CODEPATH*ORG

Welcome to class!

Turn on your camera

Rename yourself to include your pod # at the beginning of your Zoom name -
ie 6 - Emily Garvey (she/her)

Answer the following question in the chat:
What is your favorite dessert?

CODE PATH *ORG

Community Agreements |

- ⏰ We will be here and be present ⏰
- ❣️ Actively build community & create an inclusive space ❣️
- 🧠 We will be curious and have a learning mindset 🧠
- ❓ Ask for help when you need it ❓
-  We will all have our cameras on 

CODE PATH *ORG

Agenda

Check In	5 mins
Strings & Arrays	15 mins
UMPIRE Problem Walkthrough	25 mins
Breakout Sessions	25 mins
BREAK	5 mins
Breakout Sessions	30 mins
Wrap Up	15 mins

- More on strings and arrays
- A problem walkthrough
- Mock interviews with each other

CODE PATH *ORG

Survey Feedback | What we heard

Time |

Glows

- Working with peers in breakout room
- Live coding

Grows

- Need more explanation on union find problem

Strings & Arrays | 15 mins

CODE
PATH
*ORG

Activity | Check for Understanding

Time | [0:00 mn]

Consider the following code. What is this algorithm implementing?

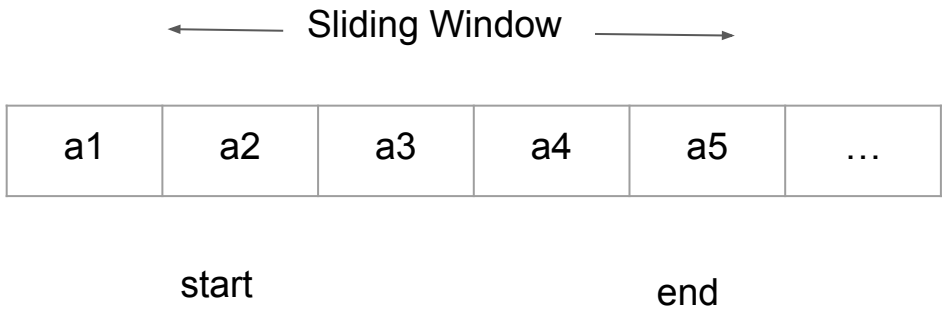
```
def solve(self, seq):
        start, end = 0, 0
        while end < len(seq):
                end += 1
                while self.start_condition(start):
                        self.process_logic1(start, end)
                        start += 1
                 self.process_logic2(start, end)
```

a) sliding window
b) DFS
c) BFS
d) none of the above

Activity | Check for Understanding

Time | [0:00 mn]

a) sliding window



We use start and end pointers to make up a window. Then we move these two pointers to make the window slide.

Sorting

- Great tool to consider if it'll help make the problem easier
  - Preprocessing arrays to simplify the problem

- Generally won't need to implement sorting
  - .sort() is sufficient
- Remember to include sort() runtime into your complexity analysis
  - O(n log n)

