CRACKING THE GAMAM TECHNICAL INTERVIEWS

Strategies, Tips, and Preparation resources

AN INSIDER'S GUIDE

Dinesh Varyani
Engineer @ Google

Cracking the GAMAM Technical Interviews

5th Edition

AN INSIDER'S GUIDE

Dinesh Varyani
Engineer @Google

Dinesh Varyani Dinesh Varyani Dinesh Varyani

Cracking the GAMAM Technical Interviews, Fifth Edition

For more information, contact
hubberspot.publisher@gmail.com

www.hubberspot.com

Table of Contents

To my mother,
who taught me that it's never too late to chase your dreams !!!

Preface

Dear Reader,

I'm Dinesh Varyani, a Cloud Engineer at Google with over 13 years of experience in software engineering. As a passionate YouTuber, blogger, Udemy instructor, and now an author, I'm excited to share my knowledge and experience with you.

The primary goal of this book is to assist you in preparing for and acing technical interviews at top tech companies like Google, Apple, Microsoft, Amazon, and Meta (GAMAM). This book offers valuable preparation resources, strategies, tips, and a comprehensive roadmap that I used to navigate Google and Amazon's technical rounds successfully. Through this book, I aim to share insights into:

- 👉 My journey of preparing for GAMAM companies
- 👉 The resources I utilized for various interview formats
- 👉 The strategies I employed to master technical topics
- 👉 The detailed monthly preparation roadmap I followed
- 👉 Effective resume tips to attract recruiters' attention
- 👉 Tracking progress throughout the preparation process
- 👉 Advice and tips for confidently answering technical interview questions

I wish you all the best in your endeavors. I'm confident that you'll find this book to be a valuable resource.

01

Coding Interview

- 👉 Preparation resources
- 👉 Do's and Don'ts in an Interview
- 👉 Things to do when you code
- 👉 Things to do when you are stuck
- 👉 Golden rules for solving a Coding problem in an Interview
- 👉 Coding Interview Patterns
- 👉 Important Algorithms you should know before your Coding Interview

Preparation resources

👉 **Important DSA topics** - Array, Binary Search, Sliding Window, Matrix, Two Pointer, Intervals, Hash Map, String, Recursion, DP, Trees, Graph, Linked List, Stack, Queue & Heap

👉 Solve LeetCode Medium level problems (at least more than 300+ covering different topics).

👉 I have created an xlsx on top/important 500 LeetCode questions and a video on How to Crack The Coding Interview?

👉 AlgoExpert's 200 handpicked coding questions (In case you want to prepare fast and only good questions)

👉 Watch my DSA playlist to revise concepts. Download all the step-by-step animated slides from here

👉 Grokking the Coding Interview: Patterns for Coding Questions - The course is excellent and has covered various coding problems segregated based on coding patterns.

Do's

- ✅ Keep a smiling face, and look confident/positive attitude person.
- ✅ Ask good clarifying questions about the coding problem e.g. size/range of the input, are there any duplicates, does input contain negative values, etc?
- ✅ Make the interview process a team effort. The more collaboration you do with your interviewer the more idea they get about how good a team player you are.
- ✅ Think out loud. Always try to explain what you are thinking about the current state of the problem.
- ✅ Always be open to saying that you don't know how certain things work.

Do's

- ✅ Always start thinking about the simpler version of the problem. Try to come up with a naive solution at first and later go for optimizing it.
- ✅ Listen carefully to what your interviewer wants and respond accordingly.
- ✅ Prepare a list of good questions related to the company, technology, work culture, etc, and always ask the interviewer at the end.

Dont's

- Never dive into solving a problem as soon as it's thrown toward you. Understand the problem, and resolve ambiguities.
- Never assume anything. Always clarify the assumptions you have with your interviewer.
- Avoid any technical jargon or famous words you know. If you do be prepared for the follow-up question.
- Never try to skip any idea or communication on which the interviewer wants to focus more.
- Not be too defensive about the mistakes that the interviewer tells you.

When you code

👉 You are expected to write production-level code.

👉 Check for edge cases.

👉 Validate input and throw meaningful exceptions.

👉 Modularize code into different functions.

👉 Write meaningful variable/method names.

👉 You are expected to know the Time and Space the complexity of the code you have written.

👉 You are expected to dry-run your code with the example given.

👉 Don't worry about the exact syntax of the code. Meaningful text can also convey the point you trying to achieve.

👉 Try to clean up the code - check for any edge cases, refactoring, removing unwanted comments (in case you comment anything), check for conditions, etc.

When you are stuck

👉 If you are stuck and unaware of any logic, just make/call a helper function (explain it will do XYZ)

👉 If you are stuck anywhere, your interviewer is the best person to help you out. Ask them for any hints or any question that clarifies your doubt. Remember the interviewer is not there to make you fail, they want you to succeed.

👉 If you are stuck, never go into silence. Always try to explain to the interviewer where you are stuck, and what you are thinking.

👉 If you are stuck in logic try to apply coding patterns - like can two pointer help, can sort help, can binary search be applied, can breadth-first search or depth-first search algorithms be applied etc.

Golden rules for solving a Coding problem in an Interview

👉 If the coding problem requires performing an operation that needs faster search in O(1), try to use **Set** or a **Map.**

👉 If the coding problem requires finding/manipulating/dealing with the top, bottom, maximum, minimum, closest, and farthest "**K**" elements among given "**N**" elements, try to use a **Heap.**

👉 If the coding problem has input as a sorted **Array**, **List**, or **Matrix**, try to use **Two Pointer** strategy or try to use **Binary Search**.

👉 If the coding problem requires trying all **Permutations** and **Combinations**, we can use either **Backtracking** or **Breadth First Search**.

👉 If the coding problem has input in the form of a **Tree** or **Graph,** then most of the time can be solved by applying Tree Traversals or Graph Traversals algorithms called **Breadth First Search (BFS)** and **Depth First Search (DFS)**.

👉 If the coding problem is around a **Singly Linked List**, and If you are stuck in traversals logic, then try to use either **Two Pointers or Slow/Fast Pointers**.

👉 If the coding problem has a recursive solution but it's hard to visualize/code, try using a **Stack** data structure with a loop.

👉 If the coding problem revolves around iterating an array, and takes $O(N^2)$ time complexity, O(1) space complexity then try to use a **HashMap/HashSet.** It makes the algorithm faster with O(N) time complexity but takes more space with O(N) space complexity.

👉 If the coding problem revolves around iterating an array and takes $O(N^2)$ time complexity, and O(1) space complexity then try to **sort** the array**.** It makes the algorithm faster with O(N log N) time complexity and O(1) space complexity.

👉 If the coding problem requires optimization around the recursive the solution, there **could** be a possibility that **dynamic programming** can be used.

👉 If the coding problem has a group of strings or some manipulation/find/storing needs to be done around the substring, there is a high possibility that either **Tries** or **HashMap** can be used.

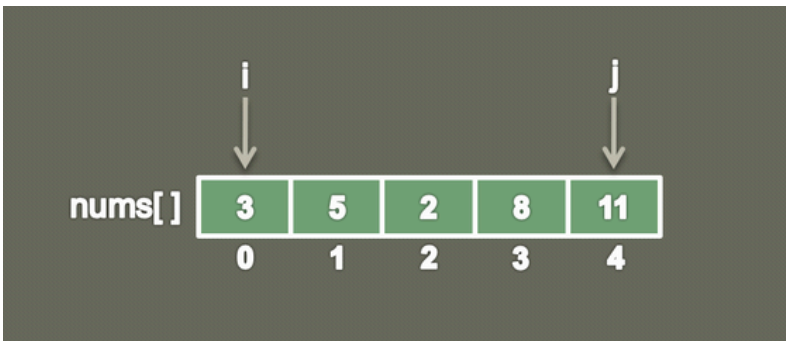
Coding Interview Patterns

1. Two Pointers

In this coding pattern, pointers are references to an array/list indexes. By using two pointers, we can process two elements at a time in a single loop. The two pointers iterate over an array/list in certain directions until some conditions are fulfilled and process two index elements simultaneously. The pattern is often used when -

👉 array/list is sorted and a comparison needs to be done between its elements.
👉 array/list elements need certain rearrangements/removal in-place.

The two pointers are typically represented by **i** and **j**.



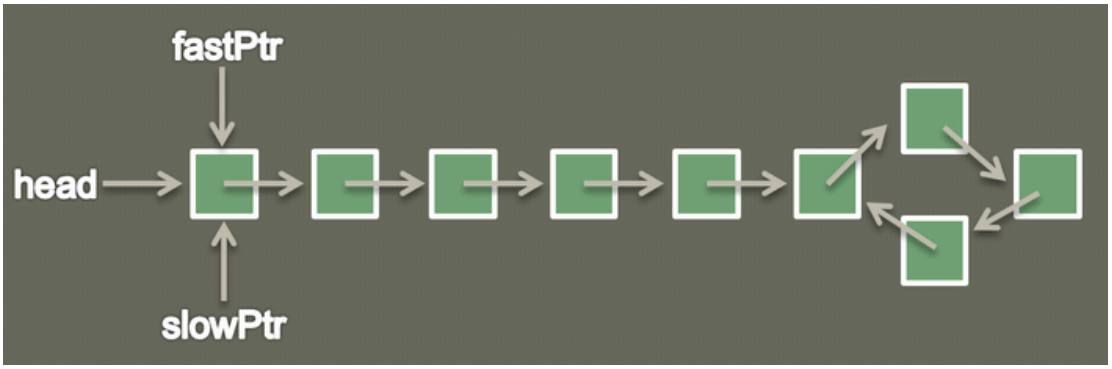
1. Two Pointers

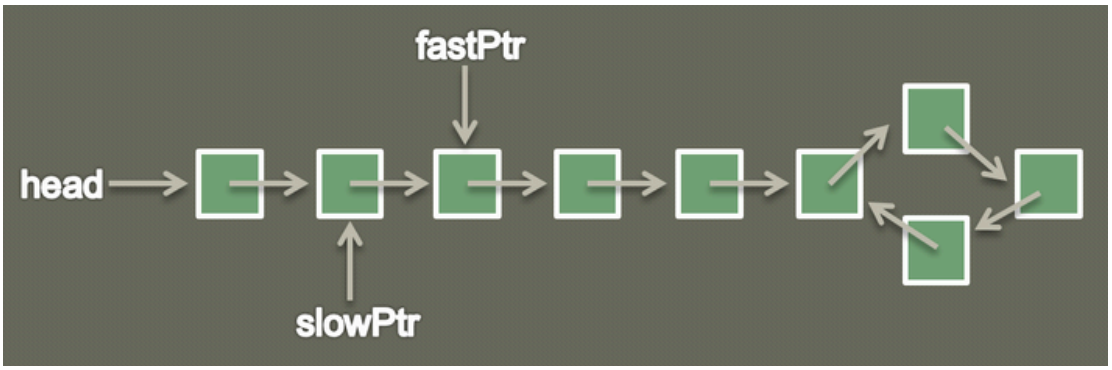
✅ Sample Questions

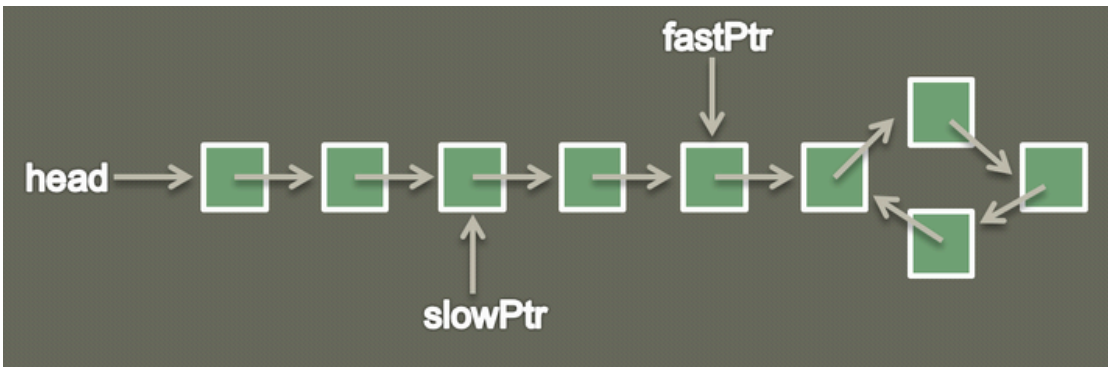
👉 Two Sum II - Input Array Is Sorted
👉 Container With Most Water
👉 Remove Duplicates from Sorted Array
👉 Next Permutation
👉 Trapping Rain Water

2. Fast and Slow Pointers

In this coding pattern, two pointers are used by name fast and slow. These pointers iterate an array/list at different speeds. The algorithm involved is termed as, Hare and Tortoise algorithm.







2. Fast and Slow Pointers

✅ Sample Questions

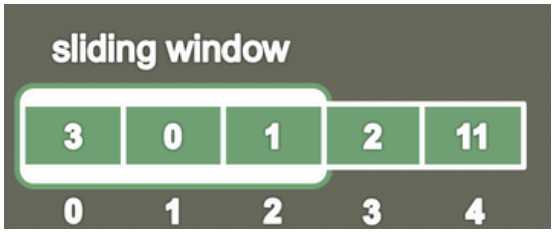
👉 Linked List Cycle
👉 Middle of the Linked List
👉 Palindrome Linked List
👉 Happy Number
👉 Circular Array Loop

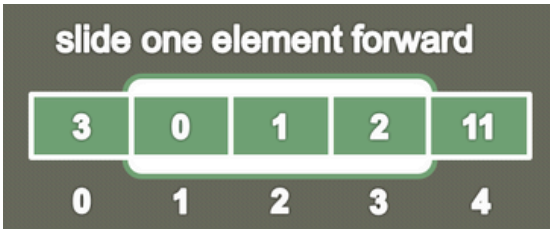
3. Sliding Window

The sliding window pattern is a common algorithmic technique used in solving problems that involve arrays, strings, or other sequence-like data structures. It involves defining a window or a subarray/substring within the given data and then iteratively moving or sliding the window to solve the problem efficiently.

Here's how the sliding window pattern typically works:

1. Initialize two pointers, "start" and "end," which define the current window.
2. Slide the window by moving the "end" pointer to the right, expanding the window.
3. Check if the current window satisfies the problem's constraints or requirements.
4. If the window satisfies the constraints, update the result or take any required action.
5. Shrink the window by moving the "start" pointer to the right, removing elements from the window.
6. Repeat steps 2 to 5 until we have processed all elements in the given data structure.





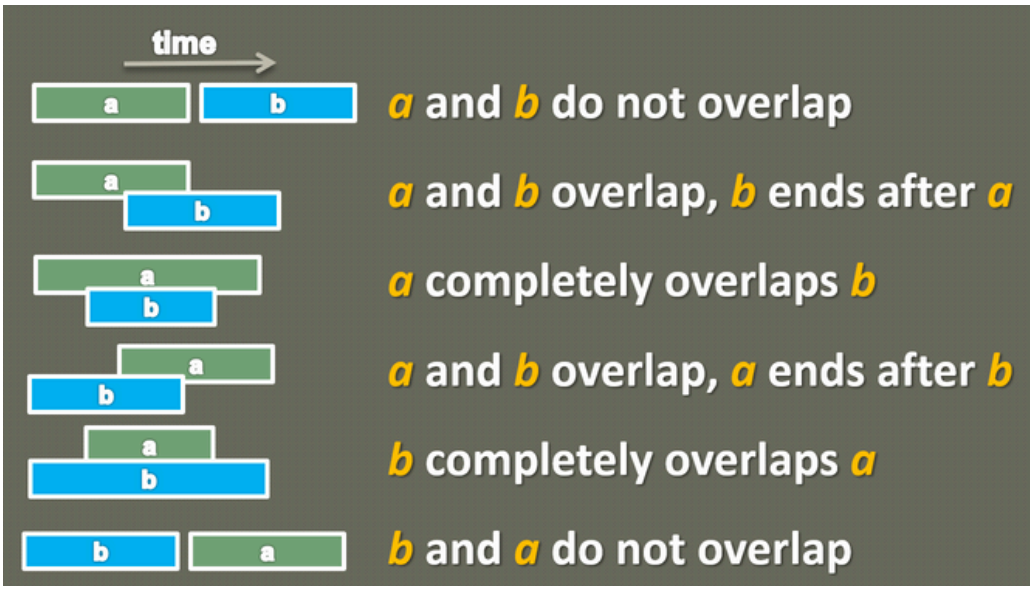
3. Sliding Window

✅ Sample Questions

👉 Longest Substring Without Repeating Characters
👉 Longest Repeating Character Replacement
👉 Sliding Window Maximum
👉 Permutation in String
👉 Fruit Into Baskets

4. Merge Intervals

The Merge Interval is a problem-solving technique used to solve problems that involve intervals or ranges that can overlap. It is used when the problem requires combining intervals that share a common overlap or intersection.

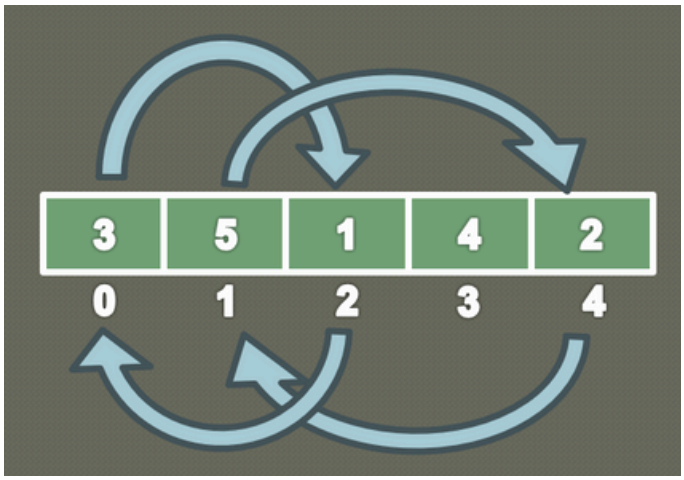


✅ Sample Questions

👉 Merge Intervals
👉 Insert Interval
👉 Non-overlapping Intervals
👉 Interval List Intersections

5. Cyclic Sort

The Cyclic Sort pattern is used for solving problems when input data lies in a fixed range. It is used when the problem requires arranging the elements of the array in a cyclic manner to achieve the desired order.

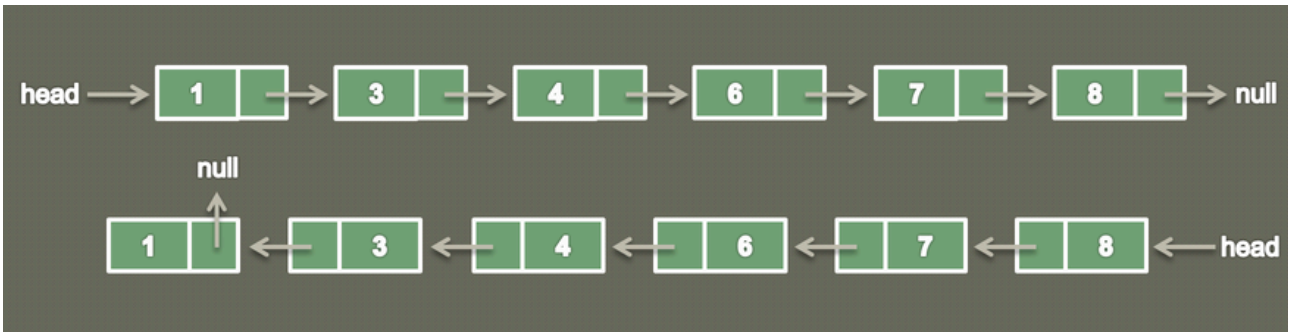


✅ Sample Questions

👉 Kth Missing Positive Number
👉 Find All Numbers Disappeared in an Array
👉 Set Mismatch
👉 Find the Duplicate Number
👉 Find All Duplicates in an Array
👉 Missing Number

6. Linked List In-Place Reversal

In this coding pattern, many a times it is required to reverse the order of the elements in the linked list in-place i.e manipulating the existing node and without using extra memory or using any additional data structures.



