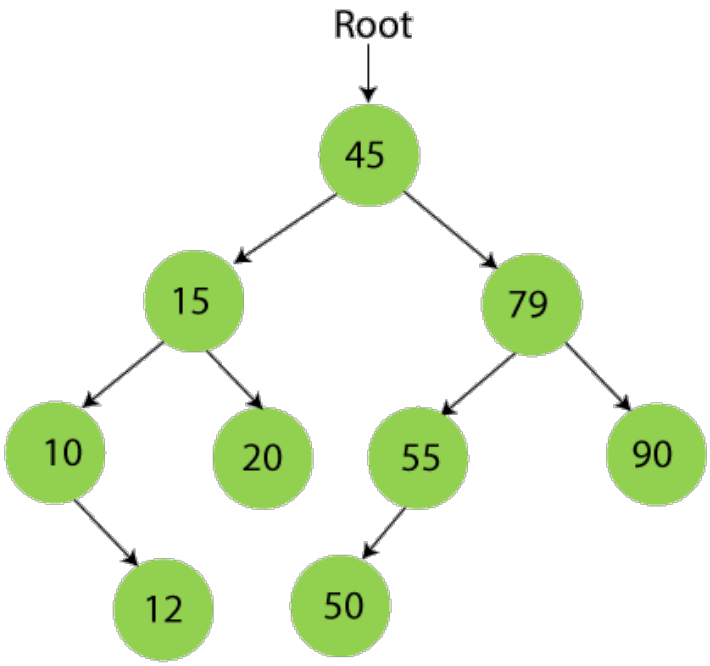
Data Structures and Algorithms Quickbook



Pre-requisite

(i) Familiarity with College Level DSA and High School Mathematics.
(ii) C++ Basics

This Quick book is more about “Understanding Why?”

Author : Yash Pratap

UNIT 1 : Before You Begin

1. Data Structures and Algorithms (DSA) from Perspective of Academia, Industry and Competitive Programming.

2. DSA, Industry and Psychometrics of Algorithm Test.

UNIT 2 : Fundamentals You Must Not Skip

3. Do it with Hands : Loops, Recursion, Math, Basic Hashing and Big O

4. Pick language and start using keyboard and compiler : C++ and STL

UNIT 3 : Practicing the Basics

5. Linear I : Arrays, Binary Search, Sorting, Sliding Window and Two Pointers

6. Greedy Algorithms, Bit Manipulation and Strings

7. Linear II : Linked List, Stacks and Queues

8. Non Linear I : Binary Tree, Binary Search Tree and Heaps

- Balanced Tree lost relevance in Interview/CP due to Heap : AVL Tree, Red Black, B/B+ Tree

9. Understanding your limitations with Tests.

UNIT 4 : Testing the depth of your understanding

10. Recursion and Dynamic Programming

11. Non Linear II : Graphs

UNIT 5 : Going Advance

12. Non SDE or Non Big Tech SDE : *Career without DSA Test as Primary Criteria*

13. SDE or Staff in Big Tech : *Career with DSA Test as Primary Criteria*

Data Structures and Algorithms from Perspective of Academia, Industry and Competitive Programming.

(i) In Academia: It is more tilted towards "P vs NP".

People are primarily trying to understand:

a. If we can reduce any given algorithm into P/NP.

b. P/NP classification helps us understand whether we can solve the problem by throwing more computing resources at it with increasing value of input.

If we can't solve the problem, whether we can use some kind of "heuristics" to get things done for all practical purposes or not.

(ii) In Industry: It is more tilted towards "debugging the algorithm."

Companies using DSA as a primary criterion wish to understand:

a. If the candidate can actually understand popular algorithms and debug issues in large codebase.

A bigger company means a bigger codebase, people already know what they are doing, and the market is already identified. So, the primary challenge of a bigger company is maintaining big infrastructure and understanding the leakages, not developing new products or solving new problems.

b. If the candidate can properly communicate the issue to other team members or not.

That's why interviewers focus primarily on the "candidate's ability to explain the algorithm" more than the "candidate's ability to solve very complex algorithms (competitive coding) within a given time."

(iii) In Competitive Programming: It is more tilted towards "optimization of algorithms."

a. Optimization of algorithms beyond a point can't be done in the majority of simple data structures and algorithms.

