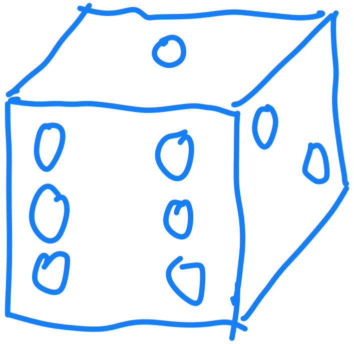
Data Structures and Algorithms
Probability in Computer Science

CS 225
Brad Solomon

November 8, 2024



Department of Computer Science

Exam 4 (11/13 — 11/15)

Autograded MC and one coding question

Manually graded short answer prompt

Practice exam will be on PL

Topics covered can be found on website

Registration started October 31

https://courses.engr.illinois.edu/cs225/fa2024/exams/

Learning Objectives

Formalize the concept of randomized algorithms

Review fundamentals of probability in computing

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

Randomized Algorithms

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

Bloom Filter

Hash Table

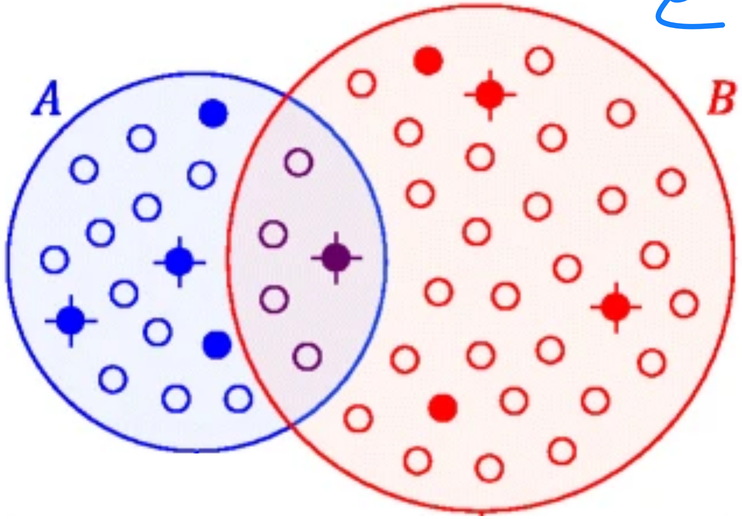
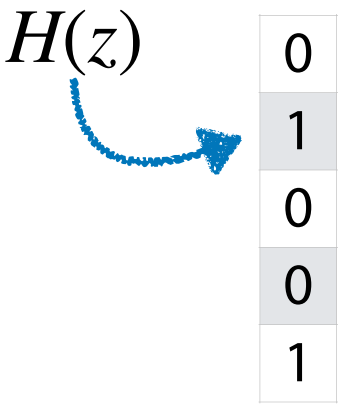
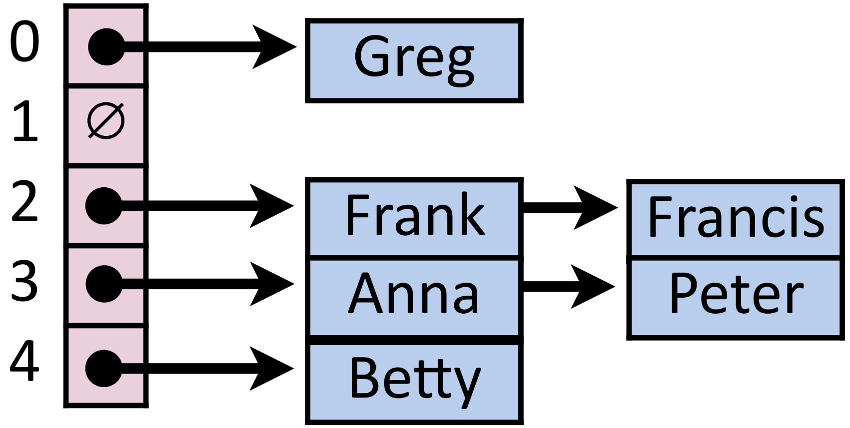


Figure from Ondov et al 2016

MinHash Sketch





$H(x)$	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.

