End to End guide on getting a Job in Tech industry

By Mohsin Ali

Content Index:

Background:

Beginning with some background introduction. I did my undergrad from LUMS in Pakistan majoring in CS, although a good school but not on the radar of all recruiters here in Bay Area. In the middle of my second year as a PhD student at University of Southern California, I decided to apply for jobs and graduate with a Masters instead. I wouldn't say I was particularly a hot candidate for companies since I never interned at top tier companies, infact over years I failed to clear Microsoft and Google interviews even for internships. I applied to 25 companies and 8 of them responded. I cleared HR and phone screens for all of them and visited their campuses for onsite interviews. Overall, I went through 56 interviews in total and secured job offers from **Facebook, Microsoft, Amazon, VMware** and **Cisco.** I spent around 8 weeks in the whole job search process where 6 weeks went to preparation and 2 weeks for interviewing. I used to prepare 60 hours a weeks in the preparation phase which means an overall effort of ~375 hours. For reference, here is my resume.

Resume Writing

Spending time on resumes is one of the most undervalued aspects of job hunts. The multiplier on gains you get on the time spent refining your resume down to the last word is huge. Writing a resume is one time effort and then it would be used in practically every application. The quality of your resume determines whether a recruiter calls you back or not and that can be the difference between getting the job. I would have spent around 75 hours refining my resume. While writing a resume, keep in mind that this resume is mostly for you to get an interview and is targeted for recruiters, there is very little value of your resume after you get an interview. I will summarize some of the aspects of resume writing but I highly recommend going through blogs/quora online for at least 20 hours to get a feel of what recruiters are looking for in resumes.

- **Organization:** As a rule of thumb, keep your resume to one page and readable within 30 seconds. Here is a suggested format template I used and here is my own resume for reference. There is some great advice on resume writing from Gayle Laakmann (Author of Cracking the Coding interview).

- **Content**: Remember recruiters are your target audiences and they have piles of resumes like yours so you have very limited time to distinguish yourself. You can't afford to dilute your resume with fillers and just focus on the strongest points. I can't emphasize enough on how crisp each and every bullet should be. Moreover, the resume should be as non technical as you can make it because the recruiter is just looking for keywords. Each bullet has a format, it starts with an action word for example, "implemented, designed etc". If any of your bullets is not starting with an action word you should seriously reconsider that line. Then, it is followed by what you did in 4-5 words which is the body of bullett. This body is usually a good place to add keywords. The ending of your bullet is crucially important. It should be presented as a valuable result or finding to make overall the bullet an achievement; Something like "improved throughput by 10 times". It is super important that each and every bullet is an achievement not a responsibility. The worst thing you can do to yourself is to write a bullet which seems like an Activity. In my experience, most of the activity bullets can be rephrased to become achievement bullets.

    - Activity (Discouraged): *Used DPDK and cache aware access mechanism to reduce the the number of cache misses and hardware interrupts*

    - Achievement (Encouraged): *Implemented an efficient cache access mechanism on top of DPDK to achieve 8 times higher throughput in some distributed systems*

- **Get Feedback (Super important):** It is important to get multiple rounds of feedback from people with different backgrounds. Every time I finished a draft of my resume, I thought this is the best version I can come up with, however after getting feedback there was always something I could improve and eventually I had to do 8 iterations over my resume. You would want one person who can check your grammar, typos, and sentence structures. Then you have resume reviewers at almost all universities career offices, who will give you insights on how a recruiter will look at your resume and then finally a technical person who helps you build a coherent story about your technical background. Don't have any more than these 3 reviewers, one for each aspect and have them review your resume twice and incorporate their comments. Its great to have a google doc version of your resume to get feedback easily from everyone through comments so you can have a follow up discussion.

- **Skills:** This is something unique that I tried. Recruiters seem to pay a lot of attention to your Skills section and see if you match the job description. They don't know what Cloud computing is but if it appears on both your resume and the job description then you are highly likely to get a call. I gathered all the 15 job descriptions I was interested in, parsed them for keywords. This will give you an extensive list of skills needed but you can't use more than one line on your resume for this so I consolidated multiple skills into something more general, like hadoop, mapreduce, hive etc all get covered by "Big Data Analytics". Similarly for all other skills. This line will essentially define what sort of job you are looking for and if it's very different from the job description you might not get an interview. Think of them as keywords you would use to lookup the job on LinkedIn that you want to end up with. Here is the skills section from my resume.