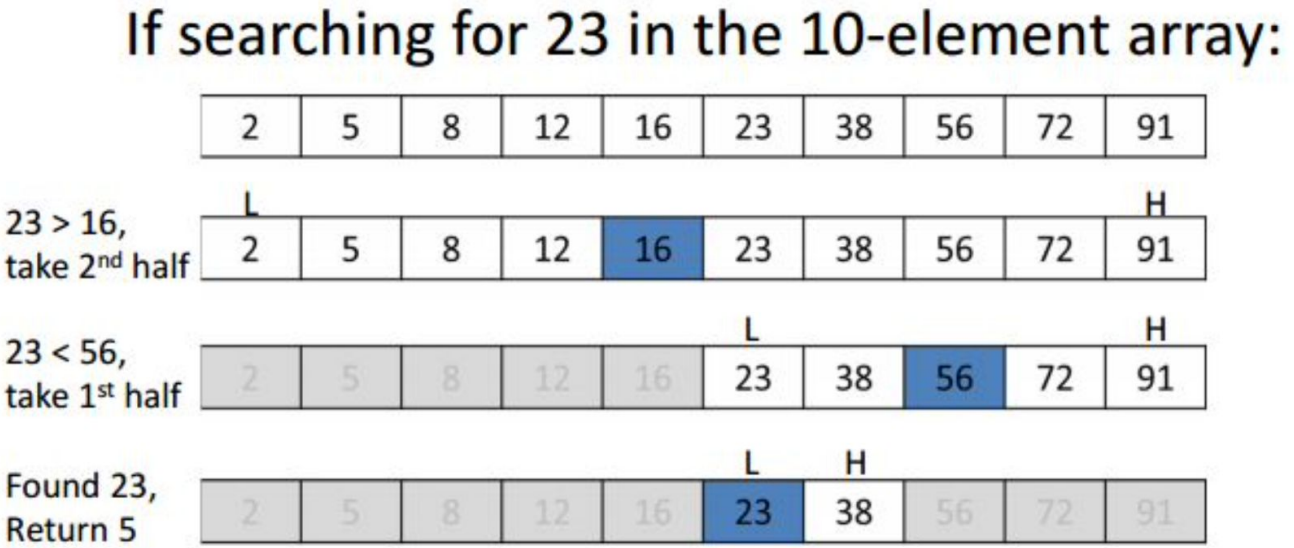
Sorting

- Understand the mechanics of popular sorting algorithms:
  - Merge sort
  - Quick sort
  - Insertion sort
  - Selection sort
  - Radix sort
- Detailed sorting algorithm explanations can be found here: https://guides.codepath.com/compsci/Sorting-Algorithms

Binary Search

- Objective: to find a value within a **sorted** list
- Take the middle element of the current search space
  - If that's the target, then you're done
  - It it's greater than the target, go left
  - Else it's less than the target, so go right
- O(log n) runtime: dividing the number of elements we search through by half each time
- Can be implemented iteratively/recursively

Binary Search



CODE PATH *ORG

sqrt(x)

- Implement int sqrt(x): compute and return the square root of x, where x is guaranteed to be a non-negative integer
- Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned

Example 1:

Input: 4

Output: 2

sqrt(x)

Example 2:

Input: 8
Output: 2
Explanation: the square root of 8 is 2.82842…, and since the decimal part is truncated, 2 is returned

sqrt(x)

Brute force: Try to square every number from 0 to target number until we reach the target

```
def bad_square_root(x):
  for i in range(x+1):
    if i * i == x:
      return i
    elif i * i > x:
      return i - 1
```

What is the runtime of this solution?

- O(square root of n)

CODE PATH *ORG

sqrt(x): optimized

We can apply binary search to find the sqrt of x

```
Find the middle number between 0 to x
If the square of the middle number is x:
  We are done
Else if the square of the middle number is > x:
  Repeat this process with the range from 0 to the middle number
Else:
  Repeat this process with the range from the middle number to x
```

What is the run time of this approach?

