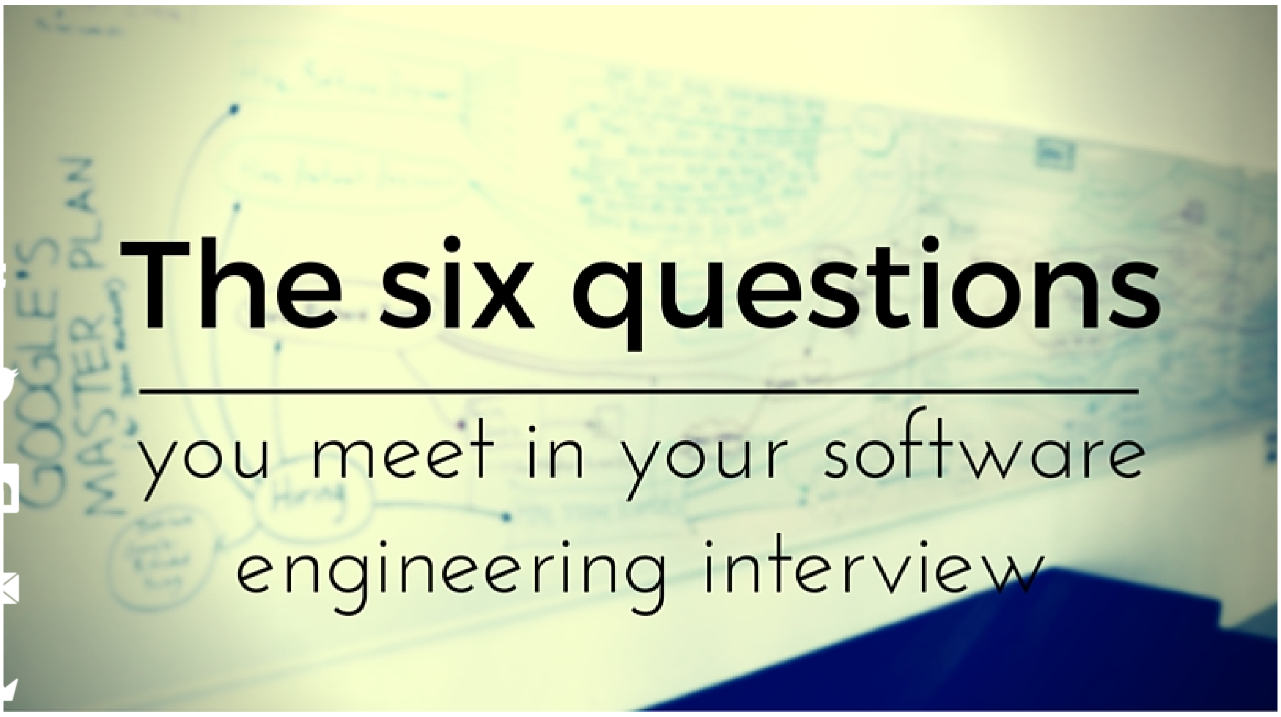
The only 6 types of questions you need to know to ace any coding interview

Posted by *Sam Gavis-Hughson*



Back when I was interviewing for jobs, I would occasionally get thrown totally off guard. I'd go into an interview and expect to be whiteboarding, only to be presented with a computer and a blinking cursor.

What do they want me to do with this? Can I run my code or not? What is does this class do? Oh god I don't even recognize this IDE!

Sometimes they'd ask me about my past experience. Shit I don't remember why we chose to use ElasticSearch.

Or they'd ask me to debug some code. Wouldn't it be nice if I remembered how to use the Eclipse debugger?

When people say "coding interview", they are usually referring to whiteboard interviews. After all, those have long been the gold standard for evaluating software engineers at modern tech companies.

Many companies continue to do whiteboard interviews simply because the other options aren't significantly better. If you're asked to code on a computer in an IDE you don't know how to use, you're at a major disadvantage compared to someone who uses it every day.

However, more and more companies are moving towards other methods of evaluation, often doing multiple interviews of different types during a single onsite visit. Each type of interview question has its own quirks and requires different preparation.

