



# Google Software Engineer Interview Guide

**GAYLE LAAKMANN**

FOUNDER / CEO CAREERCUP.COM

---

**About the Author:** *Gayle Laakmann is the founder and CEO of CareerCup.com and the author of "Cracking the Technical Interview." Gayle has worked for Google, Microsoft and Apple and has extensive interviewing experience on both sides of the table. She has interviewed and received offers from Google, Microsoft, Apple, Amazon, IBM, Goldman Sachs and a variety of other firms, and she has interviewed over 120 candidates at Google and served on its hiring committee.*

*She holds a bachelors and masters in Computer Science from the University of Pennsylvania, and is currently getting her MBA at Wharton Business School.*



**TABLE OF CONTENTS**

Table of Contents .....	2
The Google Interview Process .....	4
Resume Selection .....	4
Phone Screen / Campus Interview .....	4
Interview Day .....	4
Decision Process .....	4
Internship Recruiting Process .....	5
Google By the Numbers .....	7
Salary and Compensation .....	7
Offer and Interview Statistics .....	7
Life at Google .....	8
Job Structure .....	8
Technologies .....	8
Why People Love It .....	10
Why It May Not Be For You .....	10
Google Interview Advice .....	12
Interview Focus .....	12
Large Scale / System Design .....	12
Bit manipulation .....	13
Unusual Questions .....	14
Other Advice .....	14
General Interview Preparation advice .....	16
Soft Skills Preparation .....	16
Tech Skills Preparation .....	17
General Interview Advice .....	20
Soft Skills .....	20

Technical Skills .....	21
Google Interview Profiles .....	27
Dipa   College Hire, Software Engineer   Sept 2009 .....	27
Alex   Industry Hire, Software Engineer   Nov 2009 .....	28
Jason   Software Engineer, College Hire   April 2009 .....	30
Google Frequently Asked Questions .....	33
What language do I need to code in? .....	33
If I'm rejected, can I reapply? .....	33
I got an offer, but why won't google tell me what team i'm on? .....	33
How does 20% time work? Does it really exist? .....	33

## THE GOOGLE INTERVIEW PROCESS

### RESUME SELECTION

In selecting people to interview, Google looks for indications that a candidate is smart and a great coder. That means primarily looking at these three things:

1. Your academic performance (if you are a student or recent graduate). You don't need to have a 4.0 average, but you will have trouble getting an interview if you have much below a 3.0.
2. Your professional experience – prior jobs, etc. When you describe your jobs, make sure you say what you, personally, accomplished. That is, "I optimized Feature X by applying Y algorithm, leading to a Z% reduction in run time."
3. Your project experience. Add a "Projects" section to your resume where you can list your academic or independent projects.

### PHONE SCREEN / CAMPUS INTERVIEW

Prior to inviting you in for a full round of interviews, Google will usually conduct an initial screen. The screen will be technical and will be with an engineer. Be prepared to write code.

If you are doing a phone interview, your interviewer will often share a Google Doc with you so that you can code over the phone. Remember that there is a bit of lag between your typing and your interviewer seeing your code, so make sure to talk out loud while you're coding.

### INTERVIEW DAY

You will usually not interview for a position with a particular team, but rather, you will interview for Google in general. This means that your interviewers will come from a mix of teams.

You will interview with four or five engineers who will ask a mix of coding and algorithm questions, plus some questions about your resume or your interests.

In addition to the technical interviews, you will have a lunch "interview," which is really an informal interview. This interviewer will usually not submit any feedback on you (unless you do something egregious, like make offensive comments), so this is a great time to ask the questions you *really* want to know but are uncomfortable to ask other people (such as how long the hours are).

Google conducts "blind" interviews, meaning that one interviewer cannot reveal your performance to another interviewer. This means that if one interview goes poorly it will not bias your other interviewers.

Each interview makes his or her own decision about what questions to ask, so one candidate's experience may not be especially relevant to yours.

### DECISION PROCESS

Each of your interviewers will submit written feedback on your interview, with a score of 1 to 4. A good candidate typically gets mostly scores between 3.0 and 3.3, with one very high score (3.4+). The high score is called an

“enthusiastic endorser,” and is usually required to get an offer. You can still get hired with a single bad score (2.2 – 2.4) if your other feedback is strong.

The hiring committee will review your feedback and make a decision based on the written feedback. Your phone screen usually is not a significant factor in the decision.

In rare cases, when the hiring committee feels that no one probed a particular area, you might be scheduled for a follow up phone interview.

In reviewing your feedback, Google looks mainly at whether you’re smart and a good coder. They usually do not care much about your resume by this point.

## INTERNSHIP RECRUITING PROCESS

While interview questions and expectations are the same for intern and full time candidates, the process for interns is a bit different.

### INITIAL PHONE SCREENS

Like full time candidates, intern candidates start with one to two phone screens to assess their overall strength as an engineer. These interviews involve algorithms and coding, sometimes using Google Docs.

### CANDIDATE AND TEAM MATCHING

After that point (assuming a candidate has made the first cut), the candidate interviews with a particular team, either because the team hand-picked the candidate from a general pool of interns, or because the recruiter sent the candidate’s resume to the team.

A team might select a candidate’s resume for a variety of reasons:

- **Experience:** The candidate has particularly relevant experience.
- **Interests:** The candidate has expressed an interest in working for that team.
- **Random:** The team wanted to hire an intern and happened to select that candidate (possibly because they thought the candidate was unusually strong).

The “random” case is probably the most common.

### FINAL TEAM INTERVIEWS

Unlike full time candidates who do not interview with particular teams, intern candidates interview with the team they will eventually join. These interviews are sometimes technical, but are often simply “personality” / interests fits. That is, your interviewer is trying to assess whether or not you are interested and excited about their team.

You will usually interview with two people on the team, one of whom will be your manager.

## FULL-TIME OFFERS AFTER YOUR INTERNSHIP

At the end of your internship, if you have done well, Google will re-interview you through the regular full time process. Your internship performance will be a heavy factor in whether or not to hire you. However, poor performance on interviews can mean that a great intern does not get an offer.

