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:
Do you have any pets? What are their names?

Agenda

Check In	5 mins
Strings & Arrays	20 mins
UMPIRE Problem Walkthrough	25 mins
Breakout Sessions	45 min
Breakout Review	15 min
Wrap Up	10 mins

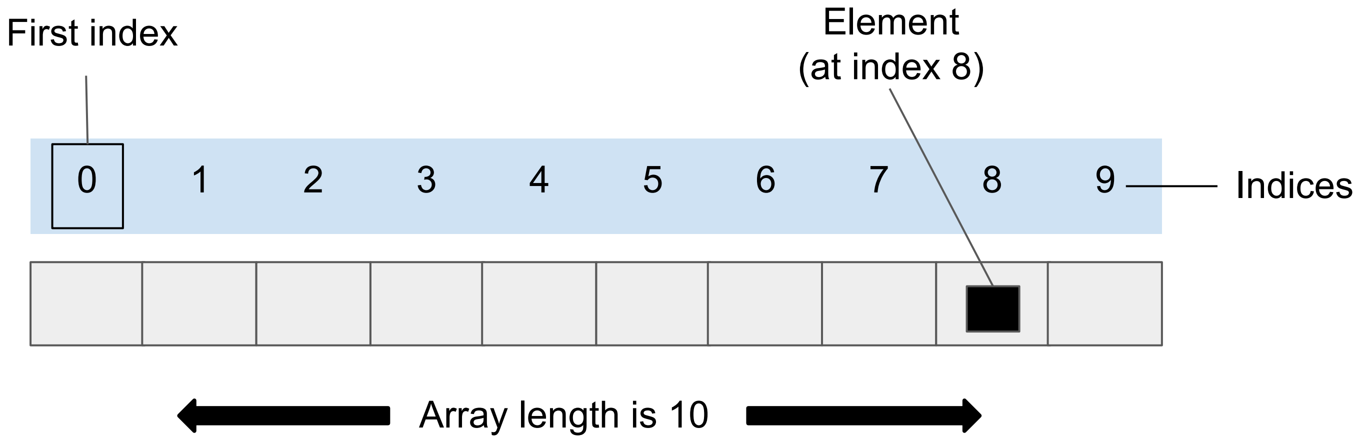
Goals

- ❑ Learn subtleties of arrays & strings
- ❑ Learn keywords to associate with array & string problems
- ❑ Learn Sliding Window Technique

Strings & Arrays | 20 mins

Arrays

Data structure that holds a “fixed” number of objects



Arrays

Strengths

- Quick index based lookups
- Amortized quick insertion at the end of the list (dynamically sized arrays)

Weaknesses

- Fixed size (in certain languages)
- Inefficient deletion and insertions in the middle of the array

Arrays Interview Questions

- Common array operations
  - Indexing, appending, inserting, removing an element, getting the length/element
  - Reversing an array
  - Getting a subarray
  - Sorting an array
- 2D arrays come up quite frequently
  - Video walkthrough of a 2D matrix question in the resource tab
- Common traversals: binary search, reverse order, matrix traversal

CODE
PATH
*ORG

Common Array Mistakes

- Getting runtime complexities wrong
  - search, remove, insert
- Using fixed/dynamic size arrays incorrectly
- Using arrays to keep track of values
  - Use a set
- Off-by-one errors, wrong indexing
  - Especially for matrices

Strings

Special kind of array, one that only contains characters

	0	1	2	3	4	5
str	G	e	e	k	s	\0
address	0x23452	0x23453	0x23454	0x23455	0x23456	0x23457

Strings Interview Questions

- Know your string operations (review the string library)
  - Indexing, appending, inserting, removing a character, getting the length
  - Sorting a string
  - Getting/finding a substring
  - Converting a character into an int (ascii value)
  - Splitting a string based on a delimiter
- Off-by-one errors
- For immutable string languages, whenever you manipulate a string, a new copy of the string is created
  - In Java, you can use StringBuffer
  - In Python, you can convert a string to a list of chars

