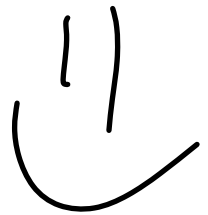
Data Structures and Algorithms

Probability in Computer Science

CS 225

Brad Solomon

April 15, 2026



The best topic!

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Announcements / Looking Ahead

One more feedback survey (releases next week)

Exam 5 / Retake exam are offered roughly at the same time

MP Puzzle is last MP, three more 'core' labs remain

One additional EC lab will release on Friday (once we start hash tables)

Learning Objectives

Formalize the concept of randomized algorithms

Review fundamentals of probability in computing

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

↳ This is Brad's opinion

Randomized Algorithms

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

Similarity between A & B

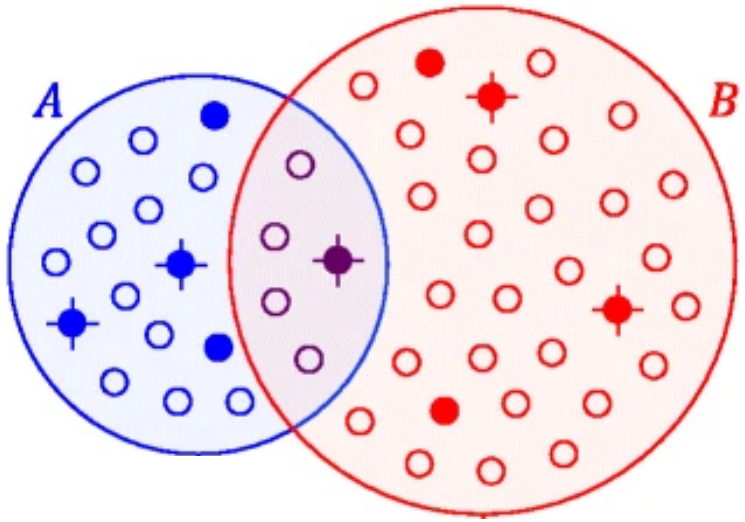


Figure from Ondov et al 2016

minhash

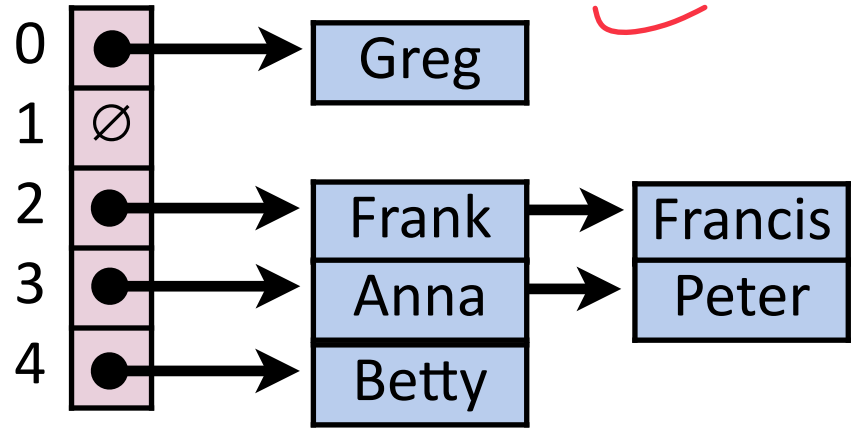
Approximate membership

$H(z)$

0
1
0
0
1

Bloom filter

Hash table



$H(x)$	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 adds coins

A faulty list

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

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

1) Probabilistic Data filtering

100 x anything good

↳ $(\frac{1}{2})^{100}$ to lose one good item

1x bad ↑] This is good!

↳ 50% of this will go away

2) Website caching

↳ Insert to cache each time we load] This is not a good solution

2.5) A simulation

3) You are trolling the user

Quick Primes with Fermat's Primality Test

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 $a^{n-1} \equiv 1 \pmod{n}$

