Create
Design
Code
Build
for
everyone

# Google Technical Internships Interview Preparation

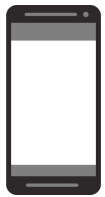
# Agenda

This guide is intended to help you prepare for Software/Site Reliability Engineering internship and Student Training in Engineering Program (STEP) interviews at Google. If you have any additional questions, please don't hesitate to get in touch with your recruiter.

1. Recruitment Process
2. Interview Tips
3. Technical Preparation
4. Extra Prep Resources

# Recruitment Process

# Recruitment Process



## 2 x 45-minute Technical Interviews with a Google Engineer

An interview coding document will be used as a virtual whiteboard for coding and algorithmic problems.

## 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.

## Project Search

Your application is reviewed by our potential project hosts and engineers to determine if there is a skill match to a project. This can result in an additional interview with a potential host to discuss their project.


## Offer

We'll take a look at the additional information gathered and if your application is approved, we will make you an offer, confirm it in writing so that you can start your internship at Google!

# Internship Project Interviews

- As we assess you, your skills and strengths throughout the interview process, we consider potential project matches here at Google. We take into consideration your interests as well as the current needs of the various Google projects before making a project match.
- During the Project Search stage, the answers to the questionnaire you have already filled out will be used by the hosts for review, so please make sure your responses are thoughtful and thorough. You are able to update your responses to the intern project preference questionnaire at any point.
- Once a potential project has been identified, you'll have a call with the engineer hosting the project to ensure that it is a fit for your skills.
- Occasionally we will not have an internship project with a strong alignment to your interests and skills. Unfortunately, if this happens, the interview process will conclude and your recruiter will let you know. We'll make a note to follow up with you about future opportunities.

# Interview Tips

# Interview Tips

## Substantiate

Make sure that you substantiate what your CV/resume says – for instance, if you list Java or Python as your key programming language, questions about this are fair game and may be asked of you.

## Explain

Explain your thought process and decision making throughout the interview. In all of Google's interviews, our engineers are evaluating not only your technical abilities but also how you approach problems and how you try to solve them. Many of the questions asked in Google interviews are open-ended because our engineers are looking to see how you break down and approach the problem.  There is no “one true answer” to 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. **Think out loud.**

# Interview Tips

## Clarify

Ask clarifying questions if you do not understand the problem or need more information. 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. Clarifying questions are encouraged!

Also, **take interviewer tips seriously** because they're trying to give you helpful hints. They may ask some additional questions or share comments when you're solving a problem. Take a minute to think through these prompts -- they can help you land on a more optimal solution!

# Interview Tips

## Improve

**Think about ways to improve the solution.** Share the different options or tradeoffs you're considering. **Be flexible.** In many cases, the first answer that springs to mind may need some refining.  It is worthwhile to talk about your initial thoughts to a question. Here are some guidelines:

- You want to get to the simplest solution as quickly as possible. You can start with a brute force solution, but then...
- **Discuss tradeoffs.** How would you improve your solution? How do you make it faster? Use less memory? Know the time and space tradeoffs of the solution/data structures you pick.
- Code. We generally don't want pseudocode. It might be acceptable in some limited cases. If you are not sure, ask the interviewer. The interviewer will guide you through.
- If you don't remember the exact interface to a library class or method, that is OK, as long as you let the interviewer know, and you postulate a substitute with a conforming interface.

# Interview Tips

## Practice

Make sure you practice writing code on paper, a whiteboard, or code in an interview coding document to prepare for the interviews over Meet. Be sure to test your own code and ensure it's easily readable without bugs.  Keep in mind, **you can choose one of the following programming languages** you're most comfortable coding in: C, C++, Go, Java, Javascript & Python

## Ask Questions

At the end of the interview, most interviewers will ask you if you have any questions about the company, work environment, their experience, etc. It's always good  to have some pre-prepared for each interview.

# Resources

## Hiring Process for Students

Read more here.

## Interviewing at Google

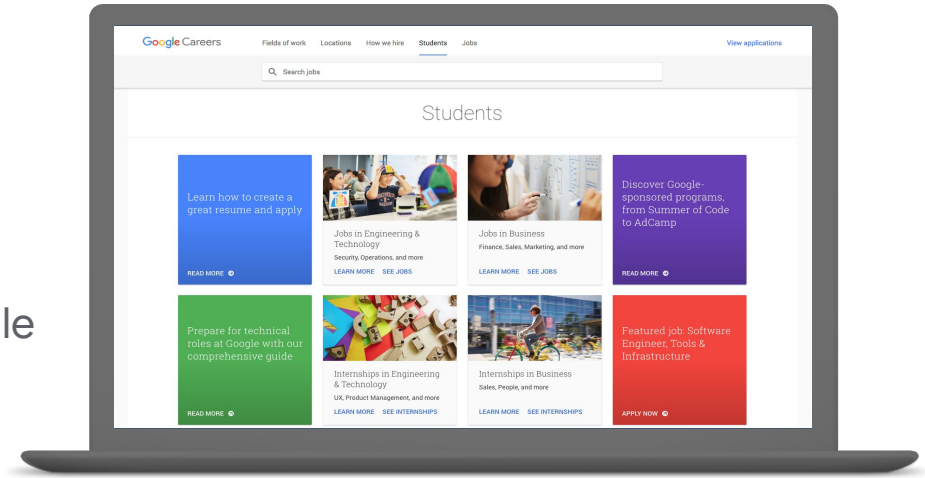
Read more here.

