Crushing the Firmware & Embedded Systems Interview

Akash Agrawal · Follow
12 min read · Sep 30, 2021

I have been working as a Firmware and Embedded Systems Engineer for almost a decade now. I have worked at companies of all sizes with large corporations to fast moving startups to very small companies with less than 20 employees. I have gone through the interview process several times and given hundreds of individual interviews. During my preparations I rarely found resources available specific for firmware/embedded systems interviews. There are actually lots of resources available for general software engineer roles including coding practice websites like Leetcode and GeeksforGeeks but nothing dedicated for firmware.

Over the years I have made syllabus and methods to prepare for firmware interviews for myself and it has worked well for me. I have worked at and got job offers from top startups like Samsara, Roku, Argo, and large companies like Intel, Apple (I currently work at Apple). Since I had so much trouble finding good resources I decided to write this guide myself to help prospective firmware engineer candidates with their interview preparations.

Most of my interview experience has been in United States but from what I know the process is very similar in other places like Europe, India and many other countries.

I have divided firmware interviews preparation into 4 parts; Coding, Theory, Design and Behavioral. Coding probably is the most important part of any technical

interview followed by theory, but the weightage of Design and Behavioral increases as well for senior level roles.

From my own experience the interviews get increasingly difficult as you apply for more senior positions, which also makes logical sense. Entry levels jobs are mostly based on coding and basic firmware theory concepts. For more senior positions you can expect some deep design questions. Always be prepared to write some code even for very senior positions. It's best to check with recruiters what kinds of questions are expected before the interview.

Technical interviews have become like sports these days. You just got to do it better than the competition. It's very hard to crack technical interviews without preparation. You don't just have to solve the coding questions, you have to solve it in under 25–30 mins. It's very hard to do it without practice. However I believe with some practice and preparation it is not very hard to crack those interviews.

How to prepare:

Getting the interview:

First steps first. You need to get interview calls before you can do anything. You are not going to get a job if you do not get the interview. Here are some of my tips to get the interview call:

1. **Resume**: Make sure your resume is updated and formatted properly. This link from Careercup (by the writer of Cracking the Coding Interview) has one of the best tips I have seen on how to write a good resume. There is also a template which you can download for your resume. I highly recommend sticking to 1 page resume. Have your resume reviewed by few of your friends or colleagues. I would also recommend getting it reviewed by a professional.
2. **Linkedin**: Linkedin has become the de facto location for recruiters to find candidates and vice versa. Keep your Linkedin profile updated even when you are not looking for positions. I always sign up for Linkedin premium whenever I am actively looking for jobs. Your profile gets listed higher in recruiter searches which makes a significant difference. I always see lot more recruiter reach out

to me when I have premium (job seekers plus) enabled. You also get to send in-mails directly to recruiters using premium. Again recruiter's response rates are much higher for in-mails than by directly applying for job postings.

3. **Referral:** Referrals are the best way to get interviews. Have a friend or an acquaintance refer you to their company. Most companies pay cash bonuses to the person who makes the referral so it's actually in their best interest that you get the job. So don't shy away from asking for referrals. If you can get your resume in the hands of a hiring manager, that's job already half done!

4. **Apply directly to company websites:** lastly directly applying to open positions at companies website works as well.

Study Routine:

It can take from few weeks to several months to prepare for technical interviews depending on your level of preparation. Everyday solve at-least 2–3 coding problems completely. So if you are using Leetcode make sure your code passes all test cases with minimal retries.

Also practice solving coding questions on whiteboard, but these days most hiring is done remotely so you might just end up doing coding excercises on some digital tool like coderpad.

Mock Interviews:

I would highly recommend doing mock interviews during your preparations. Mock interviews are a great way to build confidence and also to get meaningful feedback. You should do both coding and behavioral interview practice. For coding mock interview you can practice writing on the board (or on some code sharing tool for remote interviews) as well as speaking at the same time, which can take a while to master.

If you can not find anybody to help you with mock interviews there are some services available online both paid as well as free where you can practice technical mock interviews.

I get these comments from candidates a lot that they are not good at interviews or they really get nervous while interviewing. I get it, I am the same way. I believe preparing well and doing mock interviews are the best ways to get over these fears and building confidence.