- `O(log n)`

sqrt(x): optimized with binary search

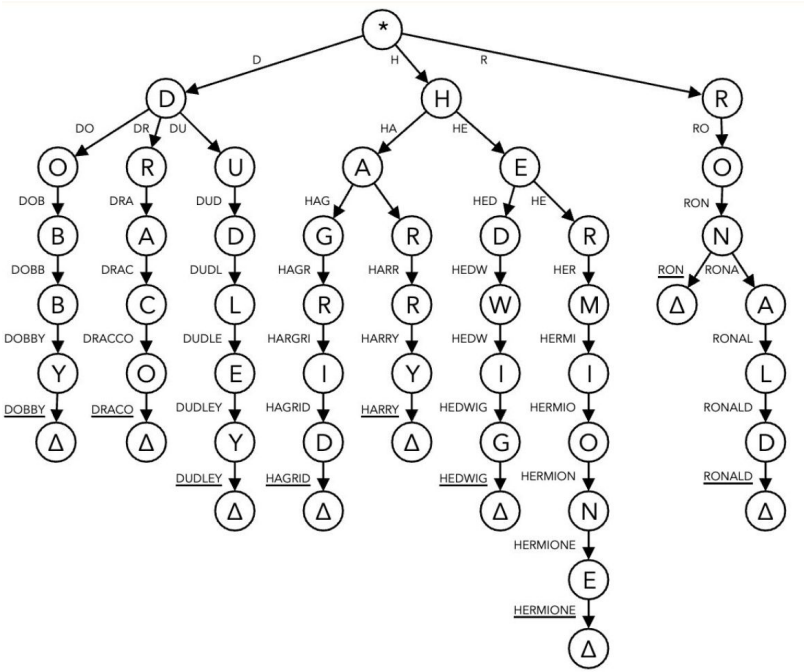
```
def square_root(x):
  if x == 0 or x == 1:
    return x
  return square_root_helper(0, x, x)

def square_root_helper(low, high, target):
  if high - low <= 1:
    return low
  mid = (high + low) // 2
  mid_squared = mid * mid
  if mid_squared == target:
    return mid
  elif mid_squared > target:
    return square_root_helper(low, mid, target)
  else:
    return square_root_helper(mid, high, target)
```

Tries

- Tries (also known as prefix trees)
  - Type of tree that stores words
  - Each node stores a character as its key and its children
  - Each path down the tree to a leaf represents a word
  - Path to internal node represent a prefix of a set of words within the trie
  - Great for efficient string lookup for a specific prefix (what are the words that begin with X?)

Tries

- Trie that stores a few Harry Potter names
- “I want to know the Harry Potter characters that have names beginning with HA”
  - Traverse from H -> A
  - Run DFS to fetch Hagrid and Harry



CODE PATH *ORG

Activity | Check for Understanding

Time | [0:00 mn]

For binary search, what is the advantage of recursive approach over an iterative approach?

a) Consumes less memory
b) Less code and easy to implement
c) Consumes more memory
d) More code has to be written

Activity | Check for Understanding

Time | [0:00 mn]

b) Less code and easy to implement

Activity | Check for Understanding

Time | [0:00 mn]

What is the worst case complexity of binary search using recursion?

a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n2)

Activity | Check for Understanding

Time | [0:00 mn]

b) O(logn)

Activity | Check for Understanding

Time | [0:00 mn]

You have to sort a list L consisting of a sorted list followed by a few “random” elements. Which of the following sorting methods would be especially suitable for such a task?

a) Insertion sort
b) Selection sort
c) Bubble sort
d) Quick sort

Activity | Check for Understanding

Time | [0:00 mn]

a) Insertion sort

Activity | Check for Understanding

Time | [0:00 mn]

Which of the following do we consider when choosing a sorting algorithm to use?

I. Space efficiency
II. Run time efficiency
III. Array size
IV. Implementation language

a) I, II, III
b) I, II
c) I, II, III, IV
d) II, III, IV
e) II, IV

Activity | Check for Understanding

Time | [0:00 mn]

c) I, II, III, IV

All of the choices are important when choosing a sorting algorithm. Space and time complexity are the characteristics by which we measure the performance of an algorithm. the array size directly affects the performance of an algorithm. In addition, the language which the algorithm is written in can also affect the performance.

UMPIRE Problem Walkthrough | 25 mins

CODE
PATH
*ORG

Max Chunks To Make Sorted

You are given an integer array arr.

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return *the largest number of chunks we can make to sort the array*.

Example 1:

Input: arr = [4,3,2,1,0]
Output: 1
Explanation:
Splitting into two or more chunks will not return the required result.
For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0, 1, 2], which isn't sorted.

Example 2:

Input: arr = [1,0,2,3,4]
Output: 4
Explanation:
We can split into two chunks, such as [1, 0], [2, 3, 4].
However, splitting into [1, 0], [2], [3], [4] is the highest number of chunks possible.

U-nderstand

When can we create a valid chunk?

5	3	6	2	6	7	9	7

U-nderstand

When can we create a valid chunk?