  Languages: Java, C, C++, Go, Python, Bash, MySQL, MATLAB, HTML, CSS
  Unix, OS Virtualization, Kernel Programming, Big Data Analytics, Cloud Infrastructure, Networking, DPDK

Getting an interview:

You might think applying through the online portal is enough but it's not true. Millions of people leave their resumes at those portals and most probably some automated systems filter out many of them before a human gets to read them. Apply through the online portal but donot solely rely on it to get you noticed.

Referrals:

Best way to get a job interview is through a referral. According to Forbes, around 50% of jobs at Cisco are filled through a referral. So reach out to your friends, colleagues, etc. and search for people you may know at a company on Linkedin and ask them if they are willing to refer you. They get referral bonuses if you are hired so they have incentive as well. At times friends of friends might be able to refer you as well so don't hesitate in asking friends if they know somebody at company XYZ. Moreover, ask your Professors if they are willing to refer you to their contacts in the industry. One of the referrals from my Professor was so strong that they offered me a job without even interviewing. Professors tend to have contacts with more senior employees and as you might expect, referral coming from a senior employee has much more weightage as compared to a referral from a new hire.

Reaching out to recruiters:

The next best thing is to find recruiters of different companies on LinkedIn and parse through their profiles for email addresses. Send them a mini cover letter saying why you are a good fit for their company and attach your resume as well. The hit ratio for such emails is very low, may be only 5% recruiters will reply but even if you get one interview out of this it's worth it. I landed a job at Facebook by a similar email and otherwise I don't think I would even have got a chance to interview with them. I have added the email as a reference. Moreover, I got an interview call from Google using a similar email.

"Hi Jaimee,

Hope you are doing well. I am a Master's Student in CS at University of Southern California and will be graduating in May 2016. Currently I have a 4.0 GPA and have taken many relevant advanced courses like advanced distributed systems which provide a good background for Facebook. I have also done research assistantships focusing on scaling large scale distributed systems. It would be great if you could have a look at my resume.

Regards,
Mohsin"

The important thing is to not get disappointed by lack of replies, these recruiters get a bunch of emails like this. I must have sent over 75 such emails but only 6-7 replied but in the end it got me the job I wanted all along. Also, don't hesitate to send emails to multiple recruiters from the same company.

Preparing for an Interview:

Why spend time on preparation

First of all, you have to realize how important it is to prepare for interviews. I used to think over years that I am good at coding, so interviewing would not be a problem for me. It took a rejection for Internship from Microsoft and a rejection for internship from Google during grad school for me to realize I needed to prepare for interviews. Actually there is very little correlation of how good a programmer you are at school with your performance in an interview. It is a totally different skill in a completely different setting and you have to work hard to acquire this skill. **Its very simple to but not easy.** You have to be consistent, sit down and practice a lot. The thing is that you are graded at a relative scale during the other interviews and all other candidates spend a significant amount of time preparing. Even if you get the most optimal solution during the interview, other candidates might have already seen that question and they would crush their interview in less than half the time you took resulting in you getting a rejection. There is a finite set of "Interview Questions" which is not more than 300 or so questions, the more you practice the better shape you are in for the interview. Actually 60% of the interviewers asked me a question very similar to some question I already had seen during my prep so I would quickly solve it and move to the next question. Therefore, within the first 15 minutes of the interview I was already ahead of a lot of the other candidates. Google is an exception to this, all their questions are new and tricky so doing enough practice would also develop a good sense of solving unknown problems.

Even if you think that **somehow** you are very good at interviewing and don't need to prepare, your performance in the interview actually determines what SDE level will you be brought in as and what salary package is offered to you. A recruiter from Facebook actually told me that they extend salary packages ranging from 100k to 125k for new grads depending on how they perform in their interviews. Not only this, the signing bonuses and stocks you get which can be vary from 5k to 150k depend on your performance in an interview.

Google's Grading Rubric

In order to prepare for the interview it's important to figure out what metrics are companies assessing you on. Grading rubric for Google looks like something below:

1. Problem solving and analytical abilities (1 point)
2. Data structures and Algorithms (1 point)
3. Coding (1 point)
    a. Ability to translate algorithms into code

    b. Ability to write bug free code
    c. Ability to write formal code (not pseudocode)
    d. Ability to write elegant code
4. Communications skills (1 point)
    a. How well can you "teach" them your solution?
    b. Also known as "Would I want to work with this person?"