Coding:

Most interviewers do not really care about what programming language you are using. For firmware roles however C and C++ are generally recommended. Especially for low level embedded software roles. C is a great language for solving pointers and bit manipulation questions. Knowledge of assembly languages can also be helpful for some low level software roles. Python is also readily used these days especially for writing test scripts and automation. I was asked specific Python (very basic) questions in some interviews. Learning a bit of Python can be helpful here.

Coding Syllabus:

Coding is definitely the most important part of firmware interviews. This is also the part most candidates are scared-of. But not to worry I have narrowed down the syllabus to the bare minimum which can help you succeed. You need to be really good at bit manipulations, arrays, strings, pointers, linked list, stacks and queues. Familiarity with binary trees, hash tables, DFS/BFS, sorting algorithms can be helpful as well. I would not worry too much about more software focused areas like dynamic programming and graph theory if you are specifically targeting low level embedded/firmware roles.

Arrays:

- binary search
- 2 pointer search
- sliding window problem
- kadane algorithm

Strings:

- Reverse string

Linked List:

- Reverse Singly Linked List
- Reverse Doubly Linked List
- Merger two linked lists
- Add, Delete, Search element of Linked List
- Find Middle of List
- Find last nth element of list
- Find if there is loop in list

Stacks and Queue:

- Implement Stack (FIFO) using Arrays
- Implement Stack using Linked List
- Implement Queue using Linked List
- Implement Queue using Arrays
- Implement Circular buffer

Bit Manipulation:

- atoi, itoa, itob, float to bin (link), atof
- Add/Sub in binary (w/o using + operator) link
- 2s complement (-ve numbers)
- endianness swap (link)
- range of 8 bit (-128 to 127 link)
- represent float in binary (link)

Memory:

- implement aligned malloc/free ** link
- implement malloc/free using static buffers (arrays)

How to answer coding questions the Google way:

I got the follwing information while preapring for Google interview. Google is very specific about how they like candidates to answer coding questions. I really like that approach and I think you can use it for other company interviews as well. So here it goes:

-When asked to provide a solution, first define and frame the problem as you see it.

-If you don't understand — ask for help or clarification.

-If you need to assume something — verbally check its a correct assumption!

-Always let your interviewer know what you are thinking as he/she will be as interested in your process of thought as your solution.

-Also, if you're stuck, they may provide hints if they know what you're doing.

-Finally, listen — don't miss a hint if your interviewer is trying to assist you!

Interviewers will be looking at the approach to questions as much as the answer:

Does the candidate listen carefully and comprehend the question?

Are the correct questions asked before proceeding? (important!)

Is brute force used to solve a problem? (not good!)

Are things assumed without first checking? (not good!)

Are hints heard and needed?

Is the candidate slow to comprehend / solve problems? (not good!)

Does the candidate enjoy finding multiple solutions before choosing the best one?

Are new ideas and methods of tackling a problem sought?

Is the candidate inventive and flexible in their solutions and open to new ideas?

Can questioning move up to more complex problem solving?

Behavioral:

The behavioral part of the interview is equally as important as the coding part and I feel you should spend some time preparing for it as well. Different companies has different names for the behavioral part of interview. Amazon calls is the Leadership Principal, Google call it 'Googliness' interview.

Some tips to prepare for this round:

prepare, write down and practice list of common questions like:

- Tell me about yourself.
- Why do you want to switch jobs?
- tell me about the hardest problem you have solved or your biggest achievement.
- Tell me about a time you wish you'd handled a situation differently with a colleague.
- Tell me about a time when someone was wrong? How did you communicate?
- *Have you handled a difficult situation with a coworker? How?*
- Can you give a story about what you should have done differently in a past project?
- What's the most complex or niche thing you know a lot about? can you explain it to me in five minutes or less?
- why should we hire you?

Prepare each question carefully and practice it several times.

-record your self

-do mock interviews

-pay attention to your voice

-general interview advice: speak slowly, loudly, clearly, practice!

-make sure people understand what you are saying. Try to explain a concept to someone and ask them if they understood.

