SoCal - Onsite Interview Prep

Our interview process for technical positions evaluates your core software engineering skills, such as coding, algorithm development, data structures, and analytical thinking.

During your interview, you'll meet with several engineers from a range of teams and seniority levels, who will ask you to solve problems in real time. Remember, it's not just a matter of getting the answer right or wrong, but how you work through the question. And even if you think one of the interviews did not go quite as planned, do not assume that the day is lost as we review the overall performance.

In general, we attempt to stay away from puzzle or "a-ha" questions (questions where you either get an answer or do not).

Coding
Coding is an important part of the Google interview. After all we are looking for software engineers. We are trying to evaluate your ability to write code that is correct, efficient, simple, readable, and maintainable. While we would love nothing more than to see compilable code, we realize that producing it in real time with someone watching you is hard. It is important to show us how you think and what implicit assumptions you make. We look for software engineering, testing, and debugging skills. We also gauge how well you understand (as opposed to remember) concepts, and what your problem solving capabilities are.

Choose the programming language in which you are most comfortable. While we prefer Java/C++/Python, if your lack of understanding of the language will prevent you from expressing your thoughts clearly, it is best to check with your interviewer whether they'd be ok with an alternative. Consider that some languages are more "interview-friendly". If you must go forward with, say, C, consider discussing with the interviewer what assumptions you can make about helper functions and/or libraries.

Work through simple examples.

Think out loud. We know it's often not natural, but we would like to learn as much as we can about the things that matter. Describe your approach to the interviewer before jumping into coding and think about the interviewer's feedback; it's much better to adjust your approach at an earlier stage, than after the whiteboard is filled.

Use helper functions. Use short, but understandable variable names. It is very difficult to find a mistake in the whiteboard code when one has 4-level nested for loops referencing i,j,l indexes.

Don't get caught up in trivialities, but be very clear with an interviewer about what shortcuts you are taking. When you have a solution, think of edge cases and step through the code.

Draw. Whiteboard is not just for writing code. Often it is very helpful to diagram.

Data Structures
While thorough understanding of the more common data structures and algorithms is integral to the interview, this is not an advanced algorithms course final. Even though it is at times helpful to lean on your knowledge of some of the less frequently used data structures or algorithms, it's likely that preparation based on deeper understanding and practice will be more beneficial than the approach skewed towards breadth.

A few other observations:

- Know your common data structures inside and out. Some examples:
    - Array
    - List
    - Stack / Queue
    - Set/Map/Hash Table (understanding the differences is important).
    - Tree (traversals, manipulation) / Trie. For example, intricacies of balanced trees do not come up often, but knowledge that such data structures exist, guarantees they provide, and rough idea how they work does.
    - Priority Queue / Heap (do you understand the difference?)
    - Graph, Adjacency matrix (traversal, path finding).
- Understand common operations and their time complexity.
- Review common applications for the data structures. What algorithms tend to go along with them? (i.e. path finding/graph/Dijkstra, or word dictionary/trie).

Concepts and Algorithms
These concepts tend to come up in the interviews often:

- Complexity analysis
- Sorting and searching
- Dynamic programming / memoization
- Object-oriented design
- Graphs & path finding
- Recursion
- Bit manipulation

System Design

Depending on your level of experience, we may ask System Design questions (your recruiter will let you know if you should expect one). These take many forms, but most touch on fleshing out product requirements and system architecture, reliability, scalability, and performance (e.g. “Design a URL shortening service”). Listen to where interviewer is steering the discussion - did they hear an interesting direction and are digging deeper into privacy or security? Logging? Testing?

Pretend you are having a dialogue with yourself - will your design work? Will it work at Google scale? Is there a client component?

- If you had never seen or thought about a problem of this sort - how would you approach it? How would you collect data and make decisions on how to proceed? What information would you need and what would you do with it?
- If you have dealt or thought about similar problems, this is a great opportunity to draw on the experience - How do you know your design works? How will you test it? What are the hot spots/pitfalls? What is the scale? Can you back-of-the-envelope it?

When asking this type of question, interviewers are usually not looking for a particular answer. As with other questions, they are hoping to learn how you think. Do not try to anticipate which points interviewer is looking for you to hit - if you are methodically working through the problem, they will frequently point you towards something interesting they have heard and will ask you to go deeper.

Some tips:

- Ultimately, we are looking for a coherent design, as opposed to a set of disconnected ideas (are you thinking about “life of a request”?)
- It is a good idea to start at the high level - by ensuring you understand the product requirements - and then proceed down the abstraction levels by “drilling in” to the more interesting aspects of the question; observe whether the interviewer is trying to channel the discussion in a particular direction.
- Do not anchor on existing solution - when asked to “Build Twitter”, we are trying to describe the problem, not asking you to guess the solution. How would you build it if it never existed?
- Diagram, diagram, diagram

The best way to prepare for such interview is to attempt to talk through a question like that. Ideally with a friend who can ask you to clarify or expand, but even talking out loud may be helpful.

Helpful Links:

How We Hire

Candidate Coaching Session: Tech Interviewing

Technical Development Guide

About our Company

Company - Google

The Google story

Life @ Google

Google Developers

Open Source Projects

Github: Google Style Guide