↳ Could use to simulate repeated coin flips

↳ webpage cache

↳ Counting in an ∞ data stream

↳ Probabilistic data cleaning

Real items have high repetition and fake/errors dont

Quick Primes with Fermat's Primality Test

$[2, p-2]$

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

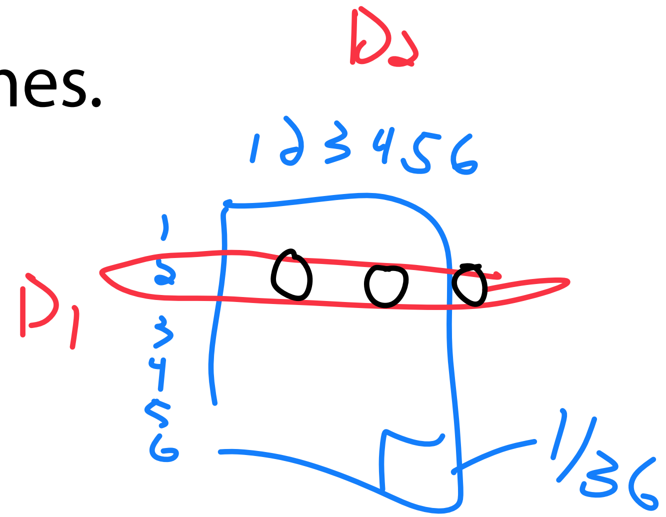
If $a=2$, $\frac{21,853}{25 \cdot 10^9 \text{ integers}}$ "pseudoprimes" – not prime but pass test

Not 100% accurate but… 99.9% accurate & fast

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 rolls 2 and D2 is even

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.

D1 is value of first dice →

DBoth is value of D1 + D2

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

$$E[D1] = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \dots \approx 3.5$$

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

$$E[DBoth] = \frac{1}{36} \cdot 2 + \frac{1}{36} \cdot (1+2) + \dots \approx 7$$

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,

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

$$E[X + Y] = \sum_x \sum_y \underbrace{Pr\{X = x, Y = y\}}_{\text{prob of event}}(x + y)$$

