Intermediate Software Engineering

Intermediate-Software-Engineering-Summer-2022 (@ Intermediate)
Personal Member ID#: 52751

String and Arrays UMPIRE Cheat Sheet

Definition

Arrays

An **array** is a data structure that holds a fixed number of objects. Because arrays have fixed sizes, they are highly efficient for quick lookups regardless of how big the array is. However, there is a tradeoff with this fast access time; any insertion or deletion from the middle of the array requires moving the rest of the elements to fill in or close the gap. To optimize time efficiency, try to add and delete mostly from the end of the array.

Arrays commonly come up in interviews, so it's important to review the array library for the language you code in.

Tips:

- Off-by-one errors can often happen with arrays, so be wary of potentially over indexing as it will throw an error
- Try to add elements to the back of an array instead of the front, as adding to the front requires shifting every element back
- In Java, arrays are a fixed size so consider utilizing an ArrayList instead if you need to dynamically alter the size of the array.

Strings

Strings are a special kind of array, one that only contains characters. They commonly come up in interview questions, so it's important to go through the string library for the language you're most comfortable with. You should know common operations such as: getting the length, getting a substring, splitting a string based on a delimiter, etc.

It's important to note that whenever you manipulate a string, a new copy of the string is created. There are different ways to reduce the space utilized depending on the language:

- In Python, you can represent a string as a list of characters and operate on the list of character instead.
- In Java, you can utilize the StringBuffer class to mitigate the amount of space utilized if you need to manipulate a string.

U-nderstand

What are some common questions we should ask our interviewer?

- Are there memory constraints?
- What's the required time complexity?
- What kind of data will the inputs be?
- Can I assume all the inputs will be valid?
- What if the input is empty?
- Could the input numbers be negative?
- Could the input array be null?
- What should we return if there is no solution to the problem?
- What should we return if there are multiple solutions to the problem?
- Could an index have a negative value?

M-atch

Are there any special techniques that we can use to help make this easier?

In the Matching step of UMPIRE, you want to think about common patterns and tricks that could apply to this problem.

In Strings/Arrays, common problem patterns include:

- **Hash and Store**
  - At each index, what information do we need? A potential solution could store the minimum number to the right of each index, key is the index. We could potentially hash that information; however, an array could serve the same purpose.
  - If you are given a sequence and the interviewer asks for `O(1)` space, it might be possible to use the array itself as a hash table. For example, if the array only has values from 1 to N, where N is the length of the array, negate the value at that index (minus one) to indicate presence of that number.
- **Two Pointer**
  - Two pointer may help us reference both sides of the array at the same time. However, since the array is not sorted, a two pointer solution may not assist us as much.
- **Sort**
  - How do we know when to split a chunk? If we know the underlying strategy on when to split a chunk (which relies on the sorted version of the data), sorting may potentially help us.
- **Pre-computation**
  - For questions where summation or multiplication of a subarray is involved, pre-computation using hashing or a prefix/suffix sum/product might be useful.
- **Sliding Window**
  - Master the sliding window technique that applies to many subarray/substring problems. In a sliding window, the two pointers usually move in the same direction will never overtake each

other. This ensures that each value is only visited at most twice and the time complexity is still O(n).

- **Traversing the array more than once**
  - Traversing the array twice/thrice (as long as fewer than n times) is still O(n). Sometimes traversing the array more than once can help you solve the problem while keeping the time complexity to O(n).

Plan

Before ANY pseudocode, have students explain a general approach to the problem. Ask the students how they would solve this problem by hand if they were to. This gives a good idea to the brute force solution. Optimizations can occur on top of this or can inspire new approaches.

❗ If students struggle to create an algorithm pose these questions:

“When do we know we can chunk off a section? Let’s say the first chunk, at what point do we know it is safe enough to separate and sort this subarray?” This should hint to the student the exact solution to the problem.

“If we are at an index and there are no smaller values to the right of that index, can we make a chunk at that point? Why or why not?” Mentors, before diving into specifics about algorithms, have students give a 1-2 sentence summary on their approach.

P-lan/Pseudocode

- Can you create any **magic** helper methods that would simplify the solution?
- Talk through different approaches you can take, and their tradeoffs
- Be able to verbally describe your approach and explain how an example input would produce the desired output

Tips:

- Try to avoid nested loops
  - This is usually a brute force solution and is $O(n^2)$ time complexity

E-valuate

Time Complexity

	Best Case	Worst Case
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	O(n)	O(n)
Deletion	O(n)	O(n)

Strings Resources

General guide

- Coding for Interviews Strings Guide

Strings in C++

- Character Arrays in C++
- Character Arrays in C++ Part 2

Arrays Resources

General guide

- InterviewCake Arrays

Python arrays

- Google developer lists guide

Java arrays

- InterviewCake DynamicArray
- ArrayList Succinctly Guide