5	3	6	2	6	7	9	7

Can we create a chunk with 5?

U-nderstand

When can we create a valid chunk?

5	3	6	2	6	7	9	7

Can we create a chunk with 5, 3?

U-nderstand

When can we create a valid chunk?

5	3	6	2	6	7	9	7

Can we create a chunk with 5, 3, 6?

U-nderstand

When can we create a valid chunk?

5	3	6	2	6	7	9	7

Can we create a chunk with 5, 3, 6, 2?

U-nderstand

When can we create a valid chunk? At a boundary, when all the numbers in the left are less than or equal to all the numbers to the right of that boundary.

5	3	6	2	6	7	9	7

CODE
PATH
*ORG

U-nderstand

When can we create a valid chunk? At a boundary, when all the numbers in the left are less than or equal to all the numbers to the right of that boundary.

5	3	6	2	6	7	9	7

Can we create a chunk with 6?

CODE
PATH
*ORG

U-nderstand

When can we create a valid chunk? At a boundary, when all the numbers in the left are less than or equal to all the numbers to the right of that boundary.

Can we create a chunk with 7?

CODE
PATH
*ORG

U-nderstand

When can we create a valid chunk? At a boundary, when all the numbers in the left are less than or equal to all the numbers to the right of that boundary.

Can we create a chunk with 9?

CODE
PATH
*ORG

U-nderstand

When can we create a valid chunk? At a boundary, when all the numbers in the left are less than or equal to all the numbers to the right of that boundary.

Can we create a chunk with 9, 7?

CODE
PATH
*ORG

M-atch

Brute Force: For every index in the array, calculate the maximum of the elements on the left side and minimum of the elements on the right side.

- If maximum of left <= minimum of right, add one to max chunks

Optimized: Preprocess the input

- Calculate the running minimum starting from the right in an array
- Go through the array and keep track of the running maximum
    - compare the running max to the minimum of the numbers to the right of it (stored in the array calculated earlier)
    - when running max <= min of the elements to the right of it, add one to chunk

M-atch

Optimized: Preprocess the input

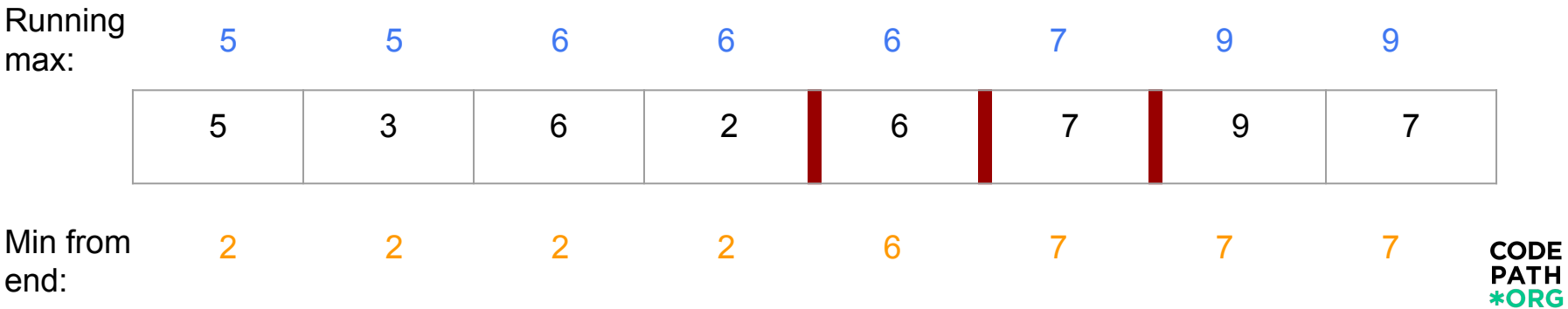
- Calculate the running minimum starting from the right in an array
- Go through the array and keep track of the running maximum
  - compare the running max to the minimum of the numbers to the right of it (stored in the array calculated earlier)
  - when running max <= min of the elements to the right of it, add one to chunk

Min from end: 2 2 2 2 6 7 7 7

M-atch

Optimized: Preprocess the input

- Calculate the running minimum starting from the right in an array
- Go through the array and keep track of the running maximum
  - compare the running max to the minimum of the numbers to the right of it (stored in the array calculated earlier)
  - when running max <= min of the elements to the right of it, add one to chunk



CODE PATH *ORG

P-lan

Get min value to the right of each element and compare it to the current one.

