Welcome!

- We'll get started in a moment😊

WOMEN WHO CODE

WOMEN WHO CODE

Strings

- What are Strings?
  - a sequence of characters.
- String questions can be treated like array questions except we have access to string methods:
  - E.g. substring, equals etc.

General Interview Tips

- Firstly it is important to clarify exactly what the question is, what are the inputs, what are the assumptions, what is the expected output?
- Communicate with the interviewer, if you need time to think by yourself, let them know. Try to clearly communicate your thought process and decisions.
- Reach consensus with interviewer about your solution before implementing.
- In your implementation, focus on readability of code.
- Run through different test cases at the end if you have time.
- Don’t forget edge cases!

Types of String Problems

- Strings are used as a basis for a large variety of questions solved with many different techniques.
- We'll focus on questions that cover unique aspects of strings:
  - Properties of words/strings: e.g. anagrams, palindromes, lexicographical etc.
  - String manipulation.
  - Prefixes/substrings/subsequences

Problem 1: Reverse String

- https://leetcode.com/problems/reverse-string/solution/
- Given a string, reverse it in place.
- In place means not using any additional data structure.

Problem 1: Reverse String

- Given a string, reverse it in place.
- In place means not using any additional data structure.

Problem 2: Valid Anagram

- https://leetcode.com/problems/valid-anagram/
- Determine whether one string is an anagram of an another.
- If the letters in one word can be rearranged to form another word, then the two words are anagrams.
- Constraints: lowercase letters only.

Problem 2: Valid Anagram

- Naive approach: find all rearrangements of one word, and see if the second word is one of them.
- N! Runtime.
- Can we make any improvements?

Problem 2: Valid Anagram

- Approach 1:
- Don’t need to generate all the rearrangements, rather we can try to generate the second word, by using the letters in the first word. If we can, the words are anagrams.
- For each letter in the first word, see if we can use an unused letter in the second word.
- Need to keep track of used indices because of duplicate letters.
- O(n^2) time complexity, O(n) space complexity.

Problem 2: Valid Anagram

- Approach 2:
- We are essentially just trying to see if the number of times a letter occurs in one word equals the number of times the same letter occurs in the other.
- Rather than for each letter, go through the second word, we can count frequency in first word first, then compare with second word.
- If the each letter in one word occur at the same frequency as in the other, then they are anagrams.
- Use a hashmap to store counts of each unique letter in one word and see if they match the second word.
- Since we are constrained with lower case letters, we can also use an array of size 26.
- O(n) time complexity, O(1) space complexity.

Problem 2: Valid Anagram

- Alternate Solution
- Another way of thinking about the problem: is there a way to identify a group of anagrams, that is all rearrangements of a word. A few different ways of doing so but the most straightforward way: sort the letters in alphabetical order.
- Two anagrams sorted in alphabetical order will be equal.
- O(nlogn) solution.