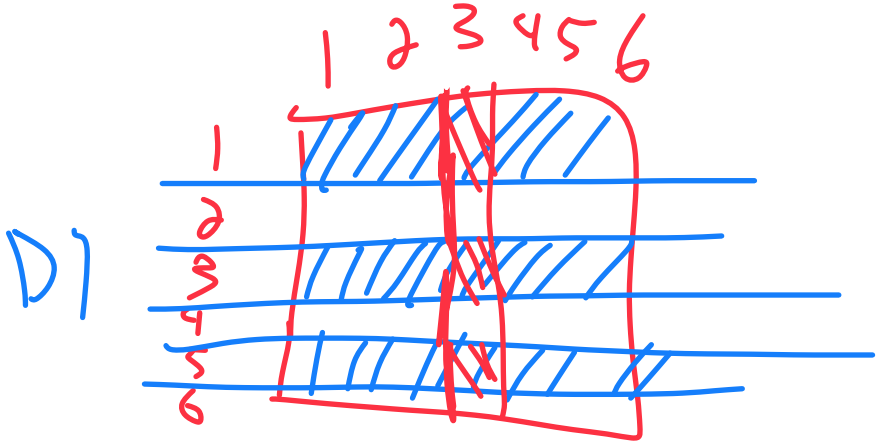
Fundamentals of Probability

Hash tables used this

Imagine you roll a pair of six-sided dice.

The **sample space** Ω is the set of all possible outcomes.

static



An **event** $E \subseteq \Omega$ is any subset.

D1 is odd

D2 is 3

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

A **random variable** is a function from events to numeric values.

Let D_1 be value of dice 1

D_2 2

let X be $D_1 + D_2$

The **expectation** of a (discrete) random variable is:

$$E[X] = \sum_{x \in \Omega} Pr\{X = x\} \cdot x$$

likelihood · value

for each state in universe

$\frac{1}{36} \cdot 2 + \frac{2}{36} \cdot 3 + \ldots$

$E[D_1] = \frac{1}{6} \cdot 1 + \frac{2}{6} + \frac{3}{6} + \frac{4}{6} + \frac{5}{6} + \frac{6}{6}$

$= 3.5$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$E[X + Y] = E[X] + E[Y]$ **(Claim)**

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

$$E[X + Y] = \sum_x \sum_y Pr\{X = x, Y = y\}(x + y)$$

prob of event

value of event

split X + Y into E_X E_Y

Fundamentals of Probability

$\sum_{Y \in W} Prob(Y) = 1$

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

$$E[X + Y] = \sum_x \sum_y Pr\{X = x, Y = y\}(x + y)$$

$$= \sum_x x \sum_y Pr\{X = x, Y = y\} + \sum_y y \sum_x Pr\{X = x, Y = y\}$$

$\sum_y$ = 1

$\sum_x$ = 1

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X+Y] = E[X] + E[Y]$$

$$E[X+Y] = \sum_x \sum_y Pr\{X=x, Y=y\}(x+y)$$

$$= \sum_x x \sum_y Pr\{X=x, Y=y\} + \sum_y y \sum_x Pr\{X=x, Y=y\}$$

$$= \sum_x x \cdot Pr\{X=x\} + \sum_y y \cdot Pr\{Y=y\} = E[X] + E[Y]$$

Fundamentals of Probability



Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

Randomization in Algorithms

1. **Assume input data is random to estimate average-case performance**

2. Use randomness inside algorithm to estimate expected running time

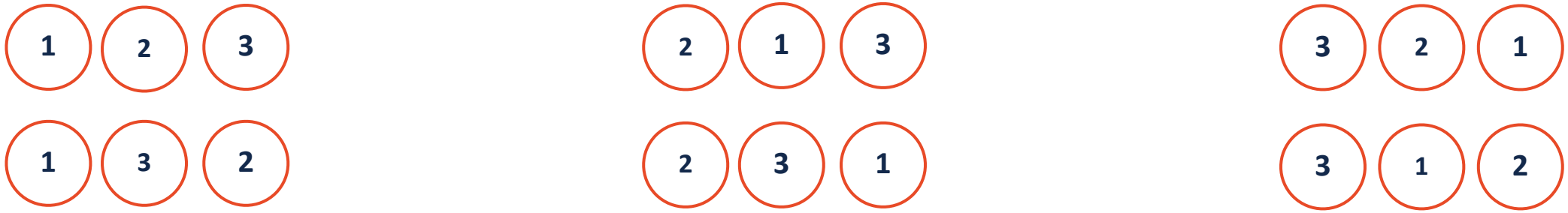
3. Use randomness inside algorithm to approximate solution in fixed time