-use SAR technique [link]

Theory:

Most firmware interviewers ask both coding as well as theory questions. It could be in the same interview or you can have separate interviews focused on either coding or theory or design.

-Difference between Serial Communication protocols like I2C, SPI and UART.

-Basic Electrical Engineering (transistors, diode, capacitors, motors)

-C++ (STL, inheritance, polymorphism, copy constructor, constructor vs destructor, overloading vs overriding operator, deep vs shallow copy, virtual functions, handles) — this is assuming your primary coding language is C but you are still familiar with C++

-What happens when you boot link link

-UART/I2C/SPI (compare, pull-ups)

-DMA

-Security (digital signing, hash, encryption)

-Software/hardware break point, JTAG

-Math: probability, basic trigonometry

-Code testing methods (how will you test this code)

-TCP/IP

OS Theory: (link)

-RTOS

-Task scheduling (FIFO, Round Robin, Priority-based)

-Threads (creation, joining, function pointer, spawn attributes)

-Mutex, semaphores, spin lock

-Context switching

-Scheduler

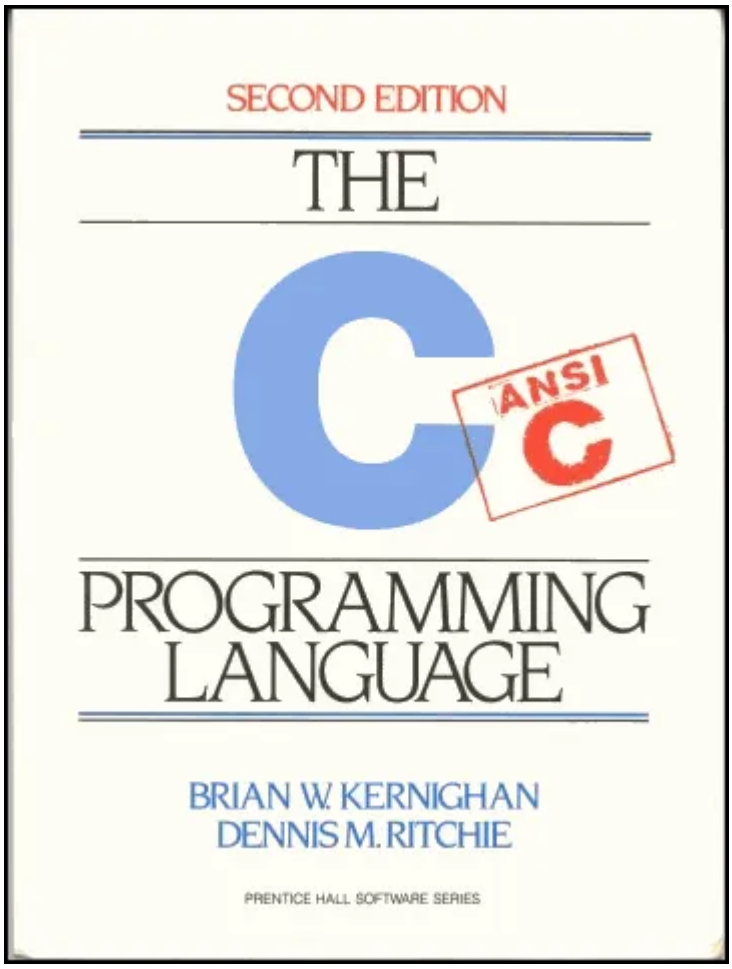
-Queues, FIFO, mailbox (multicore communication)

-G4G top OS questions, MCQs (link)

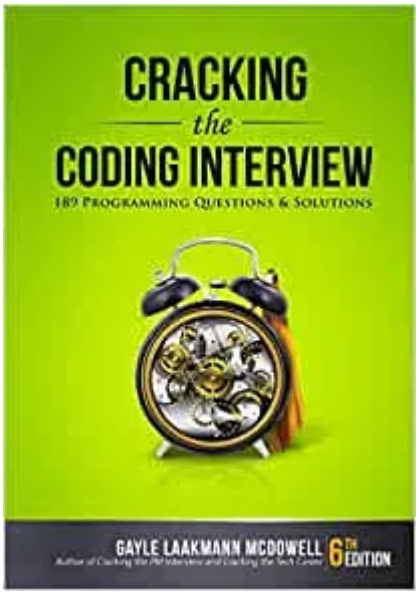
-**P**roducer consumer problem (interviewers love asking this question! Very important, both theory and writing code example)

