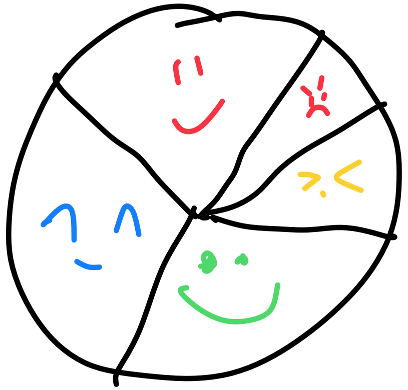
# Data Structures and Algorithms

## Probability in CS ~~2~~ Part 2

CS 225
Brad Solomon

October 25, 2023

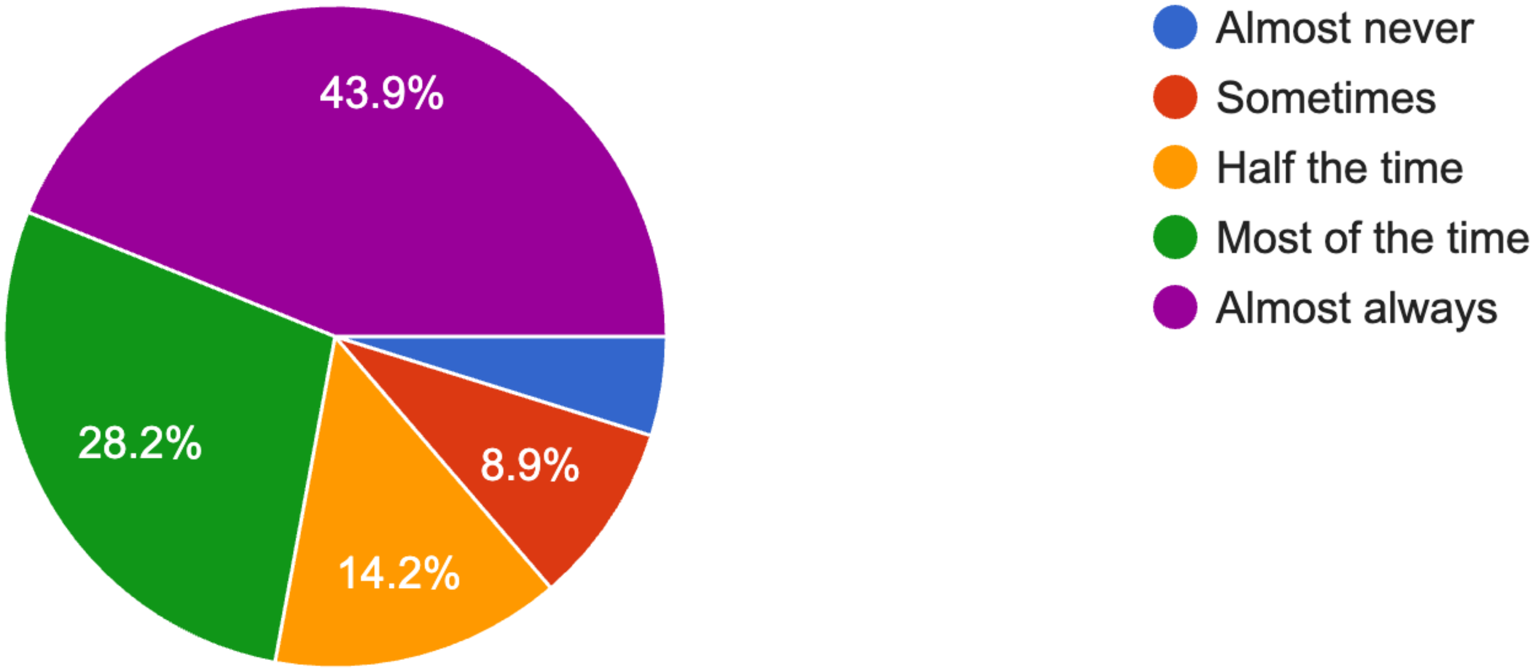


UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

# Informal Early Feedback → 70,5%

I attend lecture or watch lecture recordings:

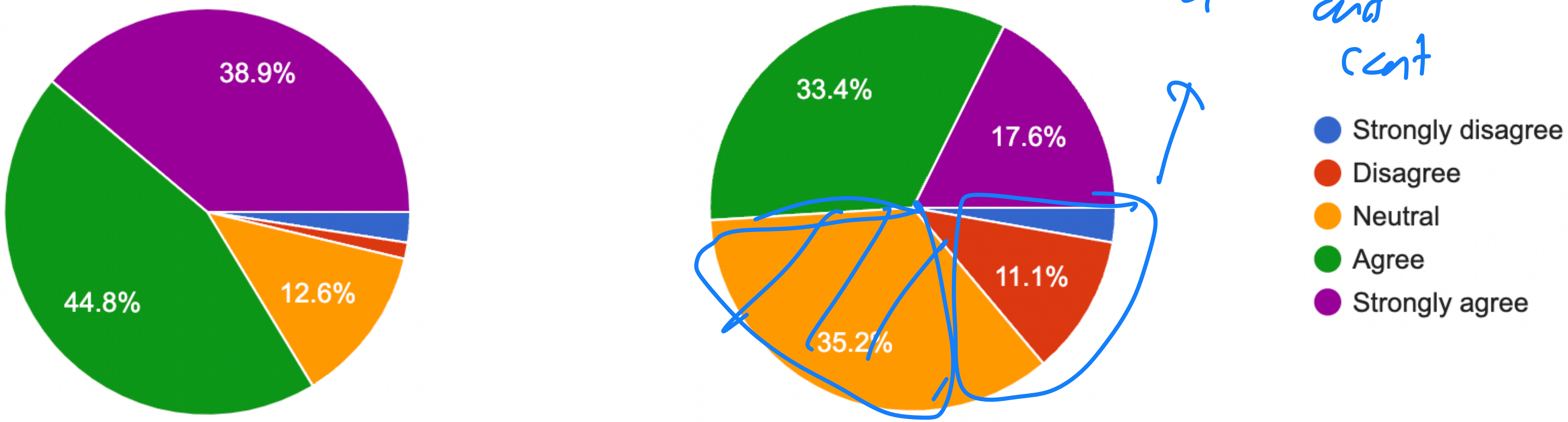
This is a lie → 30% didn't fill in



# Informal Early Feedback

The instructor is well prepared for each lecture

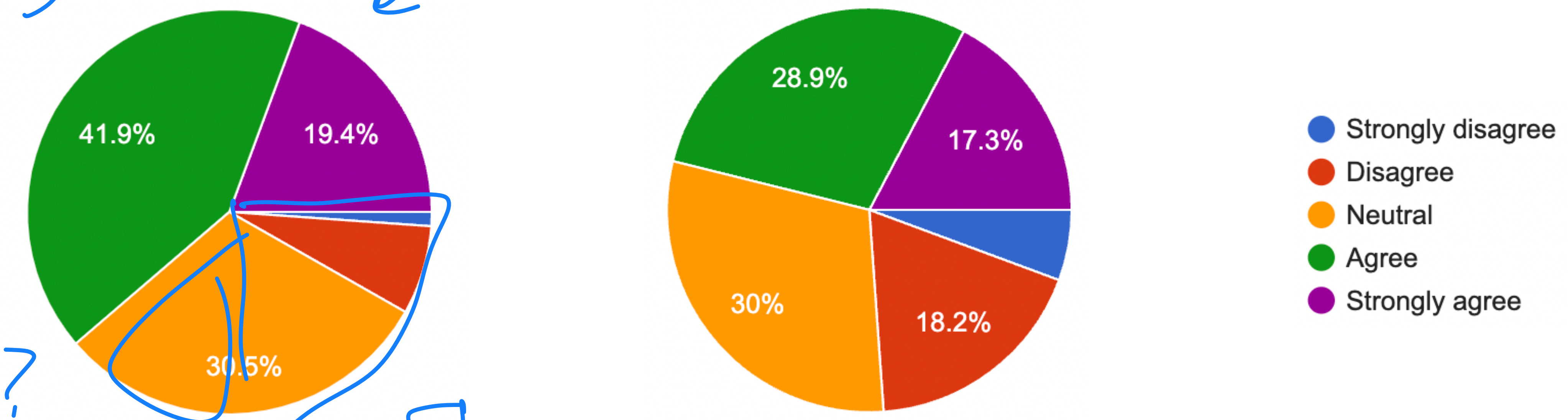
atted class and cent




I feel I can actively participate in lecture

# Informal Early Feedback

During lecture, I receive helpful and complete answers to my questions




??

