11 companies, 55 Interviews, 9 offers including Google, Amazon (and Square?). Here’s what I have to share

[](https://medium.com/@yashgirdhar?source=post_page-----293852c1c98f----------------------)

[Yashasvi Girdhar](https://medium.com/@yashgirdhar?source=post_page-----293852c1c98f----------------------)

Follow

[Sep 19](https://medium.com/@yashgirdhar/11-companies-55-interviews-9-offers-including-google-and-amazon-heres-what-i-have-to-share-293852c1c98f?source=post_page-----293852c1c98f----------------------) · 12 min read

Photo by [Artur Matosyan](https://unsplash.com/@artmatters?utm_source=medium&utm_medium=referral" \t "_blank) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

I’ve been actively interviewing for past 5 months. Sitting through so many interviews and preparing for such a long time, I’ve learned a lot of things along the way. Things that could be useful to other people too. Through this post, I aim to share all of that and add more value to that public corpus.

I’ve been a mentor on [Interviewbit Academy](https://www.interviewbit.com/academy/" \t "_blank) in parallel to my interviews and have been telling the same stuff to my students too. So there’s a small validation that these methods work generally.

I won’t be preaching any life lessons here. Instead, I’ll be sharing with you a list of actionable items that you can readily incorporate in your process. I’ve implemented these myself throughout the process and that’s why I’ll try to provide concrete examples wherever possible.

I had around 3.5 years of experience when I started and interviewed for *SSE/SDE2/L4*positions, mostly for the Android platform as it is my expertise area. But I strongly feel that this post is equally useful to the ones applying for a general SDE. The only thing is that you won’t find resources for backend system design (which I think you don’t need much after grokking).

I want to share as much as possible and that’s why it’s gonna be a long post. So take a coffee, sit back and go through it slowly.

Just to give you more visibility into the process, I’ll start with a quick summary here, and then proceed to the learnings part. Here are the companies that I interviewed for:

Offers : [Booking](https://en.wikipedia.org/wiki/Booking.com), [Truecaller](https://en.wikipedia.org/wiki/Truecaller), [Google](https://en.wikipedia.org/wiki/Google), [Amazon](https://en.wikipedia.org/wiki/Amazon_(company)), [Phonepe](https://en.wikipedia.org/wiki/PhonePe), [Curefit](https://www.crunchbase.com/organization/curefit), [Dunzo](https://en.wikipedia.org/wiki/Dunzo), [GoJek](https://en.wikipedia.org/wiki/Gojek)

Process not completed: [Grab](https://en.wikipedia.org/wiki/Grab_(company)), [Square](https://en.wikipedia.org/wiki/Square,_Inc.)

Reject : [Agoda](https://en.wikipedia.org/wiki/Agoda" \t "_blank), [Uber](https://en.wikipedia.org/wiki/Uber" \t "_blank)

The process with Grab was never completed. The recruiter never responded after my 3rd interview. Let’s leave the details for some other day.

With Square, I was told that feedback for all my interviews was great. They wanted to move forward but there was no suitable opening at that time and they would get back to me after 2–3 months.

I’ve personally documented all of my interview experiences with all these companies. So feel free to reach out to me for help with the process.

Also, this being such a long exercise, I ended up making a lot of notes on different topics like android, java, algorithmic problem solving etc.

*I’ve created a public repository of all those resources*[*here*](https://github.com/yashasvigirdhar/Interview-Preparation)*. Assume that the repository is under the most liberal license out there and feel free to use it in whatever way you want.*

Let’s start with the learning part now. I’ve divided into different sections.

**Algorithmic Problem Solving**

This was THE most time taking (and most enjoyable) part of the process. I practiced problem solving only on Leetcode and I think it’s more than sufficient.

I had solved around 200 when I started interviewing and close to 350 (50 hard and ~175 medium) when I started with Google and Amazon.

