Data Structures and Algorithms

Probability in CS 2

CS 225

Brad Solomon

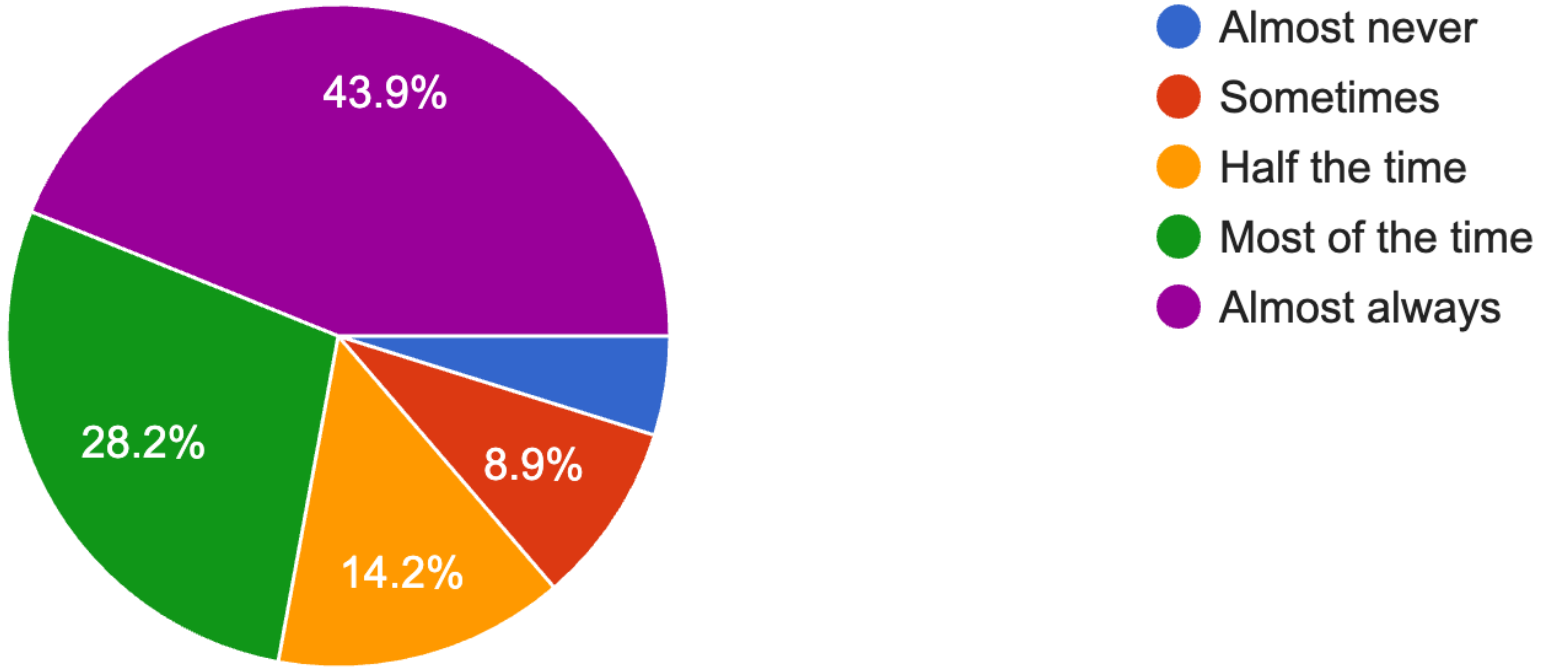
October 25, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

