CRACKING THE GAMAM TECHNICAL INTERVIEWS

Strategies, Tips, and Preparation resources

AN INSIDER'S GUIDE

Dinesh Varyani
Engineer @ Google

Cracking the GAMAM Technical Interviews

1st Edition

Dinesh Varyani

Dinesh Varyani

Cracking the GAMAM Technical Interviews, First Edition

Copyright © 2023 Dinesh Varyani.

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means whatsoever without the written permission of the author except for the use of brief quotations in a book review.

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

www.hubberspot.com

Dinesh Varyani

Dinesh Varyani

Table of Contents

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

Preface

Dear Reader,

I am Dinesh Varyani, 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.

The main objective of this book is to help you in preparing and crack GAMAM (Google, Apple, Microsoft, Amazon, Meta) technical interviews. The prime focus of this book is to share 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 type of interviews?
- 👉 What strategies I used to master different topics?
- 👉 What roadmap I followed in months of my preparation?
- 👉 Which Resume tips got me to Recruiter's eye?
- 👉 How I tracked progress of my preparation?
- 👉 My advices/tips on how to answer in a technical interviews?

Wish you all the best. I am sure you will find this book useful.

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

Preparation resources

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

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

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

4. AlgoExpert's 170 handpicked coding question (In case you want to prepare fast and only good questions)

5. Watch my DSA playlist to revise concepts.

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

✅ 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.

Don'ts

⛔ Never dive into solving a problem as soon as it's thrown towards 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 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 code - check for any edge cases, refactoring, remove 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 about any hint 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 in logic try to apply some coding patterns - like can two pointer help, can sort help, can binary search be applied, etc.

11 Golden Rules for solving a Coding problem in an Interview

Dinesh Varyani

Dinesh Varyani

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

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

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

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

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

👉 If coding problem is around a **Singly Linked List**, and 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 than try to use a **HashMap/HashSet.** It makes 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, O(1) space complexity than try to **sort** the array**.** It makes algorithm faster with O(N logN) time complexity and O(1) space complexity.

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

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

02

System Design / High Level Design Interview

Preparation resources

1. Grokking the System Design Interview - The course has step-by-step discussion and good case studies.

2. 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.

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

Time Management in System Design Interview

✅ **Requirement Clarifications** (3-5 min)

✅ **Estimations** (3-5 min)

✅ **API Design** (3-5 min)

✅ **Database Schema Design** (3-5 min)

✅ **System's Detailed Design** (20 - 25 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** - (3-5 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 requires low latency?
👉 Does the system needs to be reliable?

✅ **Estimations** (3-5 min)

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

✅ **API Design** (3-5 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 DELETE, PATCH etc
👉 Identity HTTP Status codes for different scenarios

✅ **Database Schema Design** (3-5 min)

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

✅ **System's Detailed Design** (20 - 25 **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 way to scale them