Average-Case Analysis: BST

Let $S(n)$ be the average **total internal path length** over all BSTs that can be constructed by uniform random insertion of n objects

Claim: $S(n)$ is $O(n \ log \ n)$

N=3: AllBuild() with every possible permutation of insert order

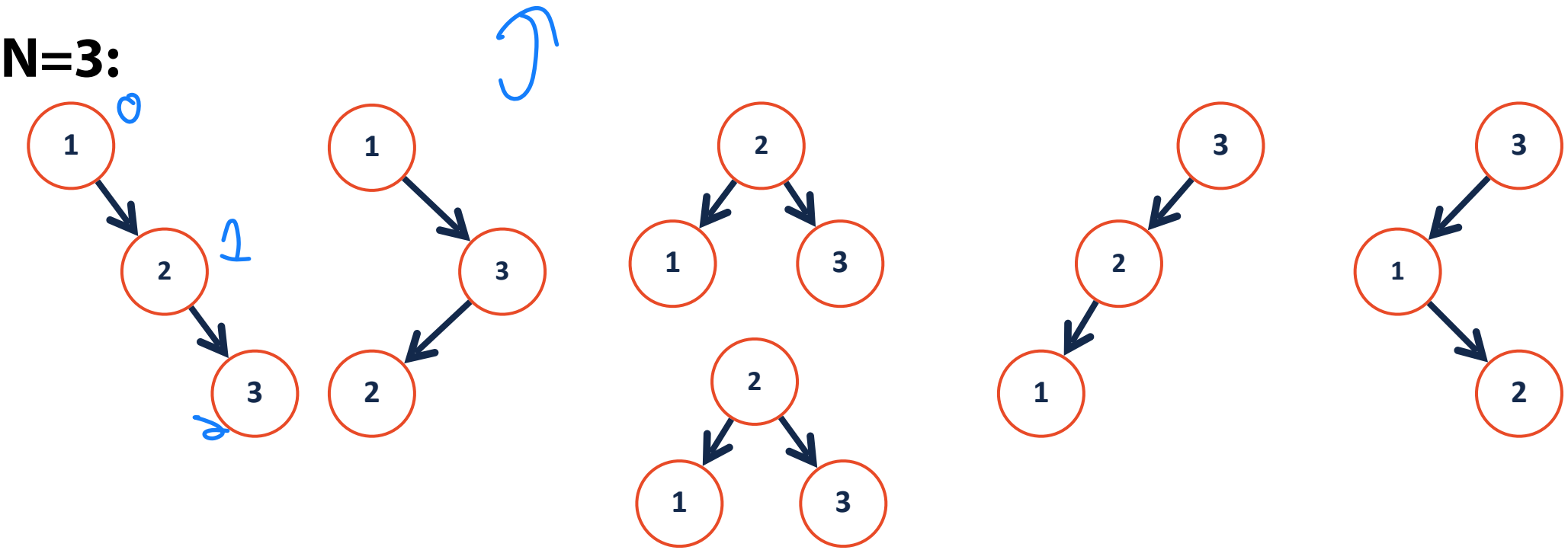


Average-Case Analysis: BST

Let $S(n)$ be the average **total internal path length** over all BSTs that can be constructed by uniform random insertion of n objects

Claim: $S(n)$ is $O(n \ log \ n)$

N=3:



Average-Case Analysis: BST

Let $S(n)$ be the **average** total internal path length **over all BSTs** that can be constructed by uniform random insertion of n objects

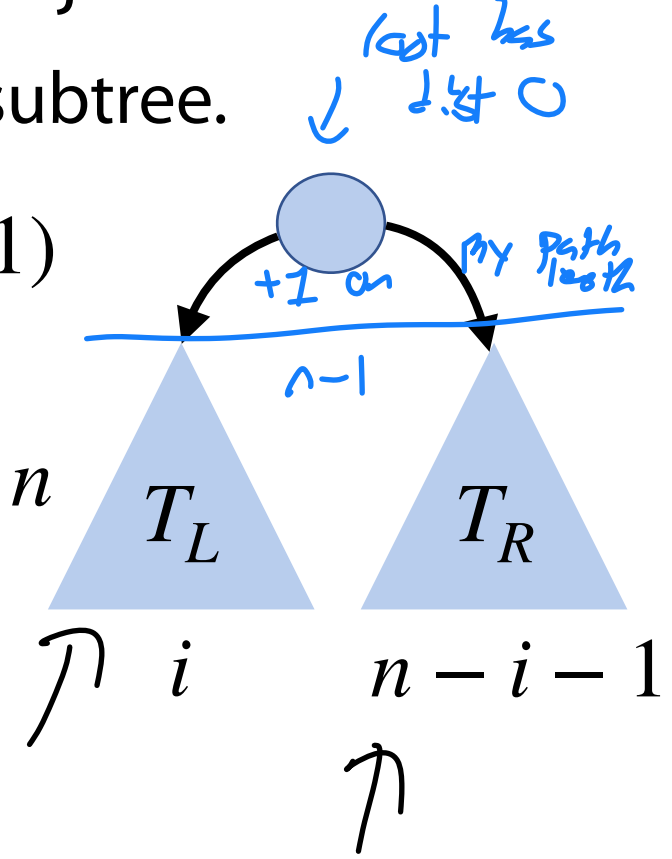
Let $0 \leq i \leq n-1$ be the number of nodes in the left subtree.