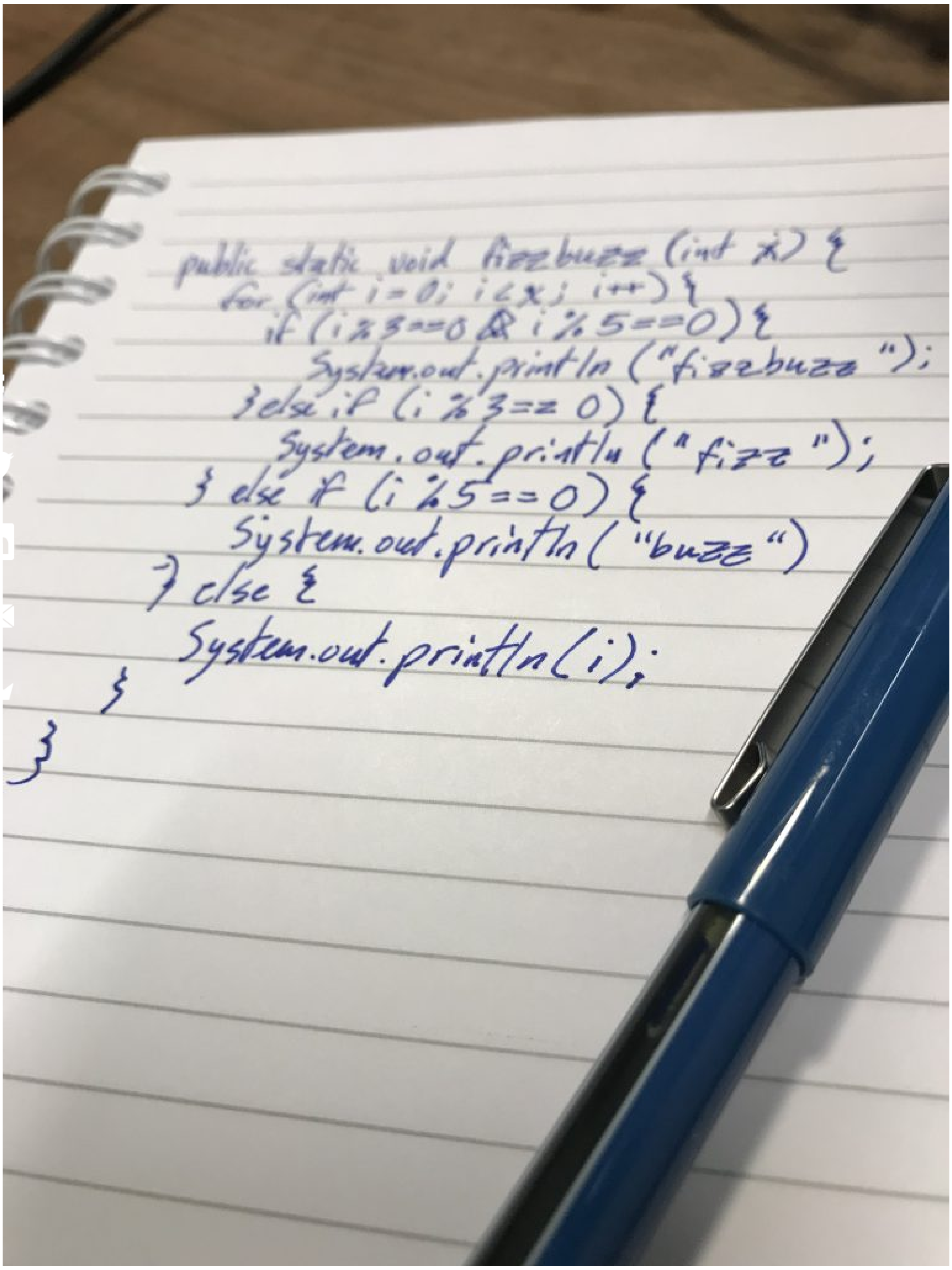
In this post, we'll go over the 6 most common types of interview questions for onsite interviews so that you'll never go into a coding interview unprepared.

Whiteboard Interview Questions

Despite the onslaught of other forms of interviewing, whiteboard interviews still reign when it comes to testing software engineers. These types of problems require you to solve an algorithmic problem (like these) on a whiteboard or sheet of paper without the aid of an IDE or a compiler.