These are some of the things that I recommend doing:

* **Discuss with others**

Even if you have solved a question, going through the discussion forum will expose you to a number of other solutions to the same problem. This is something that’ll really activate your “*problem-solving brain cells*”. There’s a global community out there which is actively solving and discussing problems on leetcode. *You CAN NOT afford to miss this gold*.

Algorithmic problem solving is something that improves proportionally with discussions you have with other people. So if you are lucky enough to have friends around you with whom you can discuss things, make use of that.

* **Mock interviews**

In a typical interview, you have to analytically think about solving the problem that’s presented to you. Along with that, you have to keep communicating with the interviewer your thought process, thinking out loud as we call it. And with both of these threads running, you have to keep reminding yourself that you have limited time.

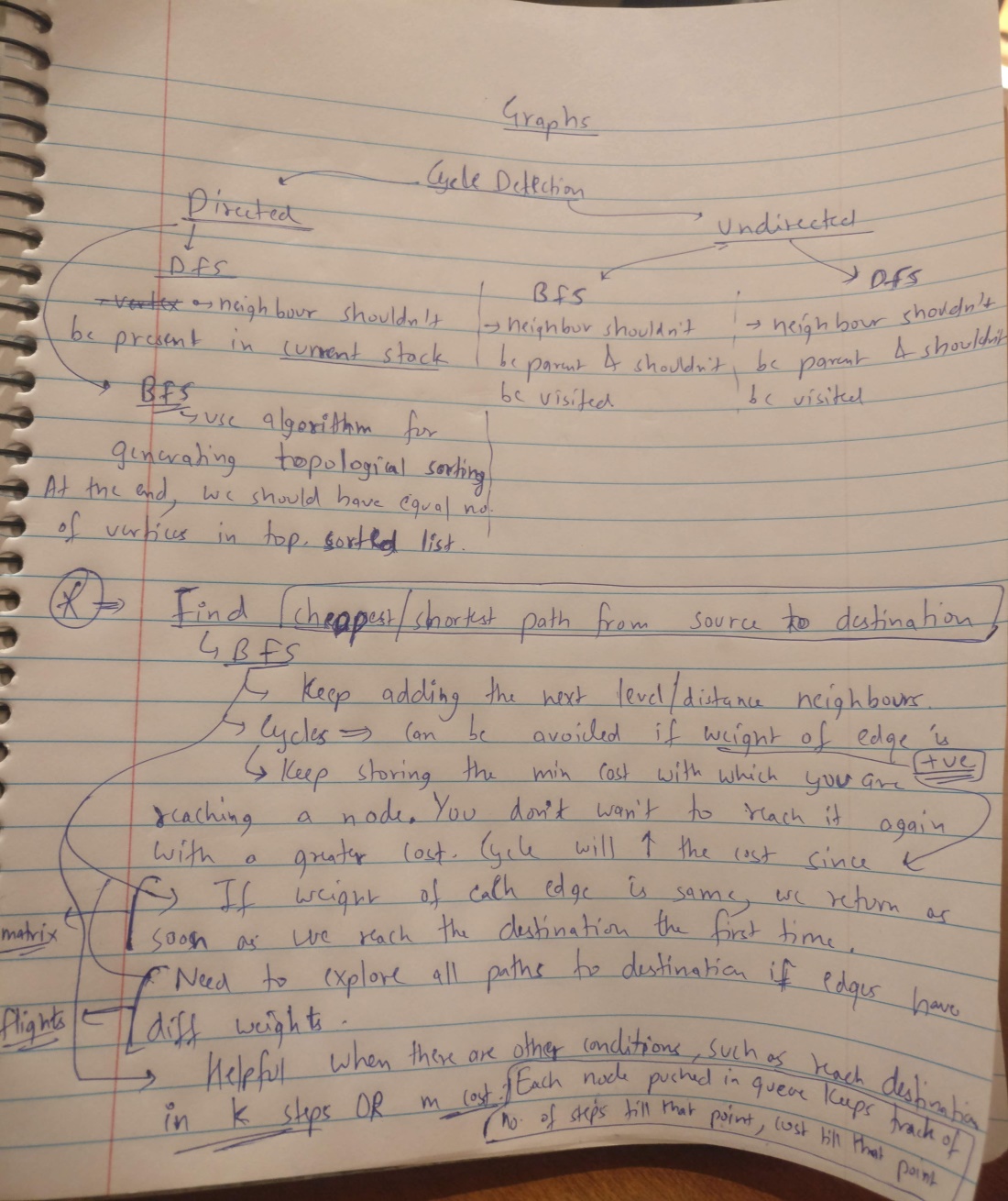
Now, our brain is not trained by default to handle a scenario like this. That’s where mock interviews come into the picture. Give at least 5–6 mock interviews before attempting for a real interview with a company you desire to work for. You might not notice consciously but your brain will eventually get trained for these scenarios,