✅ Sample Questions

👉 Reverse Linked List
👉 Rotate List
👉 Reverse Linked List II
👉 Reverse Nodes in k-Group

7. Two Heaps

The Two Heaps coding pattern is a technique used to efficiently solve problems that involve managing two sets of data in a way that allows quick access to the minimum or maximum values from each set. This pattern is often applied to problems dealing with dynamic data streams, such as online algorithms and priority queue-related tasks. The two heaps typically used are the Min Heap and the Max Heap.

A step-by-step explanation of the Two Heaps pattern -

Create two heaps: One Min Heap and one Max Heap.

👉 **Min Heap** - A binary heap where the parent node's value is smaller than or equal to its child nodes' values. The root node holds the minimum value in the heap.

👉 **Max Heap** - A binary heap where the parent node's value is greater than or equal to its child nodes' values. The root node holds the maximum value in the heap.

Divide the data into two parts -

👉 Data that will be inserted into the Min Heap - This will store elements greater than the current median.

👉 Data that will be inserted into the Max Heap - This will store elements less than the current median.

7. Two Heaps

👉 **Balance the heaps** - The difference between the number of elements in the Min Heap and the Max Heap should not exceed 1. This ensures that the median will always be at the root of either the Min Heap or the Max Heap (or the average of the roots if the heaps have equal size).

👉 **Accessing the median** -
- If both heaps have the same number of elements, the median will be the average of the roots of the two heaps.
- If the heaps have different sizes, the median will be the root of the heap with more elements.

👉 **Inserting elements** -
- Compare the new element with the current median.
- If the new element is smaller than or equal to the current median, insert it into the Max Heap.
- If the new element is greater than the current median, insert it into the Min Heap.

👉 **Rebalancing the heaps** -
- After inserting the element, check if the difference between the sizes of the two heaps is greater than 1.
- If it is, remove the root of the larger heap and insert it into the smaller heap to maintain balance.

7. Two Heaps

The Two Heaps pattern allows you to efficiently find the median or other extreme values (minimum or maximum) in the dataset without the need for sorting the entire data each time a new element is added. It is particularly useful for scenarios where the data is dynamic and continuously changing. The time complexity for insertion and retrieval of median values is O(log n), where n is the number of elements in the heaps.

✅ Sample Questions

👉 Find Right Interval
👉 Find Median from Data Stream
👉 Sliding Window Median
👉 IPO

8. Breadth First Search (BFS)

BFS is a powerful technique used to solve problems related to traversing or searching in a graph or tree data structure. It is used when the problem requires exploring or visiting all the vertices or nodes of the graph or tree in a breadthwise manner, i.e., visiting the vertices at the same level before moving to the next level.

✅ Sample Questions

👉 Binary Tree Level Order Traversal
👉 Binary Tree Zigzag Level Order Traversal
👉 Binary Tree Level Order Traversal II
👉 Minimum Depth of Binary Tree
👉 N-ary Tree Level Order Traversal
👉 Average of Levels in Binary Tree
👉 Flood Fill
👉 Find if Path Exists in Graph
👉 Max Area of Island
👉 Number of Islands
👉 Rotting Oranges
👉 Course Schedule

9. Depth First Search (DFS)

Depth-First Search (DFS) is a popular algorithm used for traversing or searching tree and graph data structures. It starts from the root (or any arbitrary node in a graph) and explores as far as possible along each branch before backtracking. The basic idea is to visit nodes and explore deeper into the data structure before backtracking to explore other branches.

✅ Sample Questions

👉 Binary Tree Preorder, Inorder, Postorder Traversals
👉 Number of Islands
👉 Validate Binary Search Tree
👉 Lowest Common Ancestor of a Binary Tree
👉 Symmetric Tree
👉 Binary Tree Maximum Path Sum
👉 Invert Binary Tree
👉 Diameter of Binary Tree
👉 Flatten Binary Tree to Linked List
👉 Kth Smallest Element in a BST
👉 All Nodes Distance K in Binary Tree
👉 Clone Graph

10. Backtracking

Backtracking is an algorithmic technique for finding solutions to some computational problems, that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

1. It has 6-7 main steps (refer to code on next page) - Implement a helper method usually named "backtrack" method.
2. The backtrack method takes a few parameters, common to many problems.
3. **Base case** - The backtrack method has a base case that defines when to add the temporary solution to the main result, and when to return.
4. **For-loop** - Most of the time we need a for-loop to iterate through the input so that we can select a candidate one by one.
5. **Select a candidate** - The first thing we do is choose a valid candidate.
6. **Explore the remaining problem** - Many times we use recursion, as we need to perform the same thing on the remaining problem. So, we recursively call the backtrack method.
7. **Remove the selected candidate** - We backtrack, remove the selected candidate and try other candidates.

10. Backtracking

```java
public List<List<Integer>> problem(int[] nums) {
  List<List<Integer>> result = new ArrayList<>();
    // 1. Call a helper method - "backtrack"
    // 2. helper method takes few parameters
    // common to many similar problems
  backtrack( ... , ... , ... , ... );

  // return the final answer
  return result;
}

private void backtrack(
    int [] nums, // input given in the problem
    List<List<Integer>> result, // result to return
    List<Integer> tempList, // temp solution
    int start // optional based on constraints
) {
    if() { // 3. conditional base case
      result.add(new ArrayList<>(tempList));
    } else {
        // 4. a loop to iterate all possible candidates
        for(int i = start; i < nums.length; i++){
          // 5. select a candidate
          tempList.add(nums[i]);
          // 6. explore remaining problem recursively
          backtrack(nums, result, tempList, i + 1);
          // 7. remove the candidate
          tempList.remove(tempList.size() - 1);
        }
    }
}
```

10. Backtracking

✅ Sample Questions

👉 Subsets
👉 Subsets II
👉 Permutations
👉 Permutations II
👉 Combination Sum
👉 Combination Sum II
👉 Palindrome Partitioning
👉 Letter Combinations of a Phone Number

11. Top K Elements

The "Top K Elements" coding pattern is a common algorithmic pattern used to find the top or bottom K elements in a collection of elements. It's particularly useful when you need to extract the K largest or smallest elements from an array, a list, or any other data structure.
Here's a general outline of the Top K Elements pattern:

1. **Priority Queue (Heap):** The most common approach to solve Top K Elements problems involves using a Priority Queue, often implemented as a heap data structure. Priority Queues allow efficient retrieval of the highest (or lowest) priority element in the collection.

2. **Selecting K Elements:** Depending on the problem statement, you may need to find the top K elements (K largest) or the bottom K elements (K smallest). The approach remains similar, but the priority queue's ordering may need to be adjusted accordingly.

3. **Iterating Through the Collection:** Start by iterating through the collection of elements. For each element, perform the following steps:
    a). If the priority queue contains fewer than K elements, add the current element to the priority queue.

11. Top K Elements

b). If the priority queue already contains K elements, compare the current element with the smallest (or largest) element in the priority queue. If the current element is greater (or smaller, depending on the requirement), remove the smallest (or largest) element from the priority queue and add the current element.

4. **Finalizing the Result:** After processing all elements, the priority queue will contain the top K elements. Depending on the problem statement, you may need to return these elements in a particular order or format.

✅ Sample Questions

👉 Kth Largest Element in an Array
👉 Top K Frequent Elements
👉 K Closest Points to Origin
👉 Kth Largest Element in a Stream
👉 Find K Closest Elements
👉 Least Number of Unique Integers after K Removals
👉 Reorganize String
👉 Task Scheduler
👉 Maximum Frequency Stack

12. K-Way Merge

The k-way merge coding pattern is a technique used to merge k-sorted arrays or lists into a single sorted array or list. This pattern is particularly useful when dealing with large datasets or streams of data that are already sorted and need to be combined efficiently. The k-way merge algorithm minimizes the number of comparisons needed to merge all the lists, resulting in improved time complexity.
Here's a step-by-step explanation of the K-way merge coding pattern:

1. **Initialize Priority Queue or Heap:** Begin by initializing a priority queue or heap data structure. This data structure will be used to keep track of the current smallest elements from each of the k-sorted lists.

2. **Insert Initial Elements:** Insert the first element from each of the k-sorted lists into the priority queue or heap. This step ensures that the smallest elements from each list are available for comparison and merging.

12. K-Way Merge

3. **Iterative Comparison and Merging:**
   - While the priority queue or heap is not empty:
     - Remove the smallest element from the priority queue or heap. This element will be the next in the merged list.
     - Insert the next element from the same list from which the smallest element was removed, if available.
     - Repeat this process until all elements from all lists have been merged.

4. **Handling Remaining Elements:** If any of the sorted lists still have remaining elements after the iterative merging process, simply append them to the merged list. Since the input lists are already sorted, no additional sorting is required.

5. **Return Merged List:** Once all elements have been merged, return the final merged list containing all elements from the input lists in sorted order.

12. K-Way Merge

✅ Sample Questions

👉 Kth Smallest Number in a Sorted Matrix
👉 Merge k Sorted Lists
👉 Find K Pairs with Smallest Sums
👉 Smallest Range Covering Elements from K Lists

13. Modified Binary Search

The Modified Binary Search is a coding pattern used to optimize certain types of binary search problems. While traditional binary search typically involves searching for a target value in a sorted array by repeatedly dividing the search interval in half, the modified binary search pattern adapts this approach for different scenarios, such as searching in rotated arrays or finding a target sum in sorted arrays.

✅ Sample Questions

👉 Binary Search
👉 Search Insert Position
👉 Find Smallest Letter Greater Than Target
👉 Find First and Last Position of Element in Sorted Array
👉 Search in a Sorted Infinite Array
👉 Search in Rotated Sorted Array

14. Tries

The Trie (pronounced "**try**") coding pattern, short for re***trie***val tree or prefix tree, is a tree-like data structure used to efficiently store and retrieve a large set of strings or sequences. It's particularly useful for tasks involving searching for words, prefixes, or patterns in a dataset.
The Trie is based on the concept of sharing common prefixes among multiple strings, making it space-efficient and fast for tasks like autocomplete, spell checking, and searching.

Structure of Trie

A Trie is composed of nodes that represent characters of the strings being stored. Each node typically contains the following components:

1. **Value:** This could be a character, representing the character associated with that node in the string, or it could be a boolean flag indicating whether the node marks the end of a word.
2. **Children:** References to other nodes representing the next characters in the strings. These references are often organized in a data structure like an array or a hashmap, where the keys correspond to characters.

14. Tries

3. **Optional: Additional metadata -** Depending on the use case, additional metadata can be stored in each node, such as word frequency, pointers to actual data associated with the word, etc.

✅ Sample Questions

👉 Implement Trie (Prefix Tree)
👉 Search Suggestions System
👉 Design Add and Search Words Data Structure
👉 Extra Characters in a String
👉 Word Search II
👉 Implement Magic Dictionary

Important Algorithms you should know before your Coding Interview

👉 Singly Linked List Reversal

👉 Floyd Cycle Detection Algorithm

👉 Sliding Window

👉 Binary Search

👉 Kadane's Algorithm

👉 Quick Select

👉 Insertion Sort

👉 Selection Sort

👉 Counting Sort

👉 Heap Sort

👉 Merge Sort

👉 Quick Sort

👉 Topological Sort

- 👉 Zigzag Traversal of a Matrix
- 👉 Preorder Traversal of a Binary Tree
- 👉 Inorder Traversal of a Binary Tree
- 👉 Postorder Traversal of a Binary Tree
- 👉 Level Order Traversal
- 👉 Breadth First Search in a Graph
- 👉 Depth First Search in a Graph
- 👉 Flood Fill Algorithm
- 👉 Kruskal's Algorithm
- 👉 Floyd Warshall Algorithm
- 👉 Dijkstra’s Algorithm
- 👉 Bellman Ford Algorithm
- 👉 Lee Algorithm

👉 Graph Bipartite

👉 Union-Find Algorithm

👉 KMP Algorithm

👉 Euclid's Algorithm

👉 Boyer–Moore Majority Vote Algorithm

👉 Dutch National Flag Algorithm

👉 Huffman Coding Algorithm

👉 Detect Cycle in a Directed Graph

👉 A* Algorithm

02

System Design / High Level Design Interview

Preparation resources

👉 Grokking the System Design Interview - The course has step-by-step discussions on distributed systems are designed. It also covers good real-life case studies.

👉 Alex Xu's System Design Interview course on ByteByteGo - The course covers all the content from his famous book (Vol 1 and Vol 2) System Design Interview.

👉 SystemsExpert videos to know how real-life System Design Interviews go.

Time Management in System Design Interview

- ✅ **Requirement Clarifications** (5-7 **min**)
- ✅ **Estimations** (5-7 **min**)
- ✅ **API Design** (5-7 **min**)
- ✅ **Database Schema Design** (5-7 **min**)
- ✅ **System's Detailed Design** (15 - 20 **min**)
- ✅ **Resolve bottlenecks and follow-up questions** (2-3 **min**)

System Design Interview Template

A System Design Interview usually lasts for 45-60 minutes. The following template will guide you on how to manage time duration across various aspects of it -

✅ Requirement Clarifications - (5-7 min)

Ask clarifying questions to understand the problem and expectations of the interviewer.

a) Functional Requirements

👉 Focused use cases to cover (MVP)
👉 Use cases that will not be covered
👉 Who will use the system
👉 Total/Daily active users
👉 How the system will be used

b) Non Functional Requirements

👉 Is the system Highly Available or Highly Consistent CAP theorem?
👉 Does the system require low latency?
👉 Does the system needs to be reliable?

System Design Interview Template

✅ Estimations (5-7 min)

👉 Latency/Throughput expectations
👉 QPS (Queries Per Second) Read/Write ratio
👉 Traffic estimates
👉 Storage estimates
👉 Memory estimates

✅ API Design (5-7 min)

👉 Outline the different APIs for required scenarios
👉 Identity request and response bodies required by APIs
👉 Identify HTTP methods APIs will target such as, GET, POST, PUT, DELETE, PATCH, etc
👉 Identity HTTP Status codes for different scenarios

✅ Database Schema Design (5-7 min)

👉 Identify the type of database (SQL or NoSQL)
👉 Design schemas like tables/columns and relationships with other tables (SQL)

System Design Interview Template

✅ System's Detailed Design (15 - 20 min)

(a) Draw/Explain high-level components of the system involving the below (if required) components -

👉 Client (Mobile, Browser)
👉 DNS
👉 CDN
👉 Load Balancers
👉 Web / Application Servers
👉 Microservices involved in fulfilling the design
👉 Blob / Object Storage
👉 Proxy/Reverse Proxy
👉 Database (SQL or NoSQL)
👉 Cache at various levels (Client side, CDN, Server side, Database side, Application level caching)
👉 Messaging Queues for asynchronous communication

(b) Identification of algorithm/data structures and a way to scale them

(c) Scaling individual components

👉 Horizontal scaling
👉 Vertical scaling

System Design Interview Template

(d) Database Partitioning

i) Partitioning Methods
👉 Horizontal Partitioning
👉 Vertical Partitioning
👉 Directory-Based Partitioning

ii) Partitioning Criteria
👉 Range-Based Partitioning
👉 Hash-Based Partitioning (Consistent Hashing)
👉 Round Robin

(e) Replication & Redundancy

👉 Redundancy - Primary and Secondary Server
👉 Replication - Data replication from active to mirrored node/database

System Design Interview Template

(f) Databases

👉 SQL - Sharding, Indexes, master-slave, master-master, Denormalization

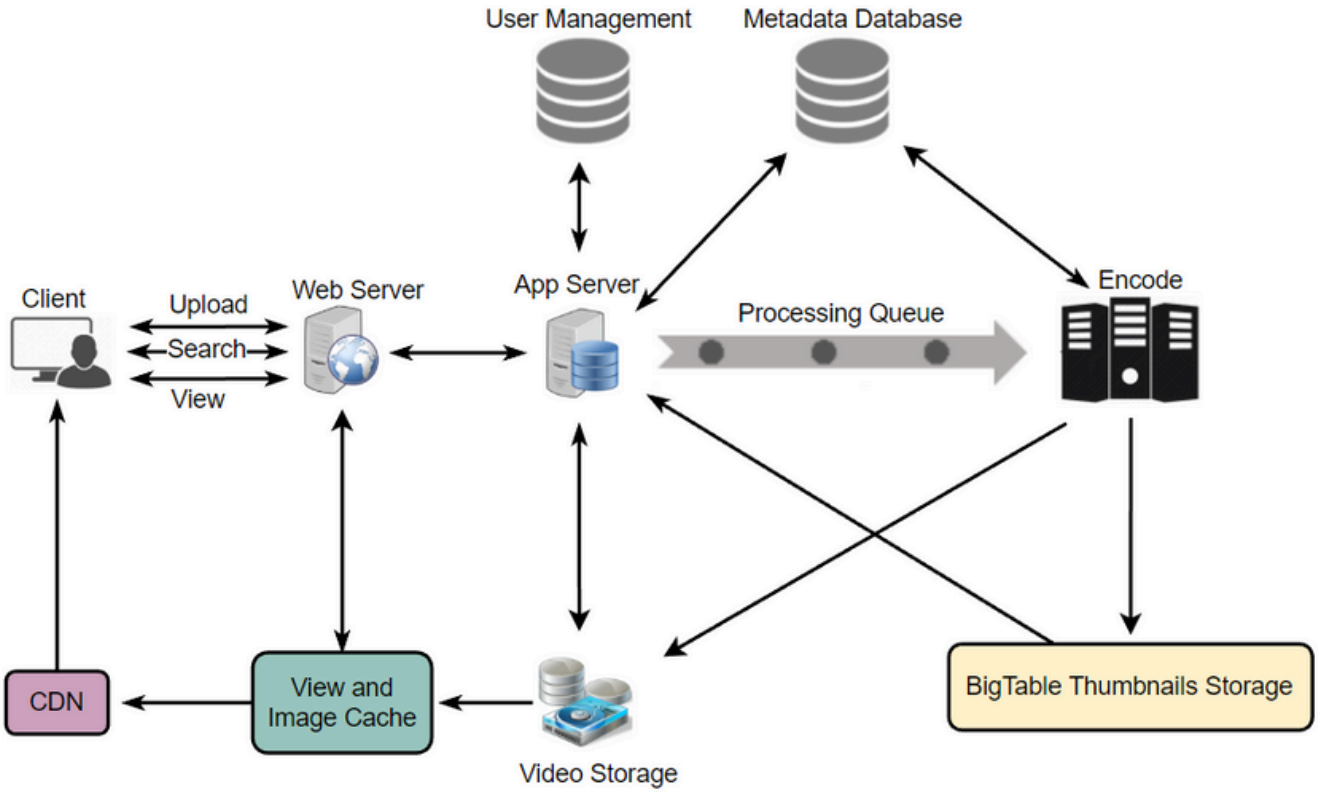
👉 NoSQL - Key-Value, Document, Wide-Column, Graph

(g) Communication Protocols and standards

IP, TCP, UDP, HTTP/S, RPC, REST, Web Sockets

✅ **Resolve bottlenecks and follow-up questions** (2-3 **min**)

Below kind of diagrams are expected in a System Design interviews



Must know Back-of-the-envelope Estimation for a System Design Interview

Back-of-the-envelope estimations

In a system design interview, the chances are high that you will be asked to estimate QPS (Queries per second) and Storage of the system using a back-of-the-envelope estimation.
There are various scales and numbers anyone appearing for System Design Interview should know. Some of the scales are as follows -

Number Scale

Name	Number	Number of Zeroes
1 hundred	100	2 Zeroes
1 thousand (K)	1000	3 Zeroes
1 million (M)	1000000	6 Zeroes
1 billion (B)	1000000000	9 Zeroes
1 trillion (T)	1000000000000	12 Zeroes
1 quadrillion	1000000000000000	15 Zeroes

Back-of-the-envelope estimations

In a system design interview, the volume of huge data is measured on the power of 2. It gets as low as bits and bytes. A byte is measured as 8 bits. Estimations become easier if we co-relate the below table with the number table and make a rough approximation. The interviewer will expect you to know these scales.