Usually, these optimizations make more sense in advanced topics like Graphs and Dynamic Programming, and most of the problems in academia and industry don't require solving these problems within a given time period on a frequent basis.

If you can solve difficult recursion and loop problems then solving all important Graph and Dynamic Programming problem isn't a big deal, they are basically recursion on steroids.

b. Doing too much competitive coding mindlessly can be counterproductive because you might develop a habit of skipping the basics, and sometimes the problem lies in the basic algorithm implementations themselves.

DSA, Industry and Psychometrics of Algorithm Test.

Engineers generally avoid writing custom algorithms on frequent basis, they prefer using algorithms implementations done by trusted libraries even in Big Tech Jobs. Algorithms aren't necessarily implemented by libraries in conventional ways, they might be modified according to hardware and language. Testing these algorithms isn't easy.

Is DSA good criteria to test competence of an engineer ?

The answer varies according to the scale of company,

1. Large Scale : Engineers are primarily hired to sustain the infrastructure.

They need people who can understand complex algorithms because the payback of optimization is huge. If Google hire 100 engineers via DSA then at any particular instance of time, they will always get 10 competent engineers who can understand and sustain important things. This is basically a good financial strategy to keep a big tech functioning. Acquisition of a company always makes more sense then development of new products. If you like to build libraries, analyze code and design large scale distributed systems then you'll be a good fit.

Role : Individual Contributor Route (Staff/Principle Engineer) .

2. Mid Scale : Engineers are primarily hired to fulfill business requirements.

They need people who can understand business requirements and convert it into a software. Even in Large Scale companies, almost 80% part of the software (internal tools) doesn't need much optimization. These jobs are similar to jobs in Mid Scale Organization so it might be possible that someone with average understanding of DSA could be found in Leadership/Management Roles in Big Tech due to proficiency in their niche.

Role : Engineering Manager Route (Tech Lead/Engineering Manager)

3. Small Scale : Engineers are primarily hired to develop prototypes and iterate till Product Market Fit. People with Product Skills are better fit. Basic understanding of DSA which doesn't become obstacle in learning is always helpful.

Role (Product) : Product Managers Route (Product Manager)

Note : If you are looking for bringing concept to reality then you should pursue academia or entrepreneurship not regular software jobs.

Psychometrics of Algorithm Test

IQ Tests are used as a "proxy" for intelligence measurement. IQ tries to measure your ability to learn diverse subjects and ability to use working memory.

IQ test predicts your ability to learn but not the ability to perform.

We can classify candidates based on IQ into 4 ranges

(i) 80 - 100 => Below Average (40% people)
(ii) 100 - 120 => Average (40% people)
(iii) 120 - 140 => Above Average (8% people)
(iv) 140+ => Higher (0.3% people)

Approximate time ratio to learn totally new abstraction for different IQ people,

Below Average : Average : Above Average : Higher = 8 : 4 : 2 : 1

Candidate's lack of fluency in language of Test can reduce IQ scores of above average person (say 120) into average (say 100). Lack of familiarity with culture / society which conducts the exam might reduce the IQ score further into below average (say 80).

Universities curriculum are designed according to average IQ people.

Lack of good educational resources in native language with similar cultural context can reduce learning speed 2 times for most average students.

It means loss of interest of average IQ students in the subject who are weak with english comprehension and doesn't belong to similar social background as majority of students.

DSA Questions has two major abstractions (i) loops (ii) recursion

It is the place where IQ will hamper your learning abilities the most.

If you have average IQ and you are decent in english then you don't have to worry.

You can even do competitive coding so cracking most algorithm interviews won't be big deal if you practice enough and use the right educational resources.

That's why, we will start with loops and recursion and drill down on it by dry run them with our hands before doing any other thing.

For below average IQ people, you should ask yourself the following question,

Can you give 2X time to DSA than most people ? Is DSA important for your career goal ?

People who are good at DSA get chance to work at Google but only 10% of them reach the position of staff engineer where they actually do algorithms and optimizations.

No one can thrive in the market for long term without identifying their real strengths.

