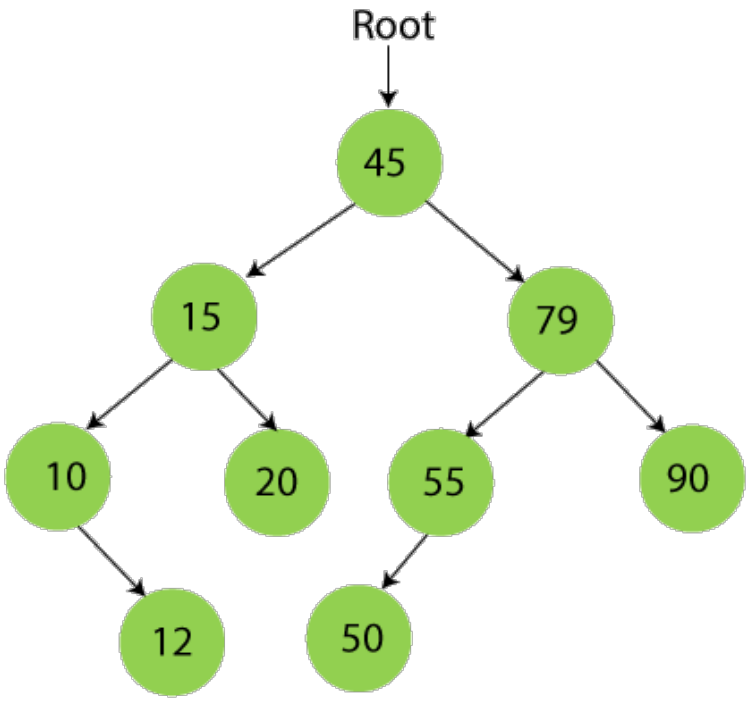
Data Structures and Algorithms Interview Guide



Pre-requisite

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

This Quick book is more about “Understanding Why?”

Author : Yash Pratap

Index

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 30 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. The 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 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.

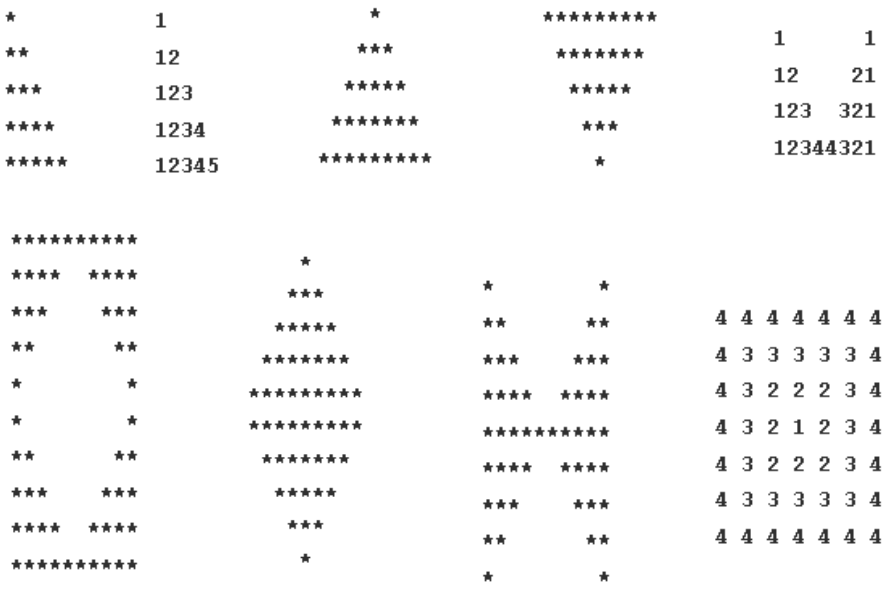
DSA can be very complex but it is less diverse which means role of IQ decrease as you practice. DSA Tests aren't IQ test due to lack of diversity in question, they are skill tests.

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

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

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 Algo)
(iii) Majority element occur more than N/2 times (Moore Voting Algo)
(iv) Maximum sum sub-array (Kadane Algo) (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

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

Sorting II

(i) Merge Sort (ii) Quick Sort

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

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

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

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)