A Technical Interview Study Guide

Nicholas White

Before diving into the study guide, I recommend spending 3-5 days getting comfortable with the programming language you'll be interviewing in. One of my favorite resources for getting up to speed with a programming language is thenewboston. He has in depth tutorials that are easy to understand in almost every programming language.

WEEK 1

Hey guys,

I'm planning on making a longer study guide over time, but here is a list of questions in order that I think will help shape a solid understanding of algorithms at the fundamental level.

Before starting these questions I would expect you to know the basics of a programming language (if statements, loops, functions, etc) and I would hope that at this point you have at least heard about data structures and algorithms (linked lists, hash maps, binary search, etc).

If you do not feel comfortable with at least one programming language I would highly recommend doing the practice problems for a programming language on HackerRank. https://www.hackerrank.com/dashboard

Find the language proficiency section and try and complete about 50% of the practice problems for one of the languages. You don't have to do them all, but if you're not comfortable with programming then algorithms will probably be very difficult for you.

Starting Point

Gayle Laakmann McDowell has some great videos explaining a lot of the important concepts involved in solving algorithm problems and cracking the coding interview.
I recommend watching these videos before beginning problems. You don't need a full understanding of the videos right now, but try and keep some of the information from them in the back of your mind as we go through the problems.
https://www.youtube.com/watch?v=GKgAVjJxh9w&list=PLOuZYwbmgZWXvkghUyMLdI90IwxbNCiWK&index=1
https://www.youtube.com/watch?v=84UYVCluClQ&list=PLOuZYwbmgZWXvkghUyMLdI90IwxbNCiWK&index=2
Feel free to continue referencing back to Gayle's videos as you go through the problems.

Algorithm Challenges for Beginners - Arrays
So to start off I think you should gain a solid fundamental understanding of arrays. Arrays are one of the most commonly used data structures in all of programming if not the most.
I'm sure you've used arrays (or lists in python) by now where you use them to store maybe a list of integers or strings (ex : [1, 2, 3] or ["hello", "whats up", "hi"]).

Solve these three array challenges below.

Question 1 - https://www.hackerrank.com/challenges/arrays-ds/problem
(Problem : printing elements of an array in reversed order)
Question 2 - https://leetcode.com/problems/monotonic-array/
(Problem : monotonic array)
Question 3 - https://leetcode.com/problems/third-maximum-number/
(Problem : find the third maximum number in an array)
Question 4 - https://www.hackerrank.com/challenges/2d-array/problem
(Problem : printing the max hourglass sum in a 2D array)
Question 5 - https://leetcode.com/problems/rotate-array/
(Problem : Rotate Array)

Solve one of the following array challenges. (Solve both recommended)
Question 6 - https://leetcode.com/problems/sort-array-by-parity/
(Problem : Sort Array By Parity)
Question 7 - https://leetcode.com/problems/flipping-an-image/
(Problem : Flipping an Image)

Arrays are extremely important and are involved in so many technical interviews that it would be strange if they didn't come up during an interview. Knowing how to traverse and manipulate arrays is essential to your success in a technical interview.
If you feel like this stuff is easy I'd say feel free to skip ahead and do some of the later problems.

Algorithm Challenges for Beginners - Binary Search

If you solved the problems above I feel confident that you have a basic understanding of arrays. Well, now that you know how arrays work, I think it's a great time to introduce binary search. I'm sure you've either heard of it or implemented it before but binary

search is so important for technical interviews. There are a bunch of problems that ask you to implement a variation of binary search to solve a specific task and these variation problems often come up in internship interviews. I've personally been asked a binary search question during an actual technical interview with a big tech company.
There are many variations of binary search that we implement depending on what the problem is asking you to do, but for now I think you should just implement a basic binary search.
Question 1 - https://leetcode.com/problems/binary-search/
(Problem : implement binary search)
Binary search is an awesome searching algorithm that can be implemented on sorted arrays only. It cuts search time from O(N) down to O(logN) which is huge in terms of time complexity!
If you don't understand time and space complexity I highly recommend the videos below.
https://www.youtube.com/watch?v=D6xkbGLQesk
https://www.youtube.com/watch?v=v4cd1O4zkGw&t=250s
Time and space are extremely important during the technical interview because you are essentially trying to find a solution to the problem with the best time and space complexities (Meaning fastest performing and least amount of extra space).
Try and really understand what binary search is doing and why it's a better solution to searching than a linear scan.
Now that we have a basic understanding of arrays, binary search, and time complexity we can move on to linked lists.