All prime numbers p, when plugged in $\equiv 1$

Some not prime #s equal 1

If $a = 2$

21,853 which are not prime but $= 1$

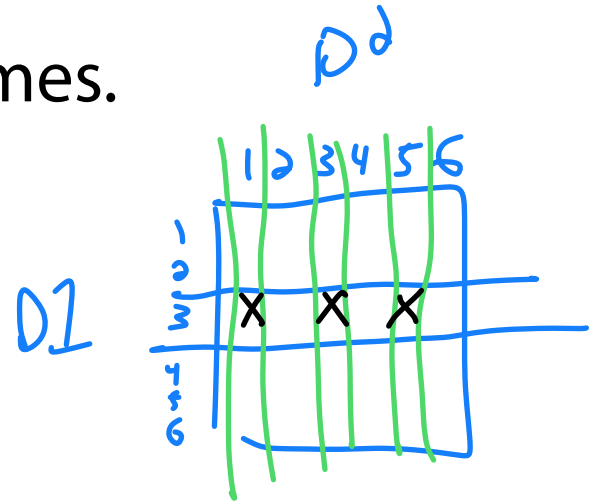
$25 \cdot 10^9$

In O(1) time get all prime and a fraction of percent of not prime

Fundamentals of Probability

Imagine you roll a pair of six-sided dice.

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



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

D1 = 3 &

D2 = odd

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 D1 be value of dice 2

D2 dice 2

P1=1, D2=2
D1=2, D2=1

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

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

prob of event · value of event

for all events

$\frac{1}{36}(1+1) + \frac{2}{36}(1+2) + \ldots$

= E[1 dice roll] * 2

E[2 dice roll] = 3.5

$= \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{3}{6} + \frac{4}{6} + \frac{5}{6} + \frac{6}{6}$