CODE PATH *ORG

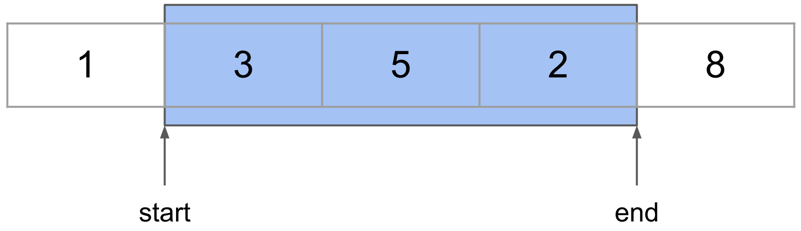
Sliding Window

- “Slide” a static/dynamically sized window along the string/array
- Use a variable/auxiliary data structure to keep track of values (eg. variable that tracks the maximum value)
- Can reduce brute force runtime to O(n) time
- Common in both array and string questions

Guide: https://guides.codepath.com/compsci/Two-pointer

Sliding Window

1. Two pointers (one for the start and one for the end of the window)
2. While loop that keeps increasing the window by incrementing the end pointer by one
3. Shrink the window by incrementing the start pointer when some condition is violated (this will depend on the problem).
    a. Use a counter or hash-map to help identify when the window is invalid
4. Update the current maximum window size, minimum window size, number of windows, etc. This will be the return value of the function.



Sliding Window

Keywords: contiguous, continuous, sequence, subarray, substring, min, max, longest, shortest

Guide with walkthrough:
https://guides.codepath.com/compsci/Two-pointer

UMPIRE Problem Walkthrough | 25 mins

Longest Substring with At Most Two Distinct Characters

https://leetcode.com/problems/longest-substring-with-at-most-two-distinct-characters/

Given a string *s*, find the length of the longest substring t that contains at most 2 distinct characters.

U-nderstand

What would the output be for these inputs?

Input: ‘aabbbbcc’

U-nderstand

What would the output be for these inputs?

Input: “aabbbbcc”

Output: 6 (either ‘**aabbbb**cc’ or ‘aa**bbbbcc**’ would work)

Input: “ababcbcbabbaadef”

Output: 6 (‘ababcbc**babbaa**def’ would be the only answer)

CODE
PATH
*ORG

U-nderstand

Edge cases?

Input: “”

Output: 0 (No characters!)

M-atch

Brute Force:
Evaluate all possible substrings and count the number of unique characters for each substring.

Time complexity?
N^2 (total number of substrings) * N (counting the number of unique characters for each substring) = N^3

How can this be improved?
Two Pointer/ Sliding Window approach seems like a good candidate to speed up run time!

CODE PATH *ORG

M-atch

What might we need to solve this using sliding window?

- Two pointers
- Perhaps some hashtable to keep track of character counts
- Maybe a counter for # of distinct characters
- Running counter for substr length and max so far

P-lan

Let’s try to work with this string: ‘ccacbbabba’

The output should be **6** from the substring ‘**bbabba**’

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ‘c’

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘c’~~ ‘cc’

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘cc’~~ ‘cca’

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ~~‘cca’~~ ‘ccac’

CODE
PATH
*ORG

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ‘ccac’

‘ccacb’ is no longer a valid substring

P-lan

```
start
c   c   a   c   b   b   a   b   b   a
                end
```

Longest valid substring so far: ‘ccac’

We need to shrink the window by moving the start pointer to the right until it’s a valid window again

P-lan

~~start~~ start

c c a c b b a b b a

end

Longest valid substring so far: ‘ccac’

P-lan

start

c c a c b b a b b a

end

Longest valid substring so far: ‘ccac’

CODE PATH *ORG

P-lan

```
            start
c   c   a   c   b   b   a   b   b   a
                        end
```

Another invalid substring, where do we move the start pointer now?

P-lan

```
         start
c  c  a  c  b  b  a  b  b  a
                  end
```

Another invalid substring, where do we move the start pointer now?