Algorithm Challenges for Beginners - Linked Lists

I feel that Linked Lists are probably the next easiest data structure to learn and they're certainly important.

The only hard part about understanding Linked Lists is that they're weird. Up until now, unless you understand data structures, you probably haven't thought about how things are working behind the scenes when you write code. You just know that there are rules and techniques in programming and you follow the rules and use those techniques to solve problems.
If you understand data structures or linked lists maybe skip this part, but if you don't I definitely recommend watching these videos.
https://www.youtube.com/watch?v=njTh_OwMljA
https://www.youtube.com/watch?v=WwfhLC16bis

Without understanding linked lists and how we might be able to loop through them and access node values it might be difficult to do the problems below. Once you understand that a linked list is a bunch of connected nodes and once you know how traverse a linked list please try out the problems below.

Question 1 -
https://www.hackerrank.com/challenges/print-the-elements-of-a-linked-list/problem
(Problem : Print the Elements of a Linked List)
Notes - Check the problem discussion boards out if you still don't understand

Question 2 -
https://www.hackerrank.com/challenges/insert-a-node-at-the-tail-of-a-linked-list/problem
(Problem : Insert a Node at the Tail of a Linked List)

Question 3 -
https://www.hackerrank.com/challenges/insert-a-node-at-the-head-of-a-linked-list/problem
(Problem : Insert a Node at the Head of a Linked List)

Linked List problems don’t come up very often in technical interviews, but they certainly could. They are one of the more simple problem types to master and expose you two other problem types such as “two pointer / slow & fast pointer” problems. Once you get used to them you might as well just sit down for a couple of days and go through them all! (all easy + medium ones)

Break point 1

Okay, so at this point we've gone over a ton of foundational material. Congrats on making it this far! The only thing I'd like to ask you now is, do you get it?
My advice right now would be to try and explain all of these concepts out loud to someone (or yourself) and it you're unable to do that, then you probably don't have a full understanding of the material yet.
What is an array?
What is binary search and why would I want to use it?
What's a linked list and how is it different than an array?
If you have even just an elementary understanding of arrays, binary searching, and linked lists then you should easily be able to answer all of these questions. Of course we have only just begun our exploration into these concepts but understanding these three things will lay the groundwork for the rest of the study guide.

Alright that's it for right now. I plan to continue this study guide and merge all of the parts together by the end of it. Congratulations on making it to the end of part 1 and good luck studying!

WEEK 2 - HASHMAPS AND HASH SETS

Hash Maps are pretty much guaranteed to come up in every technical interview one way or another. They are a very efficient data structure when we need to keep track of things.

Hash Maps are sometimes referred to as “Dictionaries” because they function similarly to dictionaries where in a dictionary you look up the definition associated with a word, in a hash map we can lookup values associated with keys. These keys and values can be any of the primitive types and we can define what goes in our hash map.

Hash Maps

If you don’t remember what hash maps / hash tables are here’s a great video to catch you up! https://www.youtube.com/watch?v=shs0KM3wKv8

Question 1 - https://leetcode.com/problems/single-number/
(Problem : Hash Tables : Single Number)
Notes - Use a hashmap to implement the solution to this problem. Don’t worry about the bit manipulation / math.

Question 2 - https://www.hackerrank.com/challenges/ctci-ransom-note/
(Problem : Hash Tables : Ransom Note)

Question 3 - https://www.hackerrank.com/challenges/ctci-ransom-note/
(Problem : Hash Tables : Ransom Note)

Question 4 - https://leetcode.com/problems/two-sum/
(Problem : Two Sum)

Notes - I'd recommend watching a Google made video example of a technical interview where they solve two sum! https://www.youtube.com/watch?v=XKu_SEDAykw
This is one of my favorite mock interview videos on the internet and it shows ideally how a technical interview will go.