Book Recommendations:

1. **The C programming Language by Brian Kernighan and Dennis Ritchie:** For firmware and embedded system engineering, deep understanding of C programming is crucial. This is a small book but has everything you need to know about the C. I would recommend going through the first 6 chapters at-least and if possible also solve all or most of the exercise questions. I would have to say this book is not an easy read but well worth the effort.



2. Cracking the Coding Interview by Gayle McDowell: This is probably the best book available currently to prepare for coding interviews however this book is written for general Software Engineering interviews so lot of the information may not be directly relevant for Firmware Engineering positions. Feel free to skip System Design, Graphs, Searching and Database section. Also if you decide to solve the exercise part, I would skip the hard questions.



Questions asked to me from my past interviews (embedded systems/firmware):

- merge two sorted linked list

- types of interrupt in linux

- diff between SPI, UART, I2C

- questions on BLE

-when to use volatile const variable at same time (answer: read only hardware registers)

-write a device driver using given spec

-implement a grey code like system (HW interview)

-design system for packets exchange

-design a traffic light system

-explain what happen in firmware update, how to uC loads new firmware

-diff between mutex and semaphores

-design opAmp

-c++ basics

- what happens when embedded sys boot

- explain compilation chain

- diff between SPI/UART

-when to use RTOS vs baremetal

-find a missing number (binary search, log n)

-find if matrix is perfect sudoku (code)

-stack problem, find if all braces opening has a corresponding closing brace

-diff between TCP/IP

-how to make sure data integrity in a data transfer

- how to connect multiple I2C slaves, clk stretching, multi voltage system, pull up

- structure padding

- array coding question

-presentation on any topic to the team

-math problems (probability, memory size)

-implement queue

-basic OS questions

-write temp sensor device driver (2s complement) — how to test

-design problem (automatic door opening system)

-find endianness

-graph search (find if 2 nodes connected)

-distance between 2 points (math)

-design a camera system

-reverse endianness

-OS: deadlock prevention, volatile, BLE

-how to sign firmware during update (code signing)

-how to overflow stack, what is PC, SP

-capacitors in parallel (what is final capacitance)

-add 2 numbers without + — operation

-find if number is multiple of two

-BLE questions, scheduler

-merge 2 sorted linked list

-array: find data above a threshold in a data array

-design a IoT light bulb

-find bugs in a code

-array: find compatible elements (using API)

-HW timer: given HW timer build an API system which can set any num of timers

-behavioral: give situation where you went out of your way, how did you solve a problem

-driver for reading a keyboard matrix

-how will you design hw, fw, sw for i2c like stream of data

-array: remove duplicates from sorted array

-discuss projects

-some trigonometry like questions

-behavioral: how will you contact/deal with external teams

-how to allocate read only and write register variable

-questions on interrupt, write syntax for function pointer

-question on allocate 12 bit ADC data in a buffer to 16 bit array elements

-boot-loader questions

-Implement malloc_align and malloc_free

-array question, move 0s to end

-lot of questions on RTOS/synchronization, mutex/semaphores

-cache coherency, virtual memory, paging, MMU, PCIe

-RTOS

- OS questions

- increment a hex string by 1 in place (no extra memory allocation)

-lot of questions on RTOS/synchronization, mutex/semaphores

-boot-loader questions

-implement ring buffer in C

-lot of questions on RTOS/synchronization, mutex/semaphores, priority inversion

-Implement malloc_align(size, alignment) and malloc_free

- find closest 5 number in a array to a given number

- set bits in a 32 bit number to a give range

- find number of set bits in a number, how will you optimize (look up table, optimize further)

-atoi() with write your own INT_MAX, INT_MIN

-find endianness

-find occurrence of 32 bit number in a buffer on every bit find_occurence(char *buff, int buf_size, uint32_t num)

-EE related questions (review schematic, register divider, ohm's law)

-how fast UART receiver need to sample?