## GOOGLE BY THE NUMBERS

### SALARY AND COMPENSATION

Below is the average salary and bonuses for US offices. Salaries do not vary significantly by US-office.

New Grad		Industry Hire (3 – 5 years)	
Salary:	\$85k	Salary:	\$96k
Bonus	\$27k	Bonus:	\$31k

### OFFER AND INTERVIEW STATISTICS

Google will not release official data on how many candidates are offered jobs, but CareerCup's best estimate is:

- After an engineering phone screen, 35% of candidates are invited on campus.
- After an onsite interview, 25% of candidates are extended offers.

This means that just 9% of those interviewed are given offers!

## LIFE AT GOOGLE

### JOB STRUCTURE

#### ROLE OF A SOFTWARE ENGINEER

Compared to at many other companies, where product managers and marketing lead the key decisions, at Google engineers take on much of this responsibility. Each project is led by a Technical Lead who coordinates with related teams and leads many of the most important decisions.

As a software engineer, you lead the key decisions for your features by soliciting input from the rest of your team. You, in turn, offer input on the decisions and direction of the team. You will also serve as an “architect” on your team.

#### REVIEWS, BONUSES AND PROMOTIONS

Because managers oversee so many people, Google relies heavily on peer reviews. You select several members of your team to review you, and your manager then compiles the reviews.

Bonuses are based on a mix of personal performance and company performance. This means, of course, in a good economy you’ll get paid much more. Google has some of the highest annual bonuses, averaging 30% in some years.

Google emphasizes promoting people while letting them stay as an “independent contributor.” That is, if you start as a Software Engineer II, your next promotion will likely be to Software Engineer III. It will not involve any major change in responsibilities.

#### SWITCHING PROJECTS

If you don’t like what you’re working on (or who you’re working with), switching projects is usually an easy and painless process. You simply need to speak with a potential team and confirm that they’d be interested in having your help. Google encourages engineers to switch every 1 – 1.5 years, and does not require interviews to switch teams.

### TECHNOLOGIES

#### LANGUAGES

Java, C++, Javascript and Python are all very common languages at Google. Backend work tends to be done in C++ or Java, with some Python, while frontend work may be done in either Python or Javascript.

For the purposes of your interview, Java, C++ or even C# will suffice.

#### MAPREDUCE



While Google did not invent MapReduce, it certainly popularized it. MapReduce is used across Google to do much of its data processing.

MapReduce is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster. Computational processing can occur on data stored either in a filesystem (unstructured) or within a database (structured).

- "Map" step: The master node takes the input, chops it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure.

The worker node processes that smaller problem, and passes the answer back to its master node.

- "Reduce" step: The master node then takes the answers to all the sub-problems and combines them in a way to get the output - the answer to the problem it was originally trying to solve.

Read more at <http://en.wikipedia.org/wiki/MapReduce>.

## OPERATING SYSTEM, VERSION CONTROL, AND OPEN SOURCE

Google relies heavily on OpenSource applications and customizes them for its purposes. Google uses its own flavor of Ubuntu and a customized version of Perforce (version control), both with their own "Googley" names (eg, Goobuntu).

All Google projects are on the same code base, making it easy to share libraries across teams.

## CODING STANDARDS

Google enforces strict coding standards to increase maintainability and readability. While these seem bizarre to the outside world, Googlers will generally tell you that the principle makes sense even if they don't agree with each and every rule.

The C++ style guide can be found at <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

Here is a comparison of "Google style" code:

Google Style	Not Google Style
<pre>bool my_method(int x) {     // Iterate through values.     for (int i = 0; i &lt; x; i++) {         something_else(i, x);     } }</pre>	<pre>bool myMethod(int x) {     // Iterate through values     for (int i = 0; i &lt; x; i++)     {         somethingElse(i, x);     } }</pre>
<ul style="list-style-type: none"> <li>• Open brace is on same line as declaration</li> <li>• Capitalization of methods is first_second.</li> </ul>	<ul style="list-style-type: none"> <li>• Open brace is not on same line as declaration</li> <li>• Capitalization of method is firstSecond.</li> </ul>

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>• Two space indents.</li><li>• Comment ends with period.</li></ul> | <ul style="list-style-type: none"><li>• Five space indent.</li><li>• Comment does not end with period.</li></ul> |
|--|--|

Some of these things may seem extremely picky – or maybe even *bad* to you – but when all code in a code base looks the same, it's so much easier on your eyes. Cross-project standards are important due to the frequency at which people switch projects.

## WHY PEOPLE LOVE IT

### THE PERKS

Google gives their engineers a ton of freedom and responsibility, and showers them with perks. The free food, the massages, 20% time – it's all true!

### THE ATTITUDE

Google is, at its core, an engineering company. It was founded by engineers (PhD students, no less) and it understands that engineers are what make the company great. As an engineer, you hold the revered position.

### 20% PROJECTS

20% projects are so unique to Google that this deserves its own heading. At Google, engineers are expected to spend 20% of their time working on a project outside of their “core” responsibilities. There is a wide range of projects.

Some people have an on-going 20% project. For example, Google News was started during someone's 20% project. This engineer regularly spent time building this product, and recruiting others to join him, until it became his “80%” project.

Other people just do occasional projects. For example, an engineer might spend a day fixing an issue that has bugged him with Gmail. Or, he might save up his time and then spend several weeks implementing a new feature for Google Labs.

Some people do projects completely off the wall. Author Gayle Laakmann (CareerCup.com Founder) spent her 20% time while she was at Google teaching two courses at the University of Washington. Several of her coworkers projects working on diversity recruiting.

The sky's the limit!

### CHALLENGING PROJECTS WITH THE RIGHT RESOURCES

Due to the large amount of data, traffic and users, Google has some of the most interesting and challenging projects. And, Google has a ton of infrastructure and technology to support building large systems, so you don't need to waste time on repetitive details. You have the right resources so let you focus on the toughest problems.

## WHY IT MAY NOT BE FOR YOU

---

## IF YOU WANT A CLOSE RELATIONSHIP WITH YOUR MANAGER

Google has a very flat hierarchy, which means that your manager might be overseeing as many as twenty or thirty people. Your manager may not have time to give a ton of direction and feedback. Some people like the lack of “managing,” others don’t.