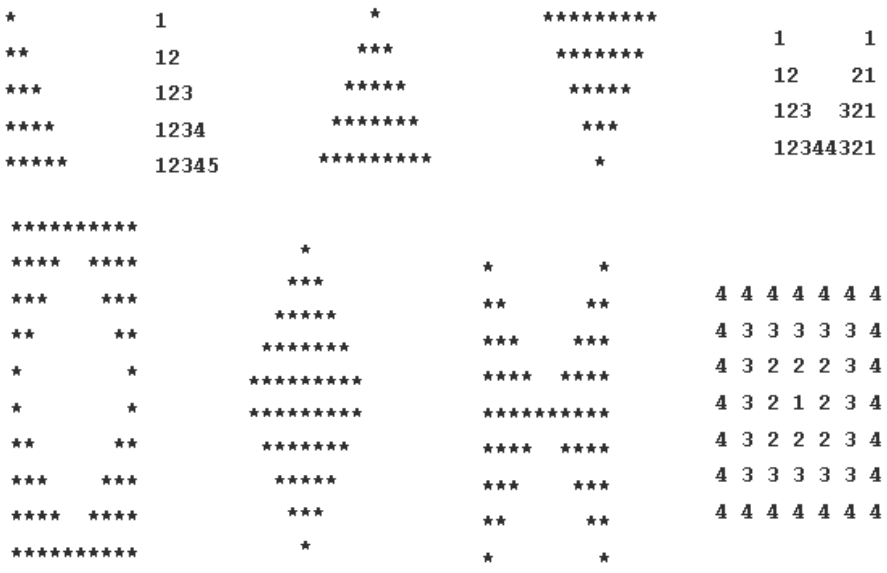
DSA can be very complex but it is less diverse which means role of IQ decrease as you practice. DSA Tests aren't IQ test due to lack of diversity in question, they are skill tests.

School exams can be a reflection of IQ due to the diversity of subjects more than DSA tests or University exams but both are generally inaccurate and biased. That's why psychologists develop separate IQ measurements tests with less bias like WAIS-4.

Do it with Hands : Loops, Recursion, Math, Basic Hashing and Big O

Source : *#takeuforward #striver_a2z_sheet*

Pattern



Math

(i) Count digit (ii) Reverse digit of number (iii) Check Palindrome (iv) GCD/HCF
(v) Armstrong Number (vi) Print all divisors (vii) Check for Prime

Recursion

(i) Print 1-N using recursion (ii) Print N-1 using recursion (iii) Sum of first N numbers
(iv) Factorial (v) Reverse an array (vi) Check if palindrome (vii) Fibonacci
(viii) Power set (ix) Combination Sum I – array of distinct integers and target sum K

Basic Hashing

(i) Count frequency of each element in array (ii) find highest/lowest frequency element

Big O

Find time and space complexity for sorting algorithms listed below

(i) Selection (ii) Insertion (iii) Bubble (iv) Merge (v) Quick

Pick a language and start using keyboard and compiler : C++ and STL

Sets and Multisets

Container	Description	Common Use Cases
unordered_set	Hash table-based set with average O(1) insertions, deletions, and lookups. No specific order.	Quick lookups, checking existence, removing duplicates.
set	Tree-based (usually red-black tree) set with O(log n) insertions, deletions, and lookups. Sorted.	Sorted collections, unique collections, range queries.
unordered_multiset	Hash table-based set allowing multiple equivalent elements. Average O(1) operations.	Counting occurrences, grouping elements with duplicates.
multiset	Tree-based set allowing multiple equivalent elements. Sorted.	Counting occurrences, maintaining sorted collections with duplicates.

Maps and Multimaps

Container	Description	Common Use Cases
unordered_map	Hash table-based map with average O(1) insertions, deletions, and lookups. Unique keys.	Fast lookups and updates using unique keys, hash tables, dictionaries.
map	Tree-based map with O(log n) insertions, deletions, and lookups. Sorted.	Sorted key-value pairs, ordered dictionaries, range queries.
unordered_multimap	Hash table-based map allowing multiple equivalent keys. Average O(1) operations.	Storing key-value pairs with duplicate keys, grouping values.
multimap	Tree-based map allowing multiple equivalent keys. Sorted.	Storing key-value pairs with duplicate keys, grouping values sorted.