Power of Two's Scale

Name	Power	Value
1 KB (Kilobyte)	2^{10}	1024 ~ 1K
1 MB (Megabyte)	2^{20}	1048576 ~ 1M
1 GB (Gigabyte)	2^{30}	1073741824 ~ 1B
1 TB (Terabyte)	2^{40}	1099511627776 ~ 1T
1 PB (Petabyte)	2^{50}	1125899906842624 ~ 1 Quadrillion

Back-of-the-envelope estimations

In a system design interview, the latency numbers play a vital role in estimations and in having the knowledge, like how much time certain components take to perform certain operations. Below are some of the latency numbers of various operations -

Latency Numbers

Operation	Time taken
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns = 10 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs

Back-of-the-envelope estimations

Latency Numbers

Operation	Time taken
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within the same datacenter	500,000 ns = 500 µs
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from the network	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	30,000,000 ns = 30 ms
Send packet CA (California) → Netherlands → CA	150,000,000 ns = 150 ms

Back-of-the-envelope estimations

In a system design interview, the **High Availability** discussion will happen for sure. It is defined as the ability of the system to be operational for a longer period of time. Below are some of the availability numbers you should know -

Availability Numbers

Availability %	Downtime per year	Downtime per month	Downtime per day
90% (one nine)	36.53 days	73.05 hours	2.40 hours
99% (two nines)	3.65 days	7.31 hours	14.40 mins
99.9% (three nines)	8.77 hours	43.83 mins	1.44 mins
99.99% (four nines)	52.60 mins	4.38 mins	8.64 secs
99.999% (five nines)	5.26 mins	26.30 secs	864.00 millisecs
99.9999% (six nines)	31.56 secs	2.63 secs	86.40 millisecs

Back-of-the-envelope estimations

Availability Numbers

Availability %	Downtime per year	Downtime per month	Downtime per day
99.99999% (seven nines)	3.16 secs	262.98 millisecs	8.64 millisecs
99.999999% (eight nines)	315.58 millisecs	26.30 millisecs	864.00 microsecs
99.9999999% (nine nines)	31.56 millisecs	2.63 millisecs	86.40 microsecs

Back-of-the-envelope estimations

In a system design, there are various big systems that involve types of blobs/objects like images, videos, audio, etc. Below are some of the approximate storage sizes of various blobs/objects -

Blob/Object Storage Sizes

Object Type	Size
char	1 B
char (Unicode)	2 B
UUID	16 B
Thumbnails	20-30 KB
Website image	200-300 KB
Mobile image	2-3 MB

Back-of-the-envelope estimations

Blob/Object Storage Sizes

Object Type	Size
Documents like books, reports, govt Ids etc	1-3 MB
Audio files like songs, recordings etc	4-5 MB
1 min 720px video	60 MB
1 min 1080px video	130 MB
1 min 4K video	350 MB

Example - Quora System Design

A System Design Interview usually lasts for 45-60 minutes. The following example "Quora System Design" will guide you on how to approach a system design interview -

✅ Requirement Clarifications - (3-5 min)

Ask clarifying questions to understand the problem and expectations of the interviewer. It is like an MVP which is the initial version of the product being designed. It has enough set of features usable by early customers who can then provide feedback for future product development. Let's see some of the functional and non-functional requirements of Quora.

a) Functional Requirements

👉 Users should be able to post questions.
👉 Users should be able to answer questions.
👉 Users should be able to upvote/downvote questions and answers.
👉 Users should be able to search questions.
👉 Users should be able to see the feed of relevant questions.

Example - Quora System Design

b) Non-Functional Requirements

The design of the system is highly affected by the non-functional requirements of the system.
Availability and **Consistency** are two major factors we need to look at while designing the system.

👉 **Availability** means that every request for data receives a response, even if it may not be the most up-to-date version of the data.

👉 **Consistency** means that all nodes in the system see the same data at the same time. Thus at any instant user is working on the same data (read and write the same data)

Other factors are -

👉 **Eventual Consistency**
👉 **Latency**

Let's discuss each of them in brief to understand its importance while designing complex systems.

Example - Quora System Design

👉 **Eventual Consistency -** In a distributed system Availability and Consistency can't be achieved simultaneously. Thus, many times we prefer Availability over Consistency. This means that system becomes highly available but can produce inconsistent reads and writes. In an eventually consistent system, multiple copies of the same data are stored on different nodes, and the nodes may not be in perfect sync with one another at all times. This means that it's possible for different nodes to have slightly different versions of the same data, but the system will eventually converge on a single, consistent state.

👉 **Latency -** It is the amount of time it takes for a request to be processed and a response to be returned. In a distributed system requests and responses can be handled by many nodes which are connected over a network. Thus, it becomes important to measure the latency of the system because of the network lag involved. It is typically measured in secs and milliseconds. The system should have low latency which means requests should be fulfilled quickly. It improves the performance of the system.

Example - Quora System Design

Quora must have the following non-functional requirements -

👉 It should be Highly Available.
👉 It should have Eventual Consistency.
👉 It should have Low Latency.

Other non-functional requirements can be discussed with the interviewer's scope in mind.

In a later section, we will see how we can achieve non-functional requirements.

Example - Quora System Design

✅ Estimations (3-5 min)

The estimations give a high-level idea about the scale of the system in the future. The scale of the system can be estimated by-

👉 Storage estimates
👉 QPS (Queries Per Second) Read/Write ratio

Let's understand both estimations in greater detail.

a) Storage estimates - Storage estimates can be estimated via the number of users using the system. It can be MAU (monthly active users) or DAU (daily active users). These numbers can be either discussed with the interviewer or you can make an assumption (with an agreement with the interviewer).

- Let's assume Quora has 300 Million MAU.
- Let's assume Quora has 40 Million DAU.
- Let's assume out of 40 Million users daily only .01% post questions on Quora.
  Thus, no. of questions each day = .01% of 40 Million
  = **4000** questions

Example - Quora System Design

👉 **Storage Requirements for Questions**

Questions may have properties/attributes like -

- question_id (unique question identifier)
- question_title (actual question)
- topic_id (questions category like sport, entertainment, etc)
- created_datetime (creation date and time)
- user_id (the user who posted it)

Let's assume that each question takes 1 KB of storage. Then, the space required for storing 4000 questions will be

= 1 KB * 4000
= 4000 KB
= 4 MB

👉 **Storage required to store per day questions = 4 MB**
👉 **Storage required to store per year questions**
= 4 MB * 365
= 1460 MB
= 1.46 GB ~ 1.5 GB
👉 **Storage required to store 10-year questions = 15 GB**

Example - Quora System Design

👉 **Storage Requirements for Answers**

Answers may have properties/attributes like -

- answer_id (unique answer identifier)
- answer_description (actual answer text)
- created_datetime (creation date and time)
- user_id (the user who posted it)

Let's assume that each answer takes 10 KB of storage. It is because the answer text can be a descriptive text.
Assume for each question we have an average of 3 answers.
Then, the space required for storing 3 * 4000 answers will be

= 10 KB * 4000 * 3
= 120000 KB
= 120 MB

👉 **Storage required to store per day answers = 120 MB**
👉 **Storage required to store per year answers**
= 120 MB * 365
= 43800 MB
= 43.8 GB ~ 45 GB
👉 **Storage required to store 10-year answers = 450 GB**

👉 **Total storage required = 15 GB + 450 GB = 465 GB**

Example - Quora System Design

b) QPS estimates - QPS can be defined as queries per second. Its total of reads and writes per second occurs in a system.
Reads are actions performed by users to view some resources such as questions, answers, etc.
Writes are actions performed by users to post some resources such as adding questions, adding answers, etc.

Reads in the system

Let's assume the number of pages visited by each user per day is 10. If we take 40 Million as daily active users then the number of reads per day will be = 10 * 40 Million = 400 Million reads per day.

Writes in the system

👉 Number of questions posted each day = 4000.
👉 Number of answers posted each day = 12000.
👉 Number of writes each day = 16000.

Example - Quora System Design

It can be seen that it's a read-heavy system because reads are more as compared to writes.

👉 Number of reads per day = 400 M
👉 Number of writes per day = 16 K
👉 Total number of queries per day
= 400 M + 16 K ~ 400 M
👉 Total seconds in a day = 24 * 60 * 60
= 86400 ~ 100 K secs
👉 **Queries per secs = 400 M / 100 K ~** 4 K queries per secs

Example - Quora System Design

✅ API Design (3-5 min)

Outline the different APIs for the required scenarios. The APIs should cover all the requirements of the system. These functional requirements are the same as what has been captured in the **Requirement Clarification** step. For Quora, some of the APIs would be -

👉 Post Questions - Add a new question

- Url - /questions
- HTTP Method - POST
- Request Body -

```
{
   "question_id": "27baf31e-7cf5-11ed-a1eb-0242ac120002",
   "question_title": "How to crack GAMAM interviews",
   "topic_id": "10",
   "created_datetime": "16/12/2023T13:45:00.000Z",
   "user_id": "37baf31e-8cf5-11ed-a1eb-0242ac560002"
}
```

- Response Body - Returns a string "Question added successfully" with status code 201. Otherwise returns a failure string with error code 4xx (client-side error) or 5xx (server-side error)

Example - Quora System Design

✅ API Design (3-5 min)

👉 Post Answer - Add a new answer

- Url - /answers
- HTTP Method - POST
- Request Body -

```
{
  "answer_id": "35caf31e-7cf5-11ed-a1eb-0242ac120010",
  "question_id": "27baf31e-7cf5-11ed-a1eb-0242ac120002",
  "answer_description": "Practice DSA and Design daily !!!",
  "created_datetime": "16/12/2023T13:45:00.000Z",
  "user_id": "30baf31e-8cf5-13ed-a1eb-0242ac260019"
}
```

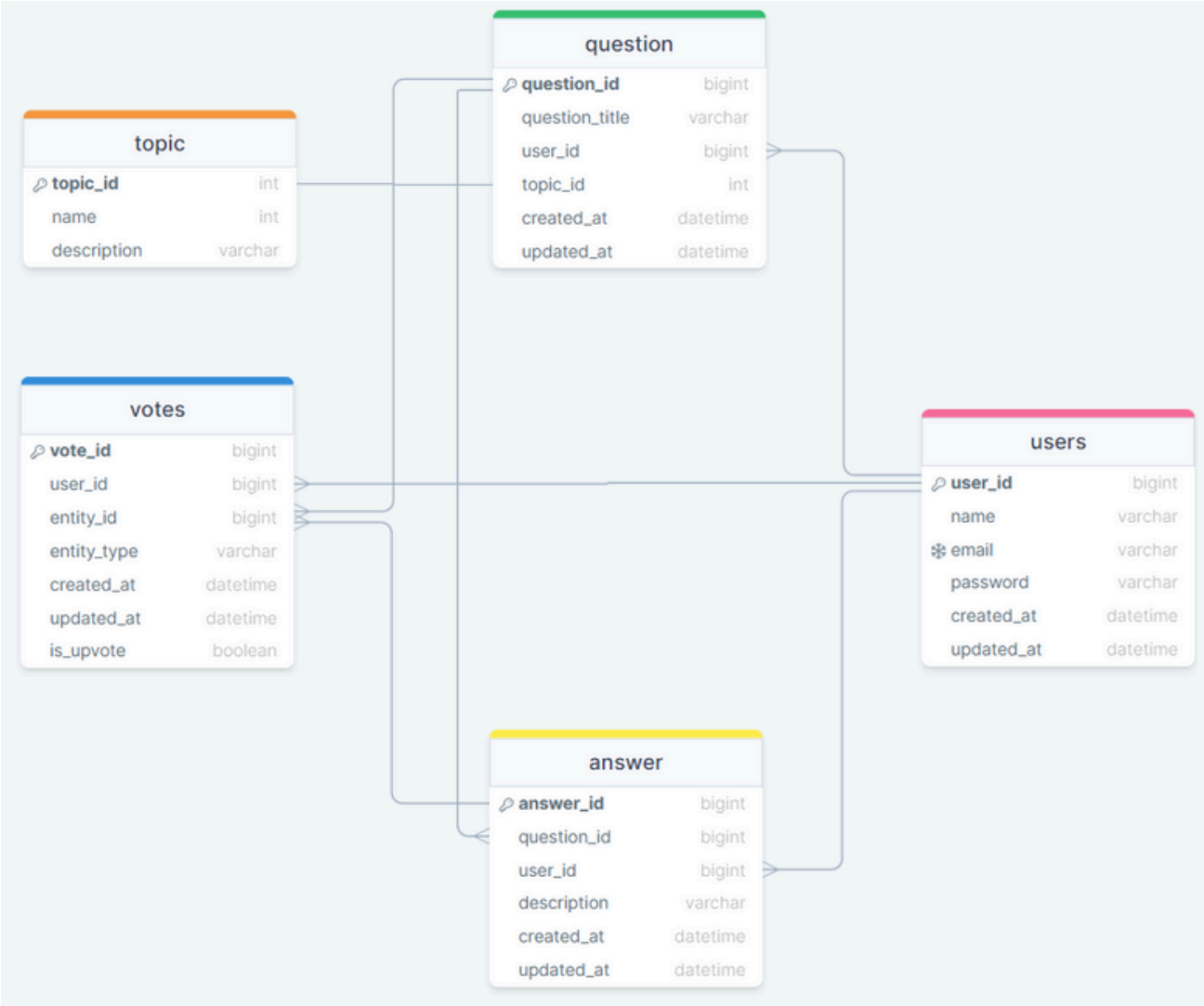
- Response Body - Returns a string "Answer added successfully" with status code 201. Otherwise returns a failure string with error code 4xx (client-side error) or 5xx (server-side error)

and other similar apis ...

Example - Quora System Design

✅ Database Design (3-5 min)

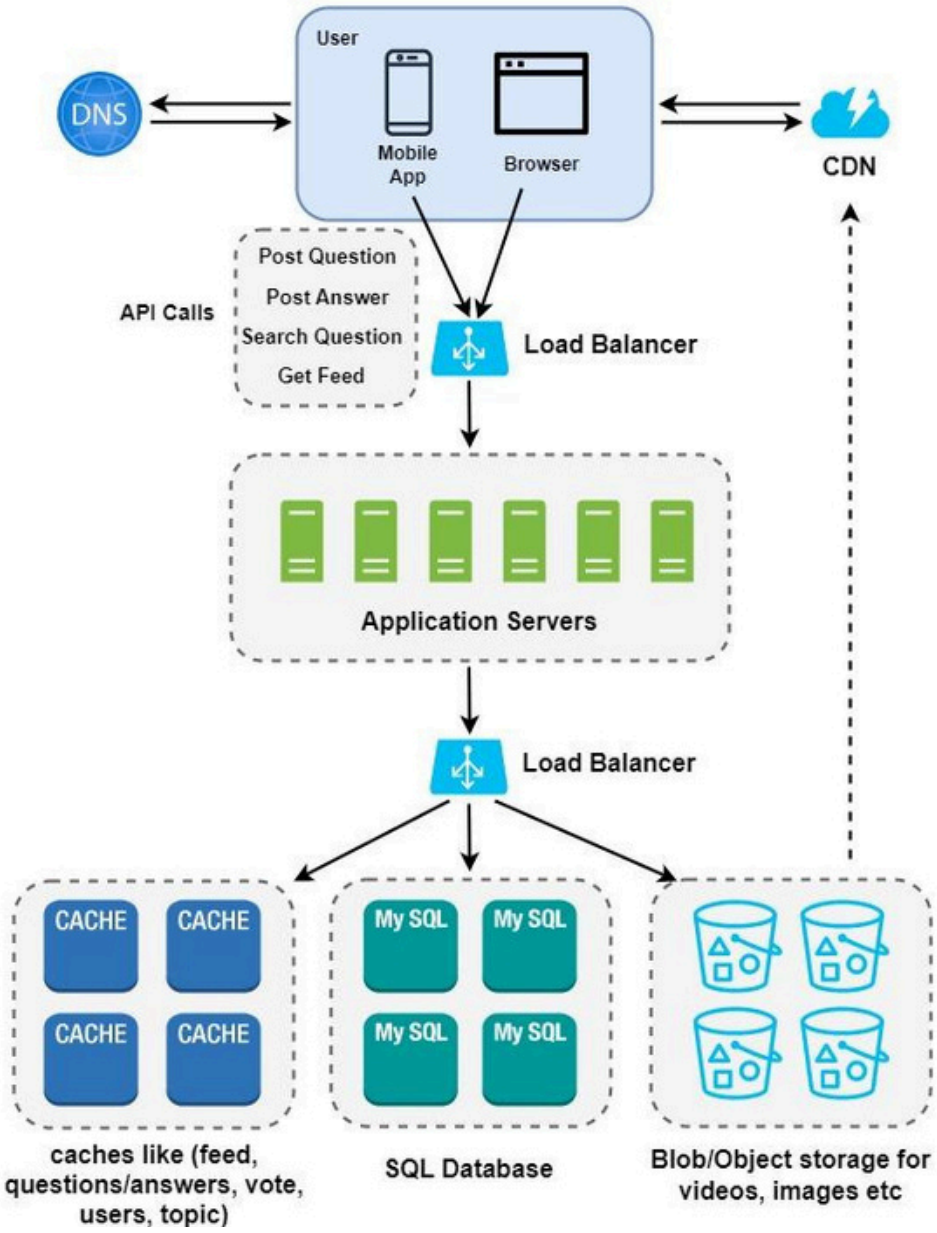
👉 Design schema-like tables/columns and relationships with other tables (SQL). For Quora's MVP below image demonstrate a good starting schema -



Example - Quora System Design

✅ System's Detailed Design (20 - 25 min)

👉 Draw/Explain high-level components of the system -



Golden Rules to answer in a System Design Interview

👉 If we are dealing with a read-heavy system, it's good to consider using a **Cache.**

👉 If we need low latency in the system, it's good to consider using a **Cache & CDN.**

👉 If we are dealing with a write-heavy system, it's good to use a **Message Queue** for Async processing.

👉 If we need a system to be an ACID complaint, we should go for **RDBMS or SQL Database**.

👉 If data is unstructured & doesn't require ACID properties, we should go for **NoSQL Database**.

👉 If the system has complex data in the form of videos, images, files, etc, we should go for **Blob/Object storage**.

👉 If the system requires complex pre-computation like a news feed, we should use a **Message Queue & Cache**.

👉 If the system requires searching data in high volume, we should consider using a **search index, tries, or a search engine like Elasticsearch**.

👉 If the system requires to Scale SQL Database, we should consider using **Database Sharding**.

👉 If the system requires High Availability, Performance, & Throughput, we should consider using a **Load Balancer**.

👉 If the system requires faster data delivery globally, reliability, high availability, & performance, we should consider using a **Content Delivery Network (CDN)**.

👉 If the system has data with nodes, edges, and relationships like friend lists, & road connections, we should consider using a **Graph Database**.

👉 If the system needs scaling of various components like servers, databases, etc, we should consider using **Horizontal Scaling**.

👉 If the system requires high-performing database queries, we should use **Database Indexes**.

👉 If the system requires bulk job processing, we should consider using **Batch Processing & Message Queues**.

👉 If the system requires reducing server load and to prevent DOS attacks, we should use a **Rate Limiter**.

👉 If the system has microservices, we should consider using an **API Gateway** (Authentication, SSL Termination, Routing etc)

👉 If the system has a single point of failure, we should implement **Redundancy** in that component.

👉 If the system needs to be fault-tolerant, & durable, we should implement **Data Replication** (creating multiple copies of data on different servers).

👉 If the system needs user-to-user communication (bi directional) in a fast way, we should use **Websockets**.

👉 If the system needs the ability to detect failures in a distributed system, we should implement a **Heartbeat**.

👉 If the system needs to ensure data integrity, we should use the **Checksum** algorithm.

👉 If asked how to limit the huge amount of data for a network request like youtube search, trending videos etc. One way is to implement **Pagination** which limits response data.

👉 If the system needs to transfer data between various servers in a decentralized way, we should go for the **Gossip Protocol**.