---

## IF YOU WANT TO BECOME A MANAGER

Because each manager oversees so many people, it’s hard to become a manager. You essentially need to “overtake” maybe twenty people. It’s probably not the best company if you want to move from developer to manager in a couple years.

However, if/when you do make that jump, it’s a big step up!

---

## IF YOU ONLY WANT TO DO ONE PROJECT, EVER

Google encourages engineers to switch companies every 1 – 1.5 years, and encourages you to spend 20% of your time working on a project outside of work. If you’re interested only in, say, image processing, you may not be encouraged to stay on that project forever.

---

## IF YOU WANT A JOB TO STAY THE SAME

Google is just ten years old – fairly young compare to most software companies – and it’s still changing rapidly. It’s growing comparatively fast, and it’s still figuring some things out from a project management perspective.

This means that Google’s culture may change over the course of several years.

---

## IF YOU WANT TO BE KNEE-DEEP IN CODE, AND ONLY CODE

While many companies have project managers and marketing to define the feature set, Google engineers take on these responsibilities. Many people love this, but some don’t. If you join Google, be prepared for a conversation about what the desired feature set is – you may not have someone else to “spec” this out for you.

## GOOGLE INTERVIEW ADVICE

### INTERVIEW FOCUS

In addition to normal software engineering questions, Google interviews focus heavily on Large Scale (or System Design) questions and Bit Manipulation.

### LARGE SCALE / SYSTEM DESIGN

To tackle system design questions, you don't necessarily need to know anything special. Just your usual problem solving skills are fine! In fact, most Google employees did not have a background in large scale programming before they started.

### HOW TO APPROACH

The general approach is as follows: Imagine we're designing a hypothetical system X for millions of items (users, files, megabytes, etc):

1. How would you solve it for a small number of items? Develop an algorithm for this case, which is generally pretty straight-forward.
2. What happens when you try to implement that algorithm with millions of items? It's likely that you will run out of space on the computer. So, divide up the files across many computers.
  - a. How do you divide up data across many machines? That is, do the first 100 items appear on the same computer? Or all items with the same hash value mod 100?
  - b. About how many computers will you need? To estimate this, ask how big each item is and take a guess at how much space a typical computer has.
3. Now, fix the problems that have occurred when you're on many computers. Make sure to answer the following questions:
  - a. How does one machine know which machine it should access to look up other data?
  - b. Can data get out of sync across computers? How do you handle that?
  - c. How can you minimize expensive reads across the network?

### EXAMPLE: DESIGN A WEB CRAWLER

1. Ask your interviewer what the web crawler needs to do. Does it just need to gather links, or does it need to index the web pages?  
*Let's assume your interviewer told you that it just needs to gather a list of the links.*
2. You probably have a basic idea of how to crawl the web – start with one link, find what pages it links to, and then go to each of these. That's breadth-first search!
3. Think about what problems you might hit. I can think of a few:
  - a. How do you know if two pages are the same? This is harder than it sounds. Two pages might have the same content but different urls (either completely different, or the same root with

different url parameters). Or, they may have different content but the same url (eg, randomly generated content). You should talk through some ways to define “equal” pages.

- b. Can you wind up in a loop? What might you do to prevent that?
  - c. How do you get a good mix of pages, so that you don’t continue traversing all pages on a single network?
4. Now that you’ve thought through a few problems and come up with solutions, design an algorithm to consider the simplest case. For now, don’t worry about the fact that you’re dealing with billions of pages. How would you design this system if it were just a small number of pages? You should have an understanding of how you would solve the simple, small case in order to understand how you would solve the bigger case.
5. Now, go back to the issues of billions of pages. Most likely you can’t fit the data on one machine, so you’ll have to divide it up across multiple machines. Consider problems like:
  - a. How will you divide up the data? Literally, what decides whether a piece of data gets put on one machine or another? Is it randomly divided? Is a piece of data hashed, and then everything with the same value mod  $m$  (where  $m$  is the number of machines) is on the same machine?
  - b. How will you figure out which computer has a particular piece of data?
  - c. Is it likely that one machine gets overloaded? Can you redistribute the data when this occurs?
6. You now have different pieces of data on different machines. Consider what problems this might cause:
  - a. A machine could fail. Do you clone the data across many machines? Or maybe save it to a disk and keep it in memory?
  - b. Since machines have to communicate across a network, you might have some latency issues. Will this be a problem? Are there better ways to divide data to reduce latency?

Note that you didn’t need to know anything about system design to tackle this problem. Just your basic technology knowledge is fine!

## BIT MANIPULATION

Many questions at Google involve bit manipulation either explicitly or implicitly. Make sure you are very comfortable with these questions.

## STUDYING BIT MANIPULATION

### BASIC PRACTICE

- Shift Left by  $x$  bits
- Shift Right by  $x$  bits
- Add
- Or
- Xor
- And

## INTERMEDIATE PRACTICE

- Subtract
- Multiply. Remember, if you get stuck here, multiple
- Get Bit (eg, get the *i*th bit of a number)
- Set Bit (eg, set the *i*th bit of a number)
- Is less than (eg, check if one number is less than another)

## ADVANCED PRACTICE

1. Check if a number is a power of two as fast as possible.
2. Given a number passed in as a string, print the binary representation of it.
3. Using bit manipulation, implement add, subtract and multiply on two integers. You cannot use any arithmetic operations, such as +, - or \*.
4. Given an integer, print the next smallest and next largest numbers that have the same number of 1's in the binary representation.

## UNUSUAL QUESTIONS

The following types of interview questions are usually not asked:

- Brain Teasers: Officially Google bans (or heavily discourages) brain teasers. But, what one interviewer feels is a brain teaser another interview may feel is a “just a math question,” so you still sometimes see these.
- C++ / Java Terminology and Trivia

## OTHER ADVICE

## PHONE INTERVIEW

Many Google questions may require you to convert between a power of two, the “english” representation of it, and the number of bytes / megabytes it represents. For your phone interview, we *strongly* recommend having this chart in front of you.