-how SPI daisy chaining work?

-why SPI faster than I2C? what is clock stretching?

-how will you figure out if 2 salve of same addresses are connected to I2C?

-why does high pullup increases rise time?

- diff SPI/UART/I2C

-add/remove elements at position n at linked list

-circular queue implement without size (using array)

-implement malloc (with fixed size array)

-lots of bit manipulation (find set bits (efficiently), set bits in range, reverse endianness)

-review schematic/code reviews

-sort linked list even/odd (maintain order)

-implement queue using linked list

-write timer function example with callback function [setTimer(*callback, duration, type ...)]

-send double pointer to function

-align malloc

-implement thread safe queue

- copy UTF-8 encoding whole characters only (check UTF-8 encoding)

- change nth bit to x

- implement sizeof

- lots of bit manipulation

- linked list seperate even/odd nodes

- simple python code/crypto-security questions

-write UART device driver (given registers)

-discuss a CPU profiler

-write coulomb counter, design H/W

-design HW/FW for a system to read 8 different Sensor values (using Analog MUX)

-RTSO basics

-Electrical Engineering/Electronics basics

Some useful links:

GitHub - theEmbeddedGeorge/theEmbeddedNewTestament.github.io

Tenouk's Ultimate C/C++ Tutorials Attributes C/C++ Preprocessor Directives C/C++ Type Specifiers Structure Member...

github.com

GitHub - mtdvio/every-programmer-should-know: A collection of (mostly) technical things every...

A collection of (mostly) technical things every software developer should know about - GitHub ...

github.com

If you find the article helpful, please consider connecting on **Linkedin**!

Let me know if there are more topics you want me to write about.