(c) **Scaling individual components** - Horizontal & Vertical scaling

(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

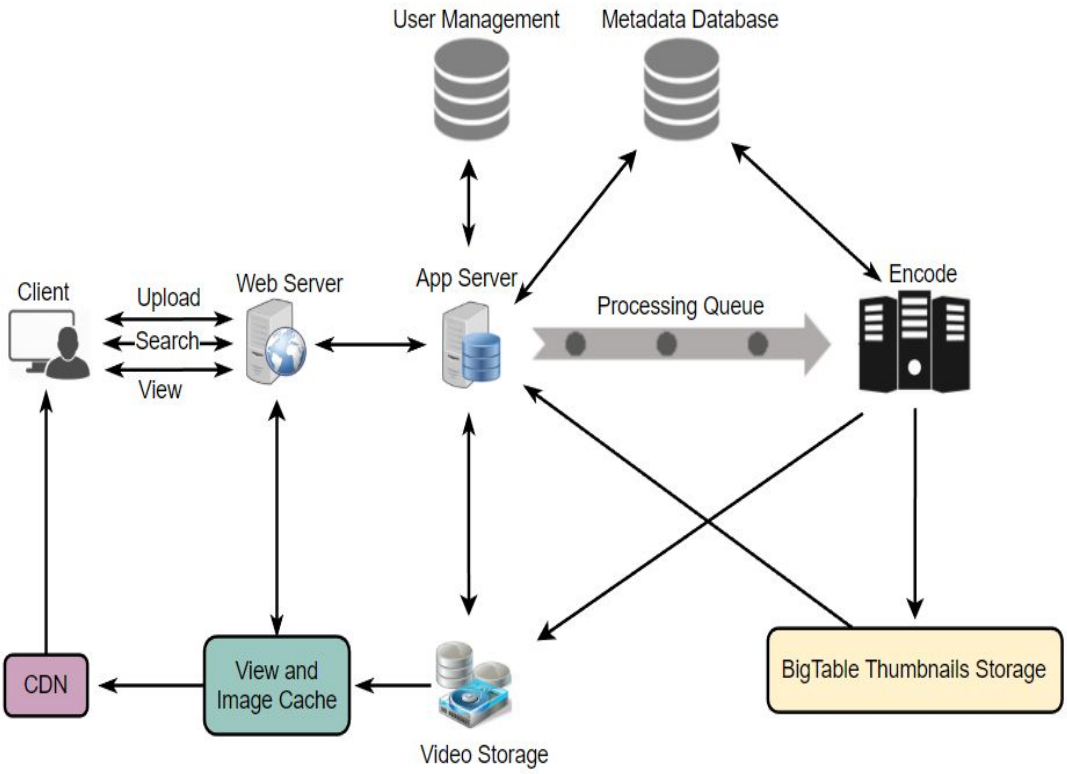
(f) **Databases**

- 👉 SQL - Sharding, Indexes, master-slave, master-master, Denormalization
- 👉 NoSQL - Key-Value, Document, Wide-Column, Graph

(g) Communication Protocols and standards like - 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



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 feed of relevant questions.

b) Non-Functional Requirements

The design of the system is highly affected by non-functional requirements of system. **Availability** and **Consistency** are two major factors we need to look 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 same data (read and write same data)

Other factors are -

👉 **Eventual Consistency** - In a distributed system Availability and Consistency can't be achieved simultaneously. Thus, many a 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 request and response can be handled by many nodes which are connected over network. Thus, it becomes important to measure the latency of system because of network lag involved. It is typically measured in secs and millisecs. The system should have low latency which means request should be fulfilled quickly. It improves performance of the system.

Quora must have 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 interviewer scope in mind.

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

✅ Estimations (3-5 min)

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

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

Let's understand the both estimation in greater detail.

a) Storage estimates - Storage estimates can be estimated via 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 interviewer).

- Let's assume Quora has 300 Million MAU (monthly active users).
- Let's assume Quora has 40 Million DAU (daily active users).
- 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

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 (user who posted it)

Let's assume that each question takes 1 KB of storage. Then, 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**

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 (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, 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**

b) QPS estimates - QPS can be defined as queries per second. Its total of reads and writes per second occuring 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 number of pages visited by each user per day is 10. If we take 40 Million as daily active users then 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.

It can be seen that its 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

✅ API Design (3-5 min)

Outline the different API's for the required scenarios. The API's should cover all the requirements of the system. These functional requirements are same what has been captured in **Requirement Clarification** step. For Quora some of the API's 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 as 201. Otherwise returns a failure string with error code 4xx (client side error) or 5xx (server side error)

✅ API Design (3-5 min)