Name	Bytes	Bytes	Bytes	Bytes	Kilobytes	Megabytes	Gigabytes
Kilobyte	Thousand	1024	$2^{10}$	$10^3$	1		
Megabyte	Million	1,048,576	$2^{20}$	$10^6$	1024	1	
Gigabyte	Billion	1,073,741,824	$2^{30}$	$10^9$	1,048,576	1024	1
Terabyte	Trillion	1,099,511,627,776	$2^{40}$	$10^{12}$	1,073,741,824	1,048,576	1024

In addition to this chart, remember to the following in front of you:

- Pencil
- Calculator
- Plenty of sheets of paper

## ONSITE INTERVIEW

For your onsite interview, make sure to thoroughly practice coding. You will be coding on a white board, so it is important to practice non-computer coding.

## GENERAL INTERVIEW PREPARATION ADVICE

Just as interviews are divided into “soft skills” (eg, personality, experience, behavioral questions) and technical skills, your preparation can be thought of this way too.

## SOFT SKILLS PREPARATION

### COMPANY / TEAM PREPARATION

Because companies will look for your knowledge and passion about them, make sure you research the company / team you'll be interviewing at. In particular, try to find some interesting recent news, or anything that describes the technology aspects. What is their system architecture? Do they utilize open source?

Before your interview, you should come up with a list of at least 10 questions to ask your interviewers. This will allow you to raise a couple of unique questions for each interviewer.

Questions generally belong to one of these three categories:

- **Passion Questions:** These questions are designed to demonstrate your passion for technology. *Example: “I’m really interested in learning about scalability, and I know [Company] has a lot of interesting scalability challenges. What sorts of opportunities are there to learn about scalability?”*
- **Insightful Questions:** These questions are intended to show how you think about technical problems. *Example: “I noticed that [Company] based its software off an open source ad server, but I’ve heard that it had a lot of issues with multiple connections. How do you handle that?”*
- **Genuine Questions:** These questions are about what you really want to know. That could be anything, of course, but here’s an example: *Example: “How much of your day do you actually spend coding? How many meetings do you have on average per week?”*

*Asking good questions can show that you’re passionate about technology, that you’re smart, or that you really, really want to work there. These are all positive signs!*

### SELF PREPARATION

Almost any company will ask you questions about yourself, and with a little bit of preparation, you can excel on these questions. CareerCup recommends creating a Preparation Grid, like this one below:

	OS Project	Internship at Microtron
Enjoyed	Designing data structures	Planning initial features
Hated	Debugging threading bugs	Working with legacy system
Most Challenging	Compressing file system	Optimizing algorithm to $O(n \log n)$
Hardest Bug	Scheduler crash	Crashing with large numbers



Learned	Emphasizing space	Designing for maintainability
---------	-------------------	-------------------------------

Along the top, as columns, you should list all the major aspects of your resume – eg, your projects, jobs, or activities. Along the side, as rows, you should list the common questions – eg, what you enjoyed most, what you enjoyed least, what you considered most challenging aspects, what you learned, what was the hardest bug you encountered, etc. In each cell, put the corresponding story.

In your interview, when you're asked about a project, you'll be able to come up with an appropriate story effortlessly. We recommend adding only a couple of words to each cell, as it will be easier to remember a story with less text. Study this grid before your interview.

*A preparation grid, like the one shown above, will help you speak clearly and intelligently about yourself and your experience. Create one, and study it before each interview.*

*NOTE: If you're doing a phone interview, you should have this grid out in front of you.*

Some additional advice:

1. *When asked about your weaknesses, give a real weakness!* Answers like "My greatest weakness is that I work too hard / am a perfectionist / etc" tell your interviewer that you're arrogant and/or won't admit to your faults. No one wants to work with someone like that. A better answer conveys a real, legitimate weakness but emphasizes how you work to overcome it. For example: "Sometimes, I don't have a very good attention to detail. While that's good because it lets me execute quickly, it also means that I make careless mistakes. Because of that, I make sure to always have someone else double check my work."
2. *When asked what the most challenging part was, don't say "I had to learn a lot of new languages and technologies."* This is the "cop out" answer (eg, you don't know what else to say). It tells the interviewer that nothing was really that hard.
3. *Remember: you're not just answering their questions, you're telling them about yourself!* Many people try to just answer the questions. Think a bit more than that, about what you say – you are communicating something in all of your answers.
4. *If you think you'll be asked "behavioral" questions (eg, tell me about a challenging interaction with a team member"), you should create a Behavioral Preparation Grid.* This is the same as the one above, but the left side contains things like "challenging interaction", "failure", "success", and "influencing people."

## TECH SKILLS PREPARATION

Your technical skills preparation can be broken down into three categories: (1) data structures; (2) algorithms; (3) concepts. We will go through each of these.

### DATA STRUCTURES

For each data structure, you should know how to implement the data structure, when to use it (pros and cons) and how to deal with space and time complexity.

Some of the most important data structures are:

- Linked lists
- Stacks
- Queues
- Trees
- Tries
- Graphs
- Vectors
- Heaps
- Hash tables

*Maybe half, or more, of interview questions involve a hash table. Make sure you are comfortable with these!*

Prior to your interview, you should be very good at implementing these data structures that you could re-write them from scratch flawlessly. Your interviewer might actually ask you to implement a binary search tree, for example, or they might ask you to write code that involves a modified version of a binary search tree. Either way, the smarter you are at implementing these data structures, the better!

---

## ALGORITHMS

For each algorithm, you should know how to implement the algorithm and you should know its space and time complexity.

Some of the most important algorithms are:

- Breadth-first search
- Depth-first search
- Tree Insert / Find
- Mergesort
- Quicksort
- Binary search

The first three of these you should be able to easily implement. The last three you should be able to do with a bit of thought.

---

## CONCEPTS

While we've listed these as "concepts," they are, of course, much more than that. Make sure you know how to work with them.