Then for a fixed i, $S(n) = (n-1) + S(i) + S(n-i-1)$

left subtree

right subtree

$$S(n) = (n-1) + \frac{1}{n}\sum_{i=0}^{n-1} S(i) + S(n-i-1) \approx cn \ ln \ n$$



Here's a slide of math you should not bother learning
(in the context of CS 225)

$$S(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} S(i)$$ (1) Guess recurrence form $S(i) = c * i\ ln(i)$

$$S(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} (ci\ ln\ i)$$ (2) Plug in recurrence

$$S(n) \leq (n-1) + \frac{2}{n}\int_{1}^{n} (cx\ ln\ x)dx$$ (3) $\sum_{i=1}^{n-1} f(i) \equiv \int_{1}^{n} f(x)dx$

$$S(n) \leq (n-1) + \frac{2}{n}\left(\frac{cn^2}{2}\ ln\ n - \frac{cn^2}{4} + \frac{c}{4}\right) \approx cn\ ln\ n$$

(4) $\int (cx\ lnx)\ dx$ can be expanded as shown above.

Average-Case Analysis: BST

Let $S(n)$ be the average **total internal path length** over all BSTs that can be constructed by uniform random insertion of n objects

$S(n) \approx (n \ log \ n)$ is provable but a weak argument! **Why?**

Source of my randomness?

Assumptions have I made?

Average-Case Analysis: BST



Let $S(n)$ be the average **total internal path length** over all BSTs that can be constructed by uniform random insertion of n objects

$S(n) \approx (n\ log\ n)$ is provable but a weak argument! **Why?**

Randomness: Input dataset is considered random

Arguably to extend analysis to 'find' we also assume query is random.