∘ 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,

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

Sum of probabilities is 1

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

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

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,

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

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

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

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

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,

$$E[X + Y] = \underbrace{E[X]}_{3.5} + \underbrace{E[Y]}_{3.5}$$

$= 7$

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

1 2 3   2 1 3   3 2 1

1 3 2   2 3 1   3 1 2

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

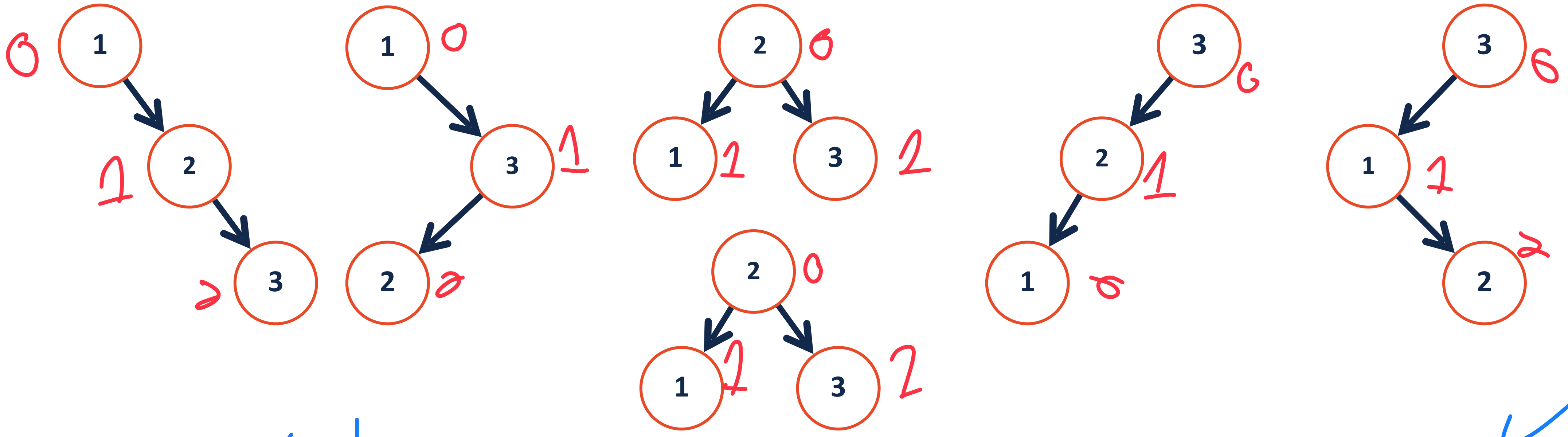
Claim: $S(n)$ is $O(n\ log\ n)$

path length

$6 \cdot 0 + 8 \cdot 1 + 4 \cdot 2 = 16/6 \approx 2.66$

trees

N=3:



$3 \log 3 \sim 4.75$

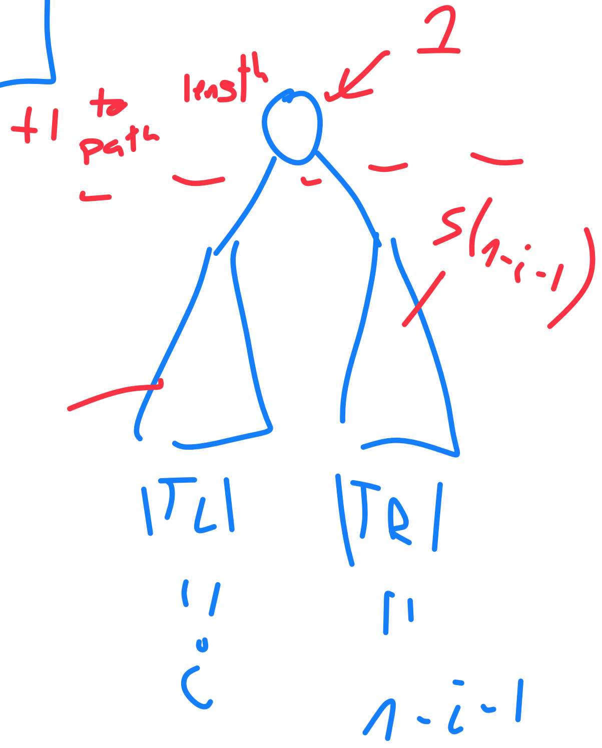
Average-Case Analysis: BST

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

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

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



S(i)

n-1 items here

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.