Question 5 - https://leetcode.com/problems/3sum/
(Problem : ThreeSum)

Notes - Now that you've completed two sum, three sum should be pretty simple since it's just an extended version of two sum! I don't think completing 4sum is necessary just because it's such a long problem it probably won't be given in an interview, but feel free to try it if you're feeling confident! Three sum is totally fair game though!

Hash Sets

Question 1 - https://leetcode.com/problems/contains-duplicate/
(Problem : Contains Duplicate)

Question 2 - https://www.hackerrank.com/challenges/two-strings/
(Problem : Two Strings)

At the end of each week, I think it might be best to solve some problems related to the week prior. That way we can refresh ourselves on old topics so that we don't forget material.

WEEK 3 - STACKS AND QUEUES

Hopefully things haven't been too difficult up until this point for anyone. To continue from here, we're going to learn about some more basic data structures. Many Universities and online classes teach one data structure at a time thoroughly until students reach a full understanding, but we're going to pursue an uncommon strategy. The strategy here is to do an initial sweep through all of the different data structures to gain an initial understanding of what types of data structures there are and afterwards we will explore each one individually.

The next data structure we're going to look at is a Queue. A Queue seems like the next easiest data structure to learn about simply because it is so similar to Linked Lists and Arrays.
Before diving into questions it might be a good idea to watch some videos.
https://www.youtube.com/watch?v=wjI1WNcIntg
https://www.youtube.com/watch?v=PjQdvpWfCmE

A Queue is very much like a Linked List or an Array, but it's used primarily in situations where we need to organize things as if they are in a line. So for example, if we were building some kind of drive thru software that tracked customer orders, we might use a queue because a drive thru typically functions as a line.
Something you need to understand is that we can write our own data structures and all a data structure really is, is a way to organize data. Hopefully that makes sense.

Here's a pretty simple problem. Try and solve it using a Queue.
Question 1 - https://leetcode.com/problems/number-of-recent-calls/
(Problem : Number of Recent Calls)

I also have a video solution for this problem if you're having a tough time.
https://www.youtube.com/watch?v=HlmNEfcgyjM&t=22s

Something you need to take away from all of this is that a Queue, A Linked List, and an Array are all just structures for our data. We are using these structures to organize our data which makes it easier to solve problems.

So Queues are known for their FIFO (first in first out) style of organization while another very important data structure, a Stack, is known for its LIFO (last in first out) style of organization. Stacks are extremely important and come up in so many problems that I would consider them to be a major priority when studying for technical interviews.
A stack is basically a stack. Imagine you're stacking pancakes. The last pancake you put on the stack will be the first one you grab to eat. It's not much more difficult than that.
So for example you might have a queue to manage something like a drive thru where a car rolls up to the drive thru and gets put at the end of the queue.

Queue example

```
drive_thru = ['red car', 'blue car', 'yellow car']
// green car pulls up to drive thru
drive_thru.add('green car')
drive_thru.poll(); // removes red car from front
print(drive_thru)     // [‘blue car’, ‘yellow car’, ‘green car’]
```

Stack example

```
pancakes = ['pancake 1', 'pancake 2', 'pancake 3']
// finished cooking pancake 4
pancakes.push('pancake 4')  // adds pancake 4 to the end
pancakes.pop(); //  removes pancake 4 from the end
print(pancakes)  //  [‘pancake 1’, ‘pancake 2’, ‘pancake 3’]
```

This is just pseudocode but hopefully it kind of shows you some scenarios where you can use these data structures. These data structures are simply just for organizing data and they have these cool built in functions like push, pop, add, and poll that efficiently carry out operations on the data.

We might dive into more of the details involved in implementing these methods in the future but for now just try and understand linked lists, arrays, queues, and stacks, and think about when, where, and why you might want to use them.

Queues are often used when performing Breadth First Search. Queues are the perfect structure to use for BFS because as we traverse we can add the next level of nodes

Medium Level Problems

Solve one of the following array challenges. (Solve both recommended)
Question 4 - https://leetcode.com/problems/task-scheduler/
(Problem : Task Scheduler)

WEEK 4 - TREES

Before we move onto trees we should do a few more problems with 1d and 2d arrays.