* **Participate in contests**

They are an extension to mock interviews except the communication part. You have to come up with solutions to questions in a limited time, covering all the edge cases and reiterating on the solution until you get all the cases right. If you don’t practice this, it might push you on the back foot in a real interview when you discover that your solution doesn’t work for all cases.

* **Maintain your personal notes**

Example of personal notes on techniques for finding a cycle in a graph.



Whenever you learn something new while solving a question, add it to this doc. There are a lot of reasons to do this. For most people, this interviewing phase extends for at least 2–3 months. So all the things that you are learning during this time, there are good chances **you won’t remember everything before your interviews**, especially the most important ones at the end. To clarify myself, I am not asking you to memorize the problems and solutions, but the techniques that you learn.

Secondly, this also helps you to **incorporate**[**Spaced Repetition Learning**](https://en.wikipedia.org/wiki/Spaced_repetition)**into your process.** Go through the entire doc every two weeks/before each major interview. You can tweak this interval based on your feedback loop.

Lastly, writing about how you solved a question and what you learnt while doing it, *creates a personalized connection with the question*. There are good chances you’ll remember the technique if you are writing it (yes, it all comes down to basics). And you are likely to find a relation between two or more questions if you are documenting the solution for both of them. That’s our ultimate goal here as I’ll explain at the end of this section.

* **Planning**

As with everything, this is the most obvious and most important factor. Along with being consistent, smartly plan your days. Don’t just blindly solve some questions each day. I used to maintain a doc where I’d list down the topics/questions I’d cover the next day. And when would I write this? The night before. Because that’s when I was able to make the best judgment of what should I focus on the next day.

So these are some of the steps that I incorporated into my routine. But apart from all this, I learned one other lesson which I’d say is the most important one:

There are a finite set of patterns/categories in problem solving questions.

Most of the problems belong to one of these patterns. Let me explain with some examples.

If you take the topic of graphs, most of the problems can be dissected into:

* Finding the no of components
* Detecting/finding a cycle.
* Finding shortest path between nodes.
* Minimum spanning tree.
* Topological sort.

Similarly, a lot of problems in DP can be [categorized into ‘Largest Subsequence Problems”](https://medium.com/@yashgirdhar/intuition-behind-largest-subsequence-problems-3f82c8b4ba41).

For trees, if you ingrain the techniques used in solving these two questions, you shouldn’t have difficulty in approaching tree questions where you have to maintain state per node:

<https://leetcode.com/problems/binary-tree-cameras/> <https://leetcode.com/problems/distribute-coins-in-binary-tree/>

And in the case of arrays, master the ‘Sliding window’ technique which applies to a lot of subarray problems.

**In-Interview guide.**

So, you’ve reached this stage. Your interviews are coming up.

You need to compare an interview to a real world scenario in your job where you are given a problem and asked for a solution. What are the things that you would keep in mind then?