👉 Post Answer - Add a new answer

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

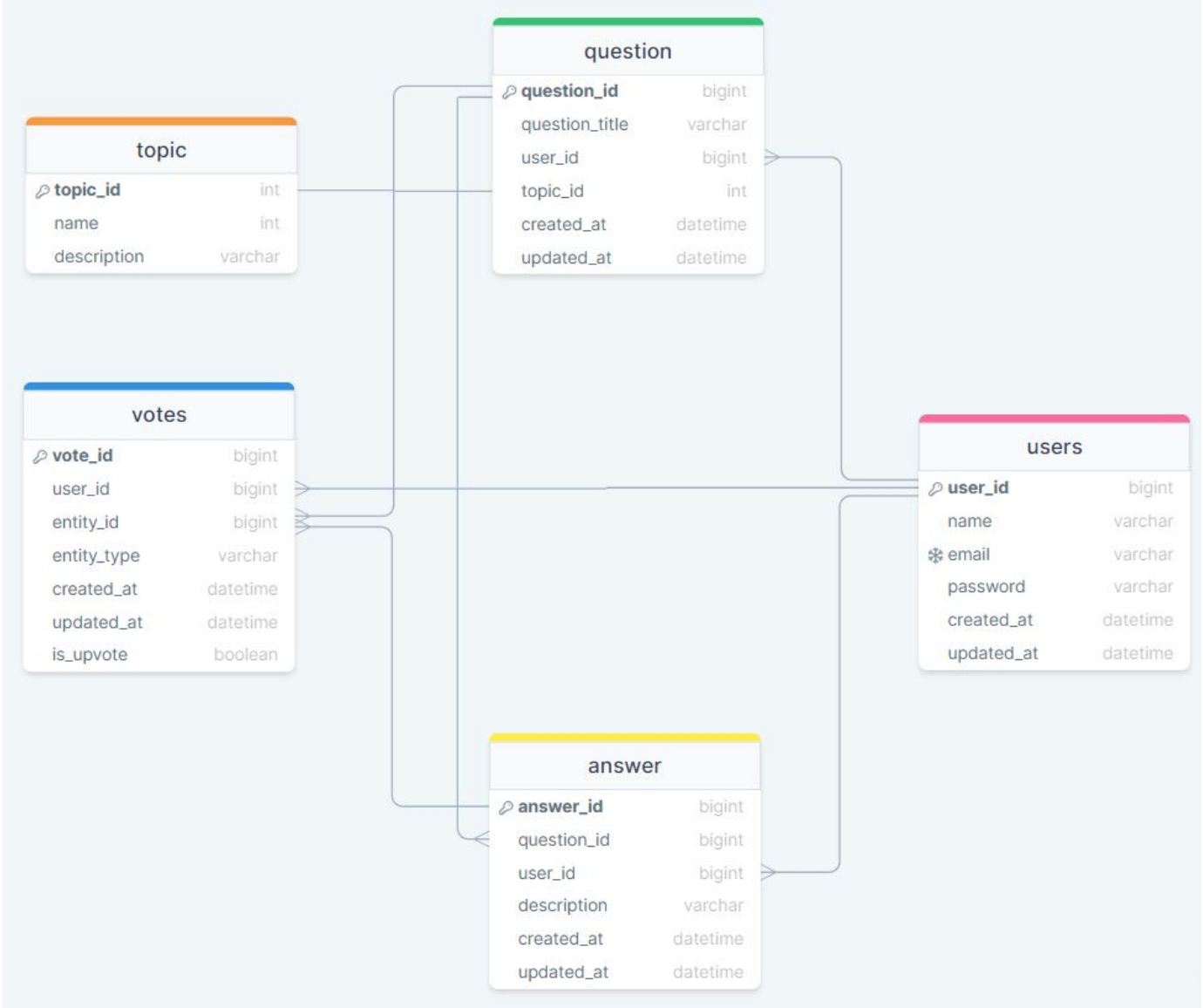
```
{
   "answer_id": "35caf31e-7cf5-11ed-a1eb-0242ac120010",
   "question_id": "27baf31e-7cf5-11ed-a1eb-0242ac120002",
   "answer_description": "Practice DSA and Design daily !!!",
   "created_datetime": "16/12/2023T13:45:00.000Z",
   "user_id": "30baf31e-8cf5-13ed-a1eb-0242ac260019"
}
```

