Data Structures

Grab a notebook not just a piece of paper and start by listing down all the data structures that has been discovered until now.

Arrays [1-D, 2-D]

Linked List [Singly, Doubly, Circular]

Stacks

Queues

Trees [Binary Trees, N-ary Trees]

Priority Queues

Unions/Disjoints ADT

Graphs

Strings

Trie

Hash maps

Start exploring these data structures by reading about them through a book or understanding them with the help of a mentor or from youtube. Get a basic working idea about all the data structures listed above. I know you are with me up until now.

Commit to Memory (CTM)

I often used this abbreviation whenever I wanted to make sure my brain has to remember something for a longer period of time. We will be using this term quite often.

Time & Space Complexity Cheat Sheet

Prepare a table and jot down the time complexity for a best, average and worst case scenarios for every data structure. Make sure you are not cheating for the cheat sheet itself and you are preparing this on your own. For initial reference you can visit https://www.bigocheatsheet.com/

Needless to say CTM the cheat sheet. This is going to help you a lot whenever you are solving any question. You can prepare this sheet multiple number of times by yourself if you have a tendency to forget things quite often. Paste this nearby on your desk for quick reference.

Identifying the Algorithms for Data Structures

Assuming you are clear with the basic data structures. Now is the time list down all the major algorithms that comes under them. Prepare an Hierarchal tree list for all the Data Structures.

Note that the questions are unlimited but the algorithms are limited.

1.Arrays

a. Sorting (You can sub-categorize into merge sort, quick sort etc.)

b. Two pointer

c. Sliding Window

d. Binary Search

e. Recursion

f. Dynamic Programming (LIS, LCS, LPS etc)

g. Ad Hoc

h. Bit Manipulation

i. Hashing

j. Insert/Update/Delete an element

k. Merge Intervals

2.Linked List

a. Sorting

b. Slow and Fast Pointer

c. Recursion

d. Hashing

e. Insert/Update/Delete an element

f. Josephus Problem

3.Stacks

a. Sorting

b. Recursion

c. Hashing

d. Dynamic Programming

e. Ad Hoc

f. Insert/Update/Delete an element

4.Queues

a. Sorting

b. Recursion

c. Hashing

d. Ad Hoc

e. Insert/Update/Delete an element

5.Trees

a. Recursive and Iterative traversals (You can sub-categorize into pre order, post order, level order and in order traversals)

b. Maximum and Minimum in a tree

c. Height of a tree

d. Swapping of nodes, sub-trees

e. Diameter of a tree

f. Binary Search Trees

g. Inorder predecessor in Binary Search Trees

h. Segment Trees/Fenwick Trees (not commonly asked but good to have knowledge about it)

i. Hashing

f. Insert/Update/Delete an element

g. Balanced Binary Tree/ Search Trees

6.Priority Queues

a. Binary heaps

b. Insert/Update/Delete an element

c. Heapify/Percolate Down

d. Get Minimum/Maximum in a heap

e. Heapsort

f. Median in an array

g. Sorting a list of large numbers/files

h. Updating a priority queue using hashmap

7.Unions/Disjoints ADT

a. Quick Union

b. slow/quick FIND

c. UNION by Height (UNION by Rank)

d. UNION by size

e. Makeset

f. Path Compression (Not really required but good to have knowledge)

8.Graphs

a. Traversals (You can sub-categorize into DFS and BFS — Always use Adjacency list)

b. Topological Sorting (DFS and BFS)

c. Number of Connected Components

d. Find if a Path exists

e. Dijkstra’s Algorithm

f. Flood fill Algorithm

9.Strings

a. Sorting

b. Recursion

c. Hashing

d. AdHoc

e. Insert/Update/Delete a character

f. KMP Search (Not required but good to have knowledge)

g. Two pointer

h. Sliding Window

i. Dynamic Programming (LIS, LCS, LPS etc)

10.Trie

a. Insert/Update/Delete a node

b. Searching a node

c. Suffix/Prefix Trees

11.Hash maps

Hashing is used in almost all the data structures and they don't have generic named approaches. We can get adapted to hashing and its uses with the help of numerous ad hoc questions.

Ex — Find the first non-repeating character in a string of length N.

Do read about difference between a hash map and a hash table and how collisions happen in them and most importantly. How do we resolve them. A bit knowledge about Linear and Quadratic probing would be great.

Please spare me if I missed any important algorithm and do tell me in the comments, I will add them into the list. You can add your own named algorithms into your own list which you think are getting used more frequently. Find below the list of some advanced not asked much (or at least I consider them advanced) algorithms and read them only if you have conquered the above list and want to spoil your work life balance. Just kidding!!

Prims/Kruskal's Algorithm

Ternary Search Trees

Fibonacci Heaps

Advanced Game Theory

AVL Trees, Red-Black Trees

(Many more...)

The Bucketing Approach

This is my self-devised approach which helped me tons for the preparation. I hope this does wonders for you as well. Take three buckets and name them easy, intermediate/medium, and hard.