If the current value is smaller than (or equal to) the min value to the right, then increase the chunk count.

I-mplement

R-eview

- Trace through your code with an input to check for the expected output
- Catch possible edge cases and off-by-one errors

E-valuate

Time complexity: **O(n)**

Storing a running min / max in either direction can reduce repeated calculations

Space complexity: **O(n)**

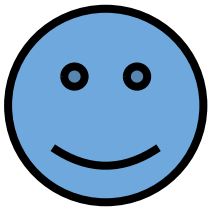
Breakout Session | 60 mins

CODE
PATH
*ORG

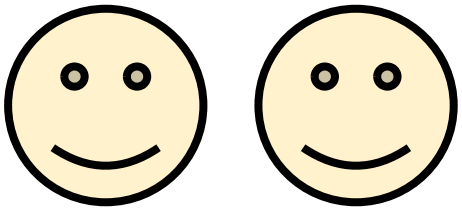
Breakout Rooms: Mock Interviews!

- We'll conduct a mock interview in your groups.
- First time: without mentors so there is no pressure!
- We'll do this multiple times during this class, so everyone will have a turn in all the roles (and sometimes, mentors will be present)

Interviewer



Interviewee



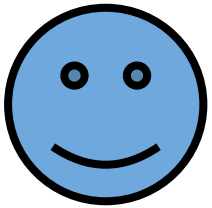
Observers

CODE PATH *ORG

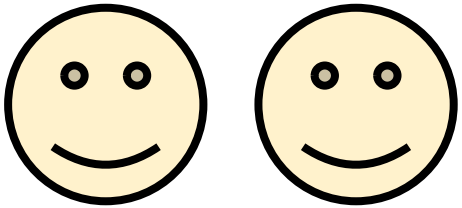
Breakout Rooms: Mock Interview Tips!

- Give everyone a few minutes to read through the question before you start
- The **Interviewer** should be the only one to open the Interviewer Script
- The **Interviewee** should share their screen
- **Observers** should follow along and be prepared to give feedback at the end
- Try to stay in character for as long as possible… but don't stress out!!

Interviewer



Interviewee



Observers

CODE PATH *ORG

Activity | Breakout Groups

60 MINS

Directions

1. Assign roles for each question so everyone can participate in a **mock interview**
2. If your pod needs help, post a message on the slack help channel and tag **@se103-tas**

Reminders

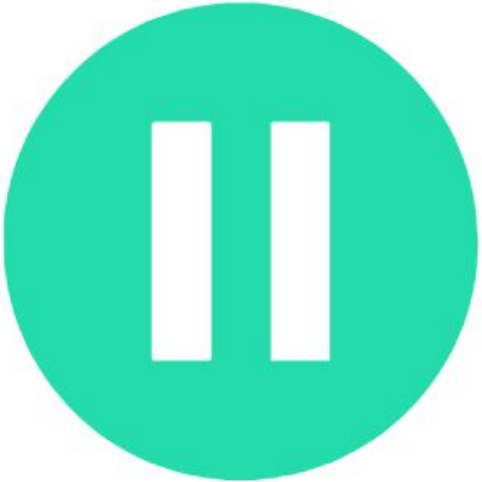
Don’t forget to turn on your cameras!

Take a 5 min break sometime during your breakout session!