We can move the start pointer one position to the right

CODE
PATH
*ORG

P-lan

~~start~~ start

c c a c b b a b b a

end

Longest substring so far: ‘ccac’

P-lan

```
                start
c   c   a   c   b   b   a   b   b   a
                                end
```

Longest substring so far: ~~‘ccac’~~ ‘bbabb’

P-lan

start

c c a c b b a b b a

end

Longest substring so far: ~~‘ccac’~~ ‘bbabba’

CODE
PATH
*ORG

P-lan

- If the current window only has 2 distinct characters, we grow the window by incrementing the end pointer
- If the current window is no longer valid (has 3 distinct characters), we shrink the window it is valid
    - Keep a mapping of characters in the window to their counts within the window to track when we can stop moving the start pointer to the right
- Update the maximum substring length along the way

P-seudocode

```
Char_to_counts = {} #dictionary that maps characters to its counts within the window

While end pointer isn’t past the end of the string:
    Update end character count in char_to_counts
    While char_to_counts has more than 2 characters: #window is no longer valid
        Decrement start character’s count
        Increment start index by 1

        #character will no longer be in the window
        Remove entry from char_to_counts if count is 0
    Update the max length
    Increment end pointer
```

CODE PATH *ORG

I-mplement (python)

```python
def longest_substr_dist_char_count_2(string: str) -> int:
        char_counts = collections.defaultdict(int)
        n = len(string)
        start = end = longest = 0

        while (end < n):
            char_counts[string[end]] += 1
            end += 1

            while (len(char_counts) > 2):
                c = string[start]
                char_counts[c] -= 1

                if (char_counts[c] == 0):
                    char_counts.pop(c)

                start += 1

            longest = max(longest, end - start)

        return longest
```

CODE
PATH
*ORG

I-mplement (java)

```java
public int lengthOfLongestSubstringTwoDistinct(String s) {
    if (s == null) {
        return 0;
    }

    Map<Character, Integer> map = new HashMap<Character, Integer>();
    int start = 0;
    int end = 0;
    int max = 0;

    while (end < s.length()) {
        char c = s.charAt(end++);
        map.put(c, map.containsKey(c) ? map.get(c) + 1: 1);

        while (map.size() > 2) {
            c = s.charAt(start++);
            map.put(c, map.get(c) - 1);

            if (map.get(c) == 0) {
                map.remove(c);
            }
        }

        max = Math.max(max, end - start);
    }

    return max;
}
```

CODE
PATH
*ORG

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)

- ‘end’ is incrementing once per loop and we stop when it reaches the length of the string
- No character is being looked at more than twice: once for ‘end’ and once for ‘start’

Space Complexity O(1)

- We assume a finite and small set of unique characters, so O(1) is a valid answer. For example, ascii would give us 256 characters, not large compared to the size of the input (potentially millions of characters)

CODE PATH *ORG

Breakout Session | 45 mins

CODE
PATH
*ORG

Activity | Breakout Groups

45 MINS

Directions

1. Answer an icebreaker question together
2. Work together through the UMPIRE steps of the problems in the Course Portal under Week 5, Session 1
3. If your pod needs help, post a message on the slack 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

Breakout Problem Review | 15 mins

Wrap Up | 10 mins

Activity | Exit Ticket

3 MINS

Answer the single 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.