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

and other similar apis …

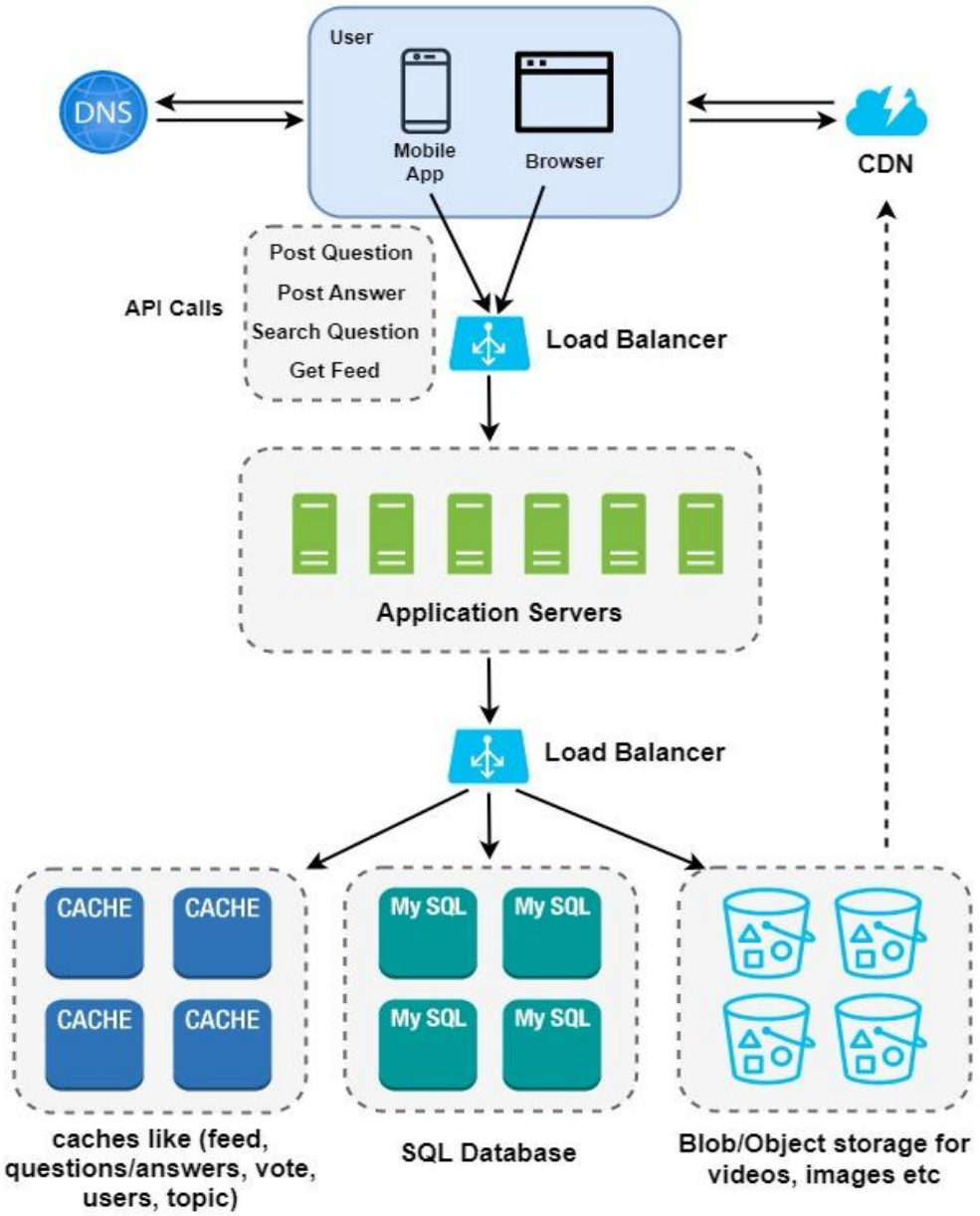
✅ Database Design (3-5 min)

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



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

👉 Draw/Explain high-level components (detailed design) of the system like the below diagram -



25 Golden Rules to answer in a System Design Interview

Dinesh Varyani

Dinesh Varyani

👉 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 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 preventing 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 **Checksum** algorithm.

👉 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**.

03

Object Oriented / Low Level Design Interview

Preparation resources

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

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

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 OOD 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.

👉 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

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

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

✅ 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.

✅ 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

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

Once we know the core classes/objects 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 …

✅ 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 that can be sketched?
👉 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
- 👉 Customer has One Shopping Cart
- 👉 Shopping Cart has Many Items

etc ...

✅ 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.