Assumptions: Input dataset is uniform random in content and order

Same assumptions then extended to query

Randomization in Algorithms

1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

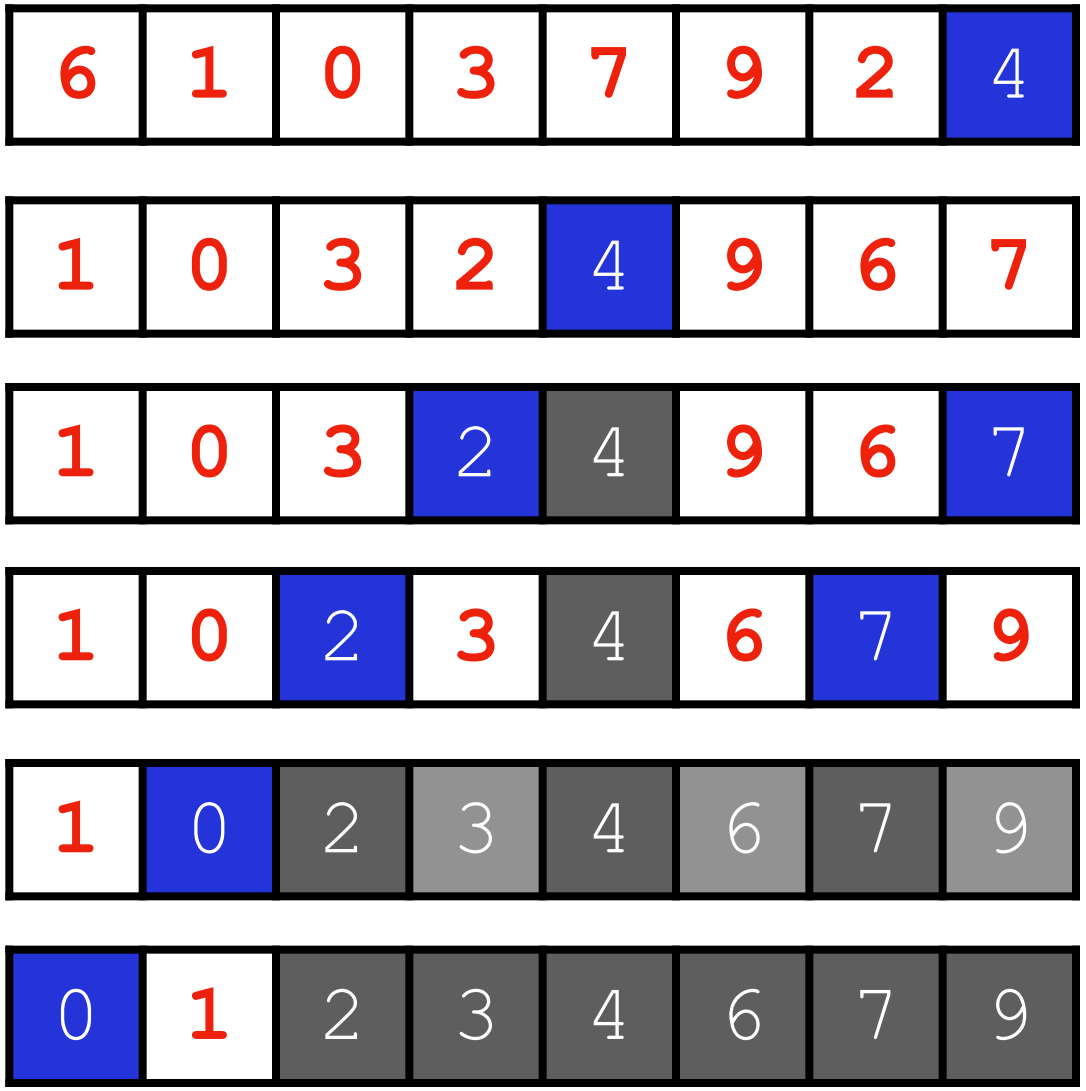
100% accuracy

May be slow

3. Use randomness inside algorithm to approximate solution in fixed time

Inaccurate but guaranteed fast

Quicksort Algorithm

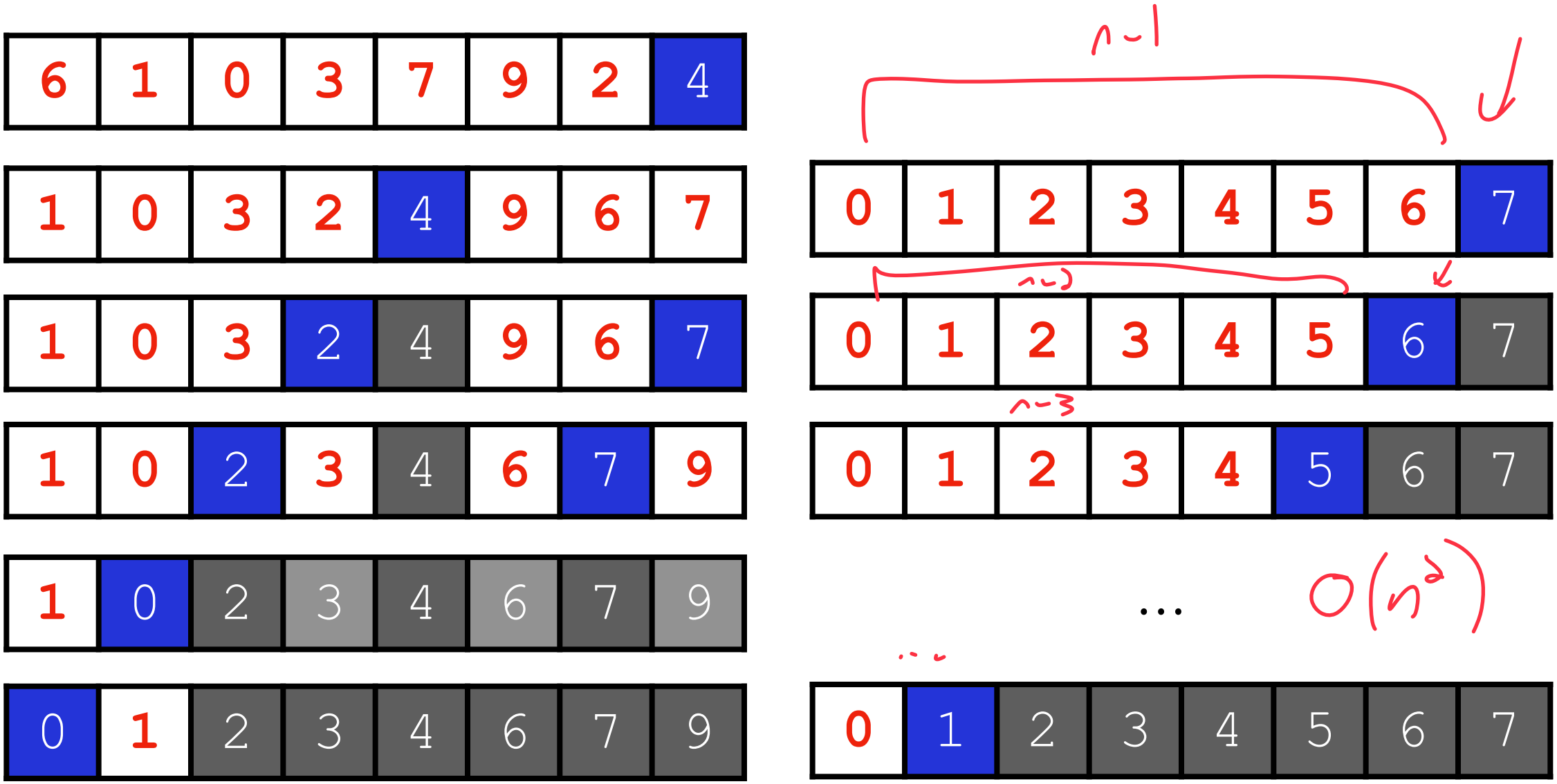