While whiteboard interviews remain popular today, they originated in a time before laptops, when it was hard to to have an interviewee code on an actual computer. Old habits die hard, and this is one of them.

However, there are also more legitimate reasons that companies continue to opt for whiteboard coding interviews. Take a moment right now to write a short function down on a piece of paper in your preferred language.



```
public static void fizzbuzz (int x) {
  for (int i=0; i<x; i++) {
    if (i%3==0 & i%5==0) {
      System.out.println ("fizzbuzz");
    } else if (i%3== 0) {
      System.out.println ("fizz");
    } else if (i%5==0) {
      System.out.println ("buzz")
    } else {
      System.out.println(i);
    }
}
```

Not so much fun, right? Once you get past learning how to draw curly braces and ampersands (surprisingly difficult), you'll probably realize how much you've been relying on various tools to write your code for you.

How do you define a java function? Who knows, the IDE does it for me.

How do I know the syntax for inserting an item into a HashMap? That usually autocompletes.

As developers, we often begin to use these tools as a crutch. A crutch that can make it hard for an interviewer to actually evaluate our underlying skill.

In a whiteboard interview, your interviewer is really looking for how you can reason about code at a high level. You can't write automated tests or do much to verify your code correctness and so you have to rely purely on your own knowledge.

Let's say you need a min priority queue and you don't know what the methods are called. You can't look it up, so you have to make an educated guess. This shows the interviewer the way you think about code organization and how you reason through things you don't already know.

Most good developers can intuitively get very close to the correct method definition, even if they don't know the data structure, because they know conventions for how good code should be designed.

Whiteboarding reveals a lot about a developer that is hidden if they're hiding behind a keyboard.

How to prepare

More than anything, whiteboard coding interviews are about familiarity. Familiarity with code on a deep level that you can't get when you are completely reliant on an IDE to help you along.

Since coding on a whiteboard is likely very foreign to you, it is almost guaranteed to go poorly unless you've done it before. That's why the most important way to prepare is to practice doing it.

In fact, I would strongly encourage you to ONLY practice using a whiteboard (or pen and paper if you don't have one), rather than practicing on the computer.

Enforcing this restriction on yourself will help you to make rapid progress, without having to do any additional work outside of the practice you're doing already. It's like listening to an audiobook while you commute to work; it kills two birds with one stone.

Here is a simple process for practicing whiteboard problems:

1. Pick a practice problem
2. Attempt to code up the solution on a piece of paper (or a whiteboard if you have one)
3. Test your code by hand
4. Once you're satisfied that your code is correct, copy it exactly into the IDE of your choice. Don't make corrections as you go. Then attempt to compile/run your code.
5. If it works, great! If not, create a list of the errors you made and be sure to explicitly check for those in the future.

With practice, whiteboard interviews don't have to be difficult. However, if you don't give them the attention they deserve, they will come back to bite you.

[convertkit form=5260926]

Conceptual Interview Questions

Say I'm interviewing you and take a look at your resume. "Oh I see you have 5 years of Java experience. Why don't you explain to me when you would use a 'finally' clause?"

This is a perfect example of what I call 'conceptual' interview questions. The goal here is to test your knowledge of concepts that you either claim to know or your interviewer thinks that you should know.

These tend to be language specific questions or very basic system design questions. They may range anywhere from what different keywords mean in a language to questions like "Tell me as many ways as you can to transfer a file between two computers." They won't require any coding.

So why might an interviewer ask one of these questions? They probably seem really easy, right?

First of all, when you say you have certain experience on your resume, that had better be true. These questions should be trivial if you know what you say you know.

Anyone with real Java experience should know that a 'finally' clause is used to clean up after a 'try-catch' statement.

Unfortunately for interviewers, it's far too common to see people putting skills on their resumes that aren't really indicative of their actual abilities. So don't be that person.

Second, and equally important, these questions test your ability to communicate clearly. Clear communication is critical between members of an organization, so your ability to explain concepts counts for a lot.

Let's consider a good example of explaining a 'finally' clause:

"The 'finally' clause is a block that runs at the end of a 'try-catch' statement every time, regardless of whether or not there was an exception. This is important because it ensures that we execute the code even if there is a return/break/continue within the 'try-catch'."