Ask questions!

Overall, attending lecture (in person) is a good use of my time.

# Suggested Improvement (To Lectures)

Brad should improve his handwriting

← Don't attend lecture

1) Lot of tools can't use!

2) Ask if my handwriting doesn't make sense!

Pacing!

Try other things

Go over all MP functions in lecture / provide pseudocode

↳ Mosaic's fault is mine

↳ Sometimes

Add more introductory content (C++)

↳ No we can't

↳ We want to add "bonus content"

# Suggested Improvement (To Lectures)

Add a weekly review session to cover topics

↳ :) If we can do it...?

AMAs for MPs

↳ Something similar!

Slow down lectures / better motivate data structures

↳ Ask questions! :)

↳ :) we can improve

Reduce size of lecture (offer more sections?)

↳ No?

More accurate captions

↳ Auto generated → Manually corrected

# Suggested Improvement (To Lectures)

Upload lecture slides earlier / make sure website matches with lecture

Upload lectures by subjects not by day

↳ Commit to trying!

Improve class as we go along :)

# Learning Objectives

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

Analyze an example of each type

# 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

Consider where randomness is & assumptions

# Randomization in Algorithms

Bad!

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

1 2 3     2 1 3     3 2 1

1 3 2     2 3 1     3 1 2

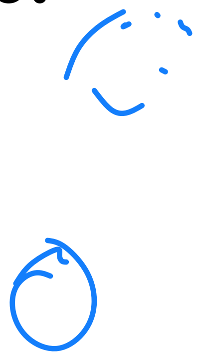
Lab_BST

# Average-Case Analysis: BST

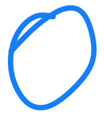
Let $S(n)$ be the average **total internal path length** over all BSTs that can be constructed by uniform random insertion of $n$ objects