1) Pick Pivot (usually last item)

2) Split array around pivot

3) Recurse on partitions

Problem: Bad pivot leads to bad Big O!



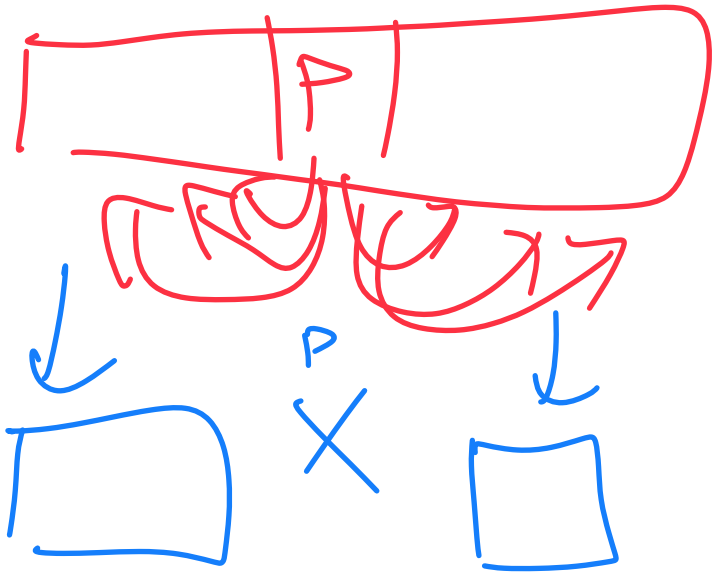
Expectation Analysis: Randomized Quicksort

In **randomized quicksort**, the selection of the pivot is random.

Claim: The expected time is $O(n \ log \ n)$ ***for any input!***

Key Idea: We never compare same pair twice!