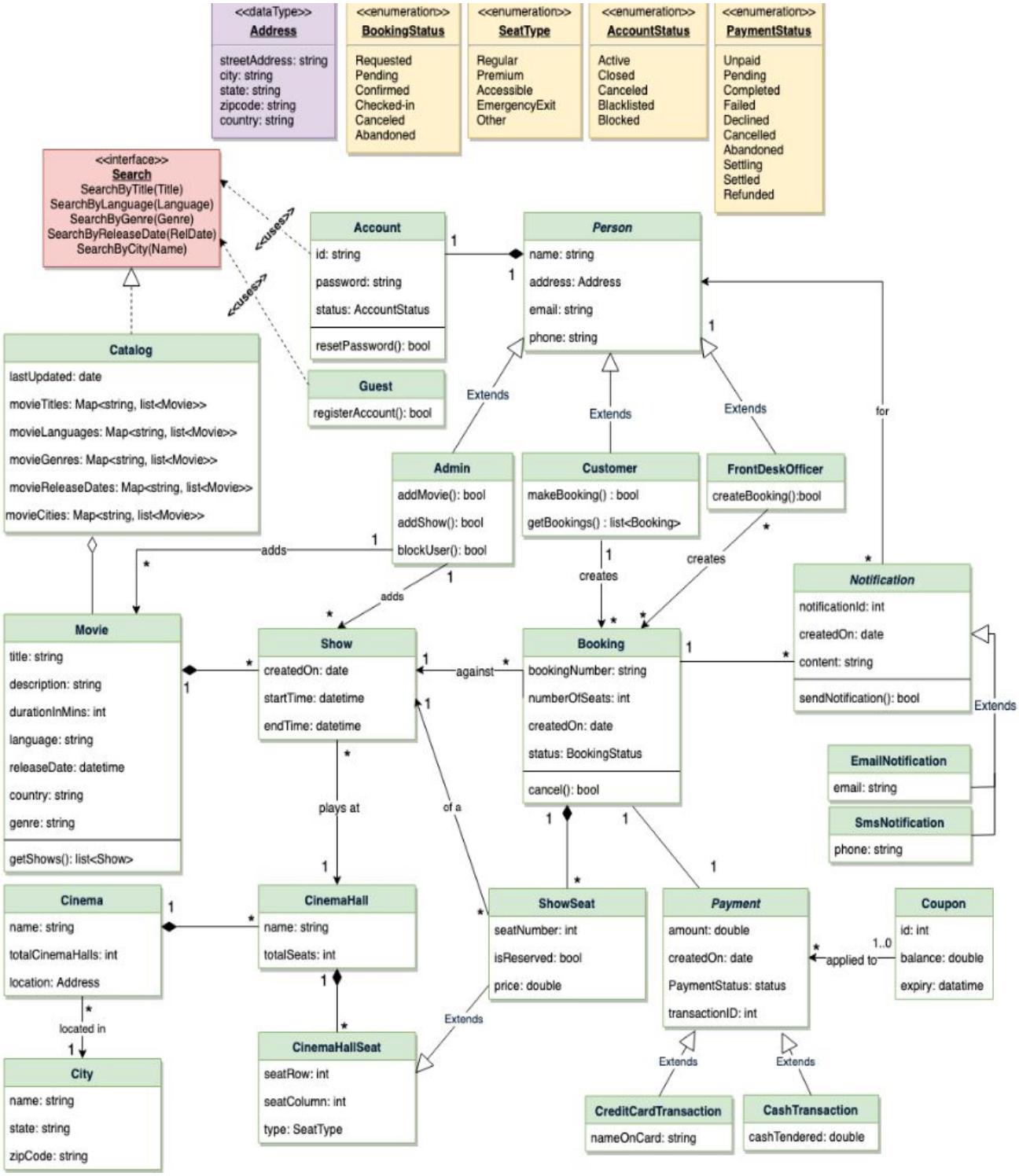
👉 Customer can add the item to shopping cart - **addItemToCart(Item item)**
👉 Customer can place an order - **placeOrder(Order order)**
etc…