## Google Code Jam Questions

Get some practice with samples from Google coding competitions here.

## Take Dean's advice!

Read it here, and try practicing coding in an interview coding document or on a whiteboard with a friend.



# Technical Preparation

# Technical Preparation

## Coding

Google Engineers primarily code in C, C++, Go, Java, Javascript & Python. We ask that you use one of these languages during your interview. For phone and interviews over Meet, you will be asked to write code real time in an interview coding document. You may be asked to:

- Construct / traverse data structures
- Implement system routines
- Distill large data sets to single values
- Transform one data set to another

# Technical Preparation

## Algorithms

You will be expected to know the complexity of an algorithm and how you can improve/change it. You can find examples that will help you prepare on TopCoder. Some examples of algorithmic challenges you may be asked about include:

- Big-O analysis: understanding this is particularly important
- Sorting and hashing
- Handling obscenely large amounts of data

# Technical Preparation

## Sorting

We recommend that you know the details of at least one n*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.  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.

# Technical Preparation

## Data Structures

Study up on as many other structures and algorithms as possible. We recommend you know about the most famous classes of NP-complete problems, such as traveling salesman and the knapsack problem. Be able to recognize them when an interviewer asks you in disguise. Find out what NP-complete means. You will also need to know about Trees,  basic tree construction, traversal and manipulation algorithms, hash tables, stacks, arrays, linked lists, priority queues.

## Hash tables / Maps

Hash tables are arguably the single most important data structure known to mankind. You should be able to implement one using only arrays in your favorite language, in about the space of one interview. You'll want to know the O() characteristics of the standard library implementation for Hash tables / Maps in the language you choose to write in.

# Technical Preparation

## Trees

We recommend you know about basic tree construction, traversal and manipulation algorithms. You should be familiar with binary trees, n-ary trees, and trie-trees at the very least. You'll want to know how it's implemented. You should know about tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

## Min/Max Heaps

Heaps are incredibly useful. Understand their application and O() characteristics. We probably won't ask you to implement one during an interview, but you should know when it makes sense to use one.

# Technical Preparation

## 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.

## 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 where a more elegant solution is recursion.

# Technical Preparation

## 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. The world is rapidly moving towards multi-core, so know the fundamentals of "modern" concurrency constructs.

# Technical Preparation

## 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 surrounds 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 – the more the better.

# Resources

## Distributed Systems and Parallel Programming

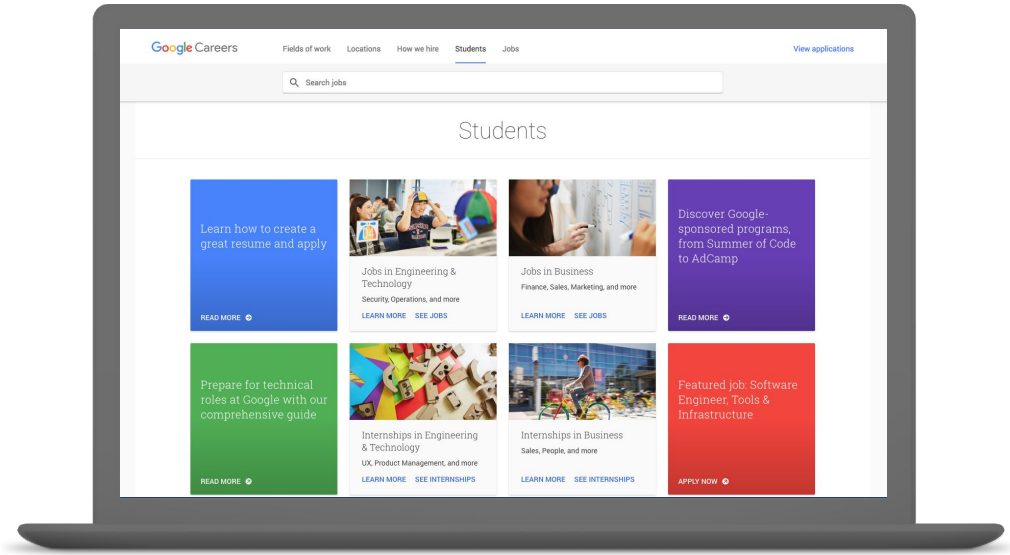
Read more here.

## Scalable Web Architecture & Distributed systems

Read more here.

## How Search Works

Read more here.



# Extra Prep Resources

# Resources recommended by Googlers

## Online Resources

Dean Jackson's ACM article
Cracking the Coding Interview Video and Cracking the Coding Interview
Five Essential Phone Screen Questions by Steve Yegge
Google+ Technical Interview Coaching Session
TopCoder Tutorials

## For Underclassmen

Guide for Technical Development
How to Get Hired - What CS students need to know

# Resources recommended by Googlers

**Books**

Programming Interviews Exposed: Secrets to Landing Your Next Job

Programming Pearls

Introduction to Algorithms

# Resources recommended by Googlers

## Learn about Google products and technologies

About Google
Google Spanner: Google's Globally-Distributed Database
Google Chubby
Industry News: Search Engine Land
Google File System
Google Bigtable
Google MapReduce

# Get some practice!

**To practice for your interview, you may want to try:**

Project Euler

ACM-ICPC Live Archive

# Best of luck with the process!