Proof: Every comparison is against a pivot, but pivot not used in recursion



Expectation Analysis: Randomized Quicksort

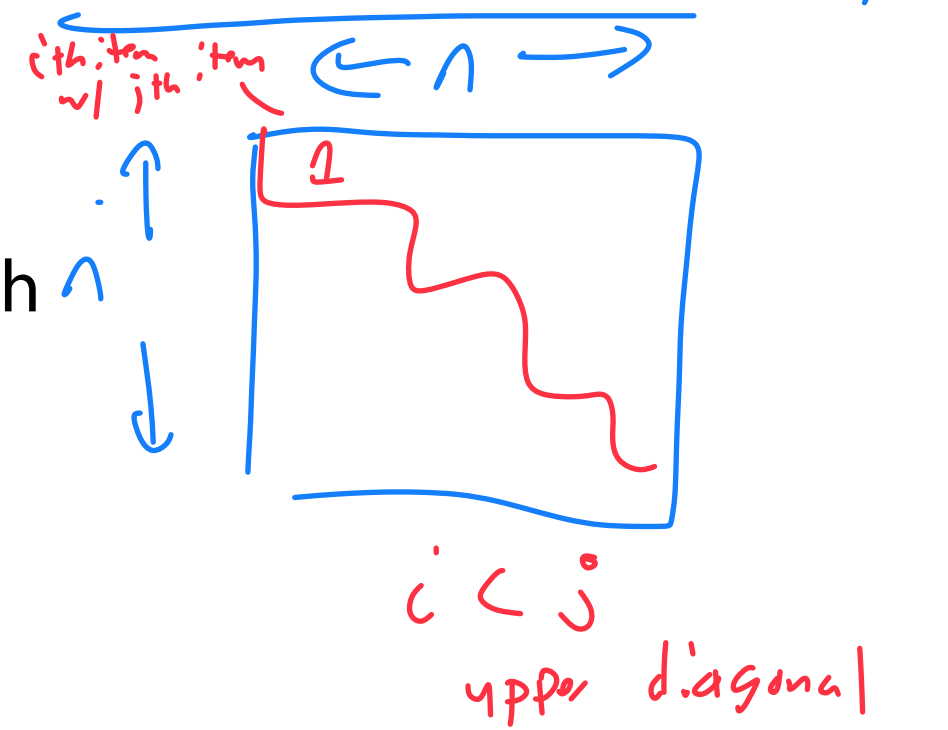
In **randomized quicksort**, the selection of the pivot is random.

Claim: The expected time is $O(n \log n)$ ***for any input!***

Extremely useful theory concept

Let X be the total comparisons and X_{ij} be an **indicator variable**:

$$X_{ij} = \begin{cases} 1 & \text{if } i\text{th object compared to } j\text{th} \\ 0 & \text{if } i\text{th object not compared to } j\text{th} \end{cases}$$



Then…

$X = \sum X_{ij}$: the # of total comparisons done in randomized quicksort

Expectation Analysis: Randomized Quicksort

In **randomized quicksort**, the selection of the pivot is random.

Claim: The expected time is $O(n \ log \ n)$ ***for any input!***

Let X be the total comparisons and X_{ij} be an **indicator variable**:

$$X_{ij} = \begin{cases} 1 & \text{if } i\text{th object compared to } j\text{th} \\ 0 & \text{if } i\text{th object not compared to } j\text{th} \end{cases}$$

Then… $$X = \sum_{i}^{n} \sum_{j=i+1}^{n} X_{i,j}$$

We can prove that $E[X] = O(n \ log \ n)$ with a **proof by induction**!

Expectation Analysis: Randomized Quicksort

To show $E[X] = O(n \ log \ n)$, we need to first get $E[X_{i,j}]$

Claim: $E[X_{i,j}] = \frac{2}{j - i + 1}.$

"guess the form"

$i < j$

Base Case: (N=2)