👉 If the system needs to scale servers with add/removal of nodes efficiently, with no hotspots, we should implement **Consistent Hashing.**

👉 If the system needs anything to deal with a location like maps, nearby resources, we should consider using **Quadtree, Geohash, etc**.

👉 Avoid using any specific technology names such as - **Kafka**, **S3**, or **EC2**. Try to use more generic names like message queues, object storage etc.

👉 If **High Availability (HA)** is required in the system, it's better to mention that system cannot have Strong Consistency. However, Eventual Consistency is possible.

👉 If asked how domain name query in the browser works and resolves IP addresses. Try to sketch or mention **DNS** (**Domain Name System**).

👉 If asked which policy you would use to evict a Cache. The preferred/asked Cache eviction policy is **LRU** (**Least Recently Used**) Cache. Prepare around its Data Structure and Implementation.

03

Object Oriented / Low Level Design Interview

Preparation resources

👉 Elevate your design game with real-world examples! Check out the Design Patterns in Action course.

👉 Grokking the Object Oriented Design Interview - A very detailed and step-by-step approach to various object oriented design case studies.

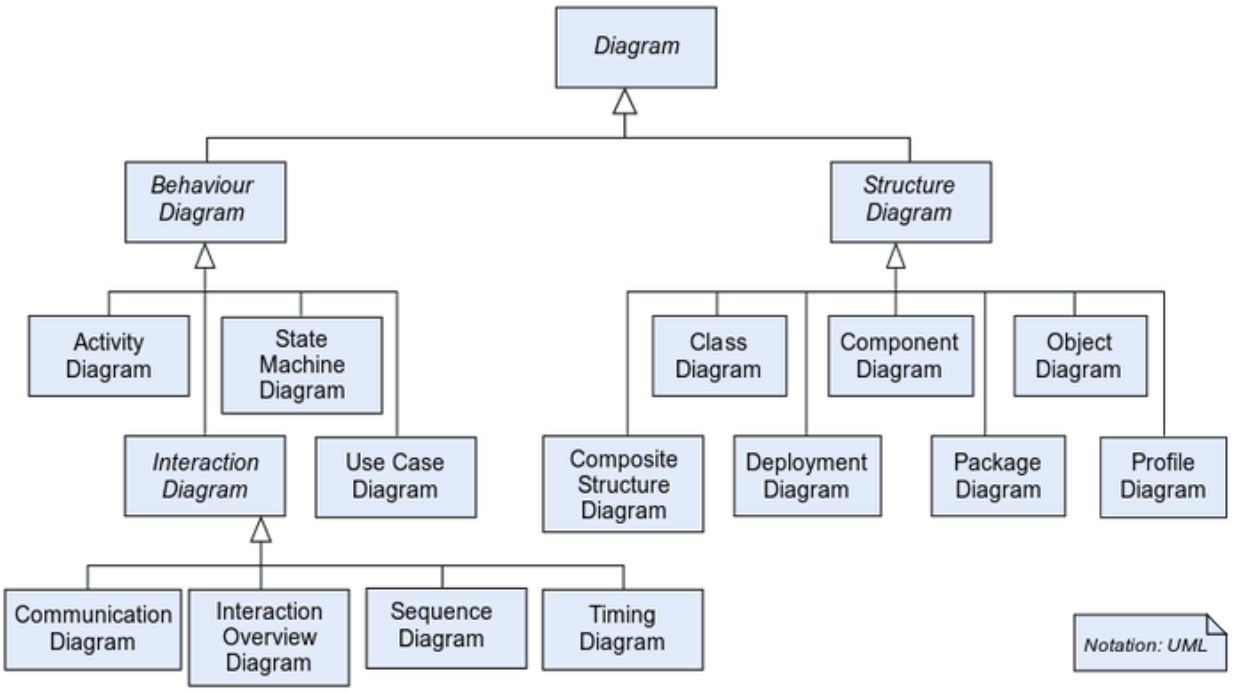
👉 Try practicing drawing UML diagrams like - Class diagrams, Use case diagrams and Activity diagrams. Try to use pen and paper or practice on Diagrams.net

UML and Class Diagrams in a Object-Oriented Design

UML (Unified Modeling Language)

The Unified Modeling Language (UML) is a general-purpose, developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. It helps to visually represent the architecture, design, and implementation of a system.

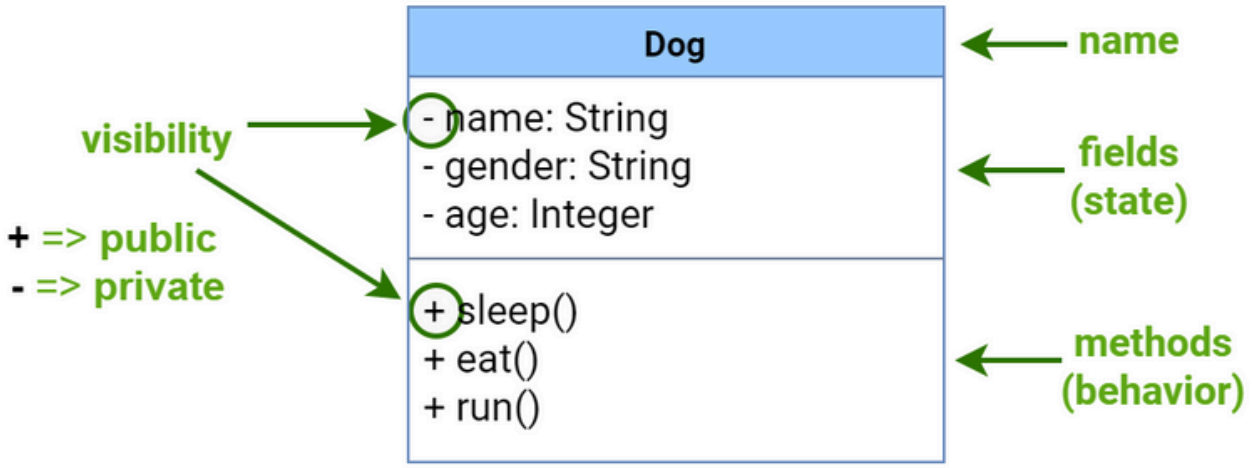
The various types of important UML diagrams are -



The most important UML diagram for Object-Oriented Design interviews is **Class Diagram**. Let's look at and discuss the diagram in more detail.

Class Diagram

The Class Diagram in the Unified Modeling Language (UML) is the most useful and important diagram that is being focused on in a low-level design interview. It is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.



In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.

Class Diagram

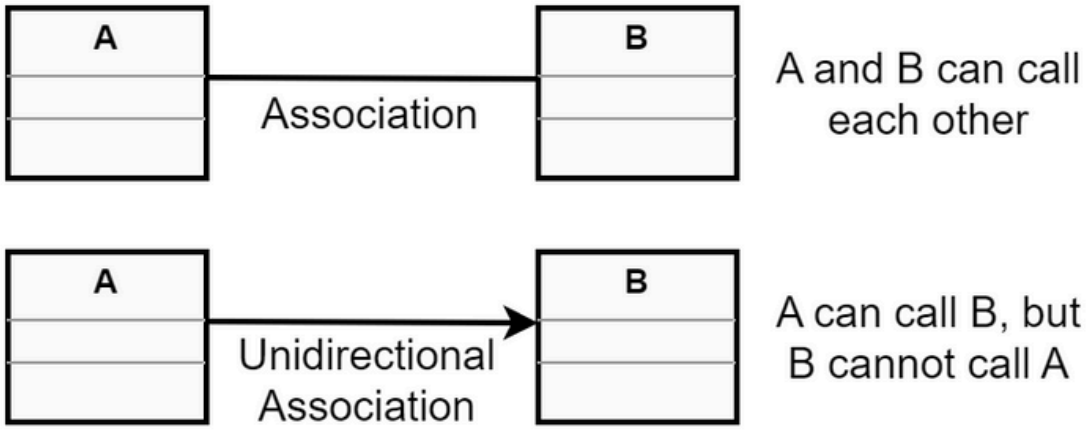
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

Relationships

The Class Diagram shows the classes, their attributes, methods, and the relationships between different classes. Several types of relationships can be represented in class diagrams, each indicating a different type of association between classes. Here are some of the most common relationships -

Association

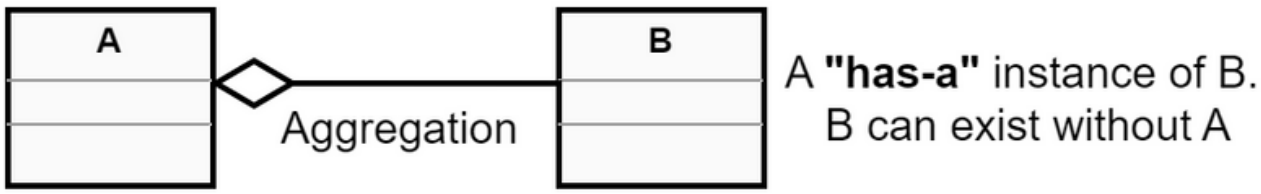
An association represents a relationship between two classes, where objects of one class are related to objects of the other class in some way. The relationship can be bi-directional or unidirectional. In a bi-directional relationship, both classes are aware of each other and their relationship. In a unidirectional relationship, two classes are related, but only one knows about the other and can call it.



For example, consider the classes "Car" and "Driver". There is an association between these classes because a car is driven by a driver. The association can be represented with an arrow pointing from the car class to the driver class, indicating the direction of the relationship.

Aggregation

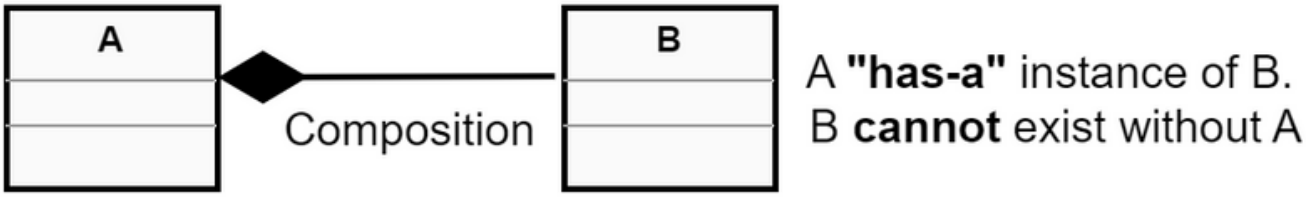
Aggregation is a special type of association that represents a "has-a" relationship between classes. It indicates that one class as a "whole" contains objects of another class as a "part" of its state. The "part" class can exist independently without the "whole" class. It is represented by the diamond symbol.



For example, consider the classes "University" and "Department". A university has several departments, and a department belongs to one university. This can be represented using an aggregation relationship, with a diamond symbol on the side of the university class pointing to the department class.

Composition

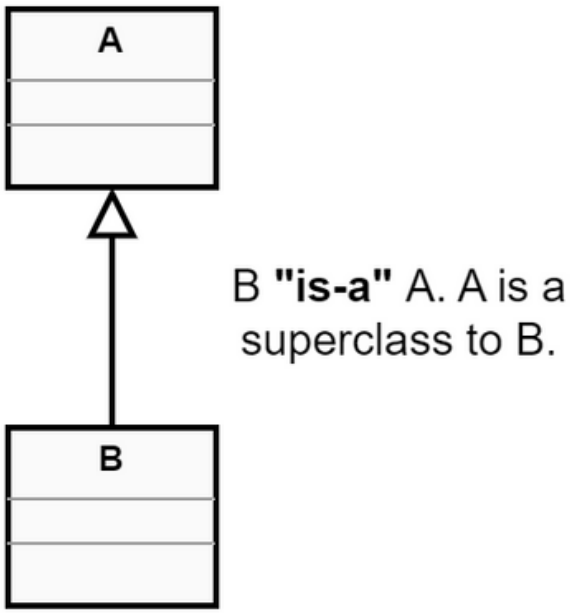
It is similar to aggregation, but it represents a stronger "whole-part" relationship between classes. It indicates that one class is made up of one or more objects of another class, and those objects cannot exist without the parent class.



For example, consider the classes "House" and "Room". A house is composed of one or more rooms, and a room cannot exist without a house. This can be represented using a filled diamond symbol on the side of the house class pointing to the room class.

Inheritance

Inheritance represents an "is-a" relationship between classes, where one class inherits the attributes and methods of another class. The inherited class is called the superclass or parent class, and the inheriting class is called the subclass or child class.

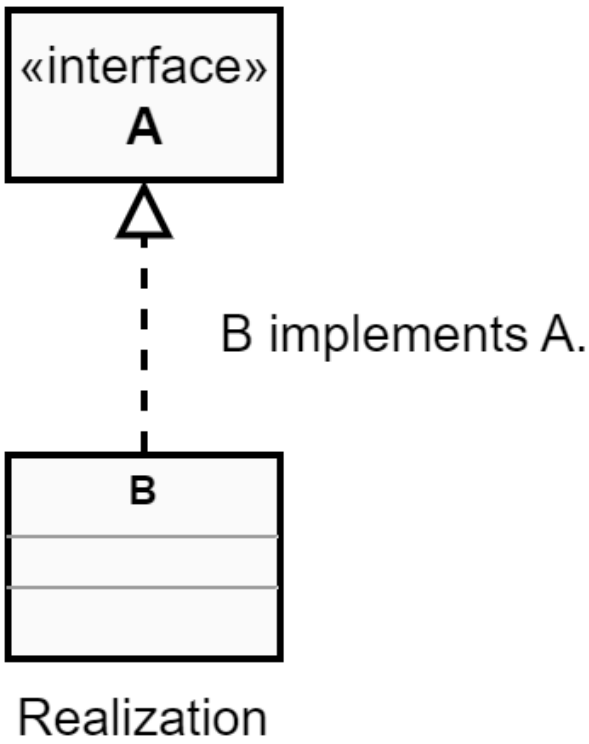


Inheritance

For example, consider the classes "Animal" and "Cat". A cat is an animal, so the class "Cat" can inherit from the class "Animal". This can be represented using an arrow pointing from the subclass to the superclass, with a hollow triangle symbol on the side of the superclass.

Realization

A realization relationship is a relationship between two entities, in which one class implements the behavior of the other class or interface. It is represented as a hollow triangle shape on the interface end of the dashed line that connects it to one or more implementers.

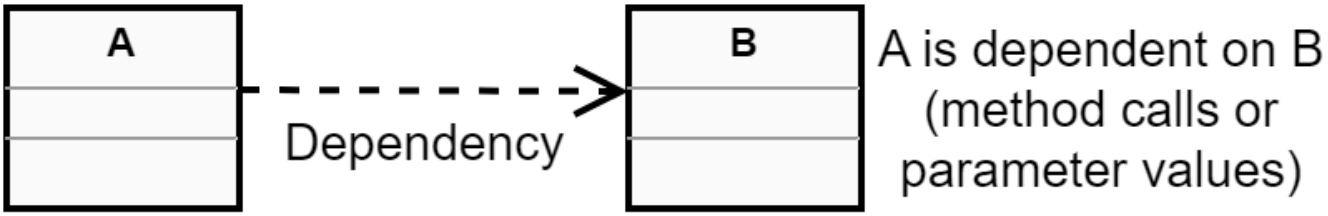


Realization

For example, consider an interface "Shape" which has the draw() method. The classes such as "Rectangle" and "Circle" implements draw() method of Shape and provide their implementation like how a Rectangle or Circle can be drawn.

Dependency

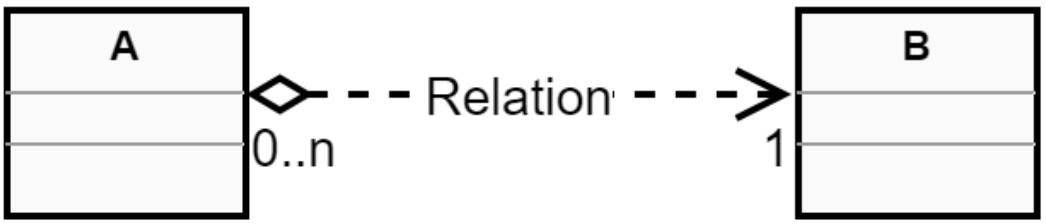
Dependency represents a weaker relationship between classes, where one class depends on another class in some way. This can be because the dependent class uses objects or methods of the other class, or because it receives parameters or returns values from the other class.



For example, consider the classes "BillingSystem" and "Customer". The billing system depends on the customer class to get information about customers' orders, billing addresses, etc. This can be represented with a dashed arrow pointing from the billing system class to the customer class.

Multiplicity

Multiplicity refers to the number of instances of one class that can be associated with the instances of another class. It specifies the range of valid cardinalities for a given association between classes. It is represented by two integers separated which tells a range. The first integer represents the minimum number of instances of the related class that must be associated with an instance of the source class, while the second integer represents the maximum number of instances of the related class that can be associated with an instance of the source class. It can be applied to all UML relationships.



For example, consider a relationship between the classes "Order" and "Item", where an order can have one or more items, but an item can only be associated with one order. It is represented as 1..* (One to Many).

Multiplicity

Other common multiplicity values include:

0..1 - Indicates that zero or one instance of the related class can be associated with an instance of the source class.
1..1 - Indicates that exactly one instance of the related class must be associated with an instance of the source class.
0..* - Indicates that zero or many instances of the related class can be associated with an instance of the source class.

Time Management in Object Oriented Design Interview

- ✅ Requirement Gathering (3-5 mins)
- ✅ Use Cases (3-6 mins)
- ✅ Identify the Core classes (3-6 mins)
- ✅ Identify the fields of each class (5-10 mins)
- ✅ Identify the Relationship between the classes (5-10 mins)
- ✅ Identify the Actions of the classes (5-10 mins)
- ✅ Code (5-8 mins)
- ✅ Follow-up questions (3-4 mins)

Object Oriented Design Interview Template

An Object Oriented Design Interview usually lasts for 45-60 minutes. The following template will guide you (with a basic example to get you an idea) on how to manage time duration across various aspects of it -

✅ Requirement Clarifications (3-5 mins)

The Object Oriented Design interview questions are often abstract and vague. Always ask clarifying questions to understand the problem and find the exact scope of the system that the interviewer has in mind. Avoid directly jumping into designing the system, as it's often considered a red flag if you clear the ambiguities later. Involve the interviewer as much as you can in the design.

👉 Focused use cases to cover (MVP).
👉 Use cases that will not be covered.
👉 Who will use the system (Actors)?
👉 How the system will be used?
👉 Avoid bringing external systems to design

Object Oriented Design Interview Template

e.g. Let's **design an online shopping site**. A few requirements would be -

👉 Users should be able to search for products based on their names.
👉 Users should be able to view/buy products.
👉 Users should be able to add/remove product items in their shopping cart.
👉 Users can place an order.
👉 Users should get notifications about orders.
👉 Users should be able to pay through different modes.

Object Oriented Design Interview Template

✅ Use cases to cover (3-6 mins)

You are expected to find possible actors of the system and write down different use cases the system will support.

👉 Actors could be -
1. Customer
2. Admin
3. System

Some of the use cases could be -

👉 Search products based on the name.
👉 add/remove/modify products in the shopping cart.
👉 Check out to buy items in the shopping cart.
👉 Make payment to place an order
👉 Send a notification to the user about the order.

Object Oriented Design Interview Template

✅ Identify the Core classes (3-6 mins)

After gathering requirements and drafting a few use cases, our understanding of what we are designing becomes clear. Now we should consider what could be the main classes of the system. You are expected to sketch a class diagram or write down class names.

The way to identify classes or entities is -

👉 Nouns in the requirements are possible candidates for Classes.

Some of the core classes could be -

👉 Product
👉 Item
👉 User
👉 ShoppingCart
👉 Order
👉 Payment
👉 Notification

Object Oriented Design Interview Template

✅ Identify the fields/properties of each class (5-10 mins)

Once we know the core classes of the system, it is expected to draw a class diagram along with class fields. Take each class identified in the above steps and add a few important properties which drive the use cases of the system. eg.

👉 **Product**
- name
- description
- price ...

👉 **User**
- name
- email
- phone ...