*Clarifying the problem statement, communicating your ideas with your teammates, thinking of all the edge cases, ready to take input from your peers, try to avoid that awkward silence when your peers are not sure what’s going on, asking people about something that you are not sure of.*

These are the exact things you need to focus on in your interview. Further elaborating on some points:

* **Ask for hints** if you are not able to reach a solution. There’s nothing wrong in asking for hints. And if you are able to take a hint and provide a solution after that, it’ll mostly be counted in your favour. I’ve asked for a hint in lot of my interviews.
* **If you end up with a jerk interviewer, DON’T let it screw your interview.** You don’t want to end your prospects of working in a company just because the person who took your interview was not the most skilled interviewer! How do you handle it? It’s difficult to generalize a single solution here but what matters is that you have to do your best. And you’d be able to give your best only if you are calm, are able to think properly and act proactively. You can’t do this if you are nervous . This is where Mock interviews come in.
* Make sure that **your interviewer is on the same frequency as you** are, all the time. Do it proactively. If you figure out that only you have been speaking for the last 25–30 secs, stop right there and ask them: If they want you to go ahead in the same direction, or deep dive into something else that you said before or stop at that point. Give them room to speak.
* **They need to know your positives in those 45 mins.**I’ve seen a lot of candidates who try to dodge interviewer from asking something they are not comfortable with. Don’t do that. It’s much worse than letting them ask and admitting that you don’t know. After the interview, they have to write a detailed feedback explaining why they think you are a good fit. Just solving the question won’t help, if you don’t give them enough evidence of your fitment.
* **Always try to come up with brute force solution first**. There are several reasons for this. First, at the beginning of the interview itself, you have given a solution that’s at least functionally correct. This gives you positive points and starts you off in a more confident position. Secondly, you now have a base that you can build upon and identify the *BUD (Bottlenecks, Duplicate work, Unnecessary work)* to optimize further. Third, your interviewer knows what you are doing so he might also provide some direction if he thinks that you are stuck.
* For the **HR and HM rounds,** **maintain a doc which has all the probable questions and their answers**. There are sometimes tricky questions in these rounds and even if there aren’t any, it’s better to prepare the content beforehand. If you are thinking about the answer at runtime, you might miss some details. These interviews also focus a lot on your ability to communicate your thoughts. So, if you already have an idea about what you are going to talk about, you can better focus on other things like this. [Here](https://github.com/yashasvigirdhar/Interview-Preparation/blob/master/HR%20and%20HM%20rounds.md) is an example doc which contains all the questions that I think you should prepare for.

*When I started driving, I couldn’t focus on the potholes coming my way because I was too busy changing gears and putting brakes. But when these things became my muscle memory, I could easily focus on the obstacles. That’s the point here. Train your mind and prepare your content as much as you can, so that you can give your best when dealing with the unforeseen obstacles.*

**Projects**

This is regarding your past projects, mostly the ones that you’ve mentioned in your resume. Before the interviews, write down each one of them in detail. Not just the project description, but also the related questions you think you could be asked. Do a dry run of explaining the projects with your friends.

If you are not able to explain a “great” project to your recruiter in simple language in less than 10 mins, then it’s of no use in an interview.

In fact it’ll just waste your time. I screwed up my uber interview specifically due to this. The project was complex and I was not able to explain it in simple terms.

Read about the [**STAR framework**](https://www.themuse.com/advice/star-interview-method) and try to use it when you have to explain ANY of your past projects in an interview.

**Learning Resources**

Apart from problem solving, there are a bunch of other things as well that we need to prepare. The [repository](https://github.com/yashasvigirdhar/Interview-Preparation) that I’ve created contains notes/helpful links about all these topics.

* **Primary programming language**: Mine is Java, so here are the things that I had prepared:

All the basic data structures such as HashMap, TreeSet, ConcurrentHashMap, ArrayList

Concurrency constructs that the language offers : volatile, synchronized, Fail-fast iterators, Thread pools, Executors, Runnables.