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 event

value of event

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,

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

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

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

↳ probabilities

↳ sum of all events is 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

My personal opinion

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

-> Is bad don't do this

2. Use randomness inside algorithm to estimate expected running time

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

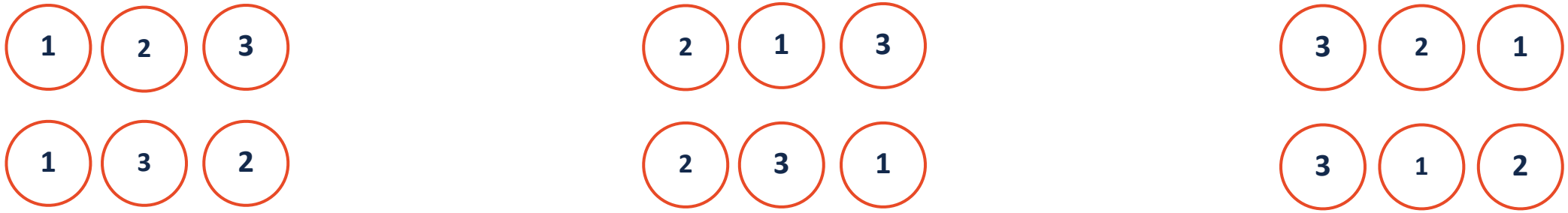
Average-Case Analysis: BST

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 $O(n \ log \ n)$

You are likely to build a "good" tree

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

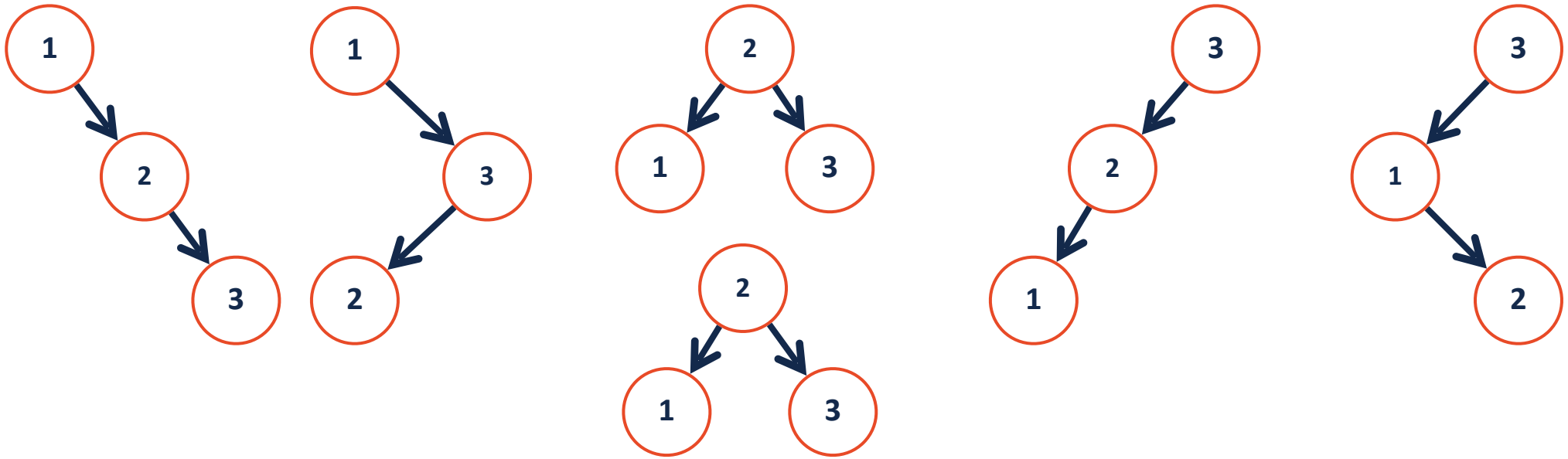


Average-Case Analysis: BST

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:



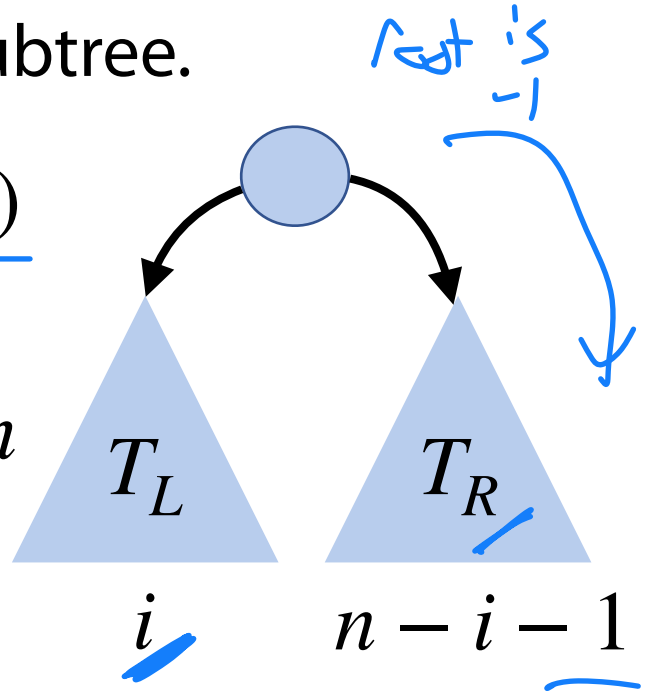
Average-Case Analysis: BST

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, $S(n) = (n-1) + S(i) + S(n-i-1)$