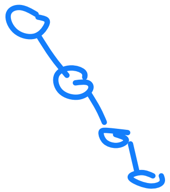
Average-Case Analysis: BST

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

real $O(n)$



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

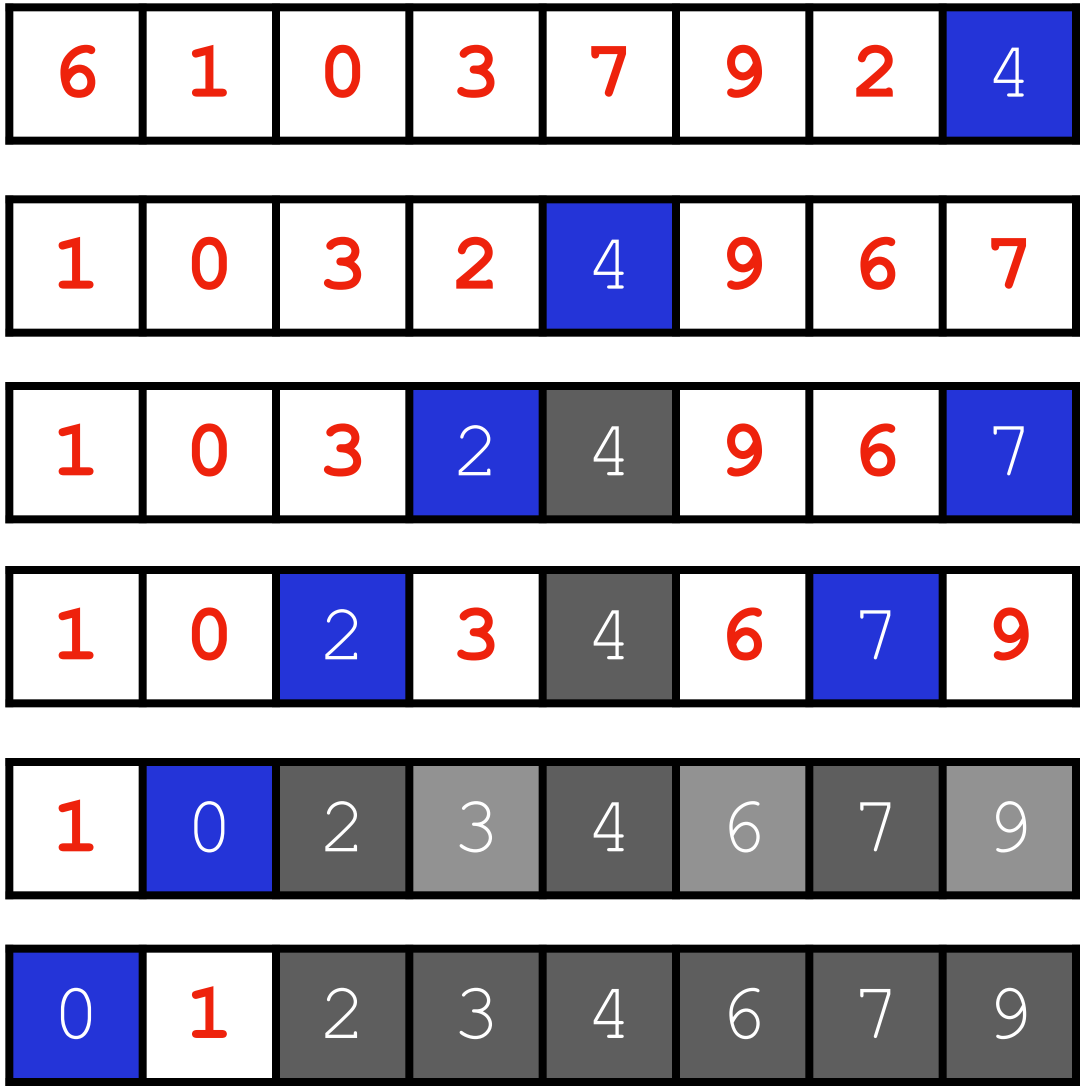
Randomization in Algorithms

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

2. Use randomness inside algorithm to estimate expected running time