Generics , Reflection, Collections

Clarity about some basic concepts such as: Why collections don’t allow primitives, Why does java have a string pool, How can you break Singleton pattern in java, How enums work

* **Android**: As I was interviewing for a role specific to Android in most companies, this was something that I prepared most. This is a vast topic and you need to narrow down the topics that you want to focus on. But there are some basic things that you are expected to know such as

*Architecture components, storage options, background processing, location, threads and processes, battery optimization, network optimization, espresso and mockito, recyclerview and major android components such as activities, services, fragments, content providers.*

I’ve compiled a list of all the questions that were asked to me in android tech and design rounds. Will be sharing them in a separate post.

* **Networks** You should know the basics, mostly around transport layer and application layer (TCP, UDP, HTTP, TLS, DNS). *‘*[*High performance Browser Networking*](https://hpbn.co/)*’* is one of the best books I’ve encountered on networking. You can read the whole book if you are interested but for interviews, Networking 101 section should suffice.
* **Concurrency**: Irrespective of the role you are applying for, a clear understanding of the basic concurrency principles is a must. ‘[Concurrency In Java](https://amzn.to/34RPOjD)’ is considered a bible by the community. The book is a must read for every software engineer I think. Again, the first five chapters should be sufficient for interviews. If you are short on time, I’ve compiled some notes from the first four chapters [here](https://github.com/yashasvigirdhar/Interview-Preparation/tree/master/concurrency/concurrency%20in%20java), but it’s strongly recommended to read through the complete chapters.
* **Operating Systems:** Being a software engineer, I think you should know about these topics since this knowledge will be used in your day-to-day job. Those fresh out of college, you need to have a strong grasp of these concepts. Acclaimed college textbook - [‘Operating System Concepts’](https://amzn.to/2QhgdEv) is a comprehensive and reliable source here. First 4–5 chapters should be sufficient for interviews but do take out time to finish it after the interviews.

There’s a Database section too but I am sure you can cover that.

**Stick to the plan**

It was not easy for me to stick to this plan for 4–5 months. So over the course of this period, I kind-of organically developed/observed some techniques which helped me to stick around for so long and make it less hectic (Actionable preaching :) ) :

* **Find a friend**with whom you can discuss things, discuss every detail of the process as it’s happening. There will be a lot of points in the process where it’s good to take a third person’s opinion on things. Deciding on when to schedule an interview, how to negotiate an offer OR, you’ll mostly be burning out at some point in this process, maybe multiple times. At that point, it’ll of course help to talk to somebody about this. But if there’s this inertia to first explain the whole process to someone and then ask their opinion, it’d be difficult to execute that as compared to just call a buddy and ask his/her opinion.
* **Feedback Loop**: This applies in general to any process in life that extends over a period of more than 3 months. You need to incorporate a feedback loop, proactively. For eg if interviews are not going well, don’t hesitate in postponing the upcoming interviews. Thoroughly document and analyze all your interviews, so that you are aware of the areas that need more work. This might sound like an obvious advice, but doing this strictly requires dedication. Deciding the order of the companies in which you want to interview is another scenario where feedback loop comes into the picture.
* If you have already quit your previous job (like I had) and are not doing anything except this interview preparation, do something to **channelize your energy**. This will greatly affect your productivity and minimize those burnouts. Running, Sports, Meditation.. whatever works for you.
* **Small checkpoints**: All this might look quite a lot in the first go and it is. So don’t aim to achieve everything at once. Set small, incremental targets and focus only on those targets. Examples of these targets can be as simple as 100 leetcode questions OR solving 2 questions in a contest. And celebrating these small wins will provide you the necessary break that you need.

*That’s all ! Congrats for making it all the way through this long post. I’ve tried my best to share everything here but might have missed certain points as this is a vast topic.*

*Please feel free to contact me here or on [linkedin](https://www.linkedin.com/in/yashgirdhar/" \t "_blank) for any questions or feedback.*

*I plan to write more posts on this topic, especially around android design interviews as that’s something for which I didn’t find much help when I was preparing.*

*Do get in touch if you want me to cover any other related topic!*

*Cheers !*