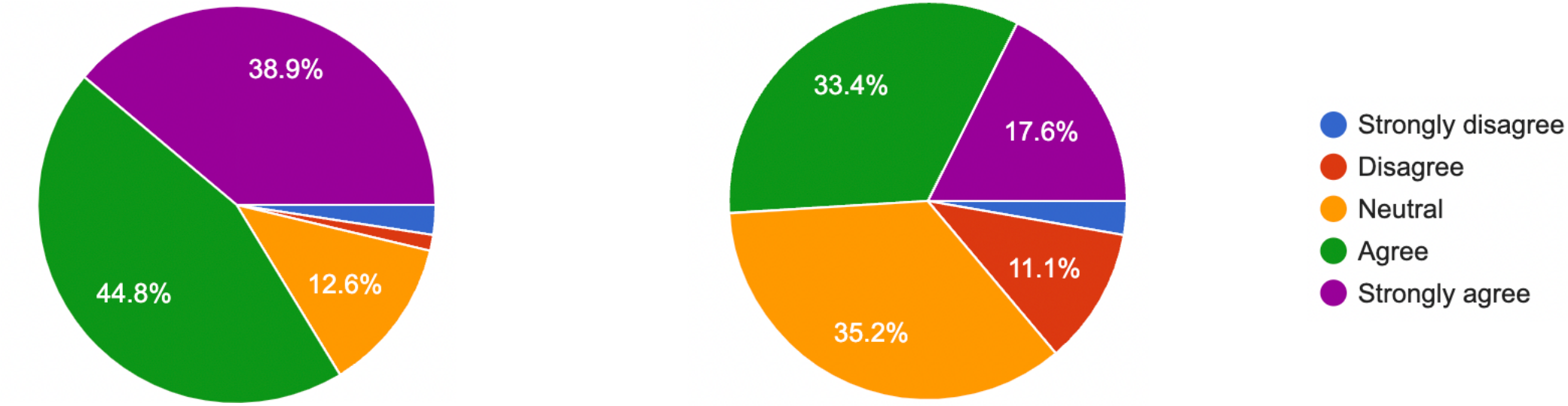
Informal Early Feedback

I attend lecture or watch lecture recordings:



Informal Early Feedback

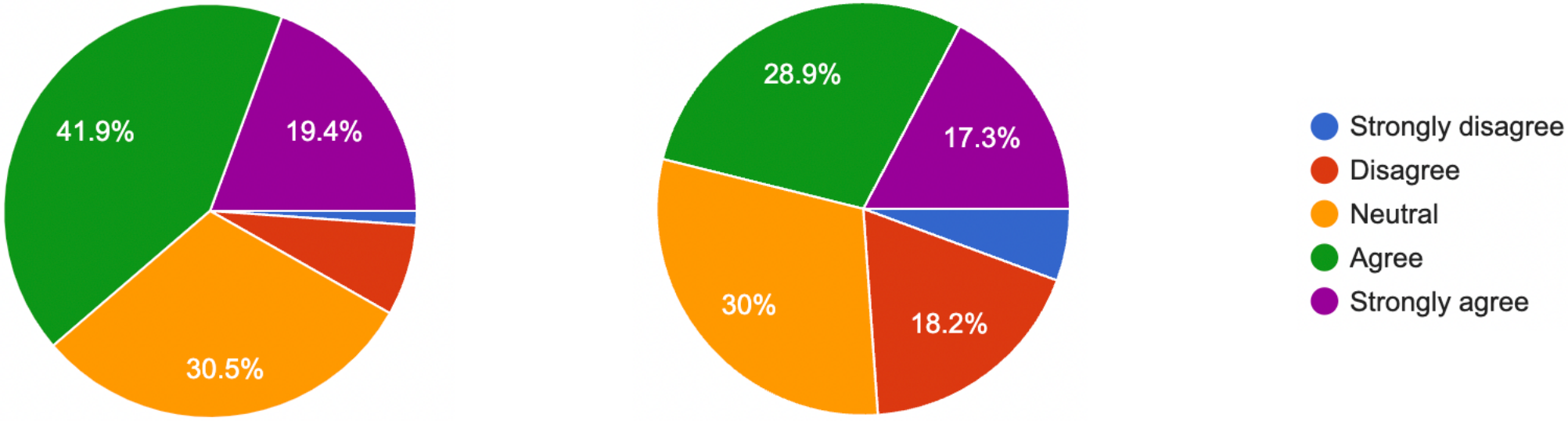
The instructor is well prepared for each lecture



I feel I can actively participate in lecture

Informal Early Feedback

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



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

Suggested Improvement (To Lectures)

Brad should improve his handwriting

Go over all MP functions in lecture / provide pseudocode

Add more introductory content (C++)

Suggested Improvement (To Lectures)

Add a weekly review session to cover topics

Slow down lectures / better motivate data structures

Reduce size of lecture (offer more sections?)

More accurate captions

Suggested Improvement (To Lectures)

Upload lecture slides earlier / make sure website matches with lecture

Upload lectures by subjects not by day

Learning Objectives

Discuss the three main types of ‘random’ in computer science

Analyze an example of each type

Randomization in Algorithms

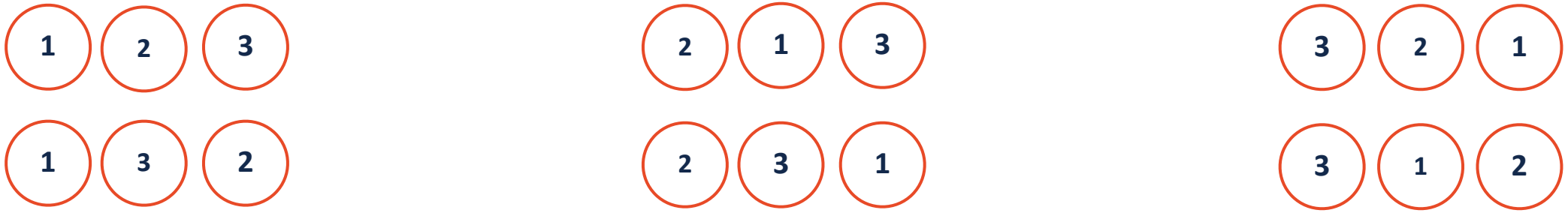
1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