Sequential Containers

Container	Description	Common Use Cases
vector	Dynamic array with O(1) random access and fast append operations.	General-purpose dynamic arrays, lists, and buffers.
list	Doubly linked list with O(1) insertions and deletions at any position.	Dynamic lists, frequent middle insertions/deletions, flexible queues/stacks.
deque	Double-ended queue with O(1) insertions and deletions at both ends.	Queues, stacks, sliding window algorithms.

Adapters

Container	Description	Common Use Cases
queue	FIFO data structure with O(1) insertions at back and deletions from front.	Buffers, task scheduling, breadth-first search (BFS).
stack	LIFO data structure with O(1) push and pop operations.	Function calls, backtracking algorithms, depth-first search (DFS).
priority_queue	Heap-based structure with O(1) access to highest/lowest element, O(log n) insertions/deletions.	Priority scheduling, Dijkstra's algorithm, accessing largest/smallest.

Algorithms

Function	Algorithm(s) Used	Time Complexity	Space Complexity	Details / Use Cases
next_permutation	Combination of swapping and reversing algorithms	O(n)	O(1)	Generates next lexicographical permutation, useful in combinatorial problems.
__builtin_popcount	CPU-specific instructions (POPCNT on x86, etc.)	O(1)	O(1)	Counts set bits in an integer efficiently, used in bit manipulation tasks.
sort	Introsort (hybrid of quicksort, heapsort, insertion sort)	O(n log n)	O(log n) (stack space)	Sorts elements efficiently, adapts to input size and distribution.
min_element	Linear scan through the range	O(n)	O(1)	Finds the smallest element in a range, optimal for linear traversal.
max_element	Linear scan through the range	O(n)	O(1)	Finds the largest element in a range, optimal for linear traversal.

Practice Problems

Implement with C++

Q1. Given the list of products in a shop, get the list of unique products via *set* and get the sorted list of products via *multiset*.

Q2. Count occurrence of all words in the given string using *unordered_multiset and multiset.* Discuss Why *unordered_multiset* is better*?*

Q3. Develop a simple cricket monitor program which track of bowling average after each bowl using *unordered_multiset* and *multiset.* Discuss why *multiset* is better?

Q4. Count occurrence of all words in the given string using *unordered_multiset* and store line numbers also with *unordered_multimap*

Q5. Find Kth lowest and Kth highest element using *priority_queue*.

Linear I : Arrays, Binary Search, Sorting, Sliding Window and Two Pointers

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don't touch the computer

Arrays I

(i) Remove duplicates from sorted array (ii) Left rotate by 1 place and by D places
(iii) Move Zeros to end (iv) Find A Missing number in Array of size N
(v) Sub Array (positive only, positive and negative both) with Sum K

Arrays II

(i) 2 Sum Problem (ii) Sort 0, 1, 2 (Dutch National Flag Algorithm)
(iii) Majority element occur more than N/2 times (Moore Voting Algorithm)
(iv) Maximum sum sub-array (Kadane Algorithm) (v) Stock Buy and Sell
(vi) Next Permutation (vii) Pascal Triangle (viii) Merge 2 sorted arrays

Binary Search

(i) Find an element in sorted array (ii) Find element insert position in sorted array
(iii) Find smallest/largest index of an element (lower/upper bound) in sorted array
(iv) First/Last Occurrence of a given number in sorted array
(v) Count the occurrence of a given number in sorted array
(vi) Search in rotated sorted array (vii) Peak Element.
(viii) Find row with maximum number of 1's in 2D array
(ix) Search in 2D matrix (x) Find square root of a number in log N

Sorting

(i) Selection Sort (ii) Bubble Sort (iii) Insertion Sort (i) Merge Sort (ii) Quick Sort

Sliding Window and Two Pointers

(i) Find an element in sorted array (ii) Find element insert position in sorted array
(iii) Find smallest/largest index of an element (lower/upper bound) in sorted array
(iv) First/Last Occurrence of a given number in sorted array

Greedy Algorithms, Strings and Bit Manipulation

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don't touch the computer

Greedy Algorithms

(i) Assign Cookies (ii) Job Sequencing (iii) Valid Parenthesis (iv) Fractional Knapsack

Strings

(i) Palindrome Check (ii) Anagram Check (iii) Isomorphic String (iv) Roman to Integer
(iv) Count frequency of string (v) count all sub-strings of length K

