Build
for
everyone

Google Interview Prep Guide
Senior Software Engineer

What's a Senior Software Engineer (SWE)?

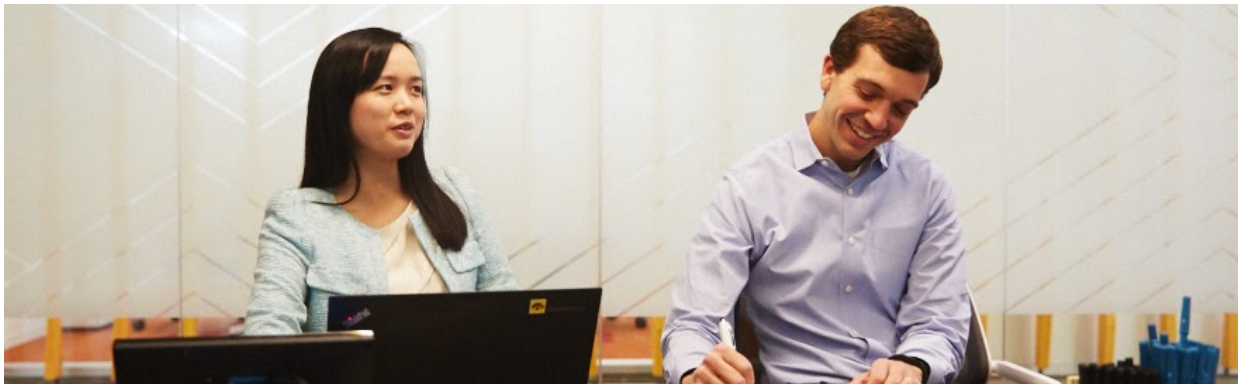
Software Engineers (referred to as "SWEs") at Google develop the next-generation technologies that change how millions of users connect, explore and interact with information and one another. As a SWE, you'll be responsible for the whole lifecycle of a project critical to Google's needs, with opportunities to switch teams and projects as you and our fast-paced business grow and evolve. Depending on the project you join, you could be involved in research, design, planning, architecture, development, test, implementation or release phases. You'll be working on products that handle information at a massive scale, bringing fresh ideas from all areas and tackling new problems across the full-stack as we continue to push technology forward.

As a Senior SWE, you'll tackle large, open-ended problems where the nature of the solution may be unclear when the problem is first presented. You'll design and implement software along with your team while providing mentorship to junior engineers and guidance on complex technical problems. You will also continue to develop and demonstrate your own expertise, while delivering solutions that are scalable and maintainable.

Some Senior SWEs may carry "Tech Lead" (TL) titles, even if they're not formally managing teams; they're leading technical direction for projects, doing the majority of design and architecture, plus writing significant amounts of code for critical components.

Why Google? Impact.

Google is and always will be an engineering company. We hire people with a broad set of technical skills who are ready to tackle some of technology's greatest challenges and make an impact on millions, if not billions, of users. At Google, engineers not only revolutionize search, they routinely work on massive scalability and storage solutions, large-scale applications and develop entirely new platforms around the world. From AdWords to Chrome, Android to YouTube, Cloud to Maps, Google engineers are changing the world one technological achievement after another.



General Interview Tips

Explain - We want to understand how you think, so explain your thought process and decision-making throughout the interview. Remember, we're not only evaluating your technical ability, but also how you solve problems. Explicitly state and check your assumptions with your interviewer to ensure they're reasonable.

Clarify - Many questions will be deliberately open-ended to provide insight into what categories and information you value within the technological puzzle. We're looking to see how you engage with the problem and your primary method for solving it. Be sure to talk through your thought process and feel free to ask specific questions if you need clarification.

Improve - Think about ways to improve the solution you present. It's worthwhile to think out loud about your initial thoughts to a question. In many cases, your first answer may need some refining and further explanation. If necessary, start with the brute force solution and improve on it — just let the interviewer know that's what you're doing and why.