- Threading
- Locks and mutexes
- Memory management (heap vs stack)
- Recursion. *If you're not comfortable with recursion, get used to it before your interview. And remember that recursive algorithms take up lots of space.*
- Randomness and combinatorics. *While you're unlikely to be tested directly on probability theory or combinatorics, it's very useful to know the basics (eg, how many ways can you rearrange a string). By knowing the mathematical answer, you'll be able to check the algorithm answer more easily.*
- Bit manipulation. *If you're not comfortable with bit manipulation, this is an easy thing to practice.*

*Computers help you too much when you code. If you want to really prepare, practice coding on paper then type it into a computer. You'll make more mistakes than you think!*

---

## HOW TO PRACTICE

This might be the best piece of advice I can give. Are you ready for it?

Ok, listen up: *don't practice on a computer!*

You've been coding all your life on a computer, and you've gotten used to the many nice things about it. But, in your interview, you won't have the luxury of syntax highlighting or code completion. Basically, you know less than you think you do!

So, instead, *practice on paper*. Select interview questions – CareerCup.com has thousands – and then write the code on paper first. Make sure to write down every last semi colon. When you've done everything you can, type it into a computer and see how you did. Practice makes perfect.

## GENERAL INTERVIEW ADVICE

### SOFT SKILLS

As stated earlier, interviews usually start and end with “chit chat” or “soft skills.” This is a time to answer questions about your resume, some general questions, and also an opportunity for you to ask questions. This part of the interview is targeted not only at getting to know you, but also at relaxing you.

#### BE SPECIFIC – NOT ARROGANT

We told you previously that arrogance is a red flag. But still, you want to make yourself sound impressive. So how do you make yourself sound good without being arrogant? By being specific!

Specificity means giving just the facts and letting the interviewer derive an interpretation. Consider an example:

- Candidate #1: “Yeah, I basically did all the hard work for the team.”
- Candidate #2: “I implemented the file system, which was considered one of the most challenging components because ...”

Candidate #2 not only sounds more impressive, but she also appears less arrogant.

#### LIMIT DETAILS AND LET YOUR INTERVIEWER ASK FOR MORE

When a candidate blabber on about a problem, it’s hard for an interviewer who isn’t well versed in the subject / project to understand it. CareerCup recommends that you stay light on details and just say the “key points.” That is, consider something like this: “By examining the most common user behavior and applying the Rabin-Karp algorithm, I designed a new algorithm to reduce search from  $O(n)$  to  $O(\log n)$  in 90% of cases. I can go into more details if you’d like.” This demonstrates the key points while letting your interviewer ask for more details if he wants to.

#### ASK GOOD QUESTIONS

Remember those questions you came up with while preparing? Now is a great time to use them!

#### STRUCTURE ANSWERS USING S.A.R

Structure your responses using S.A.R.: Situation, Action, Response. That is, you should start off outlining the situation, then explaining the action(s) you took, and lastly, describing the result.

For example: “Tell me about a challenging interaction with a teammate.”

- Situation: On my operating systems project, I was assigned to work with three other people. While two were great, the third team member didn’t contribute much. He stayed quiet during meetings, rarely infrequently chipped in during email discussions, and rarely completed his part of the projects.

- **Action:** One day after class, I pulled him aside to speak about the course and moved the discussion into talking about the project. I asked him open-ended questions on how he felt it was going, and what components he was excited about tackling. He suggested all the easiest components, and agreed to do the write-up. I realized then that he wasn't lazy – he was actually just really confused about the project. I worked with him after that to break down the components into smaller pieces, and I made sure to complement him a lot on his work.
- **Result:** He was still the weakest member of the team, but he got a lot better. He was able to finish all his work on time, and he started asking a lot more questions. We were happy to work with him on a future project.

*Structure your answers using the S.A.R. model: Situation Action Response. This will help your interviewer clearly understand what you did and why it was great.*

The SAR model helps an interviewer clearly see what you did in a certain situation and what the result was.

## TECHNICAL SKILLS

### GENERAL ADVICE

Interviews are supposed to be difficult. If you don't get every – or any – answer immediately, that's ok! In fact, in my experience, maybe only 10 people out of the 120+ that I've interviewed have gotten the question right instantly.

So when you get a hard question, don't panic. Just start talking aloud about how you would solve it.

And, one more thing: you're not done until the interviewer says that you're done! What we mean here is that when you come up with an algorithm, start thinking about the problems accompanying it. When you write code, start trying to find bugs. If you're anything like the other 110 candidates that I've interviewed, you probably make some mistakes.

### FIVE STEPS TO A TECHNICAL PROBLEM

A technical interview question can be solved utilizing a five step approach:

1. **Ask Questions:** Ask your interviewer questions to resolve ambiguity.
2. **Design an Algorithm:** Devise an algorithm.
3. **Pseudo-Code:** Write pseudo-code first, but *tell* your interviewer that you're writing pseudo-code! Otherwise, he/she may think that you're never planning to write "real" code, and many interviewers will hold that against you.
4. **Code:** Write your code, not too slow and not too fast.
5. **Test:** Test your code and carefully fix any mistakes.

## STEP #1: ASK QUESTIONS

Technical problems are more ambiguous than they might appear, so make sure to ask your own questions to resolve anything that might be unclear. You may eventually wind up with a very different – or much easier – problem than you had initially thought. In fact, many interviewers (especially at Microsoft) will specifically test to see if you ask good questions.

Good questions might be things like: What are the data types? How much data is there? What assumptions do you need to solve the problem? Who is the user?

Example Question and Answer: “Design an algorithm to sort a list”.

- *Question:* What sort of list? An array? A linked list?  
*Answer:* An array.
- *Question:* What does the array hold? Numbers? Characters? Strings?  
*Answer:* Numbers.
- *Question:* And are the numbers integers?  
*Answer:* Yes.
- *Question:* Where did the numbers come from? Are they IDs? Values of something?  
*Answer:* They are the ages of customers.
- *Question:* And how many customers are there?  
*Answer:* About a million.

Now, we have a pretty different problem: sort an array containing a million integers between 0 and 130. How do we solve this? Just create an array with 130 elements and count the number of ages at each value.

## STEP #2: DESIGN AN ALGORITHM

What do you do when you have no idea how to solve a problem? Try one of the four approaches below.

### APPROACH 1: PATTERN MATCHING

*Description:* Consider what problems the algorithm is similar to, and figure out if you can modify the solution to develop an algorithm for this problem.