A B

$i = 0$ A

$j = 1$ B

if A is pivot

↳ 1 comparison

B is pivot

↳ 1 comparison

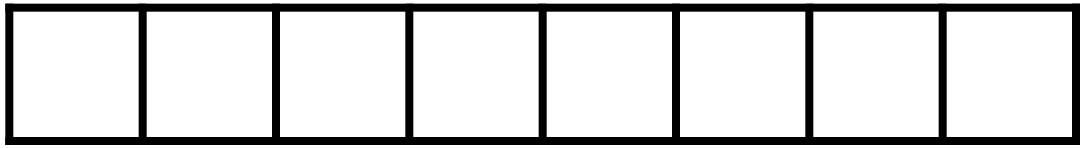
$\frac{2}{1-0+1} = \frac{2}{2} = 1$

Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \frac{2}{j-i+1}$

Induction: Assume true for all inputs of $< n$

cases for relation between i & j & p



$i < j < p$

$p < i < j$

$i \leq p \leq j$

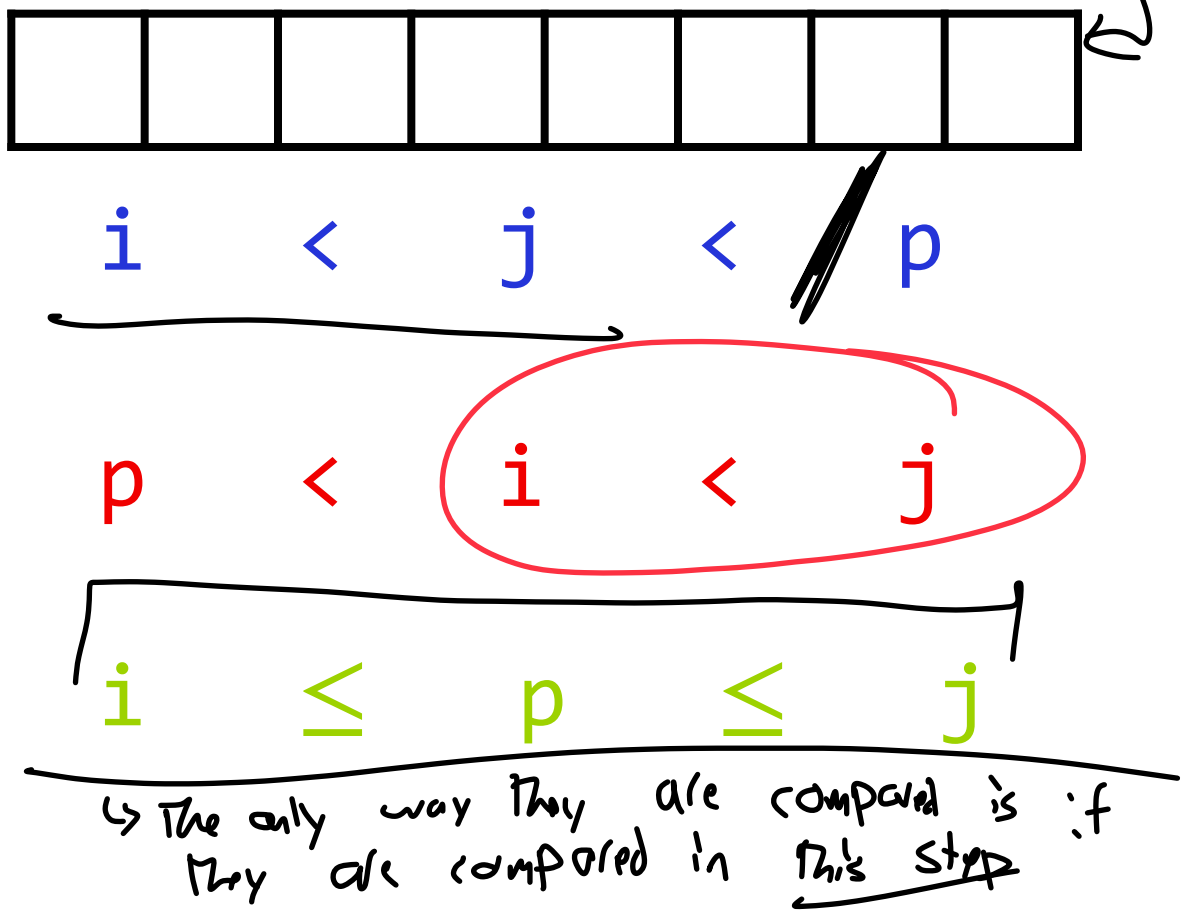
Not in same partition!

Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \frac{2}{j-i+1}$

Induction: Assume true for all inputs of $< n$





$Pr[X_{ij} | j < p] * Pr[j < p] +$

↳ This prob is on dataset smaller than n

By IH $\frac{2}{j-i+1}$

$Pr[X_{ij} | i > p] * Pr[i > p] +$

By IH $\frac{2}{j-i+1}$

$Pr[X_{ij} | i < p < j] * Pr[i < p < j]$

# of states where either $p = i$ or $p = j$

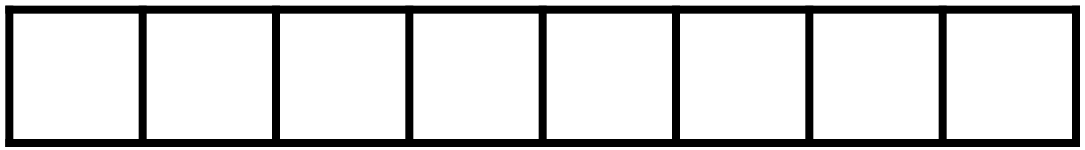
$\frac{2}{j-i+1}$

↳ The only way they are compared is if they are compared in this step

Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \frac{2}{j-i+1}$

Induction: Assume true for all inputs of $< n$



i < j < p

$Pr[X_{ij} \mid j < p] * Pr[j < p] +$

By IH, $\frac{2}{j-i+1}$ ✓

p < i < j

$Pr[X_{ij} \mid i > p] * Pr[i > p] +$

By IH, $\frac{2}{j-i+1}$ ✓

p = i or p = j

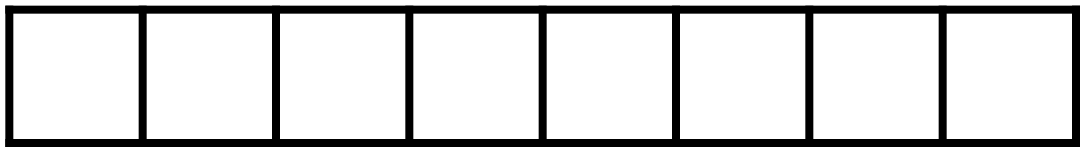
i ≤ p ≤ j

$Pr[X_{ij} \mid i < p < j] * Pr[i < p < j]$

Pivot must be either i or j — happens twice so $\frac{2}{j-i+1}$

Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \frac{2}{j-i+1}$ **Induction:** Assume true for all inputs of $< n$