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



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?**

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?**

↑ That's a weak claim

↙ source of randomness

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

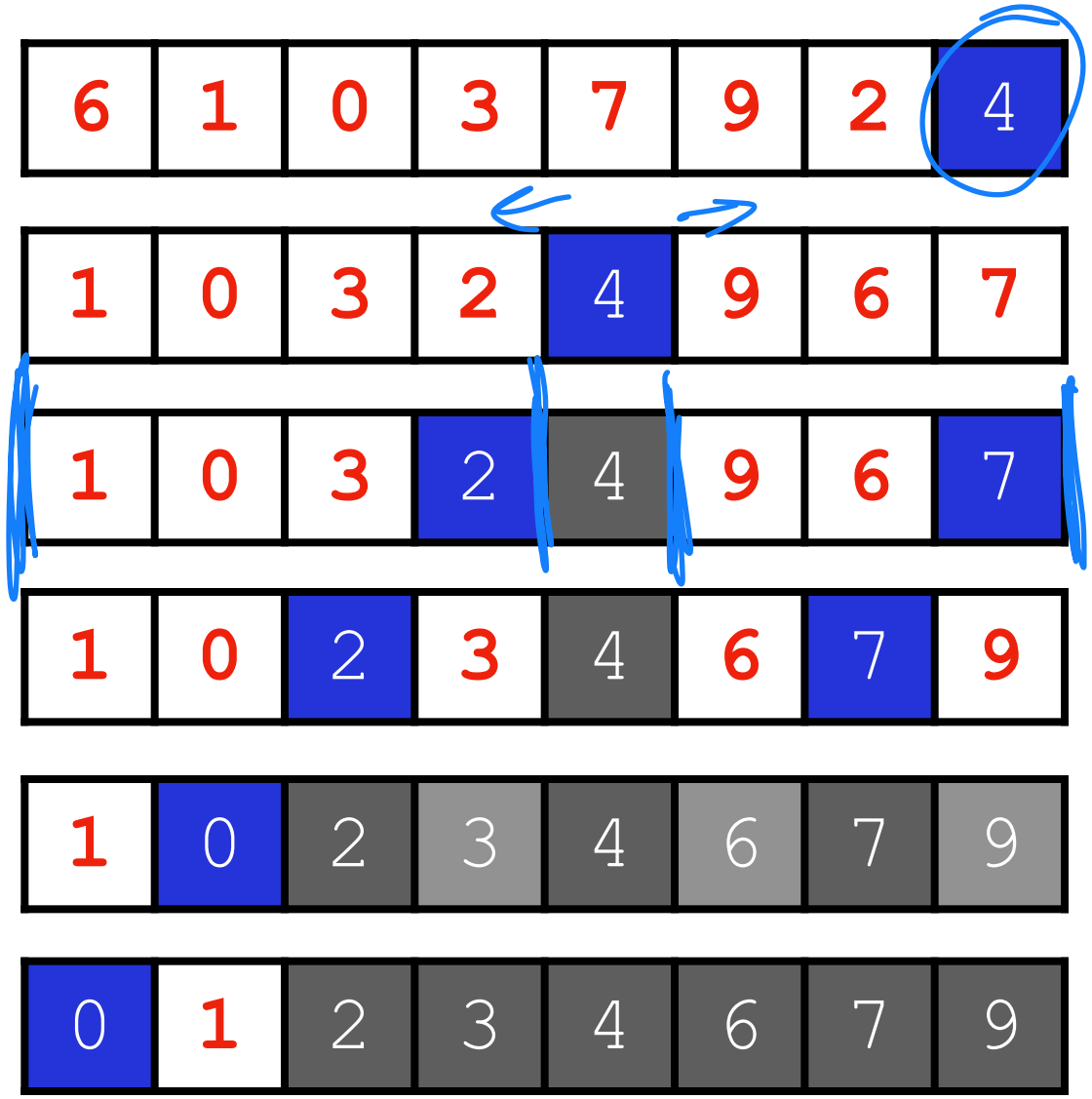
↳ this algs will work 100's of time but....