Bit Manipulation

(i) Binary Number Conversion (1s and 2s compliment)
(ii) Operations (AND, OR, XOR, NOT, SHIFT) (iii) Swap Two Numbers
(iv) Check if the i-th bit is set or not (v) Count number of set bits
(vi) Check if number is odd or not (vii) Check if number is power of 2

Linear II : Linked List, Stacks and Queues

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don't touch the computer

Linked List

(i) Singly Linked List – insert, delete, find an element, reverse
(ii) Doubly Linked List – insert, delete, reverse.
(iii) Detect a cycle and find middle in Singly Linked List (tortoise-Hare)
(iv) Sort Linked List using Merge Sort (v) Delete all occurrences

Stacks and Queues

(i) Stack and Queue using Array (ii) Stack and Queue using Linked List
(iii) Infix to Postfix/Prefix (iv) Next Greater Element
(v) Implement LRU / LFU (vi) Largest rectangle in histogram

Non Linear I : Binary Tree and Heaps

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don't touch the computer

Binary Tree

(i) Tree Traversal – In Order, Pre Order, Post-Order and Level Order
(ii) Height of Binary Tree (iii) Check if balanced Tree or not

Heap

(i) Heap Implementation – min heap/max heap (ii) Kth largest / Kth smallest using Heap

- Balanced Tree lost relevance in Interview/CP due to Heap : AVL Tree, Red Black, B/B+ Tree

Understanding your limitations with Tests.

Before any test, lets understand few important things.

If you solved all above questions then you can clearly observe that almost 60% of the questions uploaded on DSA/Competitive Coding related website such as Leetcode, Codeforces, Geekforgeeks etc belong to the topics we discussed till now and falls under easy-medium category and easily understandable if practiced properly.

Actually, these questions are primarily designed to improve website's SEO not necessarily to help candidate improve their understanding of concepts. If you are doing questions randomly then probably you are just fighting the SEO not actually learning the concepts.

If you are going with the approach like "*I will finish all array questions then I will move to another topic say Binary Search*" then you are confusing these websites with books.

Websites are designed according to SEO means "*separate article for any searchable topic*" and books are designed according to reader's flow means "*reader must not require to go beyond the book frequently in order to understand the fundamental concepts*"

If you try solving all questions of array then you'll get stuck on array for a very long time because most questions are similar and the remaining 15-20% are those which nobody can solve optimally in one go as they are exceptions and they aren't generally asked.

If you don't listen to experts in competitive coding then it might be possible that you develop impostor syndrome. You might feel other people are so smart that they possess some magical abilities to solve these exceptions just by intuition which is big lie.

So far we have understood, In Big Tech, DSA Interviews are designed to have a flow of 10% great engineers who can sustain big infrastructure at any given moment of time.

For 90% engineers in Big Tech, the job itself doesn't require complicated tasks on daily basis. So, they have luxury to do diversity hiring according to their local politics.

In western countries, diversity is defined by "gender" and "race" but in India it is defined by "caste" and "religion" but these companies only consider "gender" as diversity in India majorly because they only have to comply with US politics because their existence across the globe without any major conflicts is not possible without support of US Government.

In US, companies often fire employees abruptly which is bad for employees in short term but it improves companies decision making, innovation and quality of products. It doesn't damage employees long term due to decent social security and robust economy.

In India, firing employees abruptly could be a disaster due to no social security and not necessarily it improve innovation and quality of products as Indian market is highly price sensitive and most Indian customers don't have sufficient purchasing power.

So, these organizations aren't as unbiased or free from political influence as they claim.

Before going further,

Lets discuss DSA from University perspective with Donald Knuth's story.

Donald Knuth started his bachelors in Physics then switch to mathematics after getting exposure to IBM 650 computer. He completed his PhD in Mathematics at Cal Tech, focusing on "*finite semifields and projective planes"* topics based on coding theory and combinatorial design which were not directly related to the computing.

During PhD, he began to teach programming and he was dissatisfied with the way programming was taught, it wasn't structured properly and doesn't have good enough mathematical foundations. So, he came up with the concept of *"analysis of algorithm with asymptotic notations (Big O/Theta/Omega)"*. Later he formalize everything in his famous book *"The Art of Computer Programming"* and won *Turing Award* for his contributions.