**Claim:** $S(n)$ is $O(n \ log \ n)$ → random tree is good!

**N=0:**



**N=1:**



path length 0

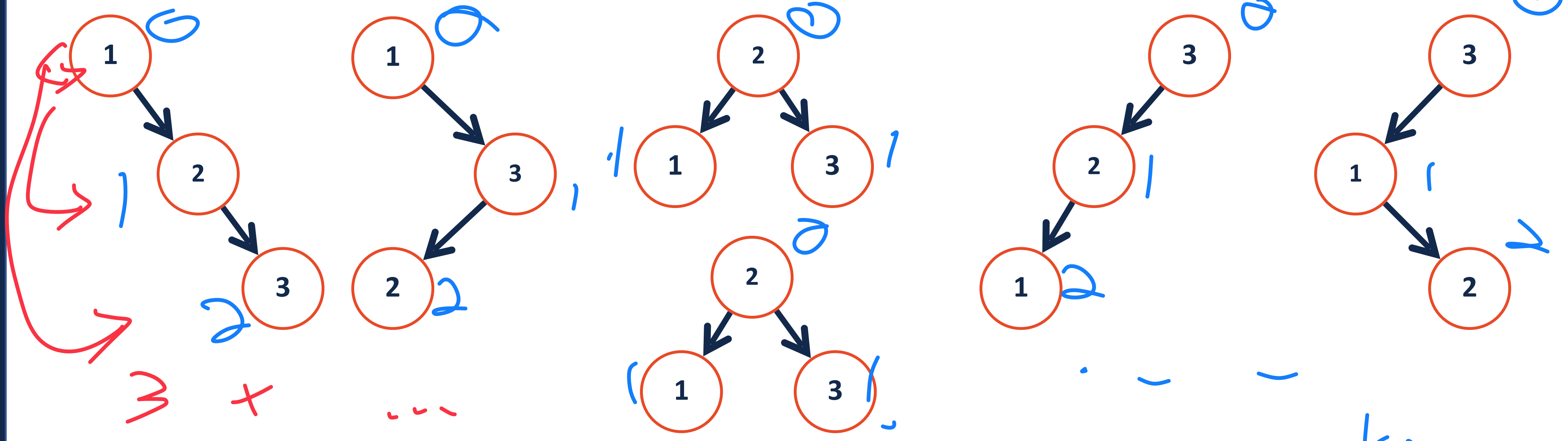
# Average-Case Analysis: BST

Assume all trees equally likely! / all queries

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

Sum all paths from root to all nodes

**N=3:**



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

$n \log n$

$3 \log 3 \sim 4.75$

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

recursion

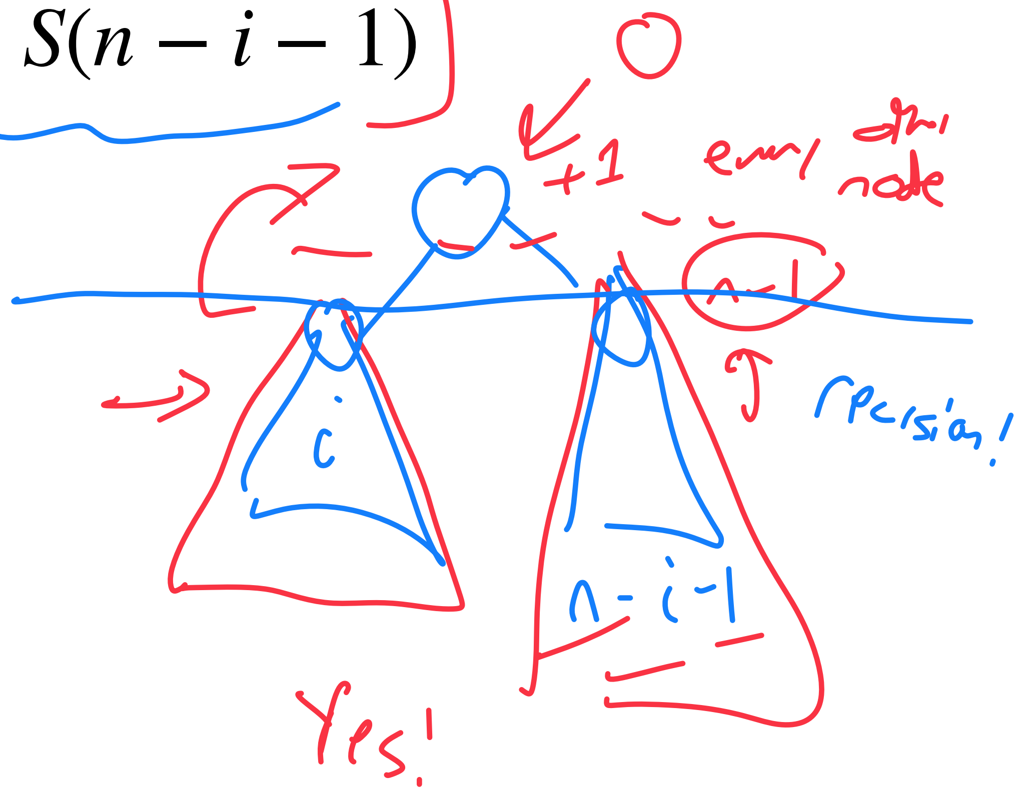
Then for a fixed $i$, $S(n) = (n-1) + S(i) + S(n-i-1)$

n total nodes

⇒ 1 node is root

i in the left

everything else



# 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

$$S(n) = (n-1) + \frac{1}{n}\sum_{i=0}^{n-1}\left(S(i) + S(n-i-1)\right)$$


$S(i) = S(0) + S(1) + S(2) \ldots S(n-1)$

$S(n-i-1) = S(n-1) + \ldots + S(2) + S(1)$

mirrors of each other!

$2\sum S(i)$

# Average-Case Analysis: BST

Least relevent proof ever

$$S(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} S(i)$$

Guessed my solution $(ci \ ln \ i)$

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

Dont like summations

Instead of discrete sum make integral

$$S(n) \le (n-1) + \frac{2}{n}\int_1^n (cx \ ln \ x)dx$$

Fact: This integral has a

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

# 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

Since $S(n)$ is $O(n\ log\ n)$, if we assume we are randomly choosing a node to insert, find, or delete* then each operation takes:

Uniform chance of asking about any node

n log n average path length of tree

n nodes

Last step divide by n

$O(\log n)$ :)