The first thing we should learn about trees is the basic traversal patterns. Please check out this explanation (https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/) on GeeksForGeeks and then proceed to solve the problems below.

Algorithm Challenges for Beginners - Trees

Question 1 - https://www.hackerrank.com/challenges/tree-preorder-traversal/problem
(Problem : Tree: Preorder Traversal)
Question 2 - https://www.hackerrank.com/challenges/tree-postorder-traversal/problem
(Problem : Tree: Inorder Traversal)
Question 3 - https://www.hackerrank.com/challenges/tree-postorder-traversal/problem
(Problem : Tree: Postorder Traversal)

Understanding these traversal patterns will help deepen your understanding of trees. Another type of traversal that you'll want to work through can be found in the problem below.

The general recursive pattern for traversing a (non-empty) binary tree is this: At node N do the following:

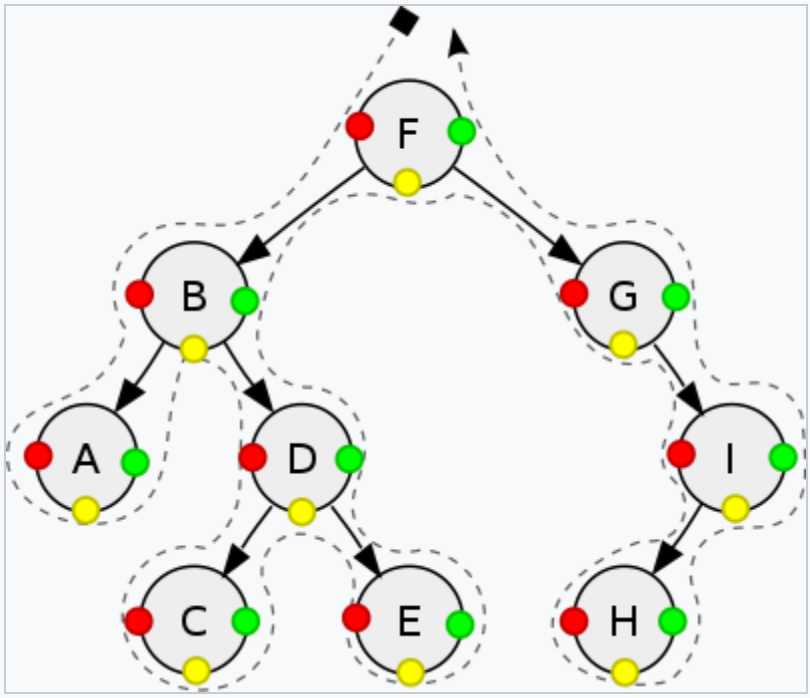
(L) Recursively traverse its left subtree. This step is finished at the node N again.

(R) Recursively traverse its right subtree. This step is finished at the node N again.

(N) Process N itself.

These steps can be done *in any order*. If (L) is done before (R), the process is called left-to-right traversal, otherwise it is called right-to-left traversal. The following methods show left-to-right traversal:

Pre-order (NLR)[edit]



Depth-first traversal of an example tree: *pre-order (red):* F, B, A, D, C, E, G, I, H; *in-order (yellow):* A, B, C, D, E, F, G, H, I; *post-order (green):* A, C, E, D, B, H, I, G, F.

1. Check if the current node is empty or null.
2. Display the data part of the root (or current node).
3. Traverse the left subtree by recursively calling the pre-order function.
4. Traverse the right subtree by recursively calling the pre-order function.

The pre-order traversal is a topologically sorted one, because a parent node is processed before any of its child nodes is done.

In-order (LNR)[edit]

1. Check if the current node is empty or null.
2. Traverse the left subtree by recursively calling the in-order function.
3. Display the data part of the root (or current node).
4. Traverse the right subtree by recursively calling the in-order function.

In a binary search tree, in-order traversal retrieves data in sorted order.[4]

Post-order (LRN)[edit]

1. Check if the current node is empty or null.
2. Traverse the left subtree by recursively calling the post-order function.
3. Traverse the right subtree by recursively calling the post-order function.
4. Display the data part of the root (or current node).

The height of a binary tree also comes up a lot in interviews so understanding what that means and how to find it will come in handy. Try the problem below!