Firmware | Coding | Coding Interviews | Embedded Systems | Interview

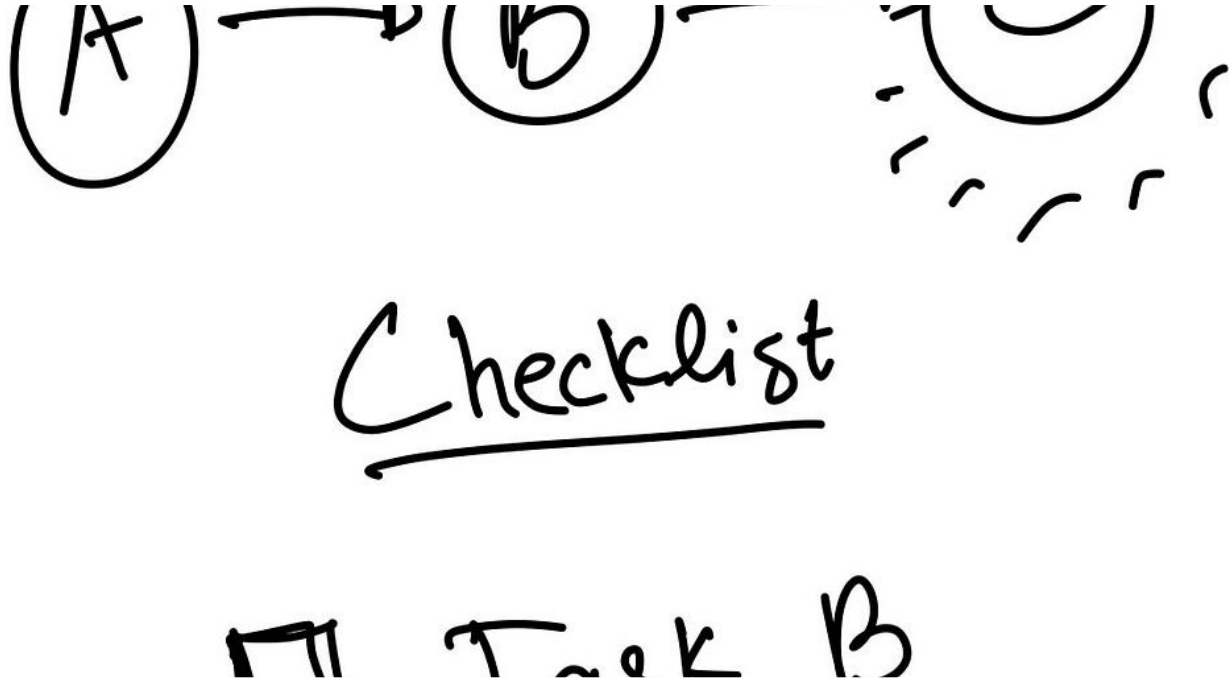
Follow

Written by Akash Agrawal

76 Followers

Firmware Engineer at Apple

Recommended from Medium



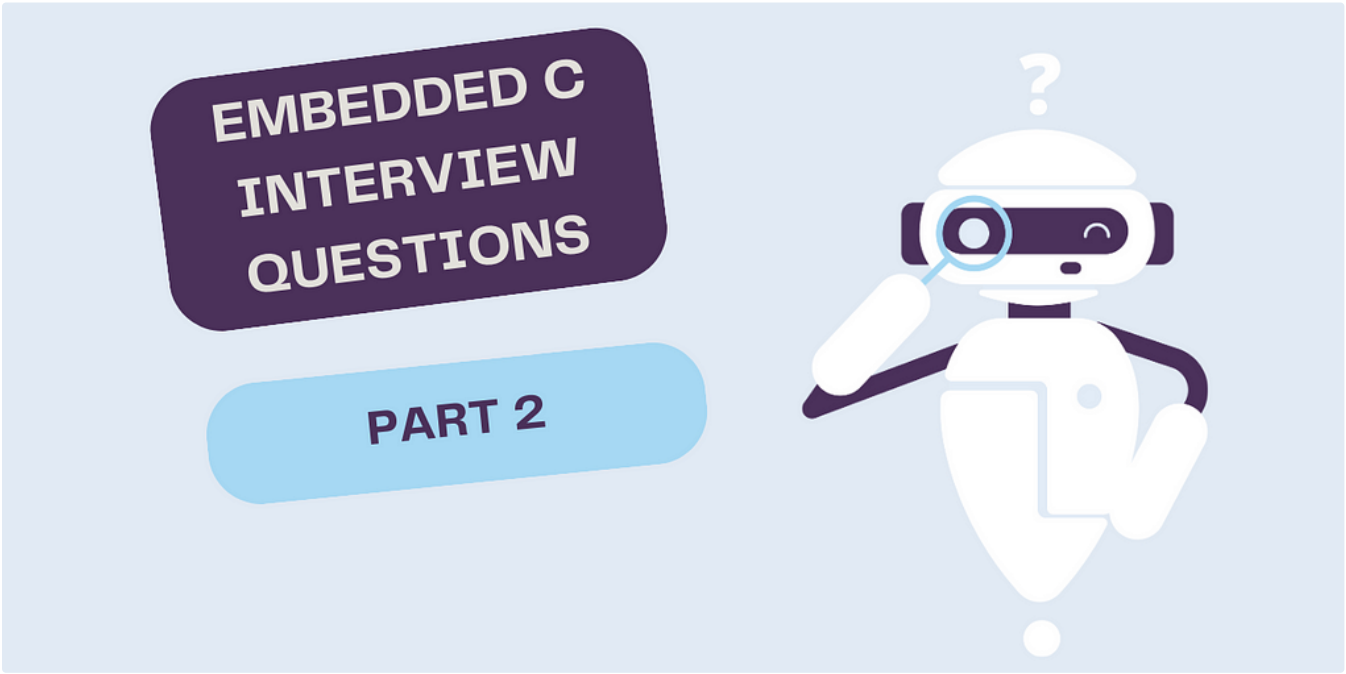
Rosmianto Aji Saputro

Why Following Roadmap Is Useless (For Embedded Engineers' Career)

When we learning embedded systems we look for roadmap to know what is the next step, and what are there to learn.

4 min read · Aug 28, 2023

6



Arjun Singh

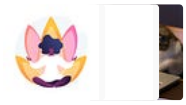
5 Embedded C Programming Interview Questions—Part 2

In the first part of this series we covered some of the most important interview questions and I can guarantee that atleast one of those (...