If you analyze the *Don Knuth's* story carefully then you'll realize that his work is more heavy on philosophy and combination of mathematics and computer science. Great research generally need inter-disciplinary approach. In early days of "*analysis of algorithms*", Knuth had no idea how to explain it to others because it doesn't fit into any existing domain of that time so he take inspiration from Richard Bellman's *Dynamic Programming* and name this process as *analysis of algorithms*.

To create thinkers like Don Knuth, University curriculum at Undergraduate level can't be fixated around some particular topic or skill with high market demand because real value is created by research which requires exposure of many subjects. It help candidate develop the ability to ask different questions and ultimately lead to new discoveries and inventions. Curriculum with too many subjects at undergraduate level provides constant flow of atleast 5% high quality research and teaching candidates out of total undergraduates and sometimes produce great scientists and inventors like Don Knuth as well.

So, far we have understood the following,

Big Tech is only interested in their top 10% engineers who sustain their big infrastructure, Coding websites like Leetcode are focused upon improving their ranking via SEO instead of helping you learn things efficiently and Universities are only interested to get top 5% research and teaching candidates out of total enrolled undergraduates.

They aren't interested to help you build relevant skills according to your strengths and market demand neither they are interested to help you build high quality professional network. No one is actually interested in your career, your goals, your personal finance.

If you see yourself as a worker of company where company decides what you should be doing on long term basis then you will never be able to develop your USP.

If you see yourself as a contributor then you'll never get identified with company and see it as a way to develop your unique strengths or a place to make bigger contribution and ultimately charge higher price for your valuable services.

Now, you can attempt the test below as you are doing it to understand your strength and weaknesses instead of others reasons.

TEST (Coding + Basic Psychometrics)
Implement with C++

Source : *#takeuforward #striver_a2z_sheet*

You have 1 hour to attempt all questions, it is highly unlikely that you'll be able to solve all the questions within 1 hour and that's why it will help us understand our weakness on the basis of 4 factors mentioned below. Try to measure these 4 factors for each question

1. Understanding Question 2. Brute force solution 3. Optimal solution 4. C++ Implementation

Go to the website *takeuforward.org and #striver_a2z_sheet* section and try to copy paste the questions and examples section of above questions into a word file.

If you feeling lazy to copy paste or feeling to look into the solutions even before trying then it might be an indication that you don't actually like DSA.

Q1. 3 Sum Problem

Q2. Merge Overlapping sub intervals

Q3. Find repeating and missing numbers

Q4. Aggressive Cows

Q5. Search an element in sorted and rotated array? Find Pivot Elements.

Q6. Median of two sorted array

Q7. Rotten Oranges using BFS

Q8. Trapping rainwater

Q9. Jump Game

Q10. Assign Cookies

Start the timer and try to solve questions in the following format,

First half hour, Try to understand problems and write their brute force solutions on paper
Second half hour, Try to find optimal solutions and also try to implement it in C++

If you have problem with,

1. Understanding Question : Either you are not good with understanding english or mathematical notations.

2. Brute force solution : You don't have the conceptual clarity

3. Optimal solution : You haven't practiced sufficient questions

4. C++ Implementation : You aren't comfortable with remembering C++ language or typing

Recursion and Dynamic Programming

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don’t touch the computer

Recursion

(i) Print Power set (ii) Combination Sum I (iii) Combination Sum II (iv) Subset Sum I (v) Subset Sum II (vi) Print All Permutation (vii) N Queen (viii) Rat in Maze

Dynamic Programming

(i) Climbing stairs (ii) Frog Jump (iii) Frog Jump with K distances (iv) Subset Sum equal to target (v) Partition Equal Subset Sum (vi) Count Subset with Sum K (vii) 0/1 Knapsack (viii) Minimum coin (ix) Rod Cutting (x) Longest common subsequence.