CODE PATH *ORG

Break Time |

Time | [0:00 mn]



[Instructions]

Breakout Problem Review | 15 mins

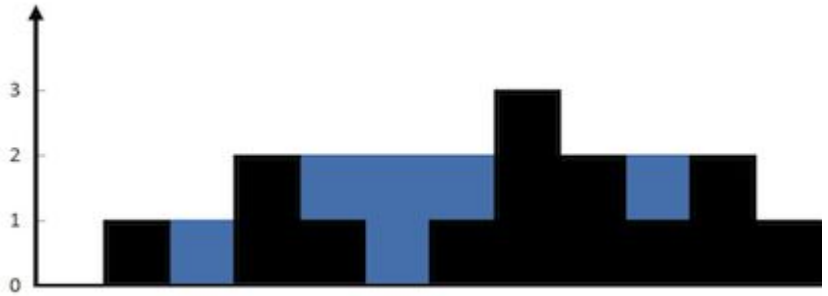
CODE
PATH
*ORG

Trapping rain water

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:

```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```



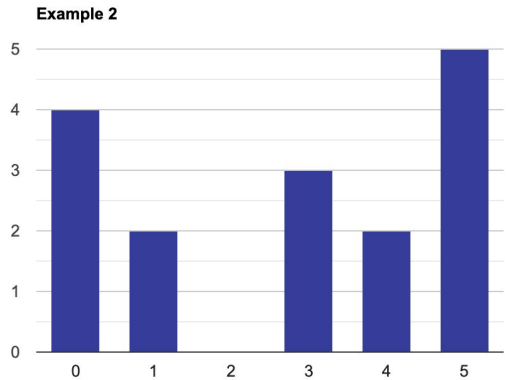
CODE PATH *ORG

Trapping rain water

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 2:

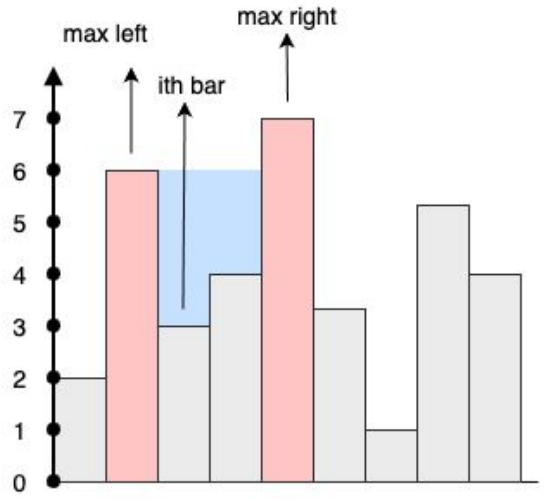
```
Input: height = [4,2,0,3,2,5]