✨ · 3 min read · Sep 20, 2023

4

Lists



Stories to Help You Grow as a Software Developer
19 stories · 704 saves

General Coding Knowledge
20 stories · 782 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 265 saves

Icon Design
36 stories · 194 saves

Lance Harvie

How to Design an Embedded Software Architecture

In the world of engineering, Embedded Software Architecture plays a critical role in the development of efficient and reliable Embedded...

7 min read · Sep 21, 2023

28

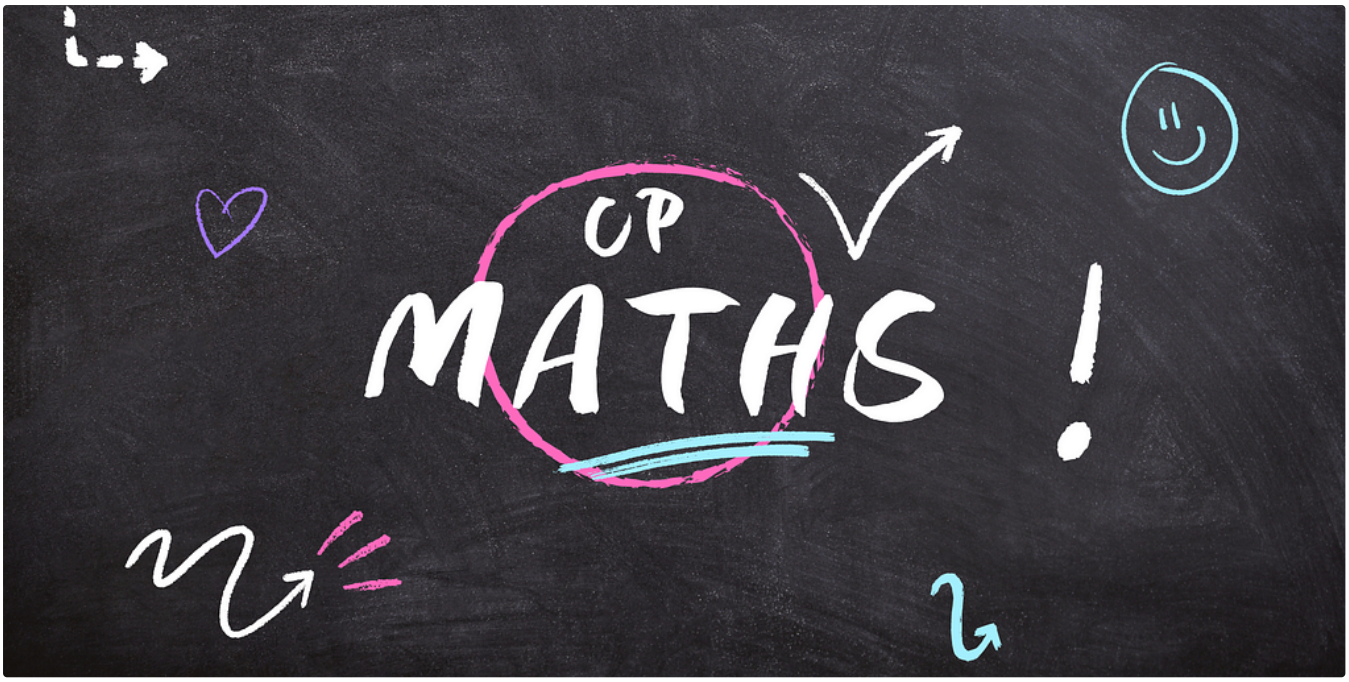
Rohit Verma

My Interview Experience at Google [L5 Offer]

Comprehensive Insights: A Deep Dive into the Journey from Preparation Through Interviews to Securing the Offer.

9 min read · Nov 26, 2023

5.5K 38



Yakub

The Essential Mathematics for Competitive Coding

Competitive coding, also known as competitive programming, is a challenging and rewarding activity where programmers solve algorithmic and...

5 min read · Jul 21, 2023

60

Arslan Ahmad in Level Up Coding

Don’t Just LeetCode; Follow the Coding Patterns Instead

What if you don’t like to practice 100s of coding questions before the interview?

5 min read · Sep 16, 2022

7.5K 45