may be slow

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

↳ will run fast but may not be correct

Quicksort Algorithm

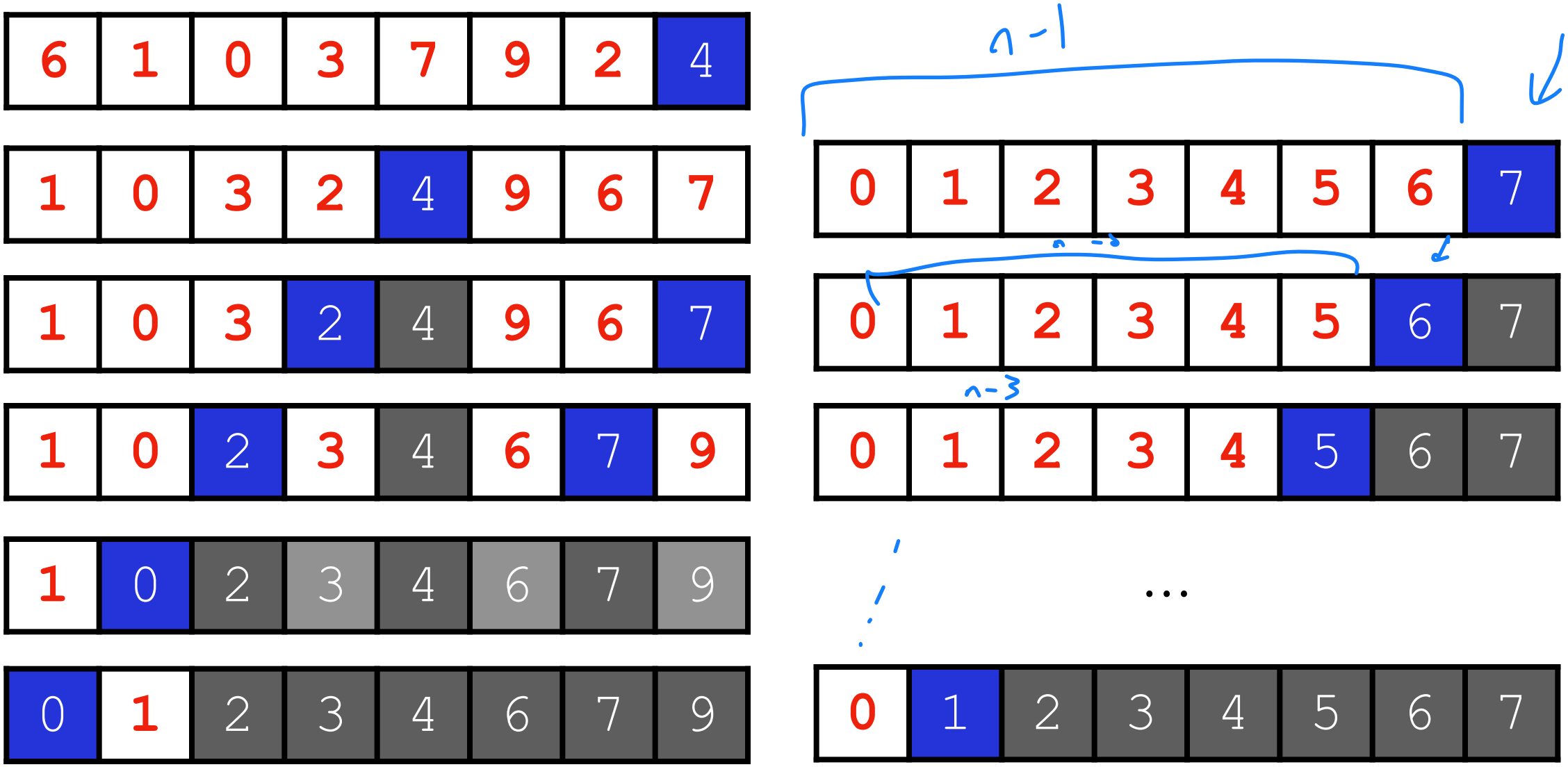


1) Pick Pivot (usually last item)

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

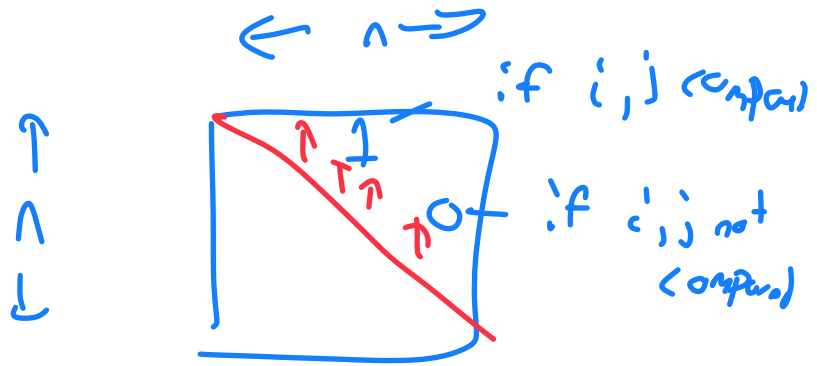
Expectation Analysis: Randomized Quicksort

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

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

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 = total # of comparisons $= \sum_i \sum_j X_{ij}$

↳ Math is clean!

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

Base Case: (N=2)

If A is pivot

A → B 1 calc

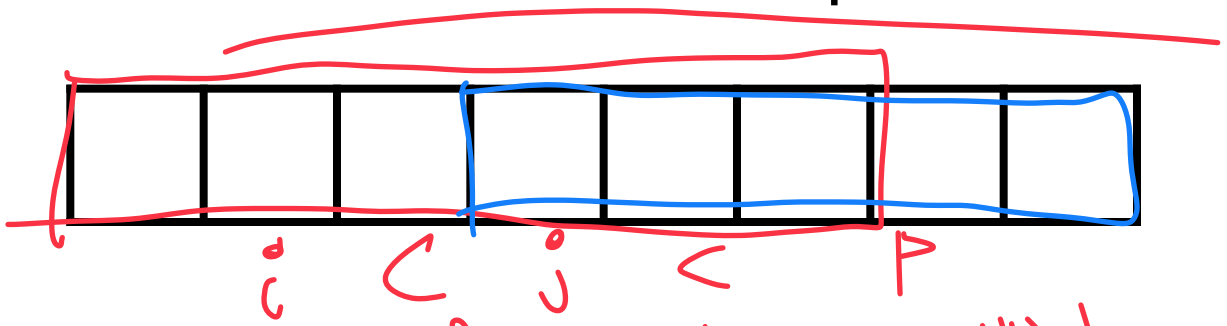
B is pivot

B → A 1 calc

Expectation Analysis: Randomized Quicksort

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

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



$= P[X_{ij} = 1 \mid j < p] \cdot Pr[j < p]$

$P[X_{ij} = 1 \mid i > p] \cdot Pr[i > p]$

$P[X_{ij} = 1 \mid i \le p \le j] \; P[i \le p \le j]$

↳ X_{ij} is 1 if $i = p$ or $j = p$

↳ if i & j are in same partition!