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

Quicksort Algorithm

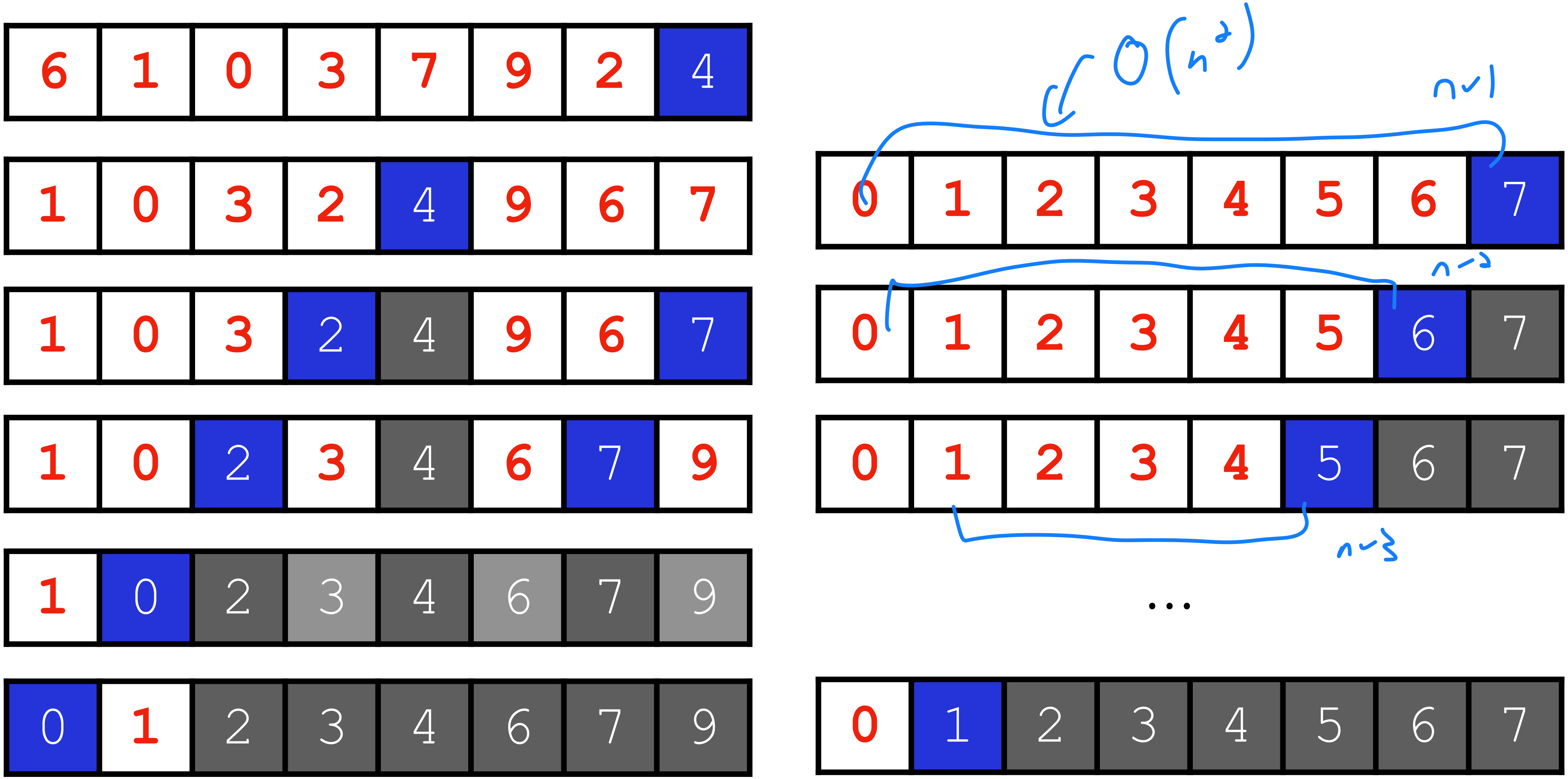


1) Pick Pivot (usually last item)

2) Split array around pivot

3) Recurse on partitions

Problem: Bad pivot leads to bad Big O!



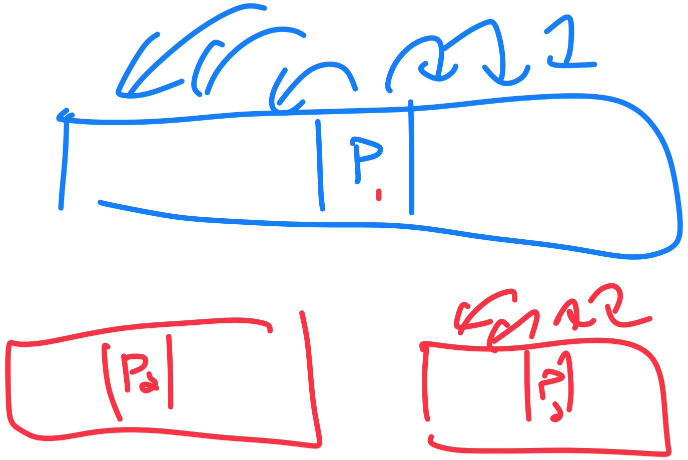
Expectation Analysis: Randomized Quicksort