Dynamic Programming II

Buy and Sell Stock I, II, III, IV, Cool down and Transaction Fee

Non Linear II : Graphs

Source : *#takeuforward #striver_a2z_sheet*

Write it with your own hands on paper, don’t touch the computer

Graph – BFS/DFS and Cycle Detection

(i) Graph Representation Matrix and Adjacency List (ii) Connected Components (iii) BFS (iv) DFS (v) Flood Fill (vi) Number of Island (vii) Number of Enclaves (viii) Cycle detection in Graph - (directed / undirected) and Safe States (ix) Bipartite Graphs (x) Distinct Island

Graph - Topological Sort

(i) Topological Sort (ii) Kahn Algorithm (iii) Course Schedule I, II (iv) Alien Dictionary

Graph - Shortest Path and Minimum Spanning Tree

(i) Dijkstra’ Algorithm (ii) Binary Maze (iii) Cheapest flight with K stops (iv) Bellman Ford (v) Floyd Warshall Algorithm (vi) Minimum Spanning Tree : Prim’s Algorithms

Now you have gained the capability to understand any advanced data structures and algorithms by overcoming the difficulty and uneasiness of dealing with the abstractions of DSA. Some of you have even begun to like DSA, and that's the real goal. It is 80% of the equation.

Going Advance : Non SDE or Non Big Tech SDE

Career without DSA Test as Primary Criteria

As we already discussed, only 10% jobs in Big Tech actually require great proficiency in DSA and rest of the jobs even in Big Tech or other companies don't require this. So lets analyze, what kind of software engineering jobs are out there mostly.

There are two types of companies B2B or B2C and almost 70% of the software jobs exists in B2B setup. In B2B, almost 70% jobs belong to back-end where reliability is more important than efficiency that's why almost 50% of B2B back-end software is written in java based technologies but in B2C almost 50% of jobs belong to front-end which uses Javascript based technologies and remaining 50% belong to back-end out of which almost 50% of back-end uses Javascript based technologies for real time applications.

You can't escape Java if you are targeting jobs in B2B companies and Javascript if targeting jobs in B2C companies.

Lets dive deeper,

In B2B, back-end software dependencies change more frequently than B2C as B2B business changes vendors more often if they don't qualify their quality standards which means we need a dependency injection frameworks like Springboot.

In B2C, back-end software dependencies doesn't change more frequently but they need ability to maintain huge amount of active users for real time application which means Javascript is better but it's can't do heavy computing tasks as it is single threaded that's why people started using Go-lang as well.

When we build back-end, most of the queries are just CRUD operation and when we need performance we can always use interfacing with C++ in both Java and Javascript or create another service in Go-lang for performance or in Python for machine learning algorithms.

Almost 80% of software jobs needs either JavaScript, Java or Python based technologies

You can pick ExpressJS (JavaScript) or Springboot (Java) or Django (Python) for back-end roles and ReactJS or AngularJS for front-end roles.

As you grow into senior role,

You must know data structures like stack, queues, graph and algorithms like mark sweep to understand garbage collection which determine the performance of a language at large scale and DSA also help us determine which tech stack and database we should select for our use case and how to design large scale or mid scale distributed systems.

In Academia or research, you need to deal with mathematics and abstract concepts a lot more than implementation of algorithms. They either work on abstract concepts or teach students and some of them become people like Donald Knuth who create entire domain. You should know as many data structures and algorithms as possible without worrying about your ability to implement them in a short time because you don't need to give any coding test at University and it will help you came up with new ways of computation.

Going Advance : Big Tech SDE or Staff

Career with DSA Test as Primary Criteria

As SDE in Big Tech, you need to practice questions which can be asked in interview.

Usually you don’t need to worry about problems involving balanced trees like B/B+ Trees, AVL, Red Black etc due to existence of heap and they can’t be asked within a short duration of interview time. Only topics which can be asked in short duration of time and convoluted beyond a point are Dynamic Programming and Graphs. That’s why they hold more weight-age at Big Tech interviews and competitive coding contests.

You need ability to communicate your solutions to the interviewer otherwise nothing works.

As SDE in Big Tech, your role is similar to engineers of other companies but your implementation speed and debugging accuracy will be much higher as coding tests train you for that and you’ll get exposure to large scale distributed systems earlier.

As you grow into senior role,

I mean to say Staff, your debugging abilities will be much higher than most engineers for a particular technology which helps Big Tech save lots of money and make you highly paid employee. Only big tech pays for high accuracy because their product work at large scale but only 10% of engineers are needed for such positions but it doesn’t mean that their salary become 2X of Engineering Manager or Product manager, probably 20% more.

The main perk could be lack of responsibilities towards people and redundant tasks which enable them to do technical work with very high efficiency.

They are called 10X engineers but this label could be very misleading because there must exists enough work of similar kind at a particular instance of time in all organizations to make the comparison valid.