Question 4 - https://www.hackerrank.com/challenges/tree-level-order-traversal/problem
(Problem : Tree: Level Order Traversal)

Tree problems can often times be solved using breadth first search and depth first search. Most of the time you'll be able to use either BFS or DFS to solve the problem and it won't matter which one you choose, but sometimes it will be obvious which one should be implemented. If there is no major difference in performance then it's best to choose the one that is easiest to implement/code or the one you're most comfortable with.

Question 5 - https://leetcode.com/problems/minimum-depth-of-binary-tree/
(Problem : Minimum Depth of BinaryTree)

Make sure you remember there is a difference between N-ary and Binary Trees. That can trip you up for a few moments if you forget during an interview!!!

Question 6 - https://leetcode.com/problems/maximum-depth-of-n-ary-tree/
(Problem : Maximum Depth of N-ary Tree)

Extra Challenges - More Involved

Question 7 - https://leetcode.com/problems/balanced-binary-tree/

(Problem : Balanced Binary Tree)

WEEK 5

Topics

- Binary Search
- Breadth First Search (BFS)
- Depth First Search (DFS)
- Dijsktra's Algorithm

Binary Search Variations

Question 1 - https://leetcode.com/problems/search-in-rotated-sorted-array/

(Problem : Search In Rotated Sorted Array)

Question 2 - https://leetcode.com/problems/search-a-2d-matrix/

(Problem : Search In A 2D Matrix)

Question 3 - https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/

(Problem : Find minimum in a rotated sorted array)

Breadth-First Search

https://www.youtube.com/watch?v=s-CYnVz-uh4&list=PLUI4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb&index=13

Depth-First Search

https://www.youtube.com/watch?v=AfSk24UTFS8&list=PLUI4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb&index=14

Congrats! You've made it pretty far! Now is a great time for us to recollect our thoughts and go over a very important topic in our technical interview preparation - searching. Believe it or not, binary search isn't the only searching algorithm we have to know. Algorithms like Breadth-First-Search, Depth-First-Search, and Dijsktra's algorithm come up all the time. They are extremely common and there's a good chance you'll have to use them during your technical interview. Dijsktra's algorithm specifically doesn't come up too much, but it's definitely good to know!

"So how can we identify when to use searching?"

"Which searching algorithm should we use?"

Binary Search problems are pretty easy to identify. There are a few big giveaways that you should use binary search. The first give away is when you're asked to implement your solution in ***O(LogN)*** time. Another big give away is when you're searching within a ***sorted*** data set. Binary search has a time complexity of O(LogN) and can only be performed on a sorted input.

https://leetcode.com/problems/search-insert-position/

Just because you've identified one of these clues in the problem **DOES NOT** mean that binary search is guaranteed to be the answer. If you've identified one of these clues in the problem, it just means that you should consider binary search as a possible solution.

We've explained and explored Binary Search, but what about Breadth First Search and Depth First Search? First let's checkout this great explanation video made by Gayle Laakmann McDowell https://www.youtube.com/watch?v=zaBhtODEL0w&t=1s

There are also problem clues that might point towards Breadth First Search or Depth First Search as potential problem solutions. Whenever you can think of the problem in terms of a graph where you have a starting node and an ending node, it might be a good idea to consider BFS and DFS as viable solutions.

Breadth First Search problems

Question 1 - https://leetcode.com/problems/word-ladder/

(Problem : Word Ladder)

Question 2 - https://leetcode.com/problems/keys-and-rooms/

(Problem : Keys and Rooms *this problem has easily understandable implementations in both BFS and DFS so try learning both implementations*)

Breadth First Search can often be used as a method of solving tree problems.

Question 5 - https://leetcode.com/problems/binary-tree-level-order-traversal/

(Problem : Binary Tree Level Order Traversal)
Question 6 - https://leetcode.com/problems/binary-tree-level-order-traversal-ii/

(Problem : Binary Tree Level Order Traversal II)

During a breadth first search it's very common to use a data structure such as a "queue" to evaluate nodes level by level.

Depth First Search problems

Depth First Search is usually implemented using recursion. It's very common to end up making an extra method / function to handle the recursion. We call these methods "helper methods".