All interviewers score you out of 4.0 and 3.5+ is an enthusiastic hire. In order to get a job at Google it's often said that you need all 4 interviewers to give you an enthusiastic hire leading to Google's obscenely high hiring bars. However situation in other companies is much better.

Brushing up Algorithms and Data Structures

Cracking the coding interview by Gayle Laakmann should be your Bible to the interview prep. Anything written in that book should be on your fingertips. I went through the book cover to cover twice but mainly focusing on first 10 chapters. Book has 150 questions and if you solve them on your own and code them on paper before looking at the answers it would brush up your problem solving skills and provide a good start. Never forget to go through all the answers in the solutions section because that is the time you are learning the most. You already knew the code you wrote so merely solving the question does not help you learn anything new but going through the solutions and realizing what you could have done better adds a lot of value to the preparation. Analytical abilities include how you can reason about the space time and runtime complexities of a solution and discuss their tradeoffs and the book goes into a lot of detail on this. Infact any advice that Gayle Laakmann gives is great, I followed her careercup posts, quora posts, Youtube videos to get insights into the interview processes.

It’s a great idea to brush up on basic data structures from CLRS which does not take a long time. However just reading them up is not enough. This grading rubric is a cryptic way of saying "Did the candidate use most efficient DS to solve the problem". Therefore, it’s important that you get enough practice on using the right DS at the right time. Moreover, you should know all the internals of all basic data structures and how they are actually implemented (yes some interviewers do ask you to implement them). If in your undergrad data structures you were not asked to implement them from scratch now would be a good time to do that before you jump into the real interview questions which use them. Even all this is not enough to ace this point, practice using these data structures is the key and will be discussed in more detail it in the next paragraph.

Improving Coding Skills

As students we are used to writing code that just works. In an interview each and every line of code that you write matters alot. You need to practice writing elegant and clean code and force yourself to use the good programming practices we all know but are too lazy to actually adopt. I strongly suggest InterviewBit for practicing to code. It has extensive coverage of all the interview style questions and their format is such that you get addicted to solving problems since there are daily goals and each question yields some points depending on how well you did in that question. Most of my prep was on this website and it significantly improved my coding habits as well. They have questions based on all the interview topics and commonly used tricks. Like you might not know "Backtracking" and "Two pointers" are 2 very popular classes of interview questions. In order to complete the interviewbit "course" you are forced to clear at least one question from each sub topic. Recently facebook also started collaborating with them to provide their own questions. I know there are others like leetcode and hacker rank but questions here are very targeted.

Another common pitfall for us students is that we are not used to test our code. You need to acquire the skill of coming up with complete and mutually exclusive set of test cases which would test all scenarios of your code. Interviewbit actually helps you acquire this as they rigorously test your implementation with all sorts of corner cases and struggling through uncommon test cases like invalid inputs you get trained to write bug free code in the actual interview. Also, even when you get all the test cases running don't forget to go through the solution code and learn from the good practices they use in their code. For example, I adopted the habit of using ternary operators whenever possible which made my code look much more cleaner. Overall I solved around 100 questions twice from Cracking the Coding Interview and around 125 questions from Interview Bit which translated to around 22000 points. Its very important to stay consistent in your prep solving 3 questions every day (21 in week) is much better than solving 21 questions over the weekend and not doing anything else for the rest of the week.

Importance of Mock interviews

People tend to bomb their first few onsite interviews. Infact I bombed my first 4 sets of onsite interviews before I got enough practice to ace the next 6. You should use mock interviews as much as possible to bomb so you actually learn from your mistakes before going in. Also companies never give you feedback on what you did wrong but your friend / Prof / Colleague can give his honest opinion on where you lacked. Moreover, If you are not well rehearsed you might take pauses here and there or stutter a few times. These are not good indicators for the interviews as it seems to make you less confident about what you are talking about. Communication skills are extremely important to an interviewer so it is important for you to practice them.

Have an interview buddy and schedule a at least 15 odd of mock interviews before your first onsite. Mock interviews also help develop a set format on how you would walk an interviewer through the solution and how you would solve ANY problem and use these mock interviews to practice your strategy. I highly recommend going through these Paid Interview Videos by Gayle Laakmann where she conducts mock interviews and gives feedbacks. Tells you a lot about the interview process and what the interviewer is looking for.