Not a real big O

# Average-Case Analysis: BST

**Summary:** All operations are on average $\Theta(log\ n)$

**Randomness:**

1) We assumed all inputs were random → real world can't assume

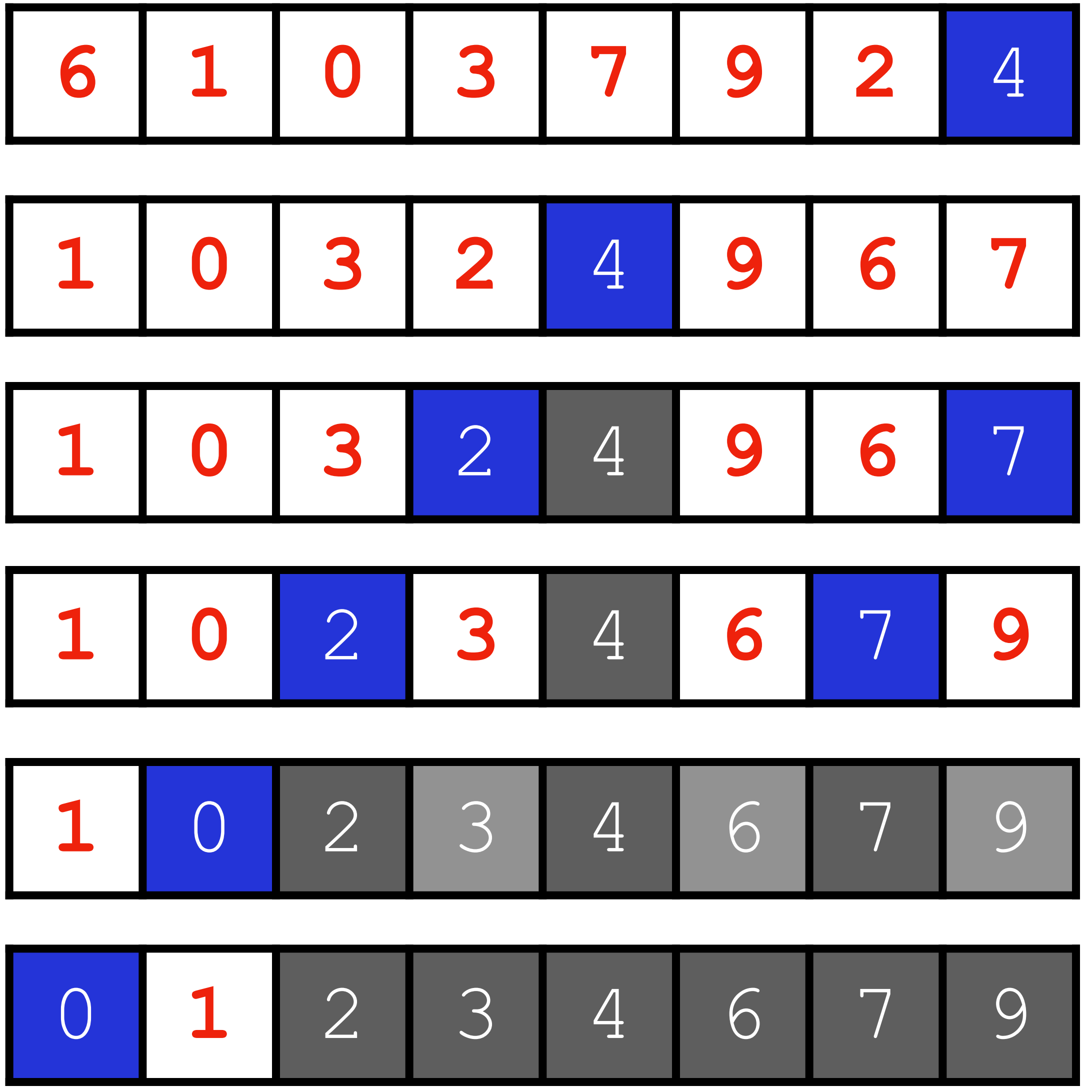
2) We assumed query is random → Not Big O!