Output: 9
```

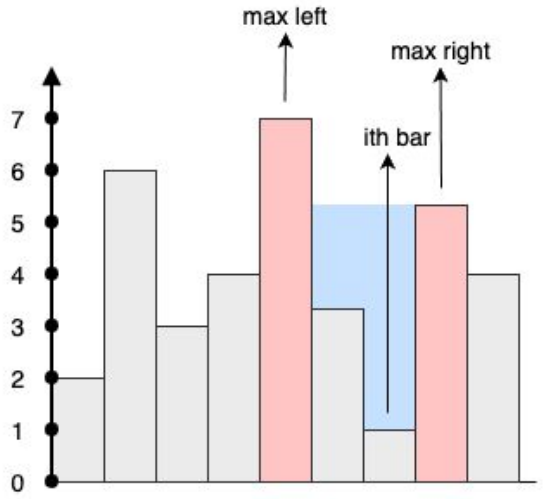


CODE PATH *ORG

Trapping rain water

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.





CODE
PATH
*ORG

Wrap Up | 10 mins

CODE
PATH
*ORG

Activity | Exit Ticket

3 MINS

Answer the 3 multiple choice in the Zoom Poll.

Exit tickets are a great way for us to check your understanding of the topics we discussed in class and identify for you what you should review after class.

Topics: Sorting and Binary Search

*Answers and explanations are available on the slides in the **Appendix** section

Exit Ticket: Question #1

Topic: Binary Search

Which of the following is true for a binary search algorithm?

a) The array being searched must be sorted before searching.
b) The array can be unsorted but will run slower than if it is sorted.
c) The array must be converted into a binary search tree before searching for the value.
d) The array needs extra space in order to accommodate swap operations that are part of the search.
e) The array must be probed in order from its beginning to its end.

Exit Ticket: Question #2

Topic: Merge Sort

Merge sort uses which of the following algorithm to implement sorting?

a) Backtracking
b) Dynamic programming
c) Divide and conquer
d) Binary search

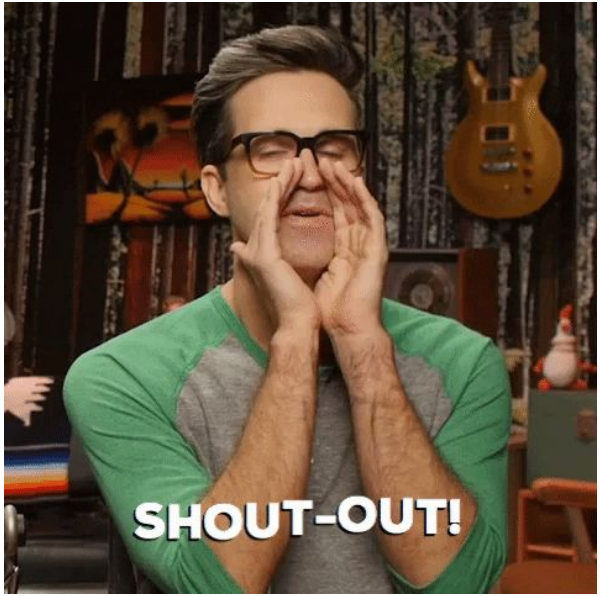
Exit Ticket: Question #3

Topic: Merge Sort

What is the average case time complexity of standard merge sort?

a) O(n log n)
b) O(n^2)
c) O(n^2 log n)
d) O(log n)

Shout Outs!



Take a moment to shoutout someone today who helped you out.

Alternatively, drop in the chat something new that you were excited to learn about today!

CODE PATH *ORG

Before you Leave |

- ❏ Complete the **Session Survey** [5 min]
- ❏ Next session is July 21, 5pm PT
- ❏ Explore the Week 8 Resources tab on the Course Portal
- ❏ Attend TA Office Hours

Appendix

Exit Ticket: Question #1

Topic: Binary Search

Which of the following is true for a binary search algorithm?

a) The array being searched must be sorted before searching.
b) The array can be unsorted but will run slower than if it is sorted.
c) The array must be converted into a binary search tree before searching for the value.
d) The array needs extra space in order to accomodate swap operations that are part of the search.
e) The array must be probed in order from its beginning to its end.

Answer: a) The array being searched must be sorted before searching.

Explanation:

The binary search presumes that the array being searched is already sorted. This is because of how it probes the values in the array. It begins at the center of the array to see if the values match. If the value is smaller than this center element, it then presumes that the value is in the lower half of the array. Otherwise, it presumes that the value must be larger and hence in the upper half of the array. It then probes the center of whichever half it has chosen, continuing to do this until the value has certainly not been found.

CODE PATH *ORG

Exit Ticket: Question #2

Topic: Merge Sort

Question:
Merge sort uses which of the following algorithm to implement sorting?

a) Backtracking
b) Dynamic programming
c) Divide and conquer
d) Binary search

Answer: C

Explanation:

Merge sort uses the technique of divide and conquer in order to sort a given array. It divides the array into two halves and apply merge sort algorithm to each half individually after which the sorted versions of these halves are merged together.

CODE PATH *ORG

Exit Ticket: Question #3

Topic: Merge Sort

Question:

What is the average case time complexity of standard merge sort?

a) O(n log n)
b) O(n^2)
c) O(n^2 log n)
d) O(log n)

Answer: A

Explanation:

The recurrence relation for merge sort is given by T(n) = 2T(n/2) + n. This is equal to O(n log n).

CODE PATH *ORG