$p < i < j$

↳ i & j are in same partition

$i \le p \le j$

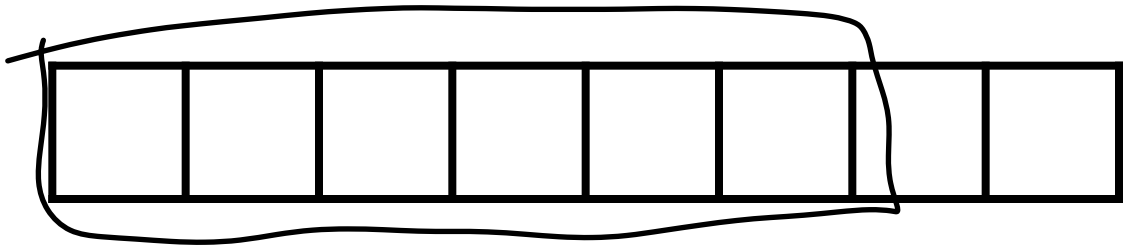
↑ is in same partition

↑ is in another partition

Expectation Analysis: Randomized Quicksort

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

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



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

By IH $\hookrightarrow \left[\frac{2}{j-i+1}\right] \cdot Pr(j<p)$

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

By IH $\hookrightarrow \left[\frac{2}{j-i+1}\right] \cdot Pr(i>p)$

sums to 1!

p < i < j

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

$\left[\frac{2}{j-i+1}\right] \cdot Pr[i<p<j]$

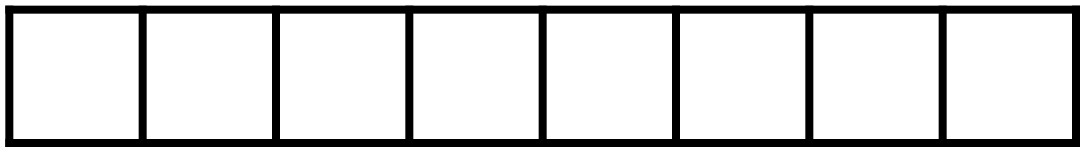
i ≤ p ≤ j

or j is P

Expectation Analysis: Randomized Quicksort

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

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



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

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

i < j < p

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

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

p < i < j

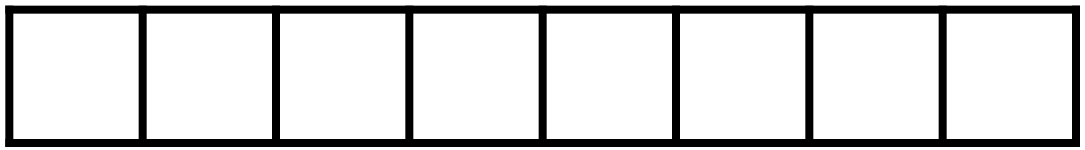
$Pr[X_{ij} \mid i < p < j] * Pr[i < p < j]$

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 $< n$

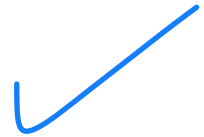


We can rewrite as: $\frac{2}{j-i+1} * (Pr[j < p] + Pr[i > p] + Pr[i \leq p \leq j]$

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

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

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

Pretend you have Harmonic series

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

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:

Assumptions:

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

↳ Saw this at the start!

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 $a^{n-1} \equiv 1 \pmod{n}$

Probabilistic Accuracy: Fermat primality test

	$a^{p-1} \equiv 1 \pmod{p}$	$a^{p-1} \not\equiv 1 \pmod{p}$
p is prime		
p is not prime		

Probabilistic Accuracy: Fermat primality test

Let's assume $\alpha = .5$

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

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

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?

Probabilistic Accuracy: Fermat primality test

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

Randomness:

Assumptions:

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!