*Example:* A sorted array has been rotated so that the elements might appear in the order 3 4 5 6 7 1 2. How would you find the minimum element?

*Similar Problems:*

*Don't make assumptions – ask your interviewer questions. Your interviewer may be specifically looking for that, or they may have just forgotten to tell you some details. Either way, make sure to ask lots of questions!*

- Find the minimum element in an array.
- Find a particular element in an array (eg, binary search).

*Algorithm:*

Finding the minimum element in an array isn't a particularly interesting algorithm (just iterate through all the elements), nor does it use the information provided (that the array is sorted). It's unlikely to be useful here.

However, binary search is very applicable. You know that the array is sorted, but rotated. So, it must proceed in an increasing order, then reset and increase again. The minimum element is this "reset" point.

If you compare the first and middle element (3 and 6), you know that the range is still increasing. This means that the reset point must be *after* the 6 (or, 3 is the minimum element and the array was never rotated). We can continue to apply the lessons from binary search to pin point this reset point, by looking for ranges where LEFT > RIGHT. That is, for a particular point, if LEFT < RIGHT, then the range does not contain the reset. If LEFT > RIGHT, then it does.

*If you're going to read anything, read about these four approaches!*

*There are four common approaches to developing an algorithm: Pattern Matching, Simplify and Generalize, Base Case and Build, and Data Structure Brainstorm.*

## APPROACH II: SIMPLIFY & GENERALIZE

*Description:* Change a constraint (data type, size, etc) to simplify the problem. Then try to solve it. Once you have an algorithm for the "simplified" problem, generalize the problem again.

*Example:* A ransom note can be formed by cutting words out of a magazine to form a new sentence. How would you figure out if a ransom note (string) can be formed from a given magazine (string).

*Simplification:* Instead of solving the problem with words, solve it with characters. That is, we're cutting characters out of a magazine to form a ransom note.

*Algorithm:*

We can solve the simplified ransom note problem with characters by simply creating an array and counting the characters. Each spot in the array corresponds to one letter. First, we count the number of times each character in the ransom note appears, and then we go through the magazine to see if we have all of those characters.

When we generalize the algorithm, we do a very similar thing. This time, rather than creating an array with character counts, we create a hash table. Each word hashes to the number of times the word appears.

## APPROACH III: BASE CASE AND BUILD

*Description:* Solve the algorithm first for a base case (eg, just one element). Then, try to solve it for elements one and two, assuming that you have the answer for element one. Then, try to solve it for elements one, two and three, assuming that you have the answer to elements one and two.

*Example:* Design an algorithm to print all permutations of a string. For simplicity, assume all characters are unique.

*Test String:* abcdefg

*Case "a" → {a}*

*Case "ab" → {ab, ba}*

*Case "abc" →*

This is the first "interesting" case. If we had the answer to  $P("ab")$ , how could we generate  $P("abc")$ . Well, the additional letter is "c", so we can just stick c in at every possible point. That is:

ab → cab, acb, abc

ba → cba, bca, bac

*Algorithm:* Use a recursive algorithm. Generate all permutations of a string by "chopping off" the last character and generating all permutations of  $s[1... n-1]$ . Then, insert  $s[n]$  into every location of the string.

**NOTE: Base Case and Build Algorithms often lead to natural recursive algorithms.**

#### APPROACH IV: DATA STRUCTURE BRAINSTORM

*Description:* This is hacky, but it often works. Simply run through a list of data structures and try to apply each one.

*Example:* Numbers are randomly generated and stored into an (expanding) array. How would you keep track of the median?

*Brainstorm:*

- Linked list? Probably not – linked lists tend not to do very well with accessing and sorting numbers.
- Array? Maybe, but you already have an array. Could you somehow keep the elements sorted? That's probably expensive. Let's hold off on this and come back if it's needed.
- Binary tree? This is possible, since binary trees do fairly well with ordering. In fact, if the binary search tree is perfectly balanced, the top might be the median. But, be careful – if there's an even number of elements, the median is actually the average of the middle two elements. The middle two elements can't both be at the top. This is probably a workable algorithm, but let's come back to it.
- Heap? A heap is really good at basic ordering and keeping track of max and mins. This is actually interesting – if you had two heaps, you could keep track of the biggest half and the smallest half of the elements. The biggest half is kept in a min heap, such that the smallest element in the biggest half is at the root. The smallest half is kept in a max heap, such that the biggest element of the smallest half is at the root. Now, with these data structures, you have the median elements at the roots. If the heaps are no longer the same size, you can quickly "rebalance" the heaps by popping an element off the one heap and pushing it onto the other.

#### FINDING PROBLEMS WITH YOUR ALGORITHM

Don't forget to think about:



- What's the space and time complexity?
- What if there is a lot of data?
- Does your design cause other issues? (eg, if you're creating a modified version of a binary search tree, did your design impact the time for insert / find / delete?)
- If there are other issues, did you make the right trade offs?
- If they gave you specific data (eg, mentioned that the data is ages, or in sorted order), have you leveraged that information? There's probably a reason that you're given it.

Also, remember that the four approaches can be "mixed and matched." That is, once you've applied "Simplify & Generalize", you may want to implement Pattern Matching next.

---

### STEP #3: PSEUDO-CODE

Writing pseudo-code first can help you outline your thoughts clearly and reduce the number of mistakes you commit. But, *make sure to tell your interviewer that you're writing pseudo-code first and that you'll follow it up with "real" code.* Many candidates will write pseudo-code in order to 'escape' writing real code, and you certainly don't want to be confused with those candidates.

---

### STEP #4: CODE

**Use Data Structures Generously:** Where relevant, use a good data structure or you can even define your own. For example, if you're asked a problem involving finding the minimum age for a group of people, consider defining a data structure to represent a Person. This shows your interviewer that you care about good design.

**Don't Crowd Your Coding:** This is a minor thing, but it can really help. When you're writing code on a whiteboard, start in the upper left hand corner – not in the middle. This will give you plenty of space to write your answer.