Practice - You won't have access to an IDE or compiler during the interview so practice writing code on paper or a whiteboard. Be sure to test your code and ensure it's easily readable without bugs. Don't stress about small syntactical errors like which substring to use for a given method (e.g. start, end or start, length) — just pick one and let your interviewer know.

Leadership Interviews

Beyond the technical preparation, you'll also be asked questions on leading teams and projects. People management interviews dive into how you would support and grow your teams, covering:

Leadership - Be prepared to show examples of how you've resolved complex situations. How did you ensure you dealt with team challenges in a balanced way? You may also be asked hypothetical questions, so be prepared to talk through how you would influence, solve problems and drive improvements. How would you take ownership and stay creative while moving quickly?

Working with people & teams - Think about how you develop and retain team members. How would you address a skills gap or personality conflict? How would you ensure your team is

diverse and inclusive? How could you spot burn out? How would you organize day to day work activities? How would you convince a team to adopt a new technology?

Googleyness - We also want to make sure this is a place you'll thrive, so we'll be looking for signs around your comfort with ambiguity, your bias to action and your collaborative nature. Be prepared to talk about how you would support a team to help them navigate tough challenges and changes. Think about how to effectively lead in a non-hierarchical team environment and what your personal leadership style is.

Applying the right framework - What's your personal Project Management philosophy? How do you apply your framework to projects you manage? Be prepared to explain and justify your methodology. Be able to discuss and compare different project management methodologies and their relative merits (e.g. tradeoffs between flexibility and process in an agile environment). Why did you use a particular approach?

Navigating complexity and ambiguity - How do you deal with ambiguous situations and problems? How do you handle projects without defined end dates? How would you prioritize multiple projects of varying complexity? How do you balance process versus execution? What are signals that too much or too little process is in place? Can you give examples of projects where you demonstrated leadership even if you weren't a formal manager?

Delivering results - Provide examples of projects where you were the end-to-end owner. How do you evaluate the success or failure of a project? What are some strategies for handling competing visions on how to execute a project? Be prepared to discuss how to use data effectively to move critical decisions forward and how to measure impact.

Coding & Algorithm Interviews

Coding - You should know at least one programming language well, preferably C++, Java, Python, Go, or C. You'll be expected to know APIs, Object Oriented Design and Programming, how to test your code, as well as come up with corner cases and edge cases for code. Note that we focus on conceptual understanding rather than memorization.

Algorithms - Approach the problem with both bottom-up and top-down algorithms. You'll be expected to know the complexity of an algorithm and how you can improve/change it.

Algorithms that are used to solve Google problems include sorting (plus searching and binary search), divide-and-conquer, dynamic programming/memoization, greediness, recursion or algorithms linked to a specific data structure. Know Big-O notations (e.g. run time) and be ready to discuss complex algorithms like Dijkstra and A*. We recommend discussing or outlining the algorithm you have in mind before writing code.

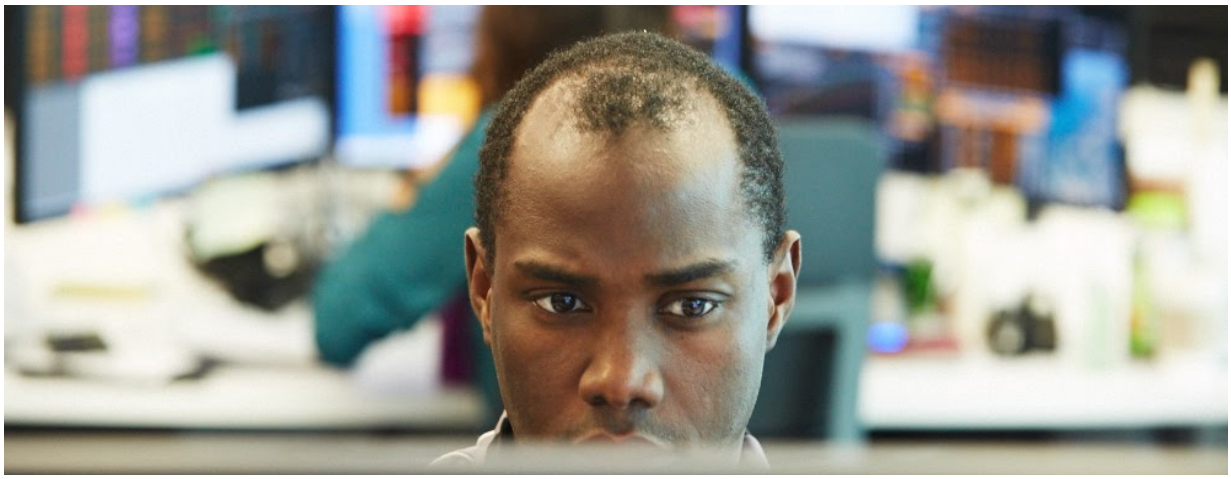
Sorting - Be familiar with common sorting functions, and the kind of input data on which they're efficient or inefficient. Think about efficiency means in terms of runtime and space used. For example, in exceptional cases insertion-sort or radix-sort are much better than the generic QuickSort/MergeSort/HeapSort answers.