✅ 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



Dinesh Varyani

Dinesh Varyani

04

API Design Interview

Preparation resources

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

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

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

4. Follow the link to understand how apis should be designed.

05

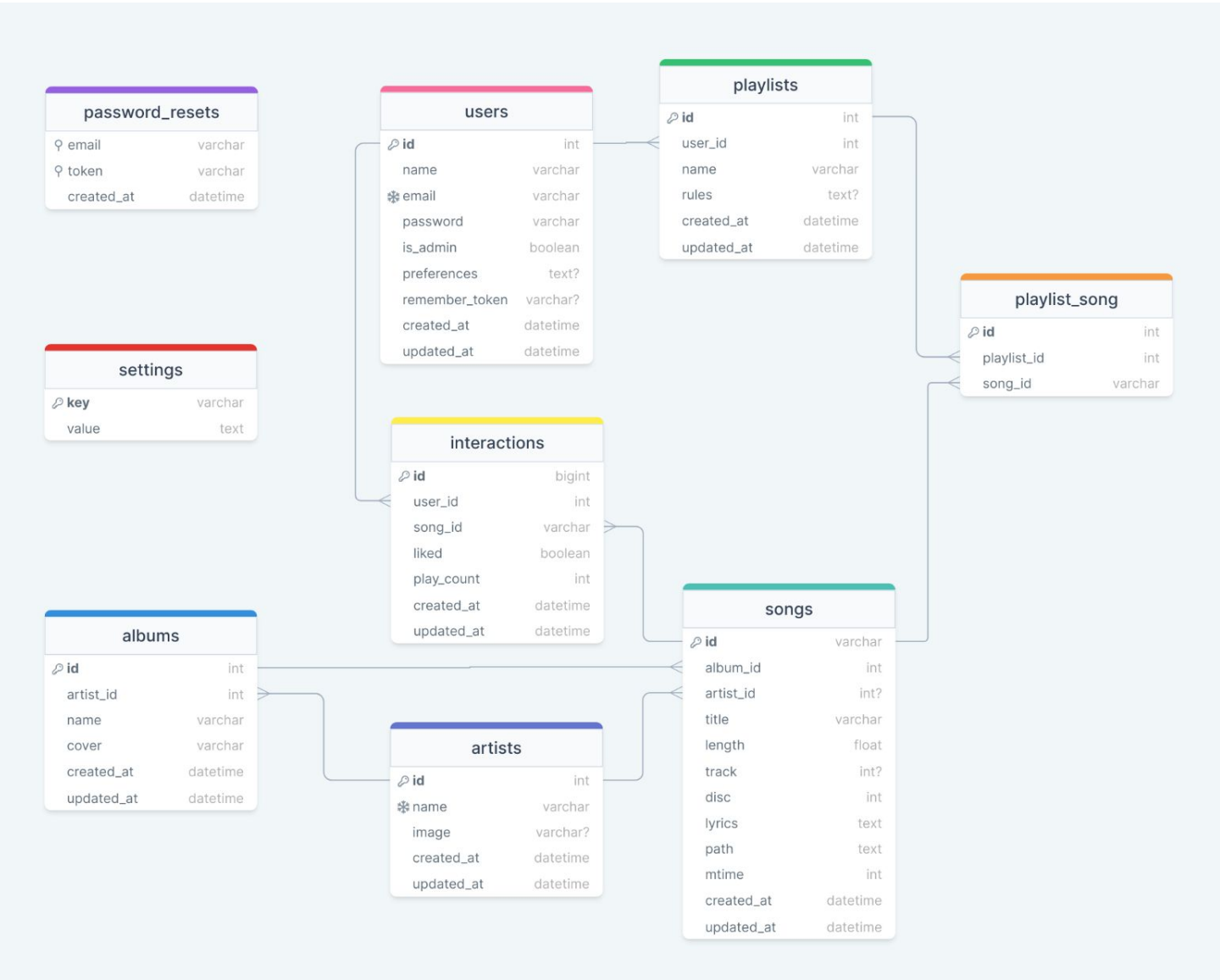
Schema / Database Design Interview

Preparation resources

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

2. Try practicing drawing of 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



Dinesh Varyani

Dinesh Varyani

06

Behavioral Interview

Preparation resources and Tips

1. Watch Jeff H Sipe's YouTube channel for behavioral questions.
2. Check out a list of good Behavior Interview questions.
3. STAR Pattern - Situation, Task, Action, Result
4. Prepare questions and career stories around Amazon's leadership principles. It will cover every aspect of Behavior interviews.
5. Apply 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.