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

Randomization in Algorithms

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



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 $O(n\ log\ n)$

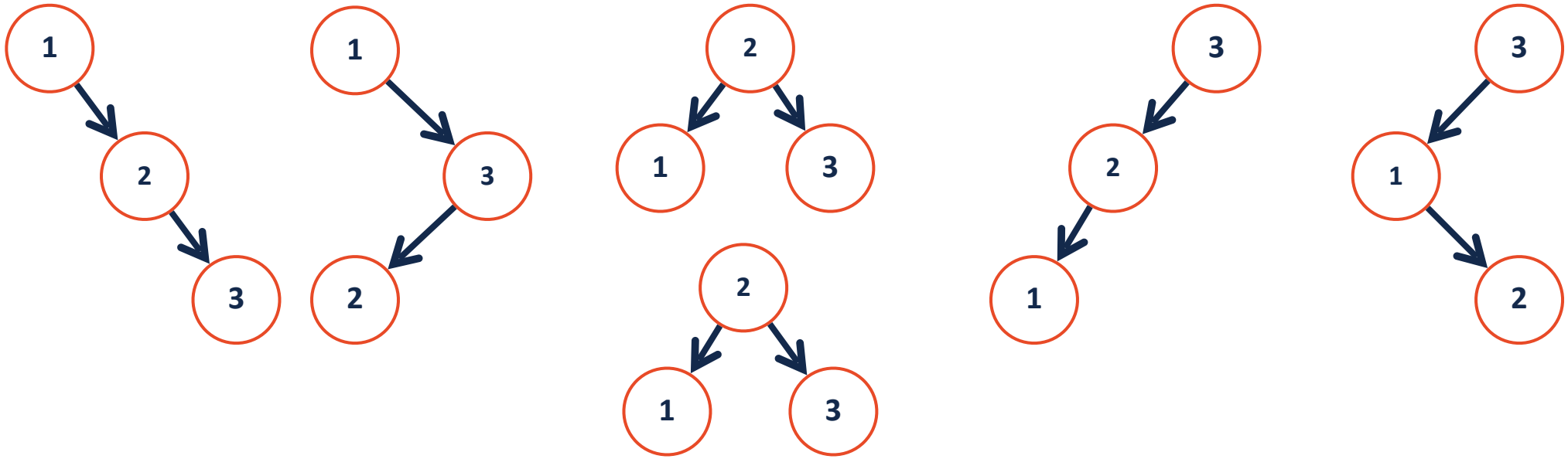
N=0:

N=1:

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

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

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

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

Average-Case Analysis: BST

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

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

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

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

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:

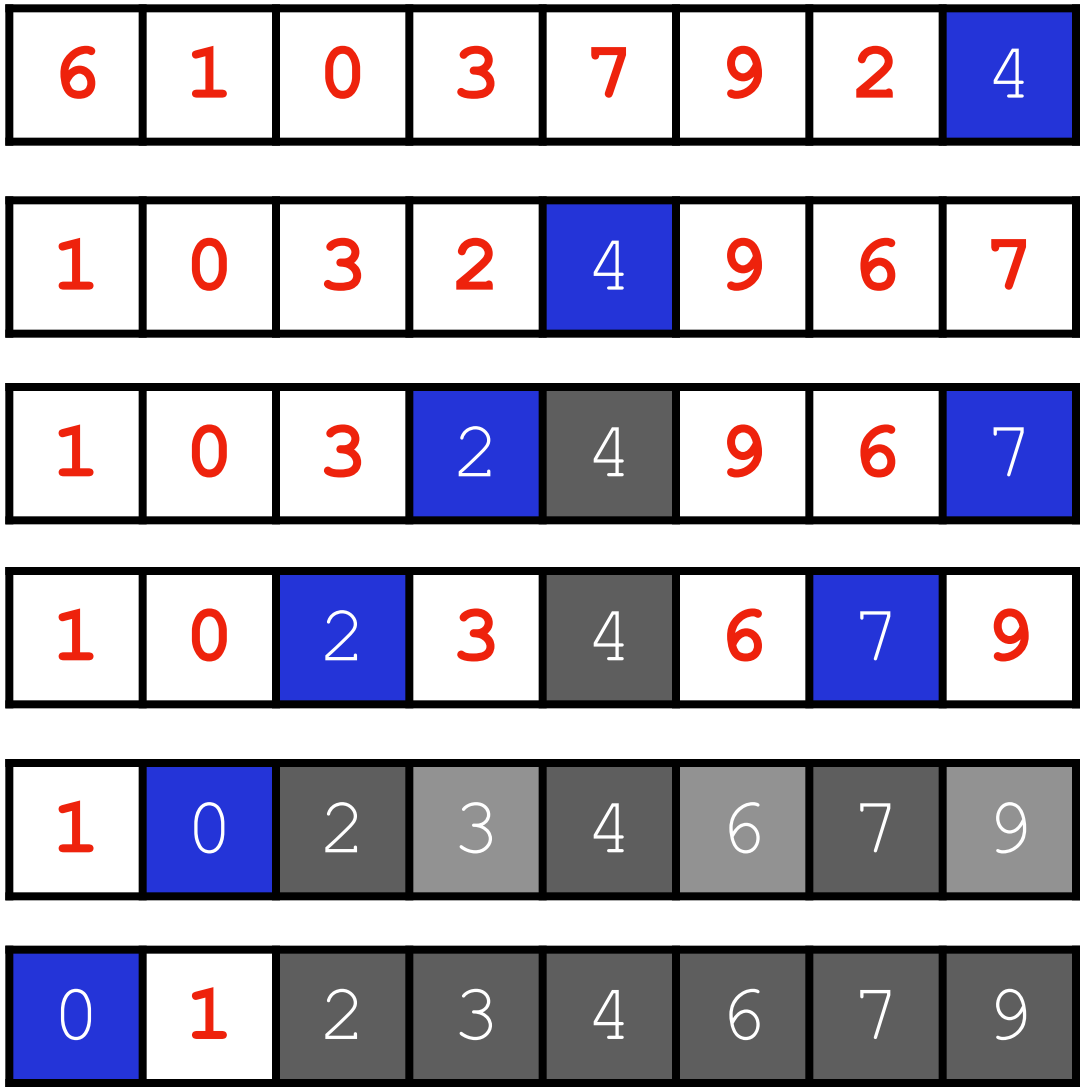
Average-Case Analysis: BST

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

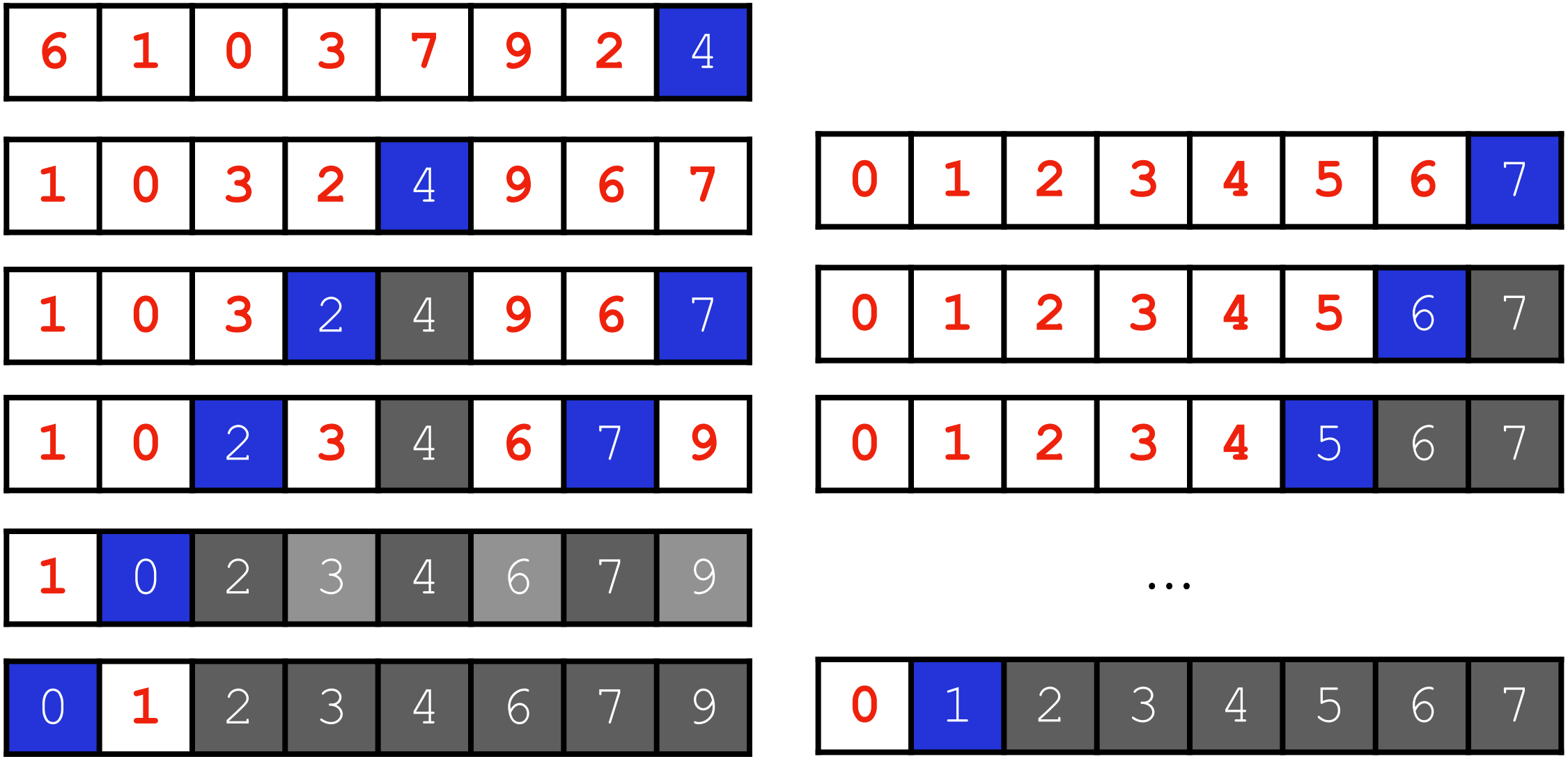
Randomness:

Assumptions:

Expectation Analysis: Randomized Quicksort



Expectation Analysis: Randomized Quicksort

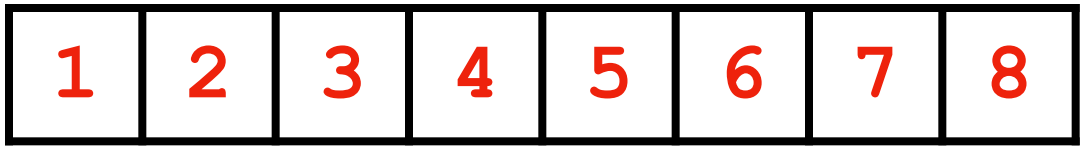


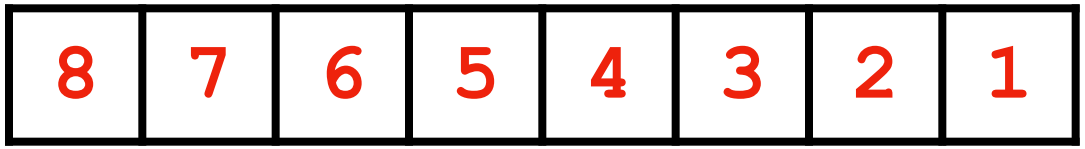
Randomization in Algorithms

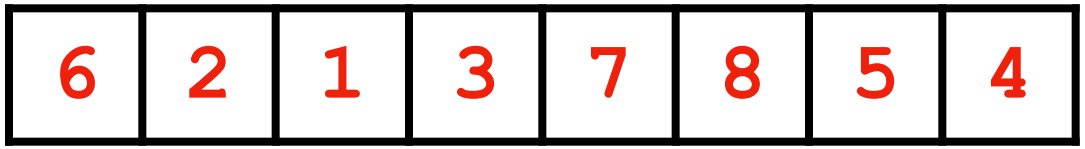
2. Use **randomness inside algorithm** to estimate expected running time

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

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







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…

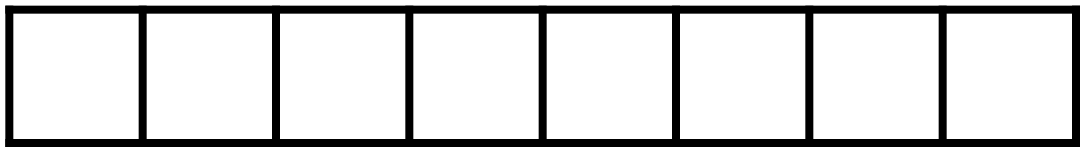
Expectation Analysis: Randomized Quicksort

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

Base Case: (N=2)

Expectation Analysis: Randomized Quicksort

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



Expectation Analysis: Randomized Quicksort

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

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 $i = 0$:

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

For $i = 1$:

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

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

Expectation Analysis: Randomized Quicksort

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

Randomness:

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!