Now read your detailed algorithm list from top to bottom and starting putting down the algorithms from each data structures you feel you know the best and you can crack them anytime they are asked into the easy bucket. Put down all the algorithms from each data structures into the intermediate bucket that you feel you are just okay with them and sometimes you are able to solve the problem and sometimes you have to see the discuss section of the problem to get the hint. And lastly put in all the algorithms you feel you don't even want to touch but always wish that they will be in your easy bucket one day. Obviously, you have to exhaust your whole list. Each algorithm should be present in any one bucket.

Ex —

Easy — [Arrays → Sorting, Two pointer, Sliding Window, Adhoc, Hashing, etc],[Linked List → Sorting, Slow and Fast Pointer …] ….

Intermediate — [Arrays → Dynamic Programming, Merge Intervals and Bit Manipulation], [Linked List → Josephus Problem (of course it's easy I am just giving an example)]…

Hard — [Graphs → Dijkstra's Algorithm] …

The Art of Solving only Quality Questions

Once you are ready with your buckets, You have to understand that at the end of your preparation you have to fetch the items in your intermediate bucket and pour them into easy bucket and fetch the items of your hard bucket into your intermediate bucket.

AIM — We must make our everyday productive without losing motivation to solve the problems. CTM

Chose any coding platform you like I personally prefer Leetcode. Assuming you are at least devoting 2 hours every-day into your preparation and you have to solve easy questions in 5–7 mins and medium question in 20–25 mins and hard question in 30–35 mins. At average you can solve 10 questions in one day. How to choose these 10 questions?

Refer to your buckets and pick out any 2 algorithms from easy and 6 algorithms from intermediate and 2 algorithms from hard bucket. Yes, you can choose randomly between the buckets.

After this search questions on leetcode with the help of algorithm and data structures tag and filter them with difficulty and sort them with the acceptance rate.

Difficulty of the question is inversely proportional to name of the bucket.

If you have good hang of the algorithm i.e., from easy bucket then choose questions with harder difficulty and vice-a-versa.

Pick out the questions which has an acceptance rate between 40–60% for each bucket. Solve these questions by timing yourself according to the difficulty level. Solve as if you are actually giving the interview. Narrate your approach to yourself, write down the test cases you used and also mention the time complexity. Do give a pat on your back once you solve the question completely. CTM

Increase the frequency of the solved questions in the list that you prepared above. For medium questions, take some time and understand the questions, write down in one liners what is given and try to remember what algorithmic question you are solving because only that approach is mostly going to give you the solution. Refrain yourself from opening the discuss section. But if you are not able to think of an approach in 3–5 mins. Refer to the discuss section and say 'Ahh! this is what was needed! Shit I was so close!' CTM

For hard questions, Follow the same approach as that of medium just that you can use about 10 mins to come up with an approach because they are usually dedicated to a specific algorithm (ex — Trie, DP) or they are made up of at most 2–3 medium algorithms. Do not have the mindset of giving up while solving the question. Use the time wisely and use the Discuss section freely but do solve it after you have understood the approach. Don't pass the question. CTM

Hard questions are meant to give you an hint of the limit of difficulty that can be expected in worst case. Try to choose medium-hard questions at first mostly so as to build up the confidence.

Follow OOPs

The only con of leetcode i feel is they have a template class ready for every solution that you are going to provide. It is recommended to follow OOPs approach. Define data members in private, use good naming conventions for variables and functions. Try to keep the code clean with the help of proper structured functions. Always present your solutions in terms of classes. Create classes for LinkedListNode, GraphNode, TreeNode etc. It is really better to develop this habit from the very start as people generally tend to forget these things in the interview.

That's it within 2 months you will be ready to outshine others who have not followed this approach.

How will I know that I'm ready?

As easy as it sounds, It's difficult to answer. It is completely dependent on the fact that how much honest have you been with your preparation. You should have solved at least 3–5 problems minimum for each algorithm in every data structure. Were you able to push most of your items from your Intermediate bucket to easy bucket and from hard bucket to intermediate bucket?

Best way to experiment this is to ask your friend to pick any question from leetcode from medium difficulty and check whether you were able to solve it. The old trial and error way.

The key is to devise the intuition behind the algorithm and that comes with experience and practicing. If you have seen enough of the questions, you are ready to solve them at any point in time. Because sometimes problems require only a specific algorithm to solve and we cannot solve them until we have seen the approach somewhere. This bucketing approach helped me a lot as it identified the topics I feel I am not strong at and made me work specifically on them smartly. CTM

You can refer to tons of curated SDE sheets from a lot of amazing youtubers out there in the end but let me know in the comments if you want me to share a list of questions that I feel important.

'Data Structures and Algorithms made Easy'

This is a gods book. I cannot put in words how important it is to read this book. This book has the appropriate algorithms that are often asked in the interviews. It has got crystal clear explanation and it talks to you itself like a mentor. I have been following this book for the last 5 years and gave a thought of creating it as a NFT lol!! Do refer this book it's easily available online, I usually prefer hard copies as we can make notes right onto that point.

Tip: Watch out videos on improving the problem solving and improving imagination on a daily basis.

Last but not the least

Let me know if you were expecting my insights on a specific topic under DSA preparation. I will add that too in this article. Suggest me some topics on which you want me to cover in my next article and do not forget to follow me on medium as well as on Instagram techmunks. I will be starting to post content on this page very soon, so as to give more crisp unique tips that can help you to standout during your interviews. It's always about the little steps we take in life.