07

Resume Tips

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

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

👉 If you have done great things in your professional career you can showcase it in a 2-3 page resume. Making a one-page resume is not always preferable.

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

👉 Provide a good introductory summary of yourself at the top.

👉 Keep important things first in the resume like - introduction, skills, projects, achievements, certifications, education, etc. Things like hobbies, and contact info can come at the end.

👉 Provide project impact via numbers and percentages rather than normal text.

- **Avoid** - Implemented the project with XYZ feature that helped in the productivity of the users.
- **Focus** - Implemented the project with XYZ feature that increased the productivity of the users by 30%.
- **Focus** - Implemented XYZ feature that reduced the manual time from 3 weeks to 10 mins.
- **Focus** - Reduced the client onboarding duration from 1 month to 1 week etc.

👉 Highlight the things you do apart from normal work like - Training, KT, sessions, etc.

👉 Showcase your achievements, certifications, and recognitions.

👉 Showcase the innovative things you have done throughout your career like working on cool projects, YouTube, Blogging, etc.

08

Preparation Strategy

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

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

👉 The more time you spent on problem (solving 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 a paper and make notes of things you missed.

👉 Revision is the key. Revise the concepts, notes, 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.

150 Days 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 revision day for things practiced in previous 14 days.

👉 Last section starting 120th day has behavioral interview questions as well.

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

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

👉 Idea is to be consistent for 150 days and not to 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 15-29 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 30-44 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 45-59 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 60-74 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 75-89 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 90-104 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 105 - 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 120 - 134 days

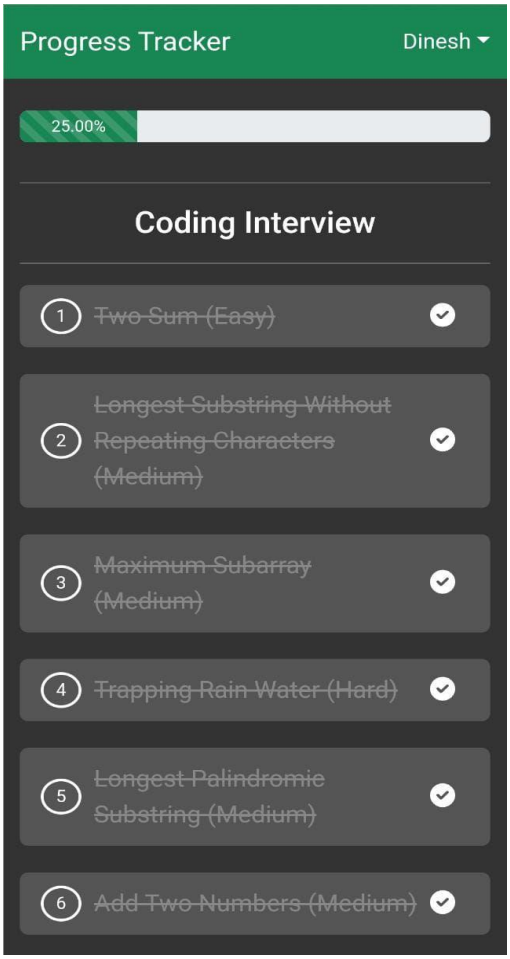
DAY 136 - 150 (Revise 1-135 days)

09

GAMAM Progress Tracker

👉 Follow the below app for GAMAM-level companies preparation. It has the best resources which will cover various topics of interviews. The app will help you in keeping 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% 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.

📌 Credits - @Arslan Ahmed, @Clement Mihailescu, @Alex Xu.

Thank you.

CRACKING THE GAMAM TECHNICAL INTERVIEW

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 to recruiter's eye?
- 👉 How I tracked progress of my preparation?
- 👉 My advices/tips on how to answer in a technical interviews?

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.