Notice in this case the answer is very simple but we clearly explain both what it is and why. You should be able to explain the concept clearly at whatever level is necessary for others to understand you.

How to prepare

Above everything else, make sure that your resume is an accurate representation of your abilities. I can't stress this enough.

That being said, it's normal to put skills on your resume that you haven't exercised in a while. If you're going to do that, they are worth some review.

If you put a specific language on your resume, make sure to go back and review the basics. Review any frameworks that you claim to know.

Primarily, you want to focus on language-specific paradigms and questions that they might ask to test your knowledge. If you google "[coding language] interview questions" it is easy to find tons of questions on whatever language you're looking for.

It is also useful to review basic computing concepts like networking or operating systems, particularly if the job is related to those fields. Interviewers often like to know that you have an understanding of the underlying system. You likely won't be expected to go into great depth, but you should know how network packets work or basic methods of task scheduling.

No need to spend a lot of time here, since most of this should be knowledge you have already, but it never hurts to bone up.

Computer Coding Interview Questions

I said earlier that whiteboard interviews came into existence because of the lack of portable computers. Well, welcome to the modern era! We're out of the dark ages now.

With the proliferation of laptops, computer coding interviews are becoming more and more common. They do a better job than whiteboard interviews because they much more closely mirror the environment that engineers work in on a daily basis.

Computer coding interviews are very similar to whiteboard interviews in terms of the questions you will be asked and the approach you should take, with the main difference being that you're coding on a computer in an IDE rather than on a whiteboard. This also means you can run your code and see if it works!

While on the surface, these interviews may seem to test the same things as whiteboard interviews, whiteboarding puts a big obstacle in your way – the whiteboard. Computer coding interviews allow your interviewer to test you at a higher level.

In a computer coding interview, you're not worrying about exact syntax. After all, you have all the tools you love to help you do that properly. You're not worrying about the correctness of your code on a low level because you can just run it and see if it works (obviously your algorithm may not work, but it's easy to see if your for loop goes out of bounds).

This ultimately allows you to focus on the bigger picture of problem solving rather than memorizing exact syntax and minutiae in your language of choice.

How to prepare

Since computer coding interview questions are generally similar in content to whiteboard interviews, I generally recommend a similar study process, minus the need to practice writing code out by hand.

To start with, it's critical to have a good foundation. Make sure you know how to implement all the most common data structures and algorithms from scratch (hashmaps, linked lists, priority queues, etc). If you're unsure whether you need to know something, the index of Cracking the Coding Interview [Affiliate Link] is a great place to look. If it's there, it's worth a review.

After you've built a strong foundation, practice problems and mock interviews is where it's at. You want to make sure that you practice until you get this truly under your belt and feel comfortable in your interview.

Lastly, it's worthwhile playing around a little with the most common IDEs for your language. You can certainly ask your recruiter ahead of time what development environment to expect, but especially if you don't get a clear answer, it's worth at least having a baseline familiarity with the options. For Java, for example, this means having used both Eclipse and IntelliJ.

System Design Interview Questions

System design interviews are a great example of a question where you'll either get one or none in your interview process. They are not as common as coding questions and many companies, like Amazon and Google, won't even ask them until a candidate has 3-5 years experience.

However, that doesn't mean you shouldn't take some time to prepare.

System design questions ask you to think at a high level about how to design a product. For example, "design a distributed datastore for a movie recommendation site." (Can you guess what company might ask a question like that?)

For these interviews, you generally won't be expected to write any code. Rather your interviewer is looking at how you critically think about the problem and its constraints and come to a reasonable solution.

These interview questions are a great way for you interviewer to test how you think in a big-picture way. Can you take this design doc and build it into an actual system? Could you design an analytics tracking system to monitor all of the orders going through the site? How about building a flow so users can upload videos?

As an engineer, there is much more to the job than just being able to write code. Sure you could be a code monkey, but if you've read this far, you're definitely looking for something more. Software engineers need to be able to take these sorts of large problems from the product team, translate them into a specific set of requirements, and then apply technology to meet those requirements.

System design interview questions are also a great proxy for testing real world engineering experience. As you gain more experience as a developer, it becomes more intuitive how to break down problems. There are common patterns that repeat over and over. These questions help your interviewer to deeply evaluate your level of experience.