We can rewrite as: $\frac{2}{j-i+1} * \underbrace{\left(Pr[j < p] + Pr[i > p] + Pr[i \leq p \leq j]\right)}$

Prob sums to 1

Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j - i + 1}$$

Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j - i + 1}$$

$$E[X] = \sum_{i=1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n - i + 1}\right)$$

Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1}\right)$$

$$E[X] = \sum_{i=1}^{n} 2(H_{n-1} - 1) \leq 2n \cdot H_n \leq 2n \ ln \ n$$

$O(n \log n)$

Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n} 2\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1}\right)$$

(1) Expand out inner sum

$$E[X] = \sum_{i=1}^{n} 2(H_{n-1} - 1)$$

(2) $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots$

$$E[X] = \sum_{i=1}^{n} 2(H_{n-1} - 1) \leq 2n \cdot H_n \leq 2n \ ln \ n$$

(3) $H_n = \theta(log \ n)$

Expectation Analysis: Randomized Quicksort



Summary: Randomized quick sort is $O(n \ log \ n)$ regardless of input

Randomness: Choice of pivot

Assumptions: I can pick a true random integer

Expectation Analysis: Randomized Quicksort



Summary: Randomized quick sort is $O(n \ log \ n)$ regardless of input

Randomness: The choice of pivot at each step

The analysis here works for any choice of input dataset!

Assumptions: Only that random numbers are actually random

While strictly not true, generally an acceptable assumption in practice

Ex: Park, Kyung Hwan, et al. "High rate true random number generator using beta radiation." AIP Conference Proceedings. Vol. 2295. No. 1. AIP Publishing LLC, 2020.

Randomization in Algorithms

1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

3. Use randomness inside algorithm to approximate solution in fixed time



Probabilistic Accuracy: Fermat primality test

Pick a random a in the range $[2,\ p-2]$

If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$

But… ***sometimes*** if n is composite and $a^{n-1} \equiv 1 \pmod{n}$

Prob of claiming something that is NOT prime is

prime is

Probabilistic Accuracy: Fermat primality test

Our label Alg says

ground truth	$a^{p-1} \equiv 1 \pmod{p}$ (label says prime)	$a^{p-1} \not\equiv 1 \pmod{p}$ (not prime)
p is prime	Positive: 100% True Positive ☺	Negative: 0% false negative
p is not prime	Positive: ε False positive	Negative: $1-\varepsilon$ True Negative ☺

$= 100\%$

This is one sided error

That's very good

(Bloom Filters lecture)

Probabilistic Accuracy: Fermat primality test

Let's assume $\alpha = .5$

50% First trial: $a = a_0$ and prime test returns 'prime!'

25% Second trial: $a = a_1$ and prime test returns 'prime!'

12.5% Third trial: $a = a_2$ and prime test returns 'not prime!'

Is our number prime?

What is our **false positive** probability? Our **false negative** probability?

2^3 ← Because one sided error we can do repeated random trials

Probabilistic Accuracy: Fermat primality test



Summary: Randomized algorithms can also have fixed (or bounded) runtimes at the cost of probabilistic accuracy.

Randomness: Selection of q

↑ The random # in my "algorithm"

Assumptions:

Can we pick random?

Probabilistic Accuracy: Fermat primality test



Summary: Randomized algorithms can also have fixed (or bounded) runtimes at the cost of probabilistic accuracy.

Randomness: The choice of α.

We can even pick more than one α if we want!

Assumptions: Only that random numbers are actually random

While strictly not true, generally an acceptable assumption in practice

Types of randomized algorithms

A **Las Vegas** algorithm is a randomized algorithm which will always give correct answer if run enough times but has no fixed runtime.

A **Monte Carlo** algorithm is a randomized algorithm which will run a fixed number of iterations and may give the correct answer.

What type of algorithm is Fermat's primality test?



What type of algorithm is randomized quick sort?

Next Class: Randomized Data Structures

Sometimes a data structure can be **too ordered / too structured**

Randomized data structures rely on **expected** performance

Randomized data structures ‘cheat’ tradeoffs!