Object Oriented Design Interview Template

✅ Identify the relationship between the classes (5-10 mins)

Once we know the core classes/objects of the system, it is expected to draw what is the relationship between the classes. The relationship mainly focuses on is-a and has-a between classes. The different types of relationships to draw or write are -

👉 Are there any classes that are very generic and more concrete classes can be sketched from it?
👉 Are there any one-to-one, one-to-many, and many-to many relationships between the classes. eg.

- 👉 Customer, Guest, and Admin inherit from User
- 👉 The customer has One Shopping Cart
- 👉 Shopping Cart has Many Items

etc ...

Object Oriented Design Interview Template

✅ Identify the possible actions of the classes (5-10 mins)

Once we are clear with the requirements, use cases and possible design of the system, etc, it is time to identify the different actions classes will perform based on their relationship.

The way to identify class actions is -

👉 Verbs in the requirements/use cases are possible candidates for actions these classes perform. Those can be taken as methods of the classes. eg.

👉 Customers can add the item to the shopping cart - **addItemToCart(Item item)**
👉 Customer can place an order - **placeOrder(Order order)**

etc...

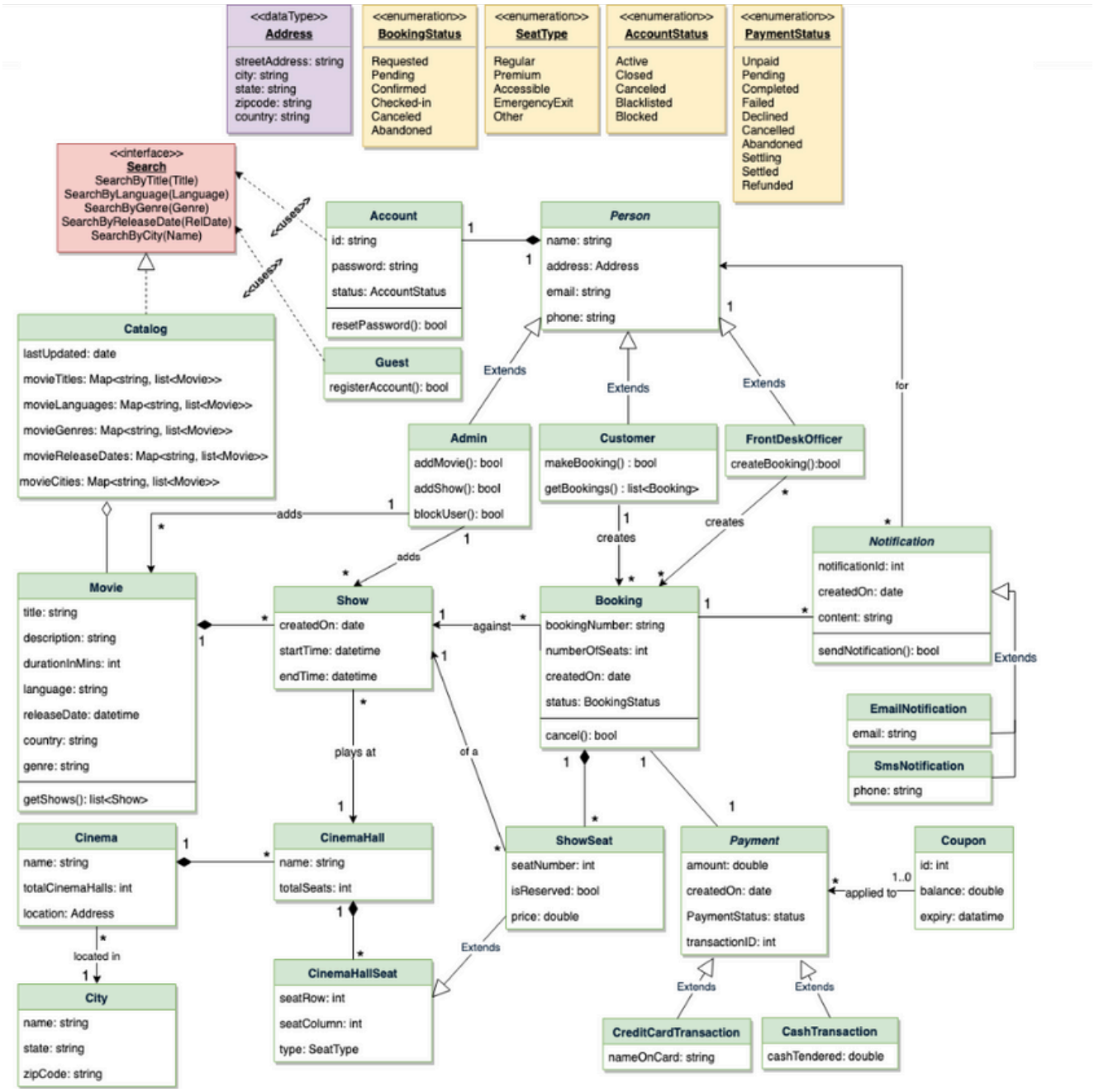
Object Oriented Design Interview Template

✅ Code (5-8 mins) (Optional)

The interviewer will ask you to write code for a specific use case by taking the above classes. The class diagram will give you an idea about the class's name, fields, and methods. You are expected to write code for the methods which fulfill the use case interviewer wants or any algorithm/data structure which handles certain use cases.

✅ Resolve bottlenecks and follow-up questions (3-4 mins)

Below kind of diagrams are expected in OOD interviews



04

API Design Interview

Preparation resources

👉 Best Practices, Implementation, and Guidelines to follow for API Design.

👉 Look for use cases like - Stripe and Twitter API Documentation.

👉 SystemsExpert also has a few case studies on API design as well.

👉 Follow the link to understand how APIs should be designed.

05

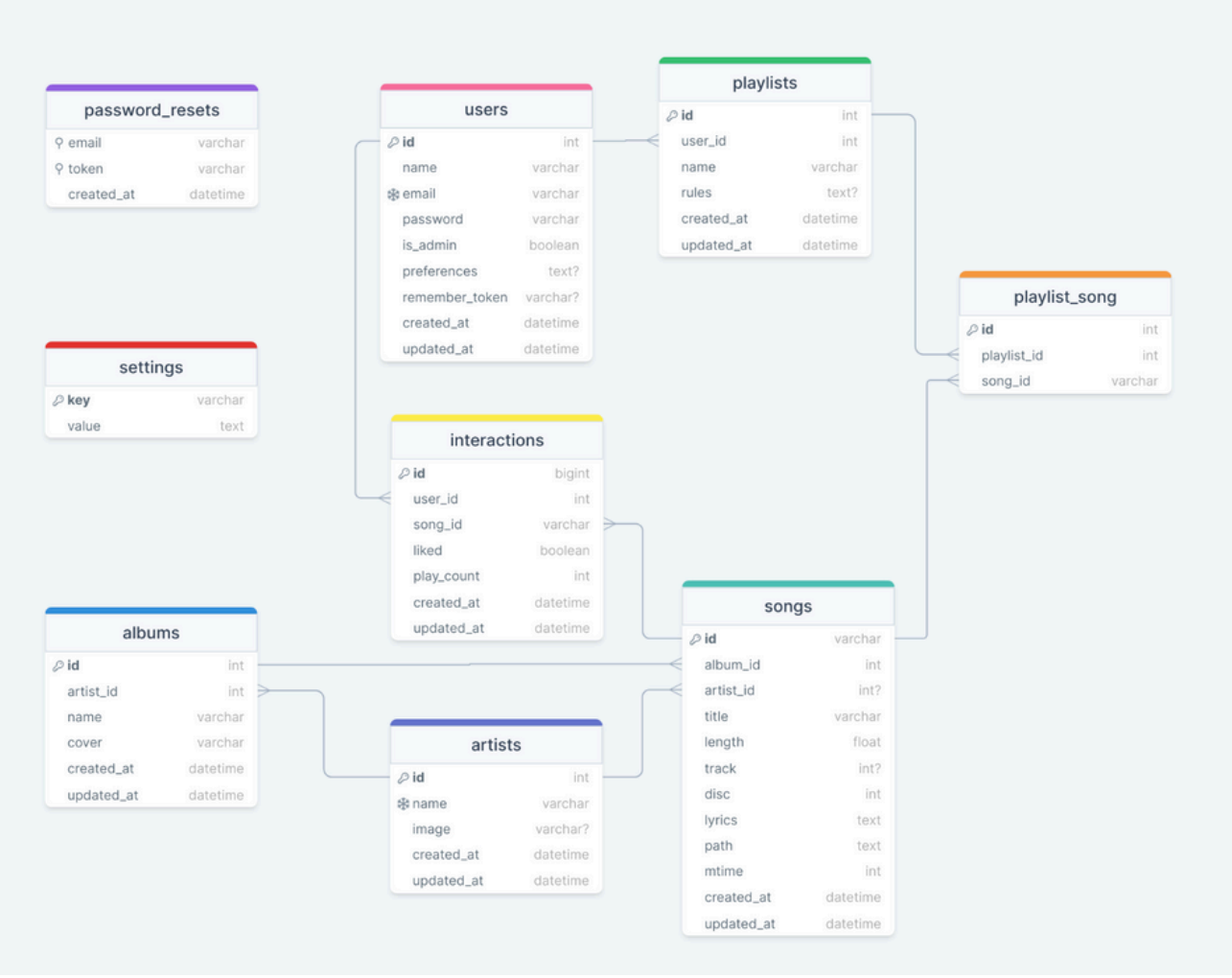
Schema / Database Design Interview

Preparation resources

👉 Grokking the Object Oriented Design Interview - Take the case studies and try to apply Objects to the Relational Mapping strategy.

👉 Try practicing drawing relational schema diagrams like - tables, columns, and relationships between tables. Try to use pen and paper or practice on drawSQL

Below kind of diagrams are expected in Schema Design interviews



06

Behavioral Interview

Introduction

Behavioral Interviews are taken to evaluate a candidate's past behavior or conduct in certain different situations to analyze/predict their future behavior towards the future employer.

Behavioral Interviews - Myths

👉 **Are Simple Interviews** - Many candidates treat Behavioral Interviews are easy and simple to crack. The reality is these interviews are often complex and very dense if not prepared well.

👉 **Are just formality/no rejection interviews** - Many candidates believe that Behavioral Interviews are just for formality. The reality is these interviews can generate red flags based on your answers and can reject your whole candidature even if other interviews have gone well.

👉 **Can be prepared in a day** - It is believed that Behavioral Interviews can be prepared a day before the actual interview. The reality is even though we know what questions can be asked, the complexity and rejection often come from the follow-up questions. Thus, sufficient time needs to be invested in its preparation.

Preparation resources

👉 Crush behavioral interviews with Mastering Behavioral Interviews course!

👉 Watch CareerVidz YouTube channel for behavioral interview questions.

👉 Watch Jeff H Sipe's YouTube channel for behavioral questions.

👉 Check out a list of good Behavior Interview questions.

👉 STAR Pattern - Situation, Task, Action, Result

👉 Prepare questions and career stories around Amazon's leadership principles. It will cover every aspect of Behavior interviews.

👉 Apply the STAR pattern to write experience stories around various questions. Do not prepare for this kind of interview a day before. Keep the stories handy for all such interviews.

Tips for Behavioral Interviews

👉 There is no right or wrong answer. Try to focus on better and positive ways to answer them.

👉 Apply the **STAR** pattern to write experience stories around the above behavioral questions.

👉 Keep the stories handy for all such interviews. Write the question and describe stories following the **STAR** pattern.

👉 If you don't have any such experience stories, try to come up with stories about your past life like school, colleague, events, etc

👉 If you don't have any such experience stories, try to come up with stories about your seniors, and colleagues and see what they have done when faced with such situations.

👉 If you don't have any such experience for a question, you should tell me that I don't have any such experience but I can tell you how would I will react in such situation.

Tips for Behavioral Interviews

👉 Avoid being disrespectful, arrogant, and confrontational to any person or company, you have worked with.

👉 Don't lie. Whatever you speak be ready with proof and explain things that look genuine.

👉 Avoid useless details, make stories/answer short, and to the point.

👉 Always highlight your good qualities in every sentence/paragraph of your answers.

👉 Do not prepare for this kind of interview a day before.

STAR Method to answer in Behavioral Interviews

STAR Method

The **STAR** method is a structured manner of responding to a behavioral interview question. The **STAR** method refers to **S**ituation, **T**ask, **A**ction, and **R**esult. Let's look at it in more detail -

👉 **Situation** - Describe the scene and provide the details about the situation or task you accomplished. The situation can be from past jobs, experience, any event, or from personal life. The interviewer must understand the exact situation and its relevance to the question.

👉 **Task** - Describe your responsibility in that situation or the goal you worked toward. It's like a list of steps that could help you accomplish the goal or situation.

👉 **Action** - Describe the actions you took to complete each and every task. These actions should describe how much contribution you made towards accomplishing the situation.

👉 **Result** - Share the outcomes of the tasks you accomplished by your actions. These results should demonstrate the impact of your actions and describe the lesson learned from them.

Story Preparation

Take Amazon's leadership principles and prepare 1-2 experience stories around it using the **STAR** method. It will cover most of your Behavioral Interview Questions. Let's take an example of one important principle - "**Customer Obsession**".

👉 **Situation** - I had just joined XYZ company and was assigned to a project that was already in the development phase for 6 months. The UI presented a screen to the Client for uploading documents. The UI showed all uploaded documents on UI, along with their metadata. Initially, it worked fine due to the limited no. of documents. As the documents increased in the system, this started making UI slower and slower. Soon, the problem became evident to the client and the team was asked about probable solutions.

👉 **Task** - Looking at the problem I created a few tasks that can tackle the problem efficiently such as,

1. Improving the performance of UI. (Rendering of the grid)
2. Improving the performance of backend logic.
3. Optimizing the queries that get us the data.
4. Handling and storage of the document blob efficiently (upload/download/store)
5. Improving the search, sorting, and filters on UI.

Story Preparation

👉 **Action** - Taking up tasks step by step, we implemented various improvements such as, -

1. Implemented Client Side Pagination. Instead, of rendering all the documents on UI, we paginated it to show a max of 10-20 records. This made the rendering of the UI faster.
2. Implemented Server Side Pagination to make queries faster to fetch limited data.
3. Implemented Document blob fetching/preview on user action, instead of fetching along with document metadata. The idea was to show only document metadata and interested document preview was done via download API.
4. Implemented filters, search, and sorting along with paginated queries i.e. removed all the UI processing to the backend.
5. Initially, the Document blob was stored in a database. It resulted in a slowness of queries. Later, we migrated it to a blob storage solution.

Story Preparation

👉 **Result** - After implementing the above actions step by step, it resulted in a drastic improvement in the performance of the system. The queries which took almost 1 minute to fetch the data of X no. of documents, now took nearly 1 sec. The clients were happy and it resulted in trust and new initiatives for the future.

07

Resume Building

Introduction to Resume Building

Building a resume is a vital step in the journey towards securing meaningful employment. It serves as a personal marketing tool, showcasing your skills, experiences, and qualifications to potential employers.

Crafting an effective resume requires attention to detail, strategic thinking, and a clear understanding of what employers are looking for. Here's an introduction to the art of resume building, encompassing key elements and best practices.

Understanding the Purpose

The primary purpose of a resume is to land you an interview. It's your opportunity to make a strong first impression and convince hiring managers that you're the ideal candidate for the job. A well-crafted resume should highlight your relevant skills, experiences, and accomplishments, tailored to the specific job you're applying for.

Structural Components

A typical resume comprises several essential sections:

👉 **Header** - Your name and contact information should be prominently displayed at the top of the page. Include your phone number, email address, and LinkedIn profile (if applicable).

👉 **Professional Summary/Objective -** This brief section provides a snapshot of your qualifications and career goals. A professional summary is ideal for an experienced professionals, while an objective is more suitable for entry-level candidates.

👉 **Skills -** List relevant skills that are directly applicable to the job you're seeking. Include both hard skills (technical abilities) and soft skills (communication, leadership, teamwork, etc.).

👉 **Work Experience -** Detail your relevant work history in reverse chronological order, starting with your most recent job. Include the company name, your job title, and dates of employment. Describe your responsibilities and achievements using action verbs and quantifiable results.

Structural Components

👉 **Education** - Provide information about your educational background, including degrees earned, institutions attended, and graduation dates. You can also mention relevant coursework, academic honors, or certifications.

👉 **Additional Sections (Optional) -** Depending on your background, you may include additional sections such as:

a). **Projects** - Highlight significant projects you've completed, especially if they demonstrate relevant skills or accomplishments.

b). **Volunteer Work -** Showcase any volunteer experiences that demonstrate your commitment to your community or relevant skills.

c). **Languages -** If you're proficient in multiple languages, it can be advantageous to list them.

👉 **Formatting and Design -** Keep your resume clean, professional, and easy to read. Use a clear and legible font (such as Arial or Calibri) with consistent formatting throughout. Organize information logically, using bullet points to emphasize key details and make your resume visually appealing. Be mindful of whitespace to avoid clutter and ensure readability.

Structural Components

👉 **Tailoring Your Resume** - Customize your resume for each job application to highlight the most relevant qualifications and experiences. Carefully review the job description and incorporate keywords and phrases that match your skills and background. This tailored approach demonstrates your genuine interest in the position and increases your chances of getting noticed by hiring managers.

👉 **Proofreading and Review -** Before submitting your resume, thoroughly proofread it to ensure accuracy and clarity. Look for spelling and grammatical errors, and consider seeking feedback from trusted peers or mentors. A fresh perspective can help identify areas for improvement and ensure your resume is polished and professional.

Preparation resources

👉 Land Your Dream Job: A Comprehensive Guide to Resume Building

Resume Building Tips

👉 Provide a valid resume name for your file e.g. - Dinesh_Varyani_Resume.pdf. Avoid names such as MyResume.pdf, Resume.docx, DineshVaryani.docx, etc

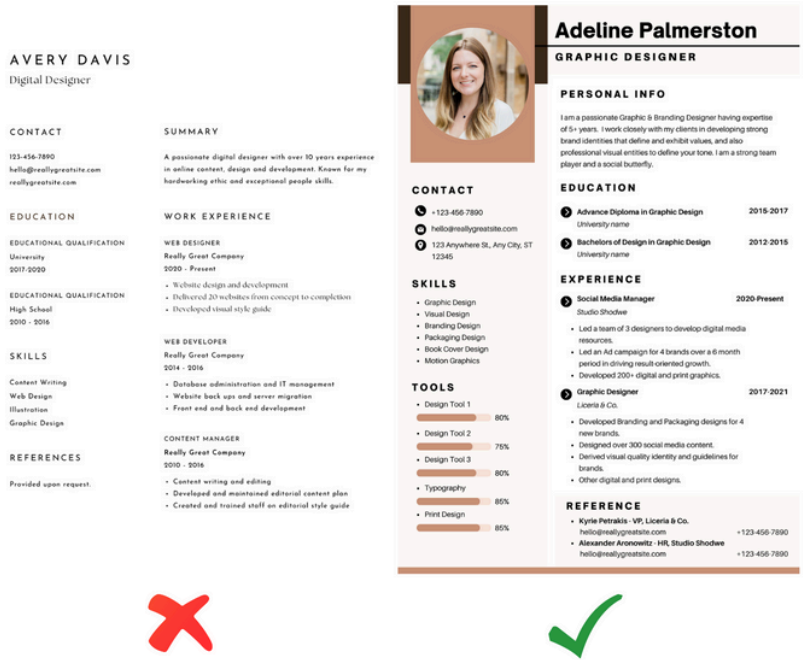
Dinesh_Varyani_Resume

Resume

👉 Keeping a One-page resume so that it's easy for the recruiter to go through it in 5-10 secs is not always correct. If you have done great things/amazing projects in your professional career, you can showcase it in a 2-3 page resume.