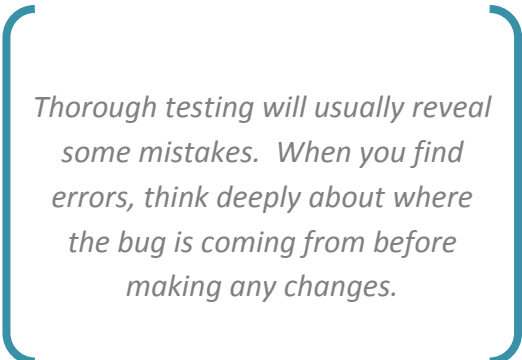
---

### STEP #5: TEST

#### Test Your Code!

Consider testing for:

- Extreme cases: 0, negative, null, maximums, etc
- User error: Can the user pass something in which will break your code?
- General cases: Test the normal case.



*Thorough testing will usually reveal some mistakes. When you find errors, think deeply about where the bug is coming from before making any changes.*

#### Fix Mistakes Carefully!

One of the worst things I saw while interviewing was candidates who recognized a mistake and tried making "random" changes to fix the error.

I would see this scenario a lot. For example, a candidate writes a function that returns a number. When she tests her code with the number '5' she notices that it returns 0 when it should be returning 1. So, she changes the last line from "return ans" to "return ans+1," without thinking through *why* this would resolve the issue.

When you notice problems in your code, really think deeply about why your code failed before fixing the mistake.

## GOOGLE INTERVIEW PROFILES

DIPA | COLLEGE HIRE, SOFTWARE ENGINEER | SEPT 2009

### CANDIDATE DESCRIPTION

Dipa is a student a senior at Carnegie Mellon University. She is majoring in Computer Science and has a 3.3 GPA.

Last year, she interviewed with Google for a Software Engineering Internship. She completed two phone screens, but did not get an offer. She ended up interning at Microsoft instead as a Software Design Engineer in Office.

She is interviewing for a software engineering position in the New York office.

### INTERVIEW QUESTIONS

#### PHONE SCREEN

1. Tell me about your senior project. What did you enjoy the most? What was the hardest part?
2. You have a set of points across the globe as longitude and latitude. How would you find all the points that are within n miles?

**Candidate Score:** 3.4

#### ON-SITE, FIRST INTERVIEW

1. Why do you want to work for Google?
2. If you could do anything in your 20% time, what would you want to do?
3. You have a binary tree where each node knows the number of nodes in its sub-tree (including itself). Given a node n and an int k, write a function to return the kth node in an in order traversal.

Can you do this non-recursively?

4. How would you figure out the number of 1 bits in an integer? If this were a repeated operation, how could you optimize this?

**Candidate Score:** 3.1

#### ON-SITE, SECOND INTERVIEW

5. What was your favorite class and why?
6. How would you design a system which spiders the links from a webpage? Specifically, how would you check if a link has already been hit?

**Candidate Score: 3.0**

---

### ON-SITE, THIRD INTERVIEW

7. What are you working on for your senior project? What would you do if you had to handle 10GB of data instead of the 5MB you expect?
8. Write code to merge two arrays in sorted order such that if an integer is in both arrays, it only gets put into the new array once.

What if you knew that one array was much longer than the other? Could you optimize it?

9. You have a huge set of stars as three dimensional coordinates. How would you find the k closest stars?

**Candidate Score: 3.5**

---

### ON-SITE, FOURTH INTERVIEW

10. Tell me about your internship at Microsoft. What was the best thing you learned there?
11. Given a bunch of points, how would you find the line which intersects the most number of points?

Write pseudo-code for this algorithm.

**Candidate Score: 2.1**

---

### CANDIDATE PERFORMANCE

Candidate did very well on the phone screen, quickly coming up with a solution involving dividing the globe into chunks.

On her on-site interviews, she did quite well on most of the questions, usually arriving at a solution with a bit of help from her interviewer. On her fourth interview, though, the interviewer ended up basically telling her the algorithm, and then asked her to write code.

She did well in coding, overall, and wrote clean code. She did make a number of minor mistakes though, and struggled with the bit manipulation problem.

Though her fourth interviewer was not very pleased with her performance, her other interviewers were. In fact, her third interviewer was so impressed with her performance that he fought to give her an offer. In part due to his persuasion, the hiring committee decided to extend her an offer for the New York office.

**Average Score: 3.02**

ALEX | INDUSTRY HIRE, SOFTWARE ENGINEER | NOV 2009

---

### CANDIDATE DESCRIPTION

Alex has spent the last three years working at Web 2.0 start-up, where he focused on the backend. He has a bachelors and masters degree from University of Illinois, where he did a thesis related to scalability of large systems. He is worried about the direction of his current company, and is now looking for other options.

---

## INTERVIEW QUESTIONS

---

### PHONE SCREEN

1. Why are you interested in Google? What would you want to work on here?
2. Imagine you had a dictionary. How would you print all anagrams of a word?

What if you had to do this repeatedly? Could you optimize it?

Using the shared Google Doc, write the code for this problem.

**Candidate Score:** 3.2

---

### ON-SITE, FIRST INTERVIEW

3. How much of your day do you spend coding? What role do you have in the architecture of your project?
4. Write code to return the nth to last element in a linked list.
5. Write code to print out a binary search tree, such that all nodes at a given depth are printed on the same line.

**Candidate Score:** 2.7

---

### ON-SITE, SECOND INTERVIEW

6. What has been the hardest project at your current job? What have you enjoyed the most?
7. You have two huge files containing data which is sorted order. Write pseudo-code to return a merged file containing all the data in sorted order.

Write pseudo-code to handle this.

8. What if you now have thousands of sorted files containing millions of entries? How would you modify your algorithm to handle this?

**Candidate Score:** 2.8

---

### ON-SITE, THIRD INTERVIEW

9. Have you ever worked on open-source projects? What did you like about it? What did you dislike?
10. Write code to find the next prime number after a given number.
11. How would you write a benchmark to calculate the time of each memory access?

**Candidate Score: 3.1**

---

## ON-SITE, FOURTH INTERVIEW

12. What language do you usually code in? Do you enjoy it? Why or why not?
13. Given an array with integers and a number  $n$ , design an algorithm and write code to print all pairs of numbers that sum to this number.
14. What if you now had to find all sets of three numbers that summed to this value? How would you do this?

