Interviewing @ Google!

This guide is intended to add clarity to the role & responsibilities of a software engineer at Google. It will also give a clear overview of the recruitment process and help you best prepare for your upcoming interviews with Google. If you have any questions, please don't hesitate to get in touch.

THE OFFICE, PEOPLE AND PROJECTS

THE ROLES AND RESPONSIBILITIES

THE RECRUITMENT PROCESS

PREPARING FOR YOUR INTERVIEW

EXTRA PREP YOU MIGHT FIND HELPFUL

The Office

The Googleplex is home to scores of buildings, each with its own personality. With hundreds of bikes and scooters at our disposal, getting around campus is easy. There's beach volleyball, a bowling alley, a climbing wall, over 25 cafeterias, more than 100 micro-kitchens, and seven fitness centers.

Add in Google Doodles, April Fool's jokes like AdSense for Conversations and Gmail Motion, and all the fascinating visitors to campus (Lady Gaga, Barack Obama…), and it's easy to see how Googlers are able to incorporate their individual personalities at work.

The People

It's really the people that make Google the kind of company it is. We hire people who are smart and determined, and we favor ability over experience. Although Googlers share common goals and visions for the company, we hail from all walks of life and speak dozens of languages, reflecting the global audience that we serve.

Our Mountain View campus is also home to some fellow engineers you may have already heard a bit about. Googlers range from our founders, Larry Page and Sergey Brin, to engineers that have created programming languages, operating systems, or text editors you may use everyday -- whether C, Java, Python, Unix or VIM is your preference.

Still want more info?

Company Overview

Office location & Streetview

Office pictures

California information

YouTube video

Meet Rob, Pablo & Raman

The Roles and Responsibilities of a Software Engineer

The Role

You can start at our Google Careers Page, which is broken down per location and has real-time and detailed requirements for our needs across all of the projects.

The Responsibilities

Software Engineers at Google are responsible for the whole lifecycle of a project. Depending upon the project you join, you could be involved in research, design, planning, architecture, development, testing, implementation, and release phases.

Ultimately, you are going to help build Google products!

Get a Googler's Perspective:

"My first few months at Google have all been about getting up to speed quickly. After the induction, I went straight into building a prototype for my project. I've been developing my code to the Google code styles and am now adding to the Google code base. I will be starting to use new technologies like Mapreduce and Flume and learning to quantify my performance and improvements. When I started the atmosphere was rich in learning opportunities, challenging, and rewarding.

With the support of the team it wasn't long before I was designing, creating and maintaining code to help support and contribute to the team's end goals. It's very evident that you are responsible for the full life cycle when you are a Software Engineer at Google."

To Summarize

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. Engineers at Google not only revolutionize search, they work on massive scalability and storage solutions, large-scale applications, and entirely new platforms for developers around the world. From AdWords to Chrome, Android to YouTube, Social to Local, Google engineers are changing the world one technological achievement after another.

Still want more info?

A Google engineer's career

Ask a Google engineer

Research @ Google

Life @ Google

The Recruitment Process

How we Hire

The Process

Interview Process
Google's interview process is consistent globally. This approach allows engineers the opportunity to develop their future career within Google both geographically and across project changes.

Project Matching
As we assess your skills and strengths throughout the interview process, we consider potential project matches. We take into consideration your interests as well as the current needs of the various Google projects before making a project match. It is not uncommon that your skill set will fit many projects, and sometimes we will arrange for a call with a Tech Lead of a project to help find a match.

Phone Interviews
Up to 2 x 45 min interviews with a Google Engineer. A Google doc will be used as a virtual whiteboard for coding and algorithmic problems.

On Site Interviews
Up to 5 x ~45 min interviews. You will use a whiteboard for discussions and coding tasks. You will also have lunch with an engineer (a great time to ask questions) and have a tour of the office.

Committee Review
An independent committee of Googlers review the interview feedback. They ensure our hiring process is fair and that we're holding true to our "good for Google" standards as we grow.

Offer
Once everything is reviewed and approved, we'll extend the offer, send you the details, and talk through getting started at Google!

Still want more info?

Interviewing @ Google

Hiring process for students

Preparing for your Interview

Before you Start

We highly recommend that you do your research about the interview process at Google. A great place to start would be finding out how we hire. Then check out our technical coaching video to see the format of our interviews.