↳ "worst"

**Assumptions:**

↳ Assume randomness is uniform

# Expectation Analysis: Randomized Quicksort

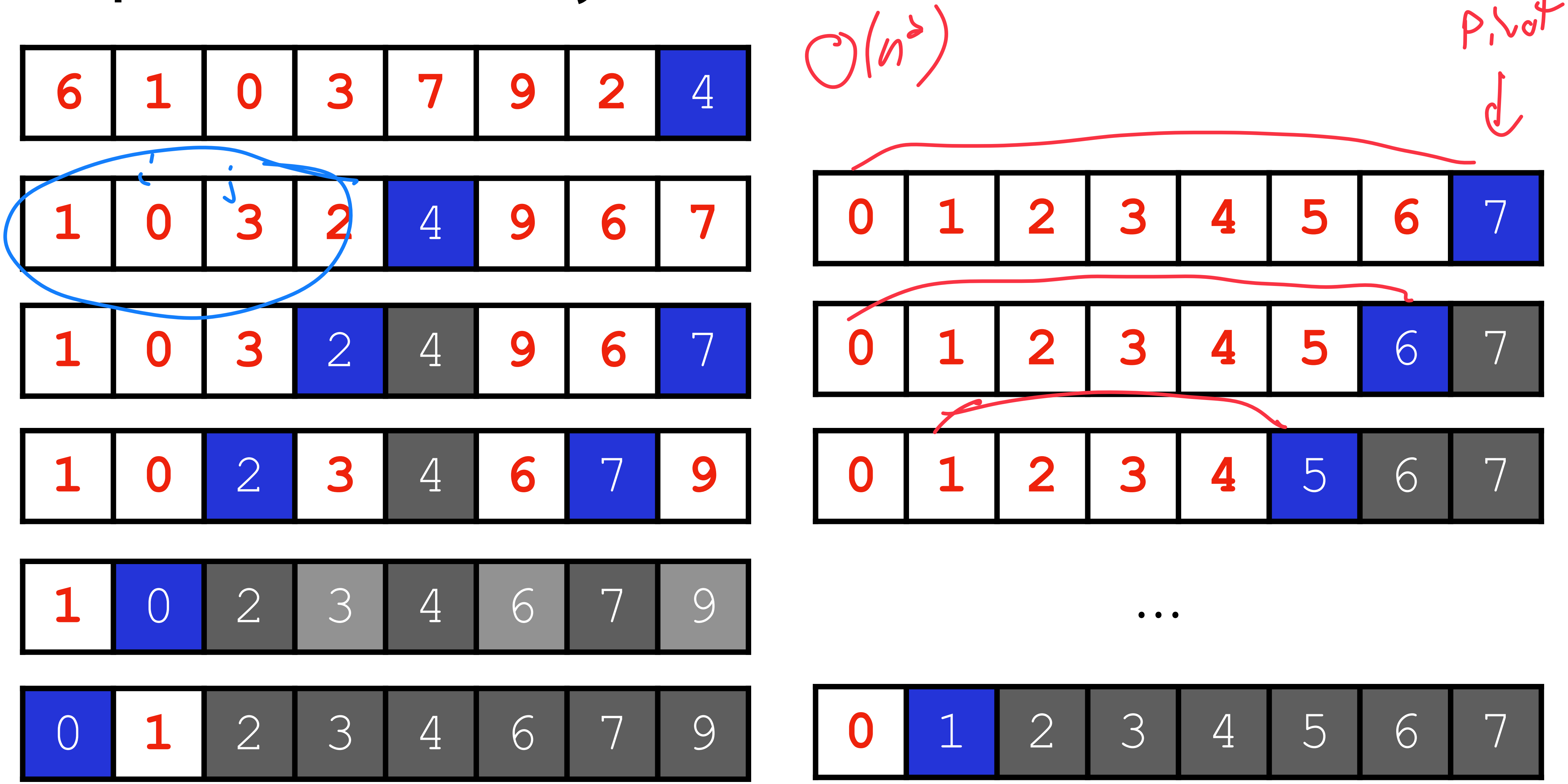



1) Pick a pivot

2) split around pivot

recurse

# Expectation Analysis: Randomized Quicksort



# Randomization in Algorithms

2. Use **randomness inside algorithm** to estimate expected running time

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

**Claim:** The expected time is $O(n \; log \; n)$ ***for any input!*** ← Avg case we assume a lot about input

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 2 | 1 | 3 | 7 | 8 | 5 | 4 |

