Hello! I am a C++ Programmer and I am posting my approach towards the internship interview preparation which might help students looking for a plan to start their Data Structures and Algorithms preparation.

**The approach:**

* I think the most important step is to understand the intricacies of the language being chosen. For instance, as a C++ programmer, one should be well-versed with pointers, the standard template library, the understanding of the internal implementation and operational complexities of the common containers like set, map, vector, etc.
* There are a lot of questions in leetcode and it seems intimidating at first as to how to choose the problems and what approach should be taken. Honestly, sometimes it takes a lot of time to solve a problem, and without a basic understanding and practice of the concepts used, it becomes increasingly difficult to solve questions which sometimes result in a loss of confidence. I recommend understanding the general patterns of the questions along with practicing the prominent questions belonging to those patterns before trying to solve similar questions on your own if you are a beginner. I will be talking about a few patterns that I have observed and studied along with the prominent questions following those patterns which might help solidify the concepts. These prominent questions are frequently asked in Companies like Meta, Amazon, Linkedin, Adobe, Citrix, Okta, MathWorks, etc. Understanding the approach to those questions with the help of the discussion forums and then solving similar questions on your own might be a better way.

**The patterns and techniques:**

[14 Patterns to Ace Any Coding Interview Question](https://hackernoon.com/14-patterns-to-ace-any-coding-interview-question-c5bb3357f6ed) talks about the important coding patterns but I am adding some related questions, identification, and techniques to better approach these problems.