Note: We also recommend checking out Steve Yegge's blog post covering everything from technical prep to mental preparation.

Interview Questions

Be prepared to write **code on a blank Google Doc or whiteboard**. Computer Science fundamentals are prerequisites for all engineering roles at Google, regardless of seniority, due to the complexities and global scale of the projects you would end up participating in. Interview topics may cover anything on your resume, especially if you have stated that you are an expert.

Where to Focus

Demonstrate your problem solving skills as applied to the question asked. If it's a coding question, it is key to provide efficient code in a short time frame to solve the problem. If it's a design question, what matters is working with your interviewer to create a high-level system, at times perhaps diving deeper on particular salient issues. If it's a general analysis question, show that you understand all particularities of the problem described and (where applicable) offer multiple solutions, discussing their relative merits. Every interviewer ultimately wants to answer the question of whether they could successfully work on a Google Engineering team with you.

Still want more info?

About us
The Google story
Google Developers
Interview Tips from Recruiters

Main Focus: Technical Preparation

Coding: Expertise in at least one programming language is required. Know a fair amount of detail about your favorite programming language. You will be expected to know about API's, OOD/OOP, and how to test code (including both corner and edge cases).

- We expect variable declaration/initialization, error checking, and effective use of appropriate data structures. Coding exercises may require synchronization primitives and concurrency libraries. Make sure your code is clean and bug-free, and avoid writing pseudo-code unless directed to do so.
- We focus on conceptual understanding rather than memorization. We are *not* looking for memorization of all Java APIs (or the equivalents in other languages). It's fine if you forget a method or a class name.
- Some interviewers care more about syntax than others - if you're not sure, ask.

Algorithms: You will likely be asked to design an algorithm and write code. You will be expected to know the complexity of an algorithm and how you can improve/change it. Be comfortable with big-O analysis and running time complexity. Be prepared to explain the running-time complexity of algorithms you know and are asked to devise.

- When writing a more complicated algorithm, make sure you have a solid idea in your head (or outlined in bullet points) and discuss the algorithm you have in mind before jumping into coding.
- Know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem. Be able to recognize different variations of these common problems.
- Review dynamic programming and know when to use it.
- Be ready to discuss complex algorithms such as Dijkstra and A*.

Data structures: Know when and how to use all of the common data structures. You will also need to know about Trees (including basic tree construction), traversal and manipulation algorithms, hash tables, stacks, arrays, linked lists, priority queues. Note: Assume you don't have to implement them if your language's standard library ships them for you.

Expert Tips:

Open up and review your Data Structures & Algorithms textbooks.

For more information on algorithms, you can also visit TopCoder and GitHub.

Nice to Know: Technical Preparation

Sorting: What common sorting functions are there? On what kind of input data are they efficient, when are they not? What does efficiency mean in these cases in terms of runtime and space used? E.g. in exceptional cases insertion-sort or radix-sort are much better than the generic QuickSort / MergeSort / HeapSort answers. You should know the details of at least one n*log(n) sorting algorithm, preferably two (say, quick sort and merge sort).

Searching: Familiarize yourself with some common search algorithms. This could be anything from simple binary search, over depth-first/breadth-first tree search, combinatorial search over solution spaces, substring searching, etc.

Mathematics: Counting problems, probability problems, and other Discrete Math 101 situations surrounds us here at Google. 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 – the more the better.

Graphs: To consider a problem as a graph is often a very good abstraction to apply, since well known graph algorithms for distance, search, connectivity, cycle-detection etc. will then yield a solution to the original problem. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros/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.

Nice to Know: Technical Preparation

Recursion: Many coding problems involve thinking recursively and potentially coding a recursive solution. Prepare for recursion, which can sometimes be tricky if not approached properly. Practice some problems that can be solved iteratively, but a more elegant solution involves recursion.

Operating systems: Be prepared to answer questions on concurrent processing and distributed 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. Understand how context switching works, how it's initiated by the operating system and underlying hardware. Know a little about scheduling. Know the fundamentals of "modern" concurrency constructs.

System design: Questions about system design or component design are common. They are typically formulated so that they are approachable to any person with a computer science background and some work experience. 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.).

- **Note**: University Graduate candidates without industry experience are *not* typically asked system design questions.