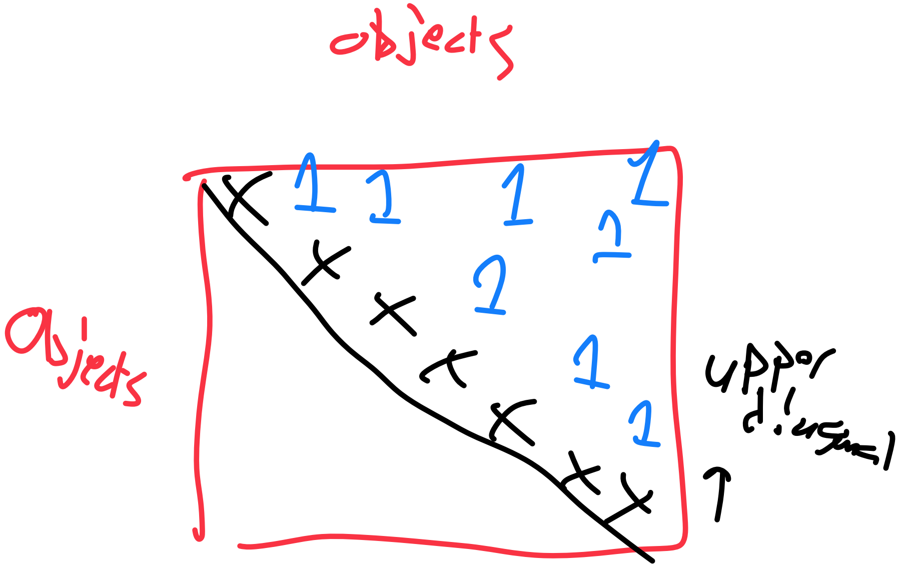
# Expectation Analysis: Randomized Quicksort

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

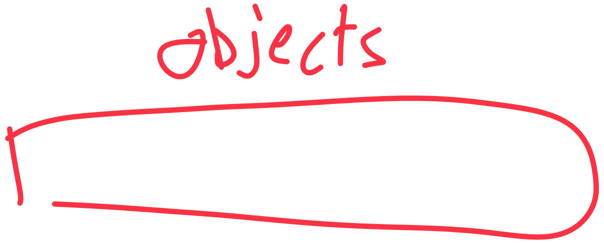
**Claim:** The expected time is $O(n\ \ log\ \ n)$ ***for any input!***

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…



# Expectation Analysis: Randomized Quicksort

$j = i+1 \rightarrow n$

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

**Base Case:** (N=2)

A B

pivot pivot

1 always

$\dfrac{2}{1-0+1} = \dfrac{2}{2}$ ✓

$i = 0$
$j = 1$

Significant mistake in lecture presentation!

X_{i,j} is the expected value of THE SINGLE PAIR. Not the total amount of comparisons!

# Expectation Analysis: Randomized Quicksort

expected value

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

smaller than n!

$(i \quad j) < p$

$p < (i \quad j)$

$i \leq p \leq j$

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

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

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

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

$+ \frac{2}{j-i+1} \cdot P[i \leq p \leq j]$

$1$

$\frac{2}{j-i+1}\left(P[j<p] + P[p<i] + P[i \leq p \leq j]\right)$

$i \mid p \mid j$, if $p = i$ or $p = j$

never be compared!

# Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j-i+1}$$

We did not have time to cover this. Briefly:

The key idea is writing out the internal sum after pulling out the 2 in the numerator. Which will show a pattern from 1/2 to … 1/(n-i)

Ex: i=0, j=i+1 -> 1/(i+1)-i+1=2

Ex: i=0, j=n-1 -> 1/(n-1)-i+1=n-i

# Expectation Analysis: Randomized Quicksort

$$E[X] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} E[X_{ij}] \qquad E[X_{ij}] = \frac{2}{j-i+1}$$

For $i = 0$:

$$\sum_{j=i+1}^{n-1} = 2(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n})$$

For $i = 1$:

$$\sum_{j=i+1}^{n-1} = 2(\frac{1}{2} + \frac{1}{3} \ldots + \frac{1}{n-1})$$

$$E[X] = 2\sum_{i=0}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k}$$

The key here is that sum of increasing fractions is O(log n) for some log.

Simplifying to 1/k makes it clear its ln

# Expectation Analysis: Randomized Quicksort

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

**Randomness:**

My algorithm choice of pivot !

**Assumptions:**

# 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:**

# 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

# 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.

# 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!