Resume Building Tips

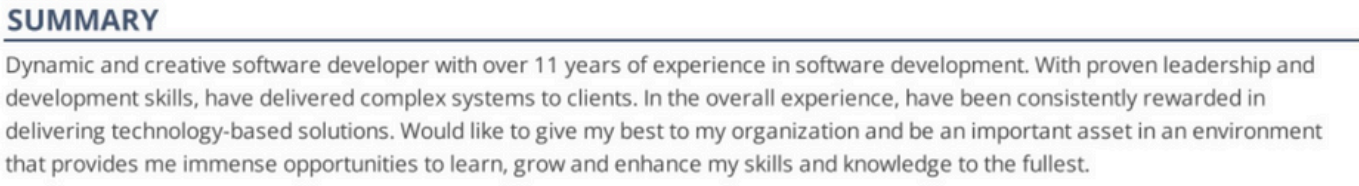
👉 Keep resume in multi-color format. Using single color will make important parts of your resume look the same. e.g. links/emails become hard to recognize from normal text.



👉 Showcase your skills/technologies in one place rather than scattered with every project you have worked on.

Resume Building Tips

👉 The resume header is the first thing the recruiter or manager will check out. Provide a good introductory summary covering your passion and ambitions at the top.



SUMMARY

Dynamic and creative software developer with over 11 years of experience in software development. With proven leadership and development skills, have delivered complex systems to clients. In the overall experience, have been consistently rewarded in delivering technology-based solutions. Would like to give my best to my organization and be an important asset in an environment that provides me immense opportunities to learn, grow and enhance my skills and knowledge to the fullest.

👉 Keep the most important things in the resume first, like the introduction summary, technical skills, projects, achievements, certifications, education, etc. These top sections will be the first thing recruiters will check out. Things like hobbies, references, and contact info can come at the end.

👉 Day-to-day work is ok and being done by everyone. But, an actual resume looks good when you showcase contributions outside work in a Company. Always highlight the things you do apart from normal work like Training, Events, Hackathons, Team bonding activities KT, Sessions, etc

Resume Building Tips

👉 Provide project impact via numbers and percentages rather than a normal description like -

❌ Implemented the project with XYZ feature that helped in the productivity of the users.

✅ Implemented the project with XYZ feature that increased the productivity of the users by 30%.

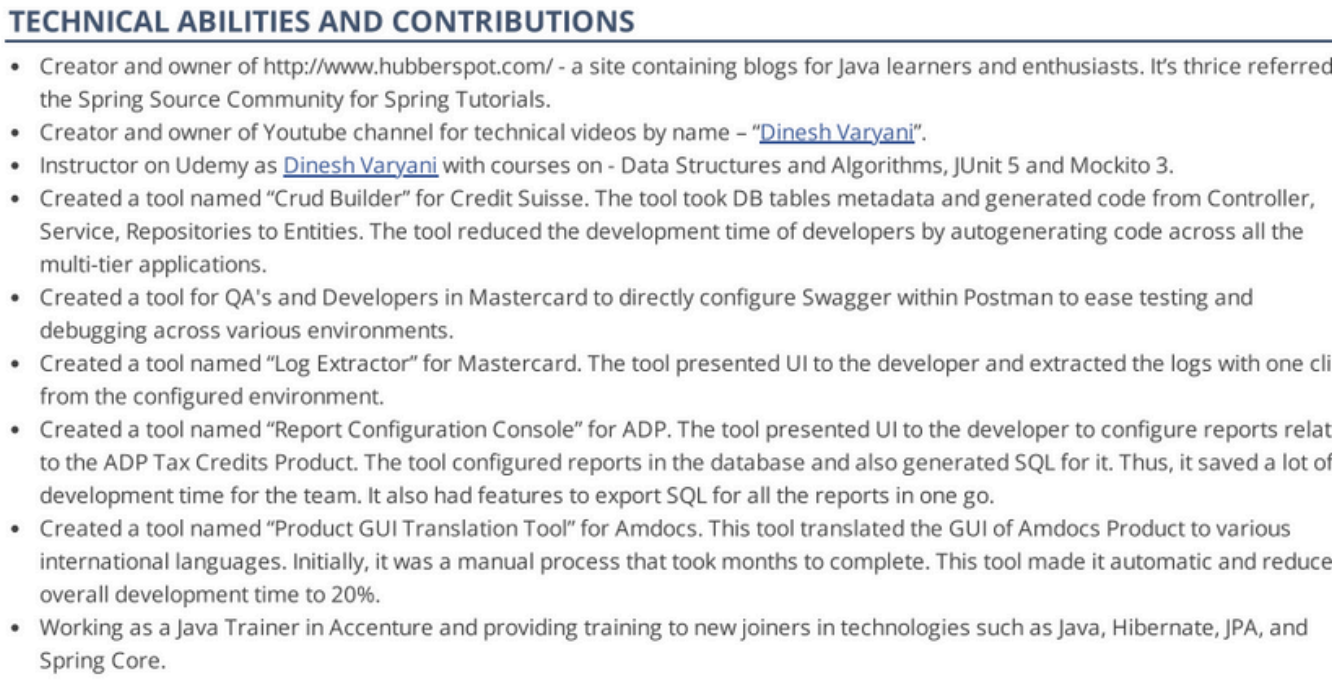
✅ Implemented XYZ feature that reduced the manual time from 3 weeks to 10 mins.

✅ Reduced the client onboarding duration from 1 month to 1 week etc.

Thus, projects outcomes in numbers and percentages shows how much impact your deliverables have done on Company's growth.

Resume Building Tips

👉 Big Tech Giants will always look for innovative and creative work you have done in your career. Thus, showcasing the innovative things you have done throughout your career like working on cool personal projects, YouTube, Blogging, Instructor, Tools, Building Apps, Coding Platforms Rank, etc



TECHNICAL ABILITIES AND CONTRIBUTIONS

- Creator and owner of http://www.hubberspot.com/ - a site containing blogs for Java learners and enthusiasts. It's thrice referred the Spring Source Community for Spring Tutorials.
- Creator and owner of Youtube channel for technical videos by name – "Dinesh Varyani".
- Instructor on Udemy as Dinesh Varyani with courses on - Data Structures and Algorithms, JUnit 5 and Mockito 3.
- Created a tool named "Crud Builder" for Credit Suisse. The tool took DB tables metadata and generated code from Controller, Service, Repositories to Entities. The tool reduced the development time of developers by autogenerating code across all the multi-tier applications.
- Created a tool for QA's and Developers in Mastercard to directly configure Swagger within Postman to ease testing and debugging across various environments.
- Created a tool named "Log Extractor" for Mastercard. The tool presented UI to the developer and extracted the logs with one cli from the configured environment.
- Created a tool named "Report Configuration Console" for ADP. The tool presented UI to the developer to configure reports relat to the ADP Tax Credits Product. The tool configured reports in the database and also generated SQL for it. Thus, it saved a lot of development time for the team. It also had features to export SQL for all the reports in one go.
- Created a tool named "Product GUI Translation Tool" for Amdocs. This tool translated the GUI of Amdocs Product to various international languages. Initially, it was a manual process that took months to complete. This tool made it automatic and reduce overall development time to 20%.
- Working as a Java Trainer in Accenture and providing training to new joiners in technologies such as Java, Hibernate, JPA, and Spring Core.

Resume Building Tips

👉 Adding your exceptional professional achievements or landmarks will make you stand out among others. Thus, showcasing your achievements, certifications, awards, and recognitions is a must.

ACHIEVMENTS AND RECOGNITIONS

- Received **STAR Team Award (Credit Suisse)** in recognition of exceptional contribution to SOW Delivery.
- Received **STAR India IT Performance Award (Credit Suisse)** in recognition of outstanding performance, accomplishment, and lasting commitments to the organization.
- Received **Spot Award (Altimetrik/Mastercard)** in recognition of outstanding performance and splendid efforts at work.
- Received **Accenture Celebrates Excellence Award - Pathfinder** for contribution to Innovative Excellence, effective project delivery by using new and innovative methods, reusing existing best practices and work methodologies to work more efficiently
- Received **Accenture Celebrates Excellence Award - Numero Uno** for contribution to Delivery Excellence in the individual category, demonstrating a key sense of accountability and ownership for the processes that impact the quality, delivery timelines, and service level metrics.
- Received **Zenith Propel Award** for demonstrating the quality of work in the Accenture Financial Services domain and having a significant impact on Accenture Success.
- Received prestigious awards through Accenture Recognize Performace Program.
- Received consistently **Top ratings** across different organizations for outstanding performance.

CERTIFICATES

- Java Standard Edition 6 Programmer Certified Professional.
- System Design certification from Educative.
- Object Oriented Design certification from Educative.
- Data Structures and Algorithms certification from Algoexpert.
- Software Olympiad 2015.
- Won Altimetrik Cricket Tournament.
- Quiz competition winner.

👉 Choose a colorful resume template that can accommodate the above tips. Websites like hiration.com provide many such templates. It has one free template that fits what a good resume should have.

08

Preparation Strategy

Preparation Strategy

👉 Try to solve at least 1 medium / 2 easy-level coding question(s) every day.

👉 Try to solve problems on your own with no help. Look for hints if provided by the coding platform.

👉 The more time you spent on the problem (solving it on your own) the more concepts will become strong.

👉 After spending over an hour if you don't get the solution, look out for the solution, write it down on paper and make notes of things you missed.

👉 Revision is the key. Revise the concepts, notes, and problems often.

👉 Try to prepare at least 1 System and 1 Object Oriented Design case studies every week.

👉 Consistency is the key (You break, You fail)

👉 Apply Pomodoro Technique (Plan, 25 mins of focused prep, 5 mins of break, repeat)

👉 Give equal importance to Behavioral Interviews as well.

09

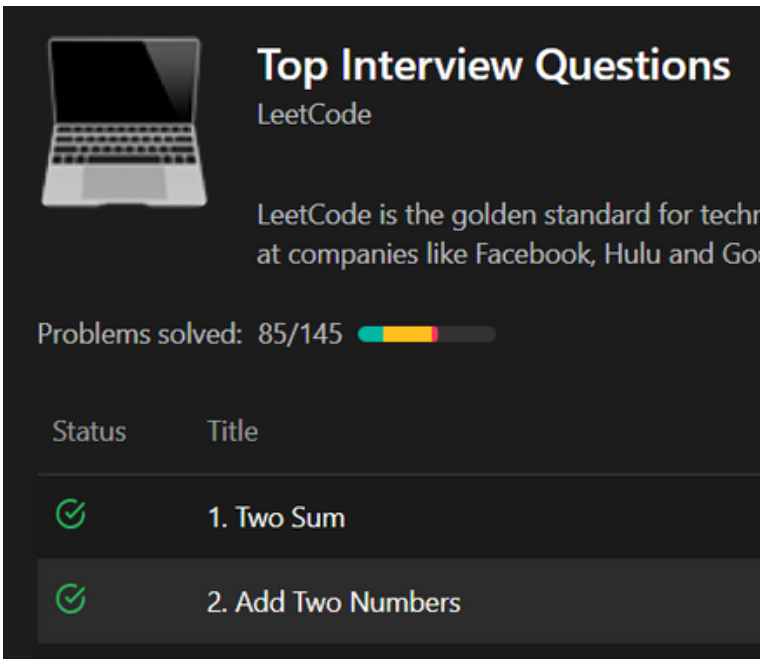
Effective LeetCode

Tips for using LeetCode effectively

👉 Solving problems in quantity won't give you quality preparation. Follow a roadmap of quality problems - 100 Days to GAMAM.

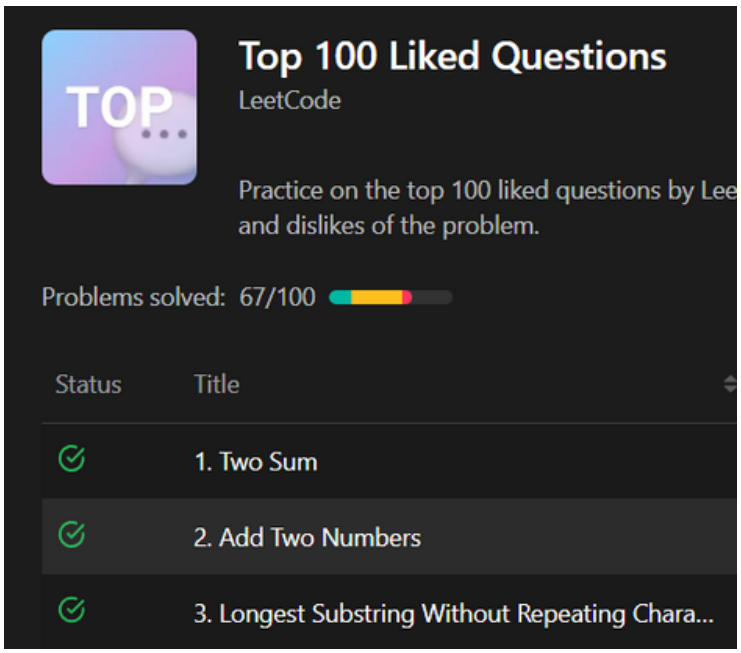
👉 Solve LeetCode problems daily without breaking consistency. Make it 1-2 hours per day. Remember the motto - "you break, you fail"

👉 Solve LeetCode curated list of Top Interview Questions (Very Important Questions)

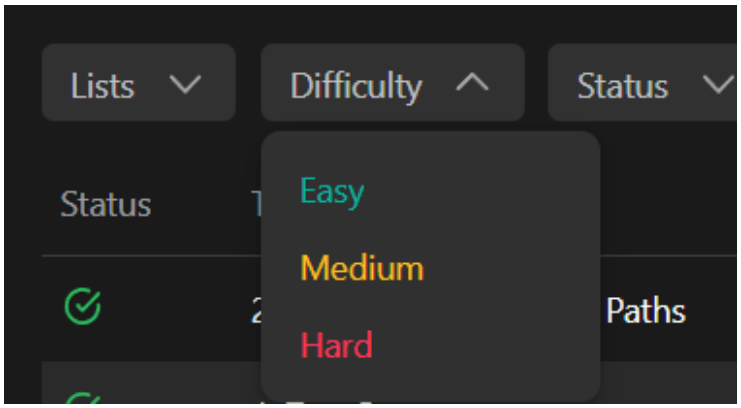


Tips for using LeetCode effectively

👉 Solve LeetCode curated list of Top 100 Liked Questions (Very Important Questions)

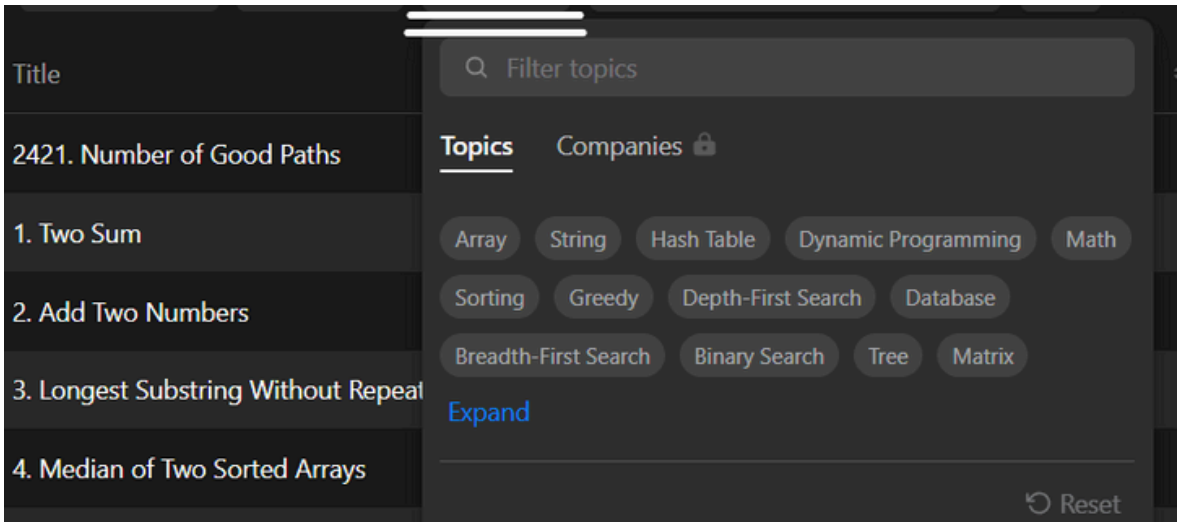


👉 If you are a beginner in coding always try to solve "Easy" level problems first.

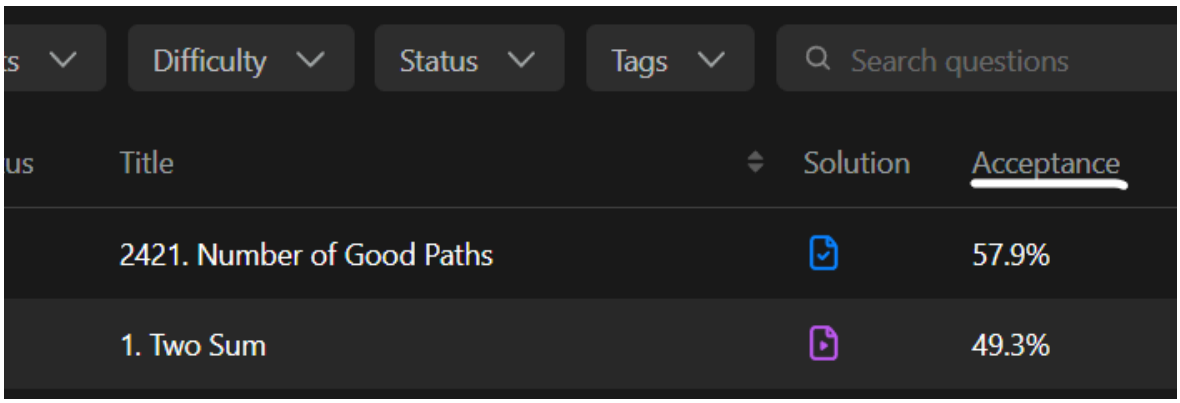


Tips for using LeetCode effectively

👉 Practice Questions from the topics in which you are weak and need confidence in the logic building. Use the Tags filter to choose specific topic questions.



👉 After Choosing any difficulty level, it's better to sort the problems by "Acceptance". This increases the chances of a successful attempt.

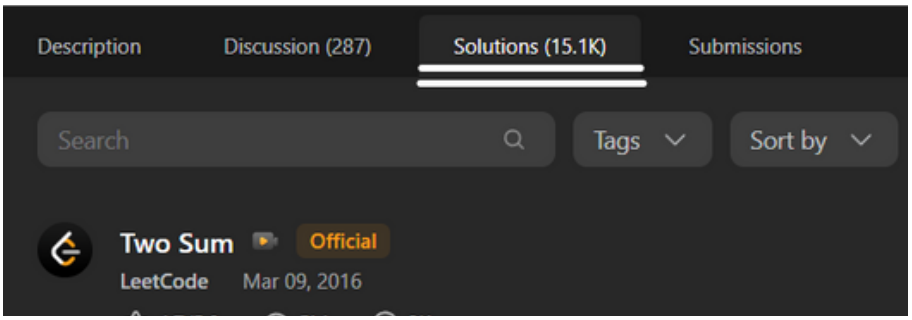


Tips for using LeetCode effectively

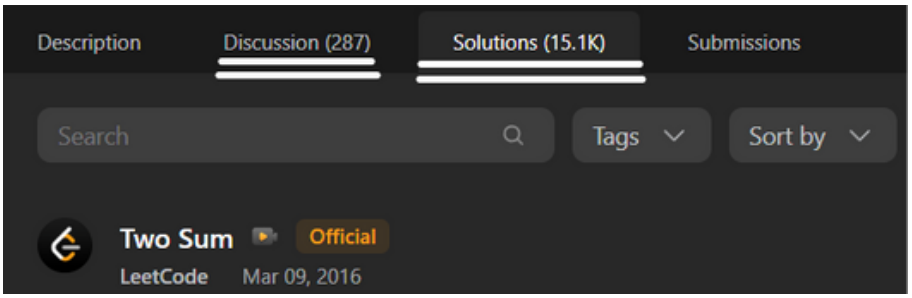
👉 If you are a beginner, Choose a topic easy-level questions and solve at least 15-20 problems on that topic to reach its medium level.

👉 If you are ***not*** a beginner, Choose a topic medium-level questions & solve at least 30-40 problems on that topic to reach its hard level.