Personally my format to attempt any problem was to divide the board into 4 sections, Code, Test cases, Examples and Candidate solutions with time and space complexities. Then the goal was to start from an example, figure out the algorithm to solve the problem, add the solution to the solution section, keep iterating until you find the most optimal solution, have the interviewer on board with your algorithm, write test cases and THEN start coding followed by thorough testing. Its super important that you don't jump into code before actually having a clear picture of your algorithm in mind. I realize it's super tempting to do this specially when you have limited time therefore the mock interview are to practice and learn to resist this temptation. Writing test cases before writing the code is something that people in industry really appreciate. I had quite a lot of interviewers compliment me when I went to the board and wrote test cases before the function prototype. Moreover, when you are done with the code the worst mistake you can make is to say you are done without actually dry running all your test cases. If your code is already correct chances are interviewer will stop you and move on to the next question or you find a bug so it’s a win-win. Having test cases already on the board helps you remember that you have to run them once you are done with writing the code.

Once you have done enough of these mock interviews, you will be on auto pilot mode for your interviews and everything should go very smoothly. Definitely buy a whiteboard and use it for these mock interviews, coding on paper and computer is very different from whiteboard coding as space is very limited. You need to familiarize yourself to whiteboard coding before actually going onsite. Also help out your friends by interviewing them, it's a great learning experience since you realize how things look when you are sitting on the other side and it helps you improve your interviewing skills. I would recommend scheduling free mock interviews on www.pramp.com which matches you with other students preparing for the same company and then you conduct mock interviews for each other in a 1 hour long session. This reduces the nervousness level when you give an actual interview.

Going in for the interview

A lot of tips in the mock interview section are valuable when going in the actual interview. Its important that you familiarise yourself with the specific interview process of the company before going in. Glassdoor is a great resource as well as quora to read up on others' experiences. You should also do research on the company before going in. For example, the team I was interviewing for at Facebook was working on Facebook ads so I read up on the ads infrastructure of Facebook which really helped since I was asked very targeted questions on improving them. Similarly don't forget to do research on the team before going in and have some very specific questions prepared for them as your questions at the end of interview show what things are important for you and you are being judged on the questions you ask. Just be calm and make sure you are oozing with confidence before you step in because it makes all the difference and for me extensive preparation was the only source of getting this confidence.

Resources

Following are all the resources I used for my interview preparation in the order of importance.

- Cracking the Coding interview
- Programming interviews exposed
- Paid Interview Videos by Gayle Laakmann
- Interview Bit for coding
- Pramp for mock Interviews
- Geeks for Geeks
- www.glassdoor.com

Must do interview questions

Each topic has some building block questions which end up being used by a lot of solutions. It is important that you already know solutions to all these problems so you don't waste time thinking up the solution to these problems and focus on the question interviewer has given you because you will be judged on that. I have included some of these questions but if you go through Interview Bit in detail you would eventually come across/cover most of these questions

- Arrays/Strings:
  - Determine if a string is a palindrome
  - Merge two sorted arrays
  - Reverse an array in place
  - Find substring
  - All sorting algorithms
  - Binary search in a sorted rotated array
  - Max profit stock problem
  - Matrix multiplication
  - Find all duplicates in an array
  - Print a matrix in a spiral manner

- Linked List:
  - Reverse a singly linked list
  - Delete/Insert a node in a linked list
  - Detect if there is a cycle in the list and return its starting point
  - Merge two sorted lists
  - Split a list into two lists one has even indexes other has odd indexes

- Trees:
    - Check if tree is balanced
    - All traversals, recursive and iterative implementations
    - BFS/DFS
    - Construct a BST from a sorted array
    - Check if two trees are mirror image of each other
    - Find max path sum in the tree, negative nodes possible
    - Lowest common ancestor of 2 nodes in a tree

- Backtracking:
    - Find all permutations or combinations
    - Find all possible subsets
    - N queens problem
    - Convert numbers into words according to letters on an old phone keypad

- Hashtables
    - Questions where you need to keep track of multiple occurences of same object
    - Questions where you want to have a 2 tuple as a key

- Dynamic programming:
    - Given you can climb 1,2, or 3 stairs in one step, how many ways of reaching the top
    - How many ways to go from top left of a grid to bottom right of the grid with some obstacles in between
    - Implement both bottom up and top down solutions for both of the above

Keep in mind that this is not an exhaustive list, only a starting point. You should be doing at least an order of magnitude more questions than the ones listed above.

Best of luck in your job search, if you need any more information, feel free to drop me an email at mohsin80211@gmail.com.