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



Pivot compares all items

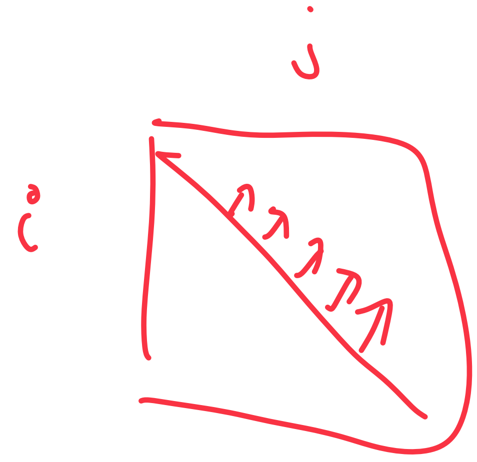
Expectation Analysis: Randomized Quicksort

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… $X = \sum_{i} \sum_{j} x_{ij}$

$i = [1, n]$

index start @ 1

$j = [i+1, n]$

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

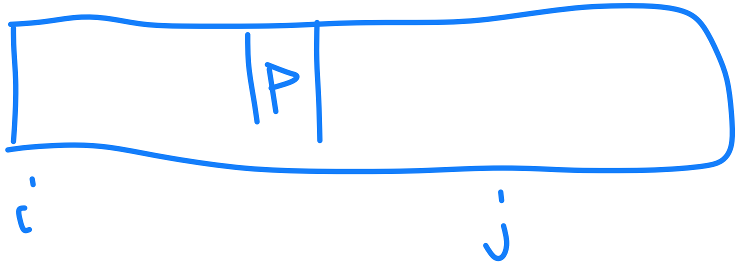
$\doteq E[X_{ij}]$

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 $E[X_{i,j}]$

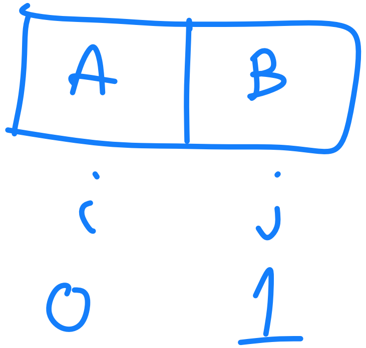
Claim: $E[X_{i,j}] = \dfrac{2}{j-i+1}$.