👉 Always check out multiple ways of solving the same problem. Check out the **Solutions** Tab.



👉 Always look for Time & Space Complexity after solving the problem in the **Discussions** and **Solutions** Tab.

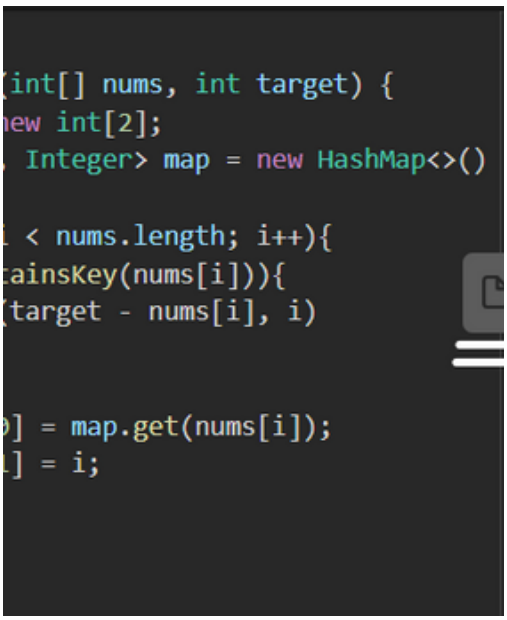


Tips for using LeetCode effectively

👉 Target to solve quality problems with at least -
✅ **100** Easy Level
✅ **250** Medium Level
✅ **75** Hard Level

👉 Don't spend more than 45-60 minutes, if you are stuck in a problem. After the above time limit -
✅ Check Hints
✅ Check Solutions

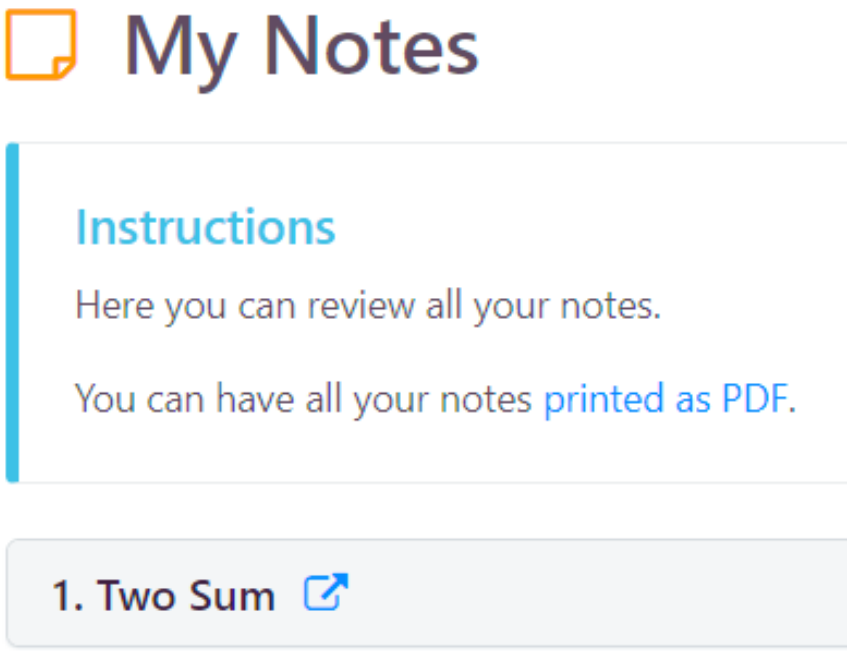
👉 If you are checking out the Solutions of others. Make sure you write the notes around what you missed under the **Notes** tab.



Tips for using LeetCode effectively

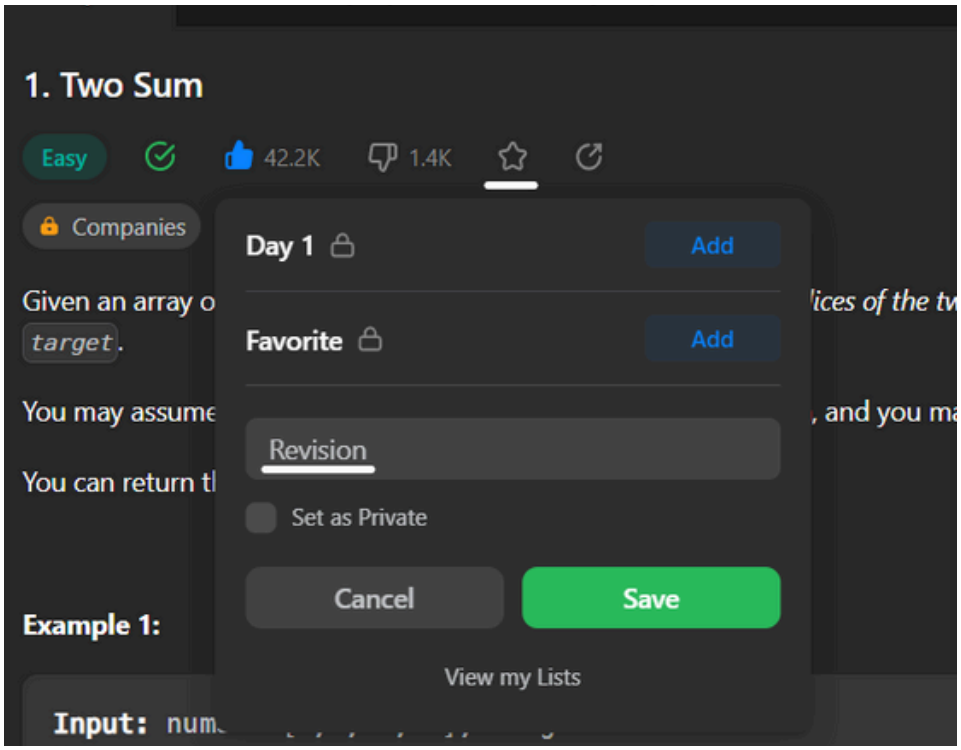
👉 If you are checking out the Solutions of others. Make sure you understand the solution, write down the solution in a paper, code it in the LeetCode editor, and dry run.

👉 Notes you write in the **Notes** tab are very useful and LeetCode provides a way (**Notebook**) to print it for personal use and revision.



Tips for using LeetCode effectively

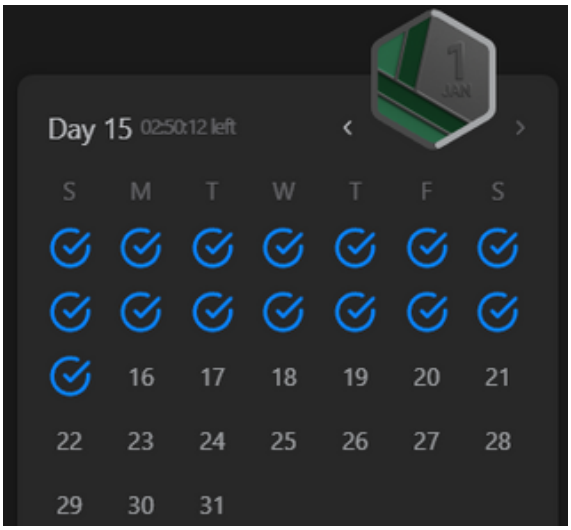
👉 Revision is a must. If you are checking out the Solutions of others. Make sure you are adding that problem in a Revision List. You can create a custom list via the LeetCode editor.



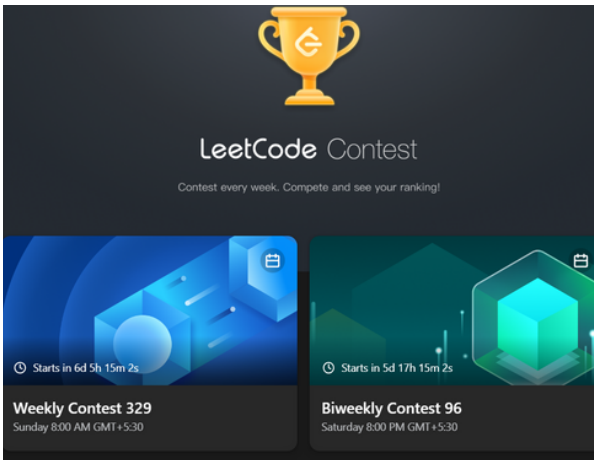
👉 Revision is a must. As per Ebbinghaus's Forgetting Curve, keep revising the problems on which you faced difficulty often. It should be revised in the span of 1st day, 7th day, 30th day.

Tips for using LeetCode effectively

👉 Once you are comfortable with LeetCode practice and have solved over 200+ problems, try to solve Daily LeetCode challenge.

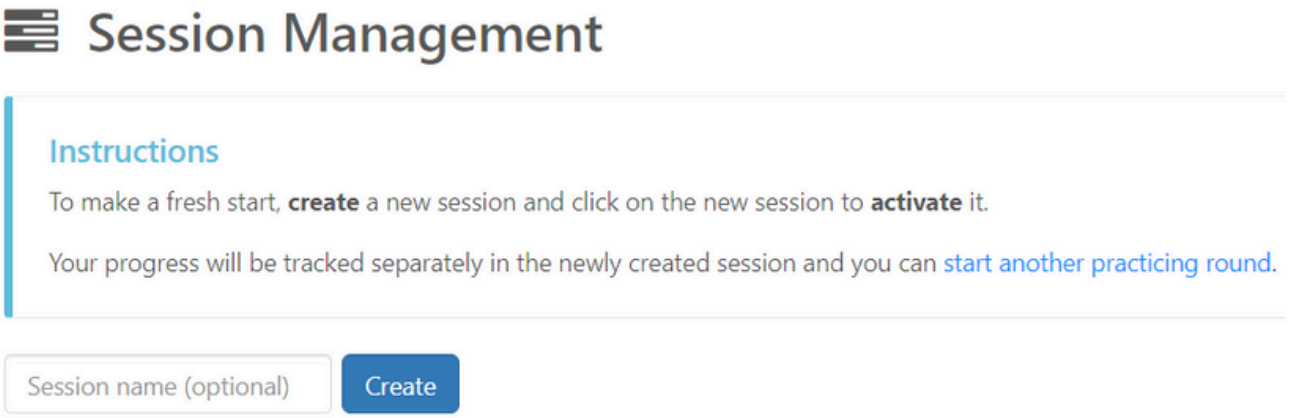


👉 Once you have solved over 150+ problems, try to participate in Weekly/Biweekly Contest using the Contest tab.



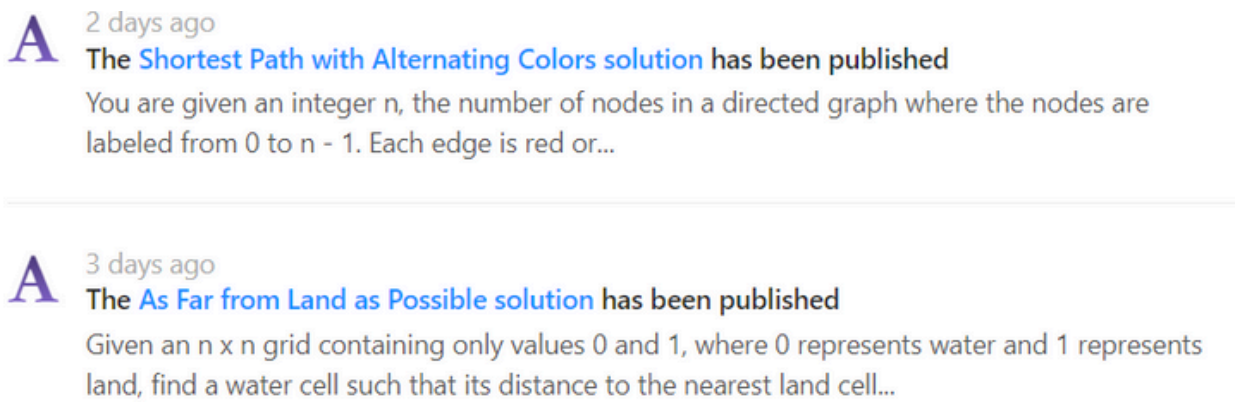
Tips for using LeetCode effectively

👉 Always keep track of your progress by creating a new session using the Session Management tab.



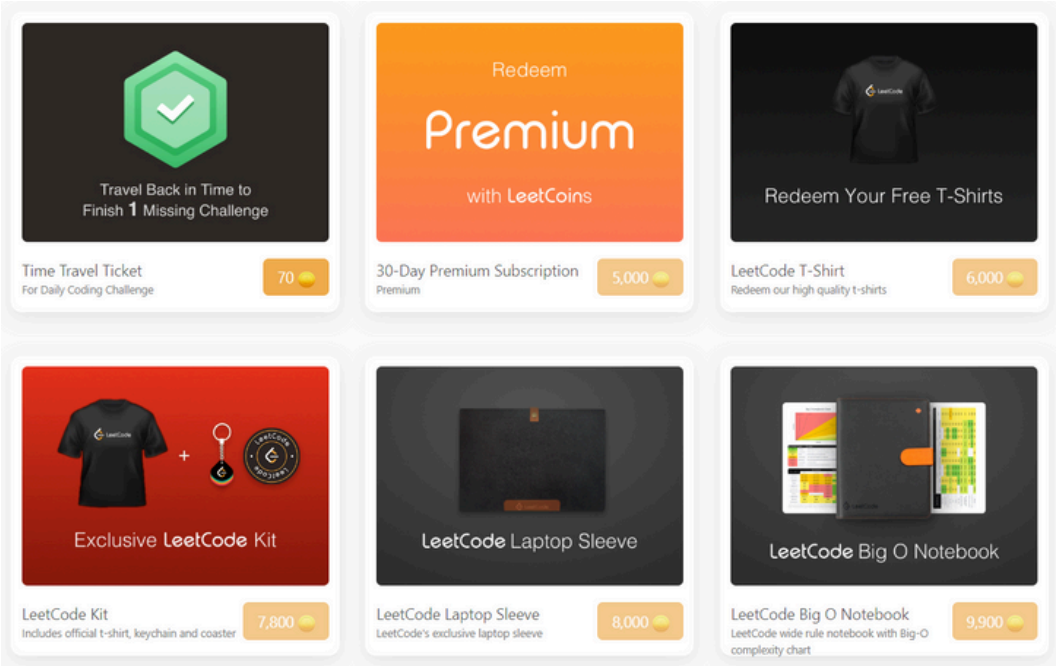
👉 Once you are comfortable with LeetCode practice and have solved over 200+ problems, try to solve Daily LeetCode challenge.

👉 Keep looking for articles posted on the homepage of LeetCode. It covers detailed solutions to a given problem.

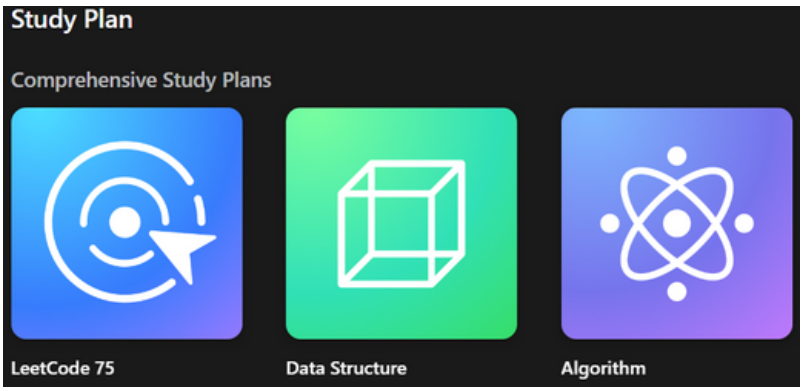


Tips for using LeetCode effectively

👉 Motivation Tip - Keep checking in daily, solve the Daily LeetCode challenge, participate in Contest to earn LeetCode points, and redeem gifts.

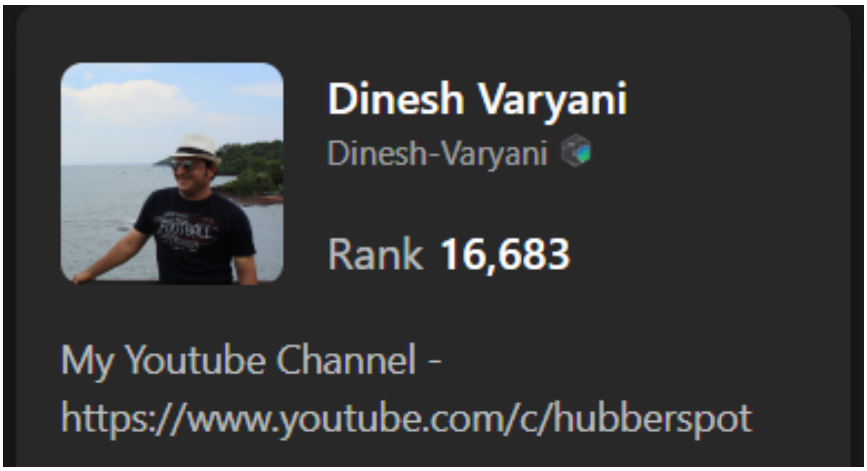


👉 Try to complete free study plans and assessments, those are great and quick resources to master DSA.

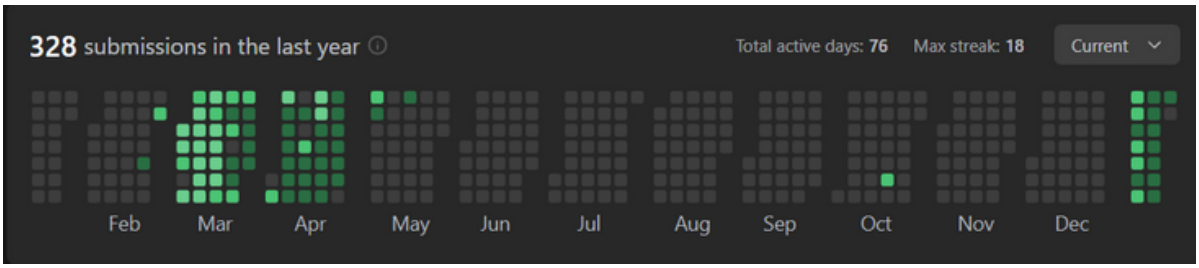


Tips for using LeetCode effectively

👉 Track your LeetCode rank every day. It will motivate you to improve it more by solving problems daily.



👉 Track your consistency via the LeetCode submission graph. It shows the total active days, max streak, and number of submissions in the last year.



10

150 Days Roadmap to GAMAM

150 Days to Roadmap to GAMAM

👉 Coding questions are from LeetCode and in order (Easy, Medium, Hard)

👉 Every 6 days have coding questions.

👉 Every 7th day have one system design and one low level design questions.

👉 System design and Low-level design questions are taken from the resources mentioned in previous sections.

👉 Every 15th day is a revision day for things practiced in previous 14 days.

👉 The last section starting the 120th day has a behavioral interview questions as well.

👉 Choose days and questions based on your time and availability.

👉 If you miss any question on a particular day, just carry it over to the next day.

👉 The idea is to be consistent for 150 days and not solve all questions in hurry.