**Candidate Score: 2.2**

---

## CANDIDATE ASSESSMENT

Though Alex hasn't worked on low-level systems for a while, he was able to think through the benchmark question remembering the just the basics. The interviewer had to help him a bit here and there by reminding him of some details, but he was able to quickly apply this information.

On the large scale / system design questions, he did reasonably well. He didn't get the algorithm parts of the questions very quickly, but he understood a lot general approaches to scalability from his work and academic experience.

His interviewers were very concerned about his coding ability. Though he was able to implement the general idea just fine, his code was messy and disorganized. He frequently had duplicate code, rather than pulling out code into common method. He also made a number of mistakes, and never tested his own code well.

The hiring committee decided not to extend an offer to Alex. They had some minor concerns about his intelligence – he seemed very smart on certain problems, but then on others he struggled more than they would have liked. The biggest issue, however, was his code. They needed him to write good, clean code with limited bugs, and they didn't see that from him. The recruiter said that they wouldn't be moving forward with him, and encouraged him to reapply in six months.

**Average Score: 2.8**

JASON | SOFTWARE ENGINEER, COLLEGE HIRE | APRIL 2009

---

## CANDIDATE DESCRIPTION

Jason graduated from University of Washington in December 2008, and spent several months traveling. He is now interviewing for a position with Google.

His GPA is lower than what Google usually hires – 2.7 out of 4.0 – but he came in with a strong recommendation from a former classmate. He has also worked extensively on an open source project to edit images, which impressed the recruiter. Two summers ago, he interned at a local start-up, which has since collapsed. Last summer, he spent his days split between the open source project, and a software idea he was exploring.

---

## INTERVIEW QUESTIONS

---

### PHONE SCREEN

1. How is work organized on the open source project you work on? How much coding do you do?
2. Write code to detect a loop in a linked list.
3. Implement a vector class (eg, resizing array).

**Candidate Score:** 3.1

---

### ON-SITE, FIRST INTERVIEW

4. What was your favorite course in college? What are the useful or practical lessons from that class?
5. Design and implement an algorithm to merge N sorted arrays.
6. Google images has a feature where you are shown related searches (eg, people who searched X also searched Y). How would you design this feature?

**Candidate Score:** 2.9

---

### ON-SITE, SECOND INTERVIEW

7. How did you get involved with the open source project? What has been the most interesting bug?
8. Imagine you have data being pulled very frequently from a large database. How would you design a MRU (most recently used) cache?
9. Implement `getValue(int id)` for the MRU cache.

**Candidate Score:** 3.0

---

### ON-SITE, THIRD INTERVIEW

10. If you worked at Google and could do anything for your 20% project, what would you do?

11. Implement malloc.

**Candidate Score:** 3.5

---

## ON-SITE, FOURTH INTERVIEW

12. Why do you want to work for Google?

13. Suppose you are working for the phone company and you run the division that gives out phone numbers. Phone numbers are assigned in one of two ways:

a. `bool GiveMeThisNumber(Person p, int number)`

Assigns the person a specific number that they requested, returning true if the number is available.

b. `int GiveMeAnyNumber(Person p)`

Assigns the person any arbitrary number that is currently unassigned.

How would you design the algorithm for those methods? You can use any data structure you would like to hold the phone numbers.

14. Implement GiveMeThisNumber and GiveMeAnyNumber.

**Candidate Score:** 3.3

---

## CANDIDATE ASSESSMENT

In the phone screen, Jason explained that he already knew Question #2 (detect loop in a linked list). The interviewer asked him to do the problem anyway, so that he could see the code. Jason was able to quickly do this, and the interviewer moved onto the next question.

On his onsite interviews, Jason struggled a lot with the algorithm to merge N sorted arrays, but did well enough on the next question to compensate for this.

The hiring committee noted in the feedback that Jason seemed to do particularly well in open-ended questions, where he wrote very well-designed code, complete with custom data structures. Likewise, Jason did well in lower-level questions, due to his extensive experience with C++.

Despite the fact that Jason struggled a lot during one question, there were no debates on the hiring committee about hiring Jason.

**Average Score:** 3.16



## GOOGLE FREQUENTLY ASKED QUESTIONS

### WHAT LANGUAGE DO I NEED TO CODE IN?

CareerCup recommends either Java, C++ or C#. You *can* write in C, but you'll struggle a lot without the object oriented niceties of a more advanced language.

### IF I'M REJECTED, CAN I REAPPLY?

Yes! Usually Google just asks that you wait six months, and then you're welcome to reapply. You should email your recruiter explaining that you recognize that you were rusty on some concepts, and that you've brushed up and are confident that you can do better.

### I GOT AN OFFER, BUT WHY WON'T GOOGLE TELL ME WHAT TEAM I'M ON?

Many people stress out about this, but it's actually a blessing in disguise. At Google, an engineer doesn't "belong" to a team like she might at many other companies. Rather, you're hired into Google in general, and you're given a ton of flexibility about what you want to work on.

Your current team is seen a temporary position, and you are expected to switch every 1 – 1.5 years. Switching teams is a very easy process that usually just involves some informal conversations with the team. You do not interview to switch teams.

Additionally, because of your 20% project, you might end up spending much of your time working on another project.

So, basically, Google isn't telling you your team because it doesn't mean much. You switch often, and you might be working on multiple teams. As long as there are multiple projects in your office that you'd be open to working on, I wouldn't worry about it.

### HOW DOES 20% TIME WORK? DOES IT REALLY EXIST?

Absolutely - and there's a wide variety of projects that people do. Here are just a few examples:

- Implementing a feature on Gmail Labs that you've wanted to see for a while.
- Fixing some bugs that bother you on Google Calendar.
- Prototyping a new design for Search.
- Creating a new automated test tool.
- Teaching a course at a local university.
- Getting involved with diversity recruiting.

There's very little oversight on it (usually – it depends on your manager). Many people will "save up" their time for a while, and then spend several days in a row working on a 20% project. Or, you might not use any for several months, and then spend 50% of your time for a while working on something. Or, maybe between switching projects, you spend some time working on a 20% project. It's up to you!