Base Case: (N=2) b/c j = i+1 as min index

Eq

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



Real

1 comparison

P=A

1

B

P=B

A

Expectation Analysis: Randomized Quicksort

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

$= \Pr[X_{i,j}=1 \mid j<p] \cdot \Pr[j<p]$

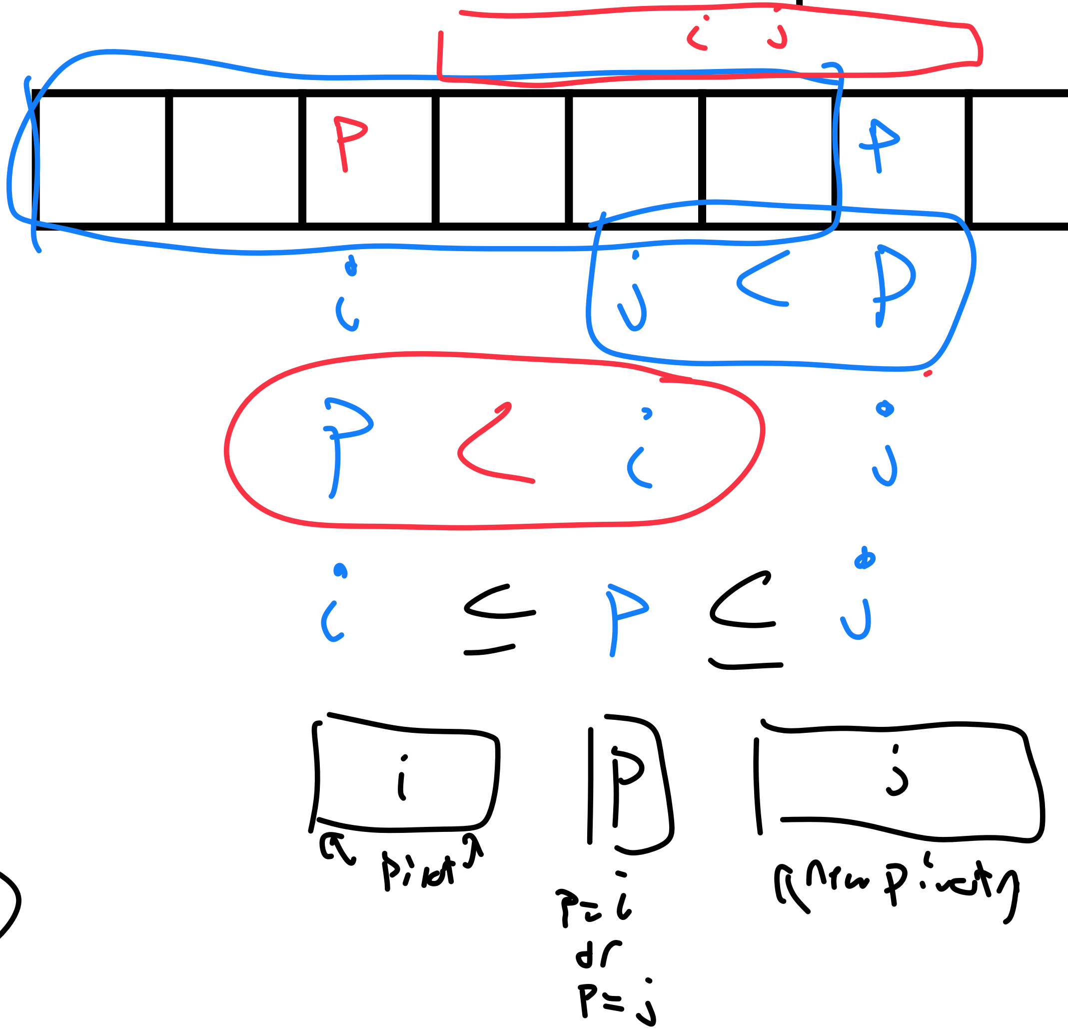
By IH $\dfrac{2}{j-i+1} \cdot \Pr[j<p]$

$+ \Pr[X_{i,j}=1 \mid p<i] \cdot \Pr[p<i]$

By IH $\dfrac{2}{j-i+1} \cdot \Pr[p<i]$

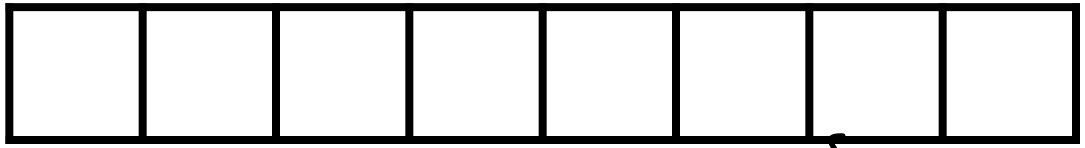
$+ \Pr[X_{i,j}=1 \mid i \le p \le j] \cdot \Pr[i \le p \le j]$

$\dfrac{2}{j-i+1} \cdot \Pr[i \le p \le j]$



Expectation Analysis: Randomized Quicksort

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



$$\frac{2}{j-i+1} \cdot \Big(\underbrace{P[j<p] + P[p<i] + P[i \le p \le j]}_{1} \Big)$$

All possible events/states!

i j p

p i j

i p j

Expectation Analysis: Randomized Quicksort

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

X is total comparisons

Expectation Analysis: Randomized Quicksort

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

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

$j=i+1$ $\quad i+2$ $\quad \ldots$

Harmonic #

$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \ldots$

$H_n = \Theta(\log n)$

Expectation Analysis: Randomized Quicksort

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

1+?

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

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

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

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

(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: None on data / distribution

↳ can we get a real random #?

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% accurate but maybe slow

Randomized quicksort $\quad O(n^2)$

↳ Avg case is good!

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

↳ Not 100% accurate & fast

↳ Fermat's primality test

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:

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.

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!