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

Exit Ticket

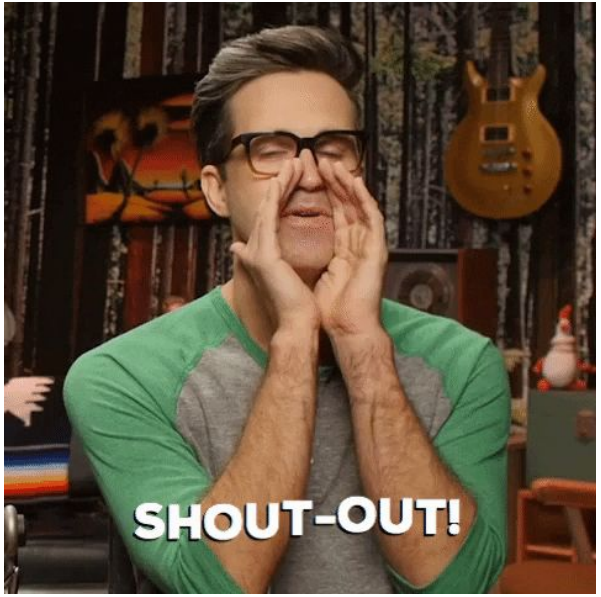
What is the runtime of the given plan?

a) $O(n*m)$
b) $O(n^2)$
c) $O(n^{2}*\log(n))$
d) $O(m*n*\log(n))$
e) $O(n*m*\log(m))$

```
# number of strings = m
# characters in a string = n
#
# iterate through all strings in the array
# for each string, sort it
# check hashtable for sorted string.
    # if not there, create an array, add the unsorted
string to the array, and add the array to the hashtable.
    # if there, add the unsorted string to the array at
the given key
# iterate through the hashtable, gather up all the arrays,
and return them
```

CODE
PATH
*ORG

Shout Outs!



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

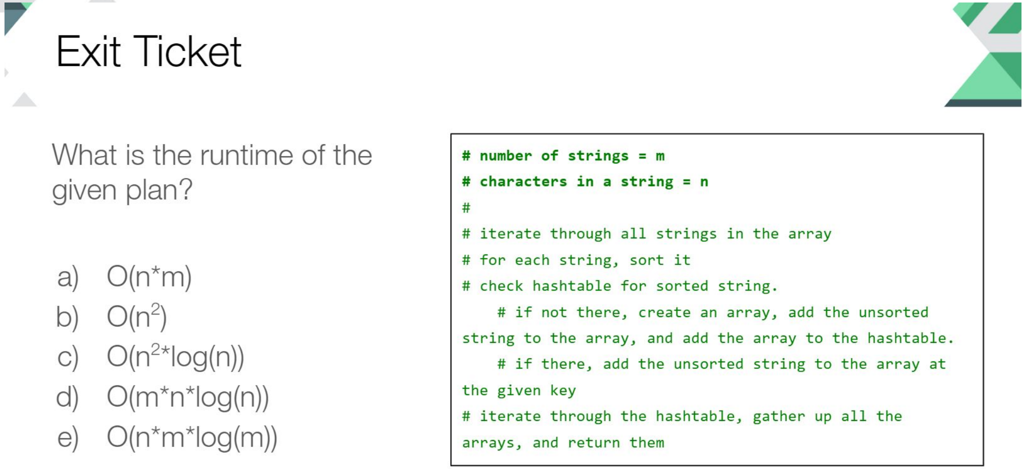
Alternatively, drop in the chat something new that you were excited to learn about today!

Before you Leave |

- ❏ Complete the [Session Survey]
- ❏ Complete the [Early-Course Survey]
- ❏ Next class: Saturday 10/1
- ❏ Explore the Week 3 Resources tab on the Course Portal
- ❏ Attend TA Office Hours
- ❏ Check out the Week 3 Career tab on: **Building Your Personal Network**

Appendix

Exit Ticket Answer



Exit Ticket

What is the runtime of the given plan?

a) O(n*m)
b) O(n^2)
c) O(n^2*log(n))
d) O(m*n*log(n))
e) O(n*m*log(m))

```
# number of strings = m
# characters in a string = n
#
# iterate through all strings in the array
# for each string, sort it
# check hashtable for sorted string.
    # if not there, create an array, add the unsorted
string to the array, and add the array to the hashtable.
    # if there, add the unsorted string to the array at
the given key
# iterate through the hashtable, gather up all the
arrays, and return them
```

d) O(m*n*log(n))
m strings are being sorted, and each string has n characters. Thus we are doing nlog(n) sort m times.