DAY 1

- Two Sum
- Best Time to Buy and Sell Stock
- Majority Element
- Move Zeroes
- Squares of a Sorted Array
- Merge Sorted Array

DAY 2

- Remove Duplicates from Sorted Array
- Remove Duplicates from Sorted Array II
- Find All Numbers Disappeared in an Array
- Intersection of Two Arrays
- Intersection of Two Arrays II
- Maximum Population Year
- Find Pivot Index

DAY 3

- Running Sum of 1d Array
- Remove Element
- Find Winner on a Tic Tac Toe Game
- Build Array from Permutation
- Third Maximum Number
- Valid Mountain Array

DAY 4

- Find Common Characters
- Sum of All Odd Length Subarrays
- Range Sum Query - Immutable
- Shuffle the Array
- Max Consecutive Ones
- Sort Array By Parity

DAY 5

- Reverse Linked List
- Remove Linked List Elements
- Remove Duplicates from Sorted List
- Merge Two Sorted Lists
- Middle of the Linked List
- Palindrome Linked List

DAY 6

- Intersection of Two Linked Lists
- Linked List Cycle
- Valid Parentheses
- Implement Queue using Stacks
- Backspace String Compare
- Next Greater Element I

DAY 7

- Design a Rate Limiter (System Design)
- Design a Library Management System (OOD Design)

DAY 8

- Binary Tree Preorder Traversal
- Binary Tree Inorder Traversal
- Binary Tree Postorder Traversal
- Maximum Depth of Binary Tree
- Invert Binary Tree
- Symmetric Tree

DAY 9

- Subtree of Another Tree
- Diameter of Binary Tree
- Balanced Binary Tree
- Merge Two Binary Trees
- Same Tree

DAY 10

- Path Sum
- Binary Tree Paths
- Cousins in Binary Tree
- Convert Sorted Array to Binary Search Tree
- Range Sum of BST

DAY 11

- Valid Palindrome
- Valid Palindrome II
- Longest Palindrome
- Longest Common Prefix
- Valid Anagram
- First Unique Character in a String

DAY 12

- Is Subsequence
- Reverse String
- Reverse String II
- Reverse Words in a String III
- Isomorphic Strings
- Remove All Adjacent Duplicates In String

DAY 13

- Defanging an IP Address
- Reverse Only Letters
- Reverse Vowels of a String
- Length of Last Word
- Add Strings
- Fizz Buzz

DAY 14

- Design Consistent Hashing (System Design)
- Design a Parking Lot (OOD Design)

DAY 15

- Revise 1-14 days

DAY 16

- Roman to Integer
- Palindrome Number
- Happy Number
- Power of Two
- Sqrt(x)
- Plus One

DAY 17

- Count Odd Numbers in an Interval Range
- Rectangle Overlap
- Add Digits
- Maximum Product of Three Numbers
- Excel Sheet Column Number

DAY 18

- Add Binary
- Counting Bits
- Number of 1 Bits
- Single Number
- Missing Number
- Reverse Bits
- Hamming Distance

DAY 19

- Binary Search
- Search Insert Position
- First Bad Version
- Valid Perfect Square
- Kth Missing Positive Number
- Kth Largest Element in a Stream

DAY 20

- Design HashMap
- Ransom Note
- Contains Duplicate
- Contains Duplicate II
- Jewels and Stones
- Unique Number of Occurrences

DAY 21

- Word Pattern
- Number of Good Pairs
- Flood Fill
- Island Perimeter
- Find if Path Exists in Graph

DAY 22

- Design A Key-value Store (System Design)
- Design Amazon - Online Shopping System (OOD Design)

DAY 23

- Fibonacci Number
- Min Cost Climbing Stairs
- Climbing Stairs
- Pascal's Triangle
- Can Place Flowers
- Maximum Units on a Truck

DAY 24

- 3Sum
- 3Sum Closest
- Non-decreasing Array
- Product of Array Except Self

DAY 25

- Merge Intervals
- Insert Interval
- Non-overlapping Intervals
- Interval List Intersections

DAY 26

- Container With Most Water
- Sort Colors
- Rotate Array
- Contiguous Array

DAY 27

- Subarray Sum Equals K
- Shortest Unsorted Continuous Subarray
- Maximum Points You Can Obtain from Cards
- Max Consecutive Ones III

DAY 28

- Permutation in String
- Wiggle Sort II
- Max Chunks To Make Sorted
- H-Index

DAY 29

- Design A Distributed Unique ID Generator (System Design)
- Design Stack Overflow (OOD Design)

DAY 30

- Revise 16-29 days

DAY 31

- Remove Nth Node From End of List
- Delete Node in a Linked List
- Remove Duplicates from Sorted List II
- Next Greater Node In Linked List

DAY 32

- Add Two Numbers
- Add Two Numbers II
- Copy List with Random Pointer
- Reverse Linked List II

DAY 33

- Swap Nodes in Pairs
- Odd Even Linked List
- Partition List

DAY 34

- Sort List
- Reorder List
- Rotate List

DAY 35

- Evaluate Reverse Polish Notation
- Min Stack
- Daily Temperatures
- Decode String

DAY 36

- Next Greater Element II
- Next Greater Element III
- Minimum Remove to Make Valid Parentheses
- 132 Pattern

DAY 37

- Design A URL Shortener (System Design)
- Design a Movie Ticket Booking System (OOD Design)

DAY 38

- Asteroid Collision
- Basic Calculator II
- Remove K Digits
- Remove Duplicate Letters

DAY 39

- Remove All Adjacent Duplicates in String II
- Flatten Nested List Iterator
- Simplify Path
- Longest Absolute File Path

DAY 40

- Open the Lock
- Shortest Bridge
- LRU Cache

DAY 41

- Longest Substring Without Repeating Characters
- String to Integer (atoi)
- Find All Anagrams in a String
- Group Anagrams
- Pancake Sorting

DAY 42

- Longest Repeating Character Replacement
- Largest Number
- Number of Matching Subsequences
- Find the Index of the First Occurrence in a String

DAY 43

- Longest Substring with At Least K Repeating Characters
- Zigzag Conversion
- Reverse Words in a String
- String Compression
- Count and Say

DAY 44

- Design Pastebin (System Design)
- Design an ATM (OOD Design)

DAY 45

- Revise 31-44 days

DAY 46

- Binary Tree Level Order Traversal
- Binary Tree Zigzag Level Order Traversal
- Construct Binary Tree from Preorder and Inorder Traversal
- Lowest Common Ancestor of a Binary Tree

DAY 47

- Binary Tree Right Side View
- Populating Next Right Pointers in Each Node
- Populating Next Right Pointers in Each Node II
- Maximum Width of Binary Tree

DAY 48

- Path Sum II
- Path Sum III
- All Nodes Distance K in Binary Tree
- Flatten Binary Tree to Linked List

DAY 49

- Count Complete Tree Nodes
- Sum Root to Leaf Numbers
- Find Bottom Left Tree Value
- Distribute Coins in Binary Tree

DAY 50

- Delete Node in a BST
- Validate Binary Search Tree
- Kth Smallest Element in a BST
- Lowest Common Ancestor of a Binary Search Tree

DAY 51

- Convert Sorted List to Binary Search Tree
- Construct Binary Search Tree from Preorder Traversal
- Binary Search Tree Iterator
- Recover Binary Search Tree

DAY 52

- Design Instagram (System Design)
- Design an Airline Management System (OOD Design)

DAY 53

- Binary Tree Maximum Path Sum
- Step-By-Step Directions From a Binary Tree Node to Another
- Maximum Level Sum of a Binary Tree

DAY 54

- Trim a Binary Search Tree
- Balance a Binary Search Tree
- Serialize and Deserialize Binary Tree

DAY 55

- Search in Rotated Sorted Array
- Search in Rotated Sorted Array II
- Time Based Key-Value Store
- Find Minimum in Rotated Sorted Array

DAY 56

- Find First and Last Position of Element in Sorted Array
- Find the Duplicate Number
- Minimum Size Subarray Sum
- Single Element in a Sorted Array

DAY 57

- Find Peak Element
- Capacity To Ship Packages Within D Days
- Koko Eating Bananas
- Peak Index in a Mountain Array

DAY 58

- Search a 2D Matrix
- Search a 2D Matrix II
- Spiral Matrix
- Spiral Matrix II

DAY 59

- Design A Web Crawler (System Design)
- Design Blackjack and a Deck of Cards (OOD Design)

DAY 60

- Revise 46-59 days

DAY 61

- Valid Sudoku
- Rotate Image
- Set Matrix Zeroes
- Game of Life

DAY 62

- Diagonal Traverse
- Matrix Block Sum
- Battleships in a Board
- Snapshot Array

DAY 63

- Number of Islands
- 01 Matrix
- Clone Graph
- Rotting Oranges

DAY 64

- Course Schedule
- Course Schedule II
- Accounts Merge
- Word Search

DAY 65

- Minimum Height Trees
- Pacific Atlantic Water Flow
- Cheapest Flights Within K Stops
- Max Area of Island

DAY 66

- Evaluate Division
- Number of Provinces
- Surrounded Regions
- Network Delay Time

DAY 67

- Design A Notification System (System Design)
- Design a Hotel Management System (OOD Design)

DAY 68

- All Paths From Source to Target
- Redundant Connection
- Shortest Path in Binary Matrix
- Number of Operations to Make Network Connected

DAY 69

- Majority Element II
- Longest Consecutive Sequence
- Insert Delete GetRandom O(1)
- Find All Duplicates in an Array

DAY 70

- Continuous Subarray Sum
- Find and Replace Pattern
- K-diff Pairs in an Array
- Custom Sort String

DAY 71

- Fraction to Recurring Decimal
- Fruit Into Baskets
- Encode and Decode TinyURL
- Minimum Area Rectangle

DAY 72

- Maximum Subarray
- Maximum Product Subarray
- Coin Change
- Coin Change II

DAY 73

- Jump Game
- Jump Game II
- Jump Game III
- Partition Equal Subset Sum

DAY 74

- Design A News Feed System (System Design)
- Design a Restaurant Management system (OOD Design)

DAY 75

- Revise 61-74 days

DAY 76

- Longest Increasing Subsequence
- Unique Paths
- Unique Paths II
- Maximal Square

DAY 77

- House Robber
- House Robber II
- House Robber III
- Decode Ways

DAY 78

- Best Time to Buy and Sell Stock II
- Minimum Path Sum
- Longest Common Subsequence
- Palindrome Partitioning

DAY 79

- Unique Binary Search Trees
- Unique Binary Search Trees II
- Target Sum
- Triangle

DAY 80

- Longest Palindromic Subsequence
- Partition to K Equal Sum Subsets
- Delete and Earn
- Palindromic Substrings

DAY 81

- Longest String Chain
- Minimum Cost For Tickets
- Delete Operation for Two Strings
- Perfect Squares

DAY 82

- Design A Chat System (System Design)
- Design Chess (OOD Design)

DAY 83

- Different Ways to Add Parentheses
- Longest Palindromic Substring
- Largest Divisible Subset
- Integer Break

DAY 84

- Matchsticks to Square
- Knight Dialer
- Minesweeper

DAY 85

- Random Pick with Weight
- Pow(x, n)
- Reverse Integer
- Multiply Strings

DAY 86

- Count Primes
- Integer to Roman
- Robot Bounded In Circle
- Angle Between Hands of a Clock

DAY 87

- K Closest Points to Origin
- Task Scheduler
- Top K Frequent Elements
- Find K Closest Elements

DAY 88

- Kth Largest Element in an Array
- Kth Smallest Element in a Sorted Matrix
- Top K Frequent Words
- Reorganize String

DAY 89

- Design A Search Autocomplete System (System Design)
- Design an Online Stock Brokerage System (OOD Design)

DAY 90

- Revise 76-89 days

DAY 91

- Sort Characters By Frequency
- Car Pooling
- Find K Pairs with Smallest Sums
- Maximum Number of Events That Can Be Attended

DAY 92

- Implement Trie (Prefix Tree)
- Word Break
- Design Add and Search Words Data Structure
- Search Suggestions System
- Remove Sub-Folders from the Filesystem

DAY 93

- Permutations
- Permutations II
- Subsets
- Subsets II

DAY 94

- Next Permutation
- Combinations
- Letter Combinations of a Phone Number
- Generate Parentheses

DAY 95

- Combination Sum
- Combination Sum III
- Combination Sum IV
- Restore IP Addresses

DAY 96

- Gas Station
- Partition Labels
- Valid Parenthesis String
- Minimum Number of Arrows to Burst Balloons

DAY 97

- Design YouTube (System Design)
- Design a Car Rental System (OOD Design)

DAY 98

- Single Number II
- Single Number III
- Maximum XOR of Two Numbers in an Array
- Divide Two Integers

DAY 99

- Sum of Two Integers
- Bitwise AND of Numbers Range
- Gray Code

DAY 100

- Sliding Window Maximum
- Trapping Rain Water
- Count of Smaller Numbers After Self

DAY 101

- Candy
- Reverse Pairs
- Subarrays with K Different Integers
- Number of Submatrices That Sum to Target

DAY 102

- Shortest Subarray with Sum at Least K
- Maximum Gap
- First Missing Positive

DAY 103

- Shuffle an Array
- Reverse Nodes in k-Group
- LFU Cache

DAY 104

- Design Google Drive (System Design)
- Design LinkedIn (OOD Design)

DAY 105

- Revise 91-104 days

DAY 106

- Basic Calculator
- Largest Rectangle in Histogram
- Longest Valid Parentheses

DAY 107

- Maximum Frequency Stack
- The Skyline Problem
- Minimum Window Substring

DAY 108

- Palindrome Pairs
- Shortest Palindrome
- Text Justification

DAY 109

- Nth Digit
- Integer to English Words
- Max Points on a Line

DAY 110

- Maximum Profit in Job Scheduling
- Median of Two Sorted Arrays
- Find Minimum in Rotated Sorted Array II

DAY 111

- Word Ladder
- Word Ladder II
- Longest Increasing Path in a Matrix

DAY 112

- Design Twitter (System Design)
- Design Facebook - a social network (System Design, OOD Design)

DAY 113

- Word Search II
- Bus Routes
- Critical Connections in a Network

DAY 114

- Shortest Path in a Grid with Obstacles Elimination
- Reconstruct Itinerary
- Making A Large Island

DAY 115

- Merge k Sorted Lists
- Find Median from Data Stream
- Smallest Range Covering Elements from K Lists

DAY 116

- Minimum Number of Refueling Stops
- Swim in Rising Water
- Longest Duplicate Substring

DAY 117

- N-Queens
- Permutation Sequence
- Sudoku Solver
- Palindrome Partitioning II

DAY 118

- K-th Symbol in Grammar
- Remove Invalid Parentheses
- Unique Paths III

DAY 119

- Proximity Service (System Design)
- Design Cricinfo (OOD Design)

DAY 120

- Revise 106 - 119 days

DAY 121

- What are your strengths and weaknesses?
- Tell me about your most challenging customer. How did you resolve their issues and make them satisfied?
- Describe a time when you had to make a decision without having all the data or information you needed.
- Which {company's} leadership principle resonates with you most?
- Tell me about a time when you were working on a project, and you realized that you needed to make changes to what you were doing. How did you feel about the work you had already completed?

DAY 122

- Nearby Friends (System Design)
- Google Maps (System Design)

DAY 123

- Edit Distance
- Regular Expression Matching
- Maximal Rectangle

DAY 124

- Can you give me an example of a time when you exceeded expectations?
- Can you describe a time when you took the lead on a project?
- Think about a time you received negative feedback. How did you deal with that?
- Tell me about a time when you had to deal with ambiguity. How did you overcome the ambiguity to reach a positive outcome?
- Have you been stressed over a certain project delivery in the past? Did it affect your work-life balance? How did you deal with it?

DAY 125

- Distributed Message Queue (System Design)
- Distributed Email Service (System Design)

DAY 126

- Split Array Largest Sum
- Burst Balloons
- Wildcard Matching

DAY 127

- Tell me about a time you have disagreed with your manager and how you handled it.
- How do you motivate others? Can you give me an example of a time you have motivated someone?
- Tell me about a time when you took a risk and failed. What did you learn from that experience?
- What obstacles have you encountered in your career? How did you overcome them?
- Tell me about a project you are proud of. How did you ensure high standards were met in delivering that project?

DAY 128

- S3-like Object Storage (System Design)
- Real-time Gaming Leaderboard (System Design)

DAY 129

- Best Time to Buy and Sell Stock IV
- Word Break II
- Russian Doll Envelopes
- Validate Stack Sequences

DAY 130

- Why do you want to work for {company}?
- Tell me about a time when you have had to work to earn someone's trust.
- Describe a time when you were given a project to work on, but your responsibilities were unclear. What did you do?
- Tell me about a time you showed initiative.
- You see a co-worker struggling with a task. What do you do?

DAY 131

- Payment System (System Design)
- Digital Wallet (System Design)

DAY 132

- Minimum Insertion Steps to Make a String Palindrome
- Minimum Cost to Cut a Stick
- Minimum Number of Taps to Open to Water a Garden
- Binary Tree Cameras

DAY 133

- Describe for me a time when you had to choose short-term sacrifices to achieve long-term gains.
- How do you deal with having to provide feedback to someone?
- Tell me about a time you failed to meet a deadline. How did you cope with that?
- Has there been a time when your contribution was overlooked and somebody else from your team took credit for it? How did you deal with it?
- Tell me about a project you are proud of. How did you ensure high standards were met in delivering that project?
- Have you been in a conflict with a fellow coworker? How did you deal with it and what was the end result?

DAY 134

- Design Uber backend (System Design)
- Design Ticketmaster (System Design, OOD Design)

DAY 135

- Revise 121 - 134 days

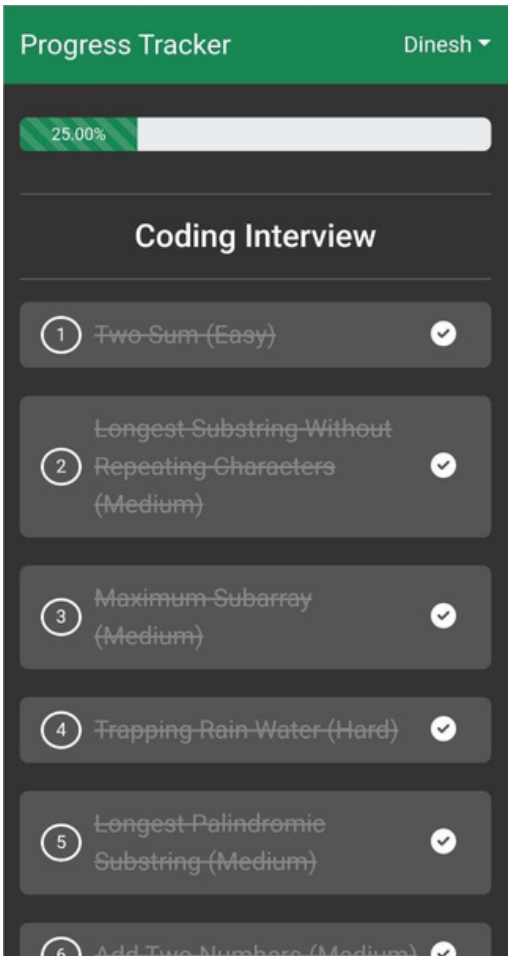
DAY 136 - 150 (Revise 1-135 days)

11

GAMAM Progress Tracker

👉 Follow the below app for GAMAM-level companies preparation. It has the best resources that will cover various topics of interviews. The app will help you keep track of your progress for interview preparation.

👉 Follow the resources (links), understand the concepts behind them, and mark them complete. Once you reach over 80-100% the progress you are well prepared for the GAMAM-level company interviews.



https://progress-tracker-5b3b0.web.app/

📌 All the best for your preparation.
📌 If you find it useful, then follow Dinesh Varyani on LinkedIn.
📌 Subscribe to my YouTube channel.
📌 For all the interview preparation resources, strategies, tips, roadmap, and mock interviews you can connect with me at - https://topmate.io/dinesh_varyani
📌 Credits - @Arslan Ahmed, @Clement Mihailescu, @Alex Xu.

Thank you.

CRACKING THE GAMAM TECHNICAL INTERVIEWS

WHAT'S INSIDE?

The main objective of this book is to help you in preparing and crack GAMAM (Google, Apple, Microsoft, Amazon, Meta) technical interviews. The book covers preparation resources, strategies, tips, and a roadmap I followed that helped me clear Google/Amazon technical rounds. The book summarizes my journey like this -

- 👉 How I prepared myself for the GAMAM companies?
- 👉 What resources I used for various types of interviews?
- 👉 What strategies have I used to master different topics?
- 👉 What roadmap I followed in months of my preparation?
- 👉 Which resume tips got me in the recruiter's eye?
- 👉 How I tracked the progress of my preparation?
- 👉 My advice/tips on how to answer in a technical interview?

Dinesh Varyani

I am working as a Cloud Engineer at Google. I am having over 12+ years of experience in Software Engineering. I am a passionate Youtuber, Blogger, Udemy & Educative Instructor, and now an Author.