WEEK 6 - Introducing Topics (Greedy, Divide & Conquer, and Dynamic Programming)

Now might be a good time to introduce some important methods for algorithm problem solving.

Greedy Algorithms - Somewhat common in technical interviews

Divide & Conquer - Not very common in technical interviews.

Dynamic Programming - - Not very common in technical interviews.

Some companies have chosen to ban dynamic programming problems from technical interviews because they are often considered some of the most difficult problems to solve. Understanding how to identify these problems is still important just in case they

ever come up. Some of them aren't too difficult and I would consider them appropriate for technical interviews.

Here are some higher level explanation videos on dynamic programming
Gayle Laakmann McDowell - https://www.youtube.com/watch?v=P8Xa2BitN3I
CS Dojo - https://www.youtube.com/watch?v=vYquumk4nWw&t=762s

After watching both of the videos above you should have some kind of idea about what dynamic programming is and why we use it. Now we can look at some more involved lecture videos by Erik Demaine at MIT.

This video is a great for Dynamic Programming -
https://www.youtube.com/watch?v=OQ5jsbhAv_M

As a follow up, I recommend watching the next 2 videos on dynamic programming as well.

Dynamic Programming II - https://www.youtube.com/watch?v=ENyox7kNKeY

Dynamic Programming III - https://www.youtube.com/watch?v=ocZMDMZwhCY

Question 1 - https://leetcode.com/problems/climbing-stairs/
(Problem : Climbing Stairs)

Notes - This is a great problem to started because the solution is directly related to the videos you just watched. Try and figure this problem out and reference the previous videos made by Gayle and CS Dojo if you end up stuck. If all else fails, reference the solution and try to understand it and how it relates to the Fibonacci example from the videos.

Question 2 - https://leetcode.com/problems/house-robber/
(Problem : House Robber)

Notes - Also very similar to the videos + problem above

Question 3 - https://leetcode.com/problems/maximum-subarray/

(Problem : Maximum Subarray)

One pattern that we can pick up on to help us identify a problem as a “dynamic programming” problem is decision making / steps. When we are asked to make decisions at each step of the problem, it is good to consider dynamic programming as a possible solution.

Ask yourself

Can I form a decision tree from this problem?
Are there repetitions within the decision tree that we can eliminate using memoization?

If so, then dynamic programming will often times end up being an optimal solution. One thing that might help organizing your thoughts is approaching the problem recursively.

While on the subject, another problem type that deals with decision making is “backtracking” which we’ll cover next. The difference between these problem types is clear because in dynamic programming we end up using variables or a data structure like an array to save our completed work and using that saved data to make the optimal next decision (to avoid repetition), but in backtracking we usually use recursion and end up making every possible decision and choosing the best one.

Another thing we might notice just from working through a bunch of dynamic programming problems is that we usually use an array to memoize. It’s usually an array with a fixed size of N+1 (N = number of elements in input) although this isn’t always the case. We might also notice that we usually add initial data to our dp array and start our iterations at i=1 rather than the beginning of the array. Once again this isn’t always the case, but it’s good to have an idea of a solution template in mind.

WEEK 7 - Backtracking

Generate Parentheses is probably one of the easiest examples I've seen when it comes to implementing a backtracking solution. I'd highly recommend going through and really trying to solve and understand the problem and why we use backtracking.
https://leetcode.com/problems/generate-parentheses/

Question 1 - https://leetcode.com/problems/subsets/

(Problem : Subsets)
Question 2 - https://leetcode.com/problems/subsets-ii/

(Problem : Subsets II)
Question 3 - https://leetcode.com/problems/permutations/

(Problem : Permutations)
Question 4 - https://leetcode.com/problems/permutations-ii/

(Problem : Permutations II)
Question 5 - https://leetcode.com/problems/combination-sum/

(Problem : Combination Sum)
Question 6 - https://leetcode.com/problems/combination-sum-ii/

(Problem : Combination Sum II)
Question 7 - https://leetcode.com/problems/palindrome-partitioning/

(Problem : Palindrome Partitioning)

This post in the leetcode discussion section is really awesome at explaining a general approach to backtracking problems.
https://leetcode.com/problems/subsets/discuss/27281/A-general-approach-to-backtracking-questions-in-Java-(Subsets-Permutations-Combination-Sum-Palindrome-Partitioning)

WEEK 8 - Heaps, Tries, etc

https://leetcode.com/problems/kth-largest-element-in-an-array/

WEEK 9 - Strings

You might be wondering why string problems are at the end of the study guide. String problems aren't necessarily a specific type of problems because there are so many of them. String problems can be broken down further into more specific types / categories of string problems.