* **Sliding window**: One of the most important patterns that might be useful in solving the string and array problems. A combination of sliding window and an unordered set is usually very powerful. The left and right pointers constitute the boundary of the window and we slide them after processing the current window. The new window usually contains most of the elements from the previous window, so instead of recalculating the new result, we process the result of the new window using the previous result, the items removed, and the items added. I recommend solving [Longest Substring Without Repeating Characters](https://leetcode.com/problems/longest-substring-without-repeating-characters/) to start with. Once you get the hang of these questions, I recommend solving [Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/). If you can understand the solution to this problem, I think you should be able to solve most of the sliding window questions.
* **Two pointers**: In this approach, we try to iterate the array or string from two different positions to find the solution. Everyone might have used this approach to solve the palindrome problem in which case we increment and decrement the left and right pointers respectively after processing. There are a few questions in which we increment or decrement only one pointer based on the condition to arrive at the solution. The prominent questions from this category are [3Sum](https://leetcode.com/problems/3sum/), and [Container With Most Water](https://leetcode.com/problems/container-with-most-water/). The two-pointer approach is also useful while solving the problem [Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/) in which case we expand from the middle.
* **Prefix sum**: This is a powerful technique for questions regarding integer subarrays. It is useful for range queries and its combination with unordered\_map is usually very helpful. You should try solving the question [Subarray Sum Equals K](https://leetcode.com/problems/subarray-sum-equals-k/) to get the hang of questions involving prefix sum. Another challenging question which I love from this pattern is [Count subarrays with more ones than zeros](https://leetcode.com/problems/count-subarrays-with-more-ones-than-zeros/).
* **Merge intervals**: It is a popular category of questions where we usually sort all the intervals first and then merge them based on the constraints of the questions. Prominent questions from this category are [Merge Intervals](https://leetcode.com/problems/merge-intervals/), and [Interval List Intersections](https://leetcode.com/problems/interval-list-intersections/).
* **Modified Binary Search**: Whenever we see a sorted array or a question where we need not visit every element and can discard ranges, binary search should come into mind. While it seems like an easy concept, there are some amazing questions, which although not obvious, can be solved using Binary Search. Prominent questions from this category are [Find First and Last Position of Element in Sorted Array](https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/), [Find Peak Element](https://leetcode.com/problems/find-peak-element/), and [Capacity To Ship Packages Within D Days](https://leetcode.com/problems/capacity-to-ship-packages-within-d-days/).
* **Monotonic**: It is not a very well known concept but seemingly O(n^2) range queries questions can be solved in O(n) using monotonic stack. The concept is simple, the monotonic stack contains elements either in increasing or decreasing order. Any time this property is not followed, we do some operations depending on the question, in order to maintain this property. Prominent questions from this category are [Next Greater Element II](https://leetcode.com/problems/next-greater-element-ii/), and [Daily Temperatures](https://leetcode.com/problems/daily-temperatures/).
* **Recursion**: One of the most important and useful concepts which in different ways is used to solve many coding questions. It is used in most of the tree questions, questions like permutations, subset generation, questions where all the solutions need to be generated/ printed. Aditya Verma has a playlist on recursion which I believe is a nice place to start with. I will be talking about some of the famous techniques to approach recursion problems. I should mention that all the below techniques follow a simple principle of trying to generate all possible solutions and check if they satisfy the constraints of the problem.
  + **Backtracking**: It is an incremental approach where we try to arrive at the solution bit by bit. At any point, if we believe that the current set of combinations cannot possibly arrive at the solution, we take a step back (discard it) and try again. The general pattern is to have a base condition and then run a loop to try and generate possible combinations and recur inside the loop in a depth-first manner. Prominent questions which can be solved using backtracking are [Combination Sum](https://leetcode.com/problems/combination-sum/), [Letter Combinations of a Phone Number](https://leetcode.com/problems/letter-combinations-of-a-phone-number/), and [Permutations](https://leetcode.com/problems/permutations/).
  + **Input/Output**: For this technique, understanding the recursion tree is important. Every recursion problem has an input and one or more outputs which becomes the input for the next recursion call. Thinking questions from a perspective of the recursion tree and generating inputs and outputs accordingly might be helpful. Aditya Verma has beautifully talked about this pattern and its related questions. Prominent questions which can be solved using this technique are [Subsets](https://leetcode.com/problems/subsets/), and [Generate Parentheses](https://leetcode.com/problems/generate-parentheses/).
  + **DFS**: When we think of DFS, we generally think of graphs. Questions involving 2D matrices where we have to traverse in all the directions to arrive at a solution also constitutes one of the important categories of questions solved by the DFS approach. In this approach, we first check the validity of the current cell, process it if it's valid and recurse in the other directions. I recommend understanding the approach of [Number of Islands](https://leetcode.com/problems/number-of-islands/), [Longest Increasing Path in a Matrix](https://leetcode.com/problems/longest-increasing-path-in-a-matrix/), [Word Search](https://leetcode.com/problems/word-search/), and [Pacific Atlantic Water Flow](https://leetcode.com/problems/pacific-atlantic-water-flow/) to get the hang of DFS questions that are usually asked in the interviews.
* **Stack**: Whenever we see a question regarding parathesis and evaluation of expressions, using a stack is usually the way to go. [Minimum Remove to Make Valid Parentheses](https://leetcode.com/problems/minimum-remove-to-make-valid-parentheses/) is a nice question to start in this domain. Other good questions that can be solved using stack are [Basic Calculator](https://leetcode.com/problems/basic-calculator/) and [Basic Calculator ||](https://leetcode.com/problems/basic-calculator-ii/).
* **Heaps**: Anytime we see a question involving the kth element (closest, smallest, largest), we should think about using Priority Queues or the QuickSelect algorithm. The regular solution involving sorting is naive and using heap is more efficient in these questions. While the heap solution is accepted in most of the interviews, QuickSelect is the most optimal solution. We should try to be well versed with this solution, which follows the ideology of the QuickSort algorithm. TECH DOSE is an awesome YouTube channel that has explained the QuickSelect algorithm well. We should also try to understand why the average case time complexity of QuickSelect is O(n) where n is the input size. Prominent questions which can be solved using QuickSelect are [K Closest Points to Origin](https://leetcode.com/problems/k-closest-points-to-origin/solution/) and [Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array/).  
  Prominent questions solved using heaps are [Meeting Rooms ||](https://leetcode.com/problems/meeting-rooms-ii/), [Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists/), and [Find Median from Data Stream](https://leetcode.com/problems/find-median-from-data-stream/) which uses two heaps. Using heaps is a powerful technique and we should try to understand how we can solve the above questions using heaps and at the same time identify the pattern. One question that I like which can be solved using heaps is [Swim in Rising Water](https://leetcode.com/problems/swim-in-rising-water/). Solving this surely improves confidence and clears the concepts. We should also be well versed with using comparators with heaps. This is very important from an implementation perspective as depending on the question we might push pair or objects into the heap.
* **Linked list**: I think questions about linked lists are more implementation based than logic-based. You should try to be proficient with altering links, deleting nodes and traversing linked lists. It is important to understand the internal implementation and what is happening from the memory point of view. For C++ programmers, mycodeschool is an amazing YouTube channel that explains the pointers along with the Data Structures pretty well. Fast and slow pointer technique (where the slow pointer say goes to the next node and the fast pointer say goes to the next's next node) is usually useful to detect merge points in two linked lists and to detect cycle in a linked list. I think if you can understand and implement [Reverse Nodes in k-Group](https://leetcode.com/problems/reverse-nodes-in-k-group/) and [Copy List with Random Pointer](https://leetcode.com/problems/copy-list-with-random-pointer/) without extra space, your linked list is solid.
* **Trees**: To start trees, do some basic tree questions including traversal (pre, in, post, level) to get the hang of it. [Binary Trees - Stanford CS Education Library](http://cslibrary.stanford.edu/110/BinaryTrees.html) has a list of 15 basic binary tree questions with Java and C++ solutions which according to me serves as a nice place to start. I am not listing prominent tree questions as I think we should strive to do as many binary tree questions as possible. The level order traversal using queues can be a baseline to solve many questions like [Binary Tree Right Side View](https://leetcode.com/problems/binary-tree-right-side-view/) and [Populating Next Right Pointers in Each Node](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/).  
  I watched a few of the Aditya Verma videos on Trees and one of the things that I have learned from his videos is to solve the tree recursion problems in three steps. This I think is a very useful way of solving questions where you do not have any idea as to how to start the question and solve it as a whole. The first step is the base condition where you think about the base cases, things like what would happen if the tree is empty etc. The second step is the hypothesis where you usually write two recursive statements to get the answer from the left and the right subtrees. You may assume that it is returning the correct answer from both the subtrees. You may also assume it like a Black box which will return us the correct answer from both the subtrees. Now, the final step is induction. Here, you think that you are at a current node and you have the answers from its left and right subtrees. Now, the question is reduced to just a single node where given you have the answers from left and right subtrees, how will you use those to return the answer from that particular node? Thinking questions from this perspective and getting the hang of it by understanding questions like [Diameter of Binary Tree](https://leetcode.com/problems/diameter-of-binary-tree/) and then approaching new questions with this technique might be helpful.
* **Graphs**: From my interview experience, I have observed that questions regarding shortest paths, Minimum Spanning Tree, and Strongly Connected Components are rarely asked. Hence, we should just be theoretically well aware of algorithms like Dijkstra's, Bellman-Ford, Floyd Warshall, Prim's, Kruskal's and concepts like relaxation. I believe Abdul Bari sir has a good playlist on YouTube regarding the same. We should concentrate more on DFS, BFS, cycle detection, and topological sort (where to calculate a particular answer, we need to first calculate the answer of its components using linear ordering). Prominent questions from this category are [Clone Graph](https://leetcode.com/problems/clone-graph/), [Alien Dictionary](https://leetcode.com/problems/alien-dictionary/), [Accounts Merge](https://leetcode.com/problems/accounts-merge/), and [Word Ladder](https://leetcode.com/problems/word-ladder/) which can be solved using BFS.
* **Trie**: Trie is a useful data structure in solving questions that involve word autocomplete and suggestions. I believe one should be well versed with its structure and operations like insert and search. Delete operation in the trie is not frequently asked. The prominent question from this category is [Word Search II](https://leetcode.com/problems/word-search-ii/).
* **Dynamic Programming**: It is usually considered one of the most intimidating topics. I think Aditya Verma's playlist on Dynamic Programming has somewhat made life easier. What I loved about the 50 videos playlist was that instead of explaining how so many different problems can be approached, he has bifurcated most of the prominent DP problems into three parent categories of 0-1 Knapsack, Longest Common Subsequence, and Matrix Chain Multiplication. Besides that, he starts every problem with the intuitive approach of recursion followed by Memoisation(top-down DP) and finally Tabulation(bottom-up DP). His playlist surely makes DP less intimidating and improves the understanding of DP in general. I recommend watching all of the 50 videos. There are many prominent DP questions which I am sure most of us have heard but two of my favourite questions which can be solved using Memoisation are [Word Break](https://leetcode.com/problems/word-break/), and [Longest String Chain](https://leetcode.com/problems/longest-string-chain/).