More Info on Distributed Systems:

Distributed systems & parallel programming

Scalable Web Architecture & Distributed systems

General Interview Tips

Substantiate

Show enthusiasm for whatever it is about the Software Engineer role that you are passionate about! Be familiar with your past experiences and comfortable communicating your work.

Explain

Explain your thought process and decision making as you work through problems. *Approach the problem as if you were trying to solve it with a colleague at work, in which case it would be more of a discussion than a test.* In Google's interviews, our engineers are evaluating not just your technical abilities but also how you approach problems and try to solve them. We want to understand how you think. This would include ***explicitly stating and checking any assumptions*** you make in your problem solving to ensure they are reasonable.

Clarify

Ask clarifying questions if you need more information or do not understand the problem. Many of the questions asked in Google interviews are deliberately underspecified because our engineers are looking to see how you engage the problem. In particular, they are looking to see which areas you see as the most important piece of the technological puzzle you've been presented.

Improve

Think about ways to improve the solution that you present. In many cases, the first solution that springs to mind isn't the most elegant and may need some refining. It's worthwhile to talk through your initial thoughts with the interviewer. If necessary, don't be afraid to start with the brute force solution and improve on it - just let the interviewer know that's what you're doing.

Practice

Practice writing code without an IDE on either a whiteboard (for onsite interviews) or Google Doc (for phone interviews). After all, this likely isn't your usual development environment. Be sure to test your own code and ensure it's easily readable without bugs.

Ask Questions

You are also interviewing Google, and you want to make sure Google is a good place for you.

Feel free to ask the same question more than once; this is a great opportunity to hear multiple engineers' perspectives on what matters to you in your work environment.

Additional Interview Tips from Google Engineers

Ask a friend to practice whiteboard coding with you, asking you questions. Step away from the problem and think out loud while answering.

When given a question, (i) make sure you completely and fully understand the question, (b) write down an initial set of edge cases, (c) verbalize every solution or idea you have, (d) code up a solution on the whiteboard or Google Doc, and (e) go through all your edge cases.

Approach the interview as a problem to be solved. No one is expected to do perfect on every aspect; the interview should explore the edges of knowledge and so occasionally may venture into “I don’t know” territory. Don’t let it fluster you.

Listen - don’t miss a hint if your interviewer is trying to assist you!

Saying what you know can be helpful, specifically when it can convey a lot of knowledge while spending minimal time. (ex: “I would normally use StringBuilder, but in the interest of time, I’ll use the + syntax because it is quicker to write.”)

Consider inventing methods as you go.
Like findFirstVowel(String s);
If the method is "interesting" the interviewer can ask you to implement.

Don’t stress small syntactical errors. If you can’t remember whether the method is a substring(start, end) or substring(start, length), just pick one and let your interviewer know. In the real world, you could just check the documentation or even autocomplete.

Test your solution with sample data/parameters. Discuss how your solutions scale. Discuss the time and size complexity of your solutions. Think about and express the Google scale implications of decisions.

You can write longer, more descriptive variable names once and then ask if it’s okay to abbreviate to save time.

Get some practice!
To practice for your interview, you may want to try:

Google Code Jam questions -- samples from Google coding competitions.

Take Dean's advice! Try practicing coding in a Google doc or on a whiteboard with a friend.

Extra Prep You Might Find Useful

Recommended by Googlers

Books

Most Recommended:

Cracking the Coding Interview
Gayle Laakmann McDowell
CareerCup, LLC, 2011
You may view the 4th Edition online here.

Additional Resources:

Programming Interviews Exposed: Secrets to Landing Your Next Job
John Mongan, Eric Giguere, Noah Suojanen, Noah Kindler
John Wiley & Sons, 13.11.2012

Programming Pearls
Jon Bentley
Pearson Education, 01.09.2000

Introduction to Algorithms
Thomas H. Cormen
Mit Press, 2009

The Algorithm Design Manual
Steven S. Skiena
Springer-Verlag London Limited, 2008
You may view the 2nd Edition online here.

Online resources

Places to Find Practice Problems

https://code.google.com/codejam

www.leetcode.com

www.projecteuler.net

Google Publications for Additional Reference:

The Google File System

Bigtable: A Distributed Storage System for Structured Data

MapReduce: Simplified Data Processing on Large Clusters

Google Spanner: Google's Globally-Distributed Database

Google Chubby

Still want even *more* info?

Google books

Google publications

Google scholar