Dev.Ed
June 2022

Technical Interviews: How to Study

Leetcode is an essential part of preparing for interviews. Here are additional exercises that would help you avoid the biggest pitfalls I see in interviews.

Exercise 1: Pattern matching.
This is a speed exercise. Open a list of leetcode problems you have never solved. Start a stopwatch. Read each problem, and without solving them or writing any code, guess their solution structure and write it down. Then move to the next one. Continue until the last problem and stop the timer. Now go through the solutions for each problem and see if your solution structure guess was right. With each exercise, try to reduce your time. Why do this exercise? One of the biggest pitfalls I see in interviews is that people do not recognize the problem's solution pattern and get lost in the weeds trying to invent one. The truth is, there is a small set of common solution patterns that nearly every problem falls into. Problems are then made unique by having their own trick that is integrated into the pattern.

Here are a few examples of common solution patterns, but there are many more. They are searchable on leetcode. It might help to start by searching for questions by category to get a feel for what they have in common. It can make the solution patterns easier to recognize.

- Two-pointer pattern solves:
  - reverse a string
  - detect linked list cycle
  - etc..
- Dynamic Programming:
  - Fibonacci number
  - pascals element
- Graph search (bfs, dfs, a*, ucs, etc):
  - count number of islands
  - print right view of binary tree
- Sliding window:
  - contains duplicate
  - max consecutive ones
- Backtracking:
  - sudoku solver
  - word search

Exercise 2: storytelling.
Gather a list of common behavioral questions and write answers to them in STAR format. Rehearse them as often as possible. I recommend making them into flash cards. Any story you mention, make sure it involves a software project that you can easily answer additional questions on. Why this exercise? Companies often care a lot about your storytelling. The things

you say and how you say them are used to understand how well you can communicate ideas to your teammates concisely. By practicing these questions, you will be able to explain the answers more easily when under interview pressure.

Once you are scheduled for an interview, some companies will send you a list of soft skills (examples), leadership principles (Amazon), or company values (Microsoft) that they test on. If you receive one of these, go through each one and come up with a story about it. For example, if one of their values is empathy, come up with a relevant story about a time when you demonstrated empathy on a team.

Exercise 3: mock interviews.
One very important part of preparing for a technical interview is practicing all these skills in front of an evaluator. Many interviewers will be more likely to evaluate you favorably if you have practiced away the bad habits before the interview. If you are unsure whether you are interviewing well in these practice sessions, find someone experienced in conducting interviews who you can ask about what good candidates do.

There are many ways to set up mock interviews including:

- Scheduling time with friends or mentors
- Using a mock interview site like pramp
- Finding a CS career community (discord, reddit, etc) and asking for mock interviews. Many of these communities will have a subsection specifically for this!
    - Discord public servers (there are probably more than this now):
        - CS Career Hub
        - Cscareers.dev
    - Reddit:
        - Search mock technical interview
        - Search mock interview
        - Search CS careers

Extra credit:
Solve a problem and…

- Explain your solution to a non-tech friend
- In a separate document, isolate every variable/function/class/etc name into a list, see if it is self explanatory when pulled out of the solution
- Try to come up with a different solution
- Evaluate your time and space complexities, then do the same for the other solutions on the internet
- Optimize your solution
- Argue for why you used that algorithm/data-structure/etc rather than another one
- Identify and walk-through edge cases
- Create unit tests
- Identify what inputs would make your solution fail (what if the guarantees for the input such as “in ascending order” or “all values are unique” were not honored?)