**Important notes:**

* Identification of the problems comes with practice. If you start following this article and then solve similar questions on your own, I believe that you will have a planned approach to most of the new problems you would see.
* Arrays and Strings are more important than Dynamic Programming for most companies. Companies like Meta do not even ask Dynamic Programming. Hence, I recommend targetting Arrays and Strings properly before moving towards Dynamic Programming.
* There are a lot of topics like Segment trees, AVL trees, Fenwick trees, and Disjoint sets which I have not covered as they are somewhat advanced and are rarely asked in the internship interviews.
* There are a lot of other techniques like Greedy and Bit Manipulation which I have not covered. Greedy is mostly intuitive and questions requiring Bit Manipulation are rarely asked. I recommend going through the chapter on Bit Manipulation from Cracking the Coding Interview book if you are interested.
* Sometimes when you have to calculate and manipulate an array in place, remember that you can traverse the array or string backwards as well. I recommend solving [Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/) to better understand what I am trying to say here.
* According to me TECH DOSE is a good YouTube channel which you can refer to understand most of the questions I have linked above. Python Programmers can refer to the channel NeetCode.
* As the next step, you should start solving the Top frequently asked company-specific leetcode questions of Apple, Microsoft, Amazon, Facebook, and Google in this particular order. Many questions would be common. I am sure once you have solved these questions, you will feel supremely confident going into the interviews.

These techniques are just like the weapons in your armoury. You will still need to practice how and where to use them. Practice hard, and do not give up. This article only covers DSA but you also need to prepare OS and Networks for companies like Citrix and Cisco. A good understanding of OOP and low-level design is also important. I will write another detailed article on how to answer technical and behavioural questions in an interview. You can contact me using my website [Aman Chopra](https://aman-chopra.github.io/) in case you have any doubts.  
Happy Coding!