When given string problems we are often asked to deal with mutation, sorting, counting, substrings, etc. Now that we have a bunch of our problem solving techniques at our disposal, we use some of these strategies and tools to come up with a solution.

One of the most common situations that will come up in string problems, is that we'll want to know the number of times each character occurs in a string. This happens all of the time and instead of using a HashMap<Character> we can use a simple int[] arr where the index of the array corresponds to the position of each character in the alphabet and the value corresponds to the number of times that specific character occurs in the string.

For example, in the string "apple" we might build an int[] arr like the one below

int[] arr char_counts = new int[26] { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

You'll notice that the value of position 0 in our array (first position in array) ('a' because 'a' is the first position in the alphabet) has a value of 1 because there is 1 occurrence of a in the string "apple".

You'll also notice that our int[] array is size 26 because there are 26 letters in the alphabet. So once again, we have an integer for each character position in the alphabet that represents the number of times that character occurs in a given string.

The int[] array is nice because once it is built we are able to access the number of times a specific character occurs based on its index in the alphabet. To get the number of times the letter 'b' occurs we can access that value by referencing char_counts[1].

This might seem like a simple concept but it becomes so useful when dealing with string problems. The int[] array is constant space since it's only size 26 and we are able to access the indices easily when dealing with characters because we can make those simple conversions from char to int.

Question 1 - https://leetcode.com/problems/is-subsequence/

(Problem: Is Subsequence)

Question 2 - https://leetcode.com/problems/repeated-substring-pattern/

(Problem: Repeated Substring Pattern)

WEEK 10 - More Advanced Problems

Sliding Window Problems

Question 1 -

https://leetcode.com/problems/longest-continuous-increasing-subsequence/

(Problem: Longest Continuous Increasing Subsequence)

Question 2 - https://leetcode.com/problems/minimum-size-subarray-sum/

(Problem: Minimum Size Subarray Sum)

Question 3 - https://leetcode.com/problems/longest-repeating-character-replacement/

(Problem: Longest Repeating Character Replacement)

Question 4 - https://leetcode.com/problems/permutation-in-string/

(Problem: Permutation in String)

Sometimes the best way to solve an array problem is using a sliding window approach. This approach is commonly used when solving problems that deal with substrings,

subsets, subarrays, etc. and involves keeping track of a beginning and ending boundary within the data you're traversing.

Sliding Window
https://leetcode.com/problems/find-all-anagrams-in-a-string/discuss/92007/Sliding-Window-algorithm-template-to-solve-all-the-Leetcode-substring-search-problem.

Arrays

Question 1 - https://leetcode.com/problems/product-of-array-except-self/
Notes - This problem is a little bit more advanced, but not impossible. It's an extremely popular problem that is given out all of the time in interviews! (or a variation)

Question 2 - https://leetcode.com/problems/set-matrix-zeroes/
(Problem: Set Matrix Zeroes)

Question 5 - https://leetcode.com/problems/first-missing-positive/
(Problem: First Missing Positive)

WEEKS 11 & 12 - SYSTEMS DESIGN

Depending on what position you're interviewing for, systems design may be a section during your technical interview. Entry level software engineers shouldn't have to worry about systems design too much but should understand basic concepts like client-server architectures, load balancing, etc. Mid-level and senior engineers will be expected to know how to design scalable systems. In my opinion, one of the best guides to systems design is https://github.com/donnemartin/system-design-primer

The best strategy for learning systems design in my opinion is to go through that guide while simultaneously scheduling Pramp interviews.

Pramp is a platform that matches you with a random person who is also studying for their technical interviews. It's a great place to meet talented young professionals and it makes it really easy to learn fast.