Data structures - You should study up on as many data structures as possible. The data structures most frequently used are arrays, linked lists, stacks, queues, hash-sets, hash-maps, hash-tables, dictionary, trees and binary trees, heaps and graphs. You should know the data structure inside out, and what algorithms tend to go along with each data structure.

Mathematics - Some interviewers ask basic discrete math questions. This is more prevalent at Google than at other companies because counting problems, probability problems and other Discrete Math 101 situations surround us. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of elementary probability theory and combinatorics. You should be familiar with n-choose-k problems and their ilk.

Graphs - Consider if a problem can be applied with graph algorithms like distance, search, connectivity, cycle-detection, etc. There are three basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list) — familiarize yourself with each representation and its pros and cons. You should know the basic graph traversal algorithms, breadth-first search and depth-first search. Know their computational complexity, their tradeoffs and how to implement them in real code.

Recursion - Many coding problems involve thinking recursively and potentially coding a recursive solution. Use recursion to find more elegant solutions to problems that can be solved iteratively.



The Large Scale Design Interview

System Design - Questions are used to assess a candidate's ability to combine knowledge, theory, experience and judgement toward solving a real-world engineering problem. In other words, can a candidate design policies, processes, procedures, methods, tests, and/or components from ground up.

Sample topics include: features sets, interfaces, class hierarchies, distributed systems, designing a system under certain constraints, simplicity, limitations, robustness and tradeoffs. You should also have an understanding of how the internet actually works, and be familiar with the various pieces (routers, domain name servers, load balancers, firewalls, etc.). Other topics include: traverse a graphs, run-time complexity of graphs, distributed hash table system, resource estimation with real systems, big product design picture, translation of an abstract problem to a system, API discussions, binary trees, cache, mapreduce, for loop problems, index, reverse link-list, compilers, memory cache, networks and also common topics.

Operating Systems - You should understand processes, threads, concurrency issues, locks, mutexes, semaphores, monitors and how they all work. Understand deadlock, livelock and how to avoid them. Know what resources a process needs and a thread needs. Understand how context switching works, how it's initiated by the operating system and underlying hardware. Know a little about scheduling. We are rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs.

Resources

Books

Cracking the Coding Interview
Gayle Laakmann McDowell

Programming Interviews Exposed: Secrets to Landing Your Next Job
John Mongan, Eric Giguere, Noah Suojanen, Noah Kindler

Programming Pearls
Jon Bentley

Introduction to Algorithms
Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein

Interview Prep

How we hire
Interviewing @ Google
Interview tips from Google Software Engineers
CodeJam: Practice & Learn
Technical Development Guide
System Design
Approaching System Design

About Google

Company - Google
The Google story
Life @ Google
Google Developers
Open Source Projects
Github: Google Style Guide

Google Publications

The Google File System
Bigtable
MapReduce
Google Spanner
Google Chubby