How to prepare

These questions are not the easiest to prepare for. Since system design questions are really meant to test real world experience, working as a software engineer is the best way to prepare.

However, there are some specific things you can do, assuming that is not an option.

First, we can start with existing examples. How does Twitter or Facebook Messenger work? What are the underlying technologies?

The coolest part is that many companies share details of how they designed parts of their systems. Uber, for example, has tons of great articles on their engineering blog (LINK) about how they built out different pieces of technology.

This first step is very much a research process. Google "facebook messenger technology" or check out sites like Gainlo where he breaks down these sorts of design problems.

Once you've seen what's out there, it's time to try this for yourself. Googling "system design interview question" will pop up plenty of examples for you to test out.

Pick a problem and try to come up with the most detailed design that you can. I recommend you do this by using the following steps:

1. Clearly define the problem. This means that we want to know, for example, how much traffic we're trying to deal with, how many users, how much data, etc. It's a really different problem if there are 10 users or 10 million users.
2. Figure out what exactly your design will cover and won't cover. If you're designing a messenger app, do you want to be able to send photos in addition to text messages? You want to reduce your design down to a minimal core set of features.
3. Design each feature separately and then bring them together. In the same way we write multiple methods in our code to abstract out the details, you want to think about this when solving design problems. It makes it so much easier when you just have discreet pieces to tie together rather than trying to do everything at once.

System design problems are as much an art as a science. They are also unlikely to make or break your interview, since there's a less clearly defined "right" or "wrong" answer to the question. Rather, doing well or poorly will sway you one way or the other depending on your other interviews.

The key here is to continue improving your skills. Keep studying and practicing and working as a software engineer and your skills will continue to improve.

Debugging Interview Questions

"Guess what?! I wrote some buggy code and you get to fix it." Debugging interview questions are exactly what they sound like: You are given some code that doesn't run as expected and you have to find and resolve any bugs.

"Guess what?! I wrote some buggy code and you get to fix it."

CLICK TO TWEET

In my entire experience interviewing for jobs, I've only ever directly seen an interview like this once; obviously they're still somewhat of a rarity. That being said, this may be my favorite form of interviewing.

If you've ever worked as a software engineer, you know how essential of a skill debugging is. As the adage goes, "programmers spend 20% of their time coding and 80% of their time debugging."

I remember a full week of investigation into a bug back in 2016 that ultimately led to a 1 line fix. The struggle is real.

So if debugging is such a critical component, it makes sense that employers should test potential engineers on that skill.

And taking it a step further, debugging is a composite skill; it combines many other necessary skills. You have to be able to read and interpret code, understand the intended functionality, locate the problem, and write new code that adequately solves it.

Debugging ability tells the interviewer a lot about a potential candidate.

As well as testing an engineer's general skill set, it also highlights two particularly important abilities (or lack thereof): Ability to read code and ability to use the available tools effectively.

Reading code is difficult. There can be so many dependencies that it is really hard to know what a given function is doing, especially when developing with larger systems. However, it's absolutely a critical skill. There's no other way to learn the codebase and start effectively contributing.

The better you are at understanding code, the faster you will be able to integrate into the company and start making productive contributions. Employers don't want to sit around for months while you muddle through different areas of the codebase.

Most organizations also have relatively poorly documented code. In the interest of moving quickly, documentation is often pushed off until "later". It's also hard to document code that is in flux, so documentation rarely gets added at the time of writing, meaning that much of the original intention can get lost.

Being able to use the tools effectively is also a critical skill. There are so many powerful tools at our disposal that it would be a shame not to use them. Why waste time doing things manually when you can automate them in 1/10th of the time.

Ability to use tools well not only shows that you care about efficiency, but is also indicative of a deeper level of past experience. It's unlikely that you would have much experience using the Eclipse debugger if you've only coded for your Intro CS class. However many seasoned engineers will have plenty of experience using that or one of the many other available debuggers.

How to prepare

It's hard to practice debugging directly, unless you have a specific buggy project that you're working on. However, that doesn't mean that you can't get better at debugging interviews.

The key is to focus on the auxiliary skills. This means improving your ability to quickly read and interpret code and learning how to debug code effectively, both with and without debugger tools. You may end up in a situation where the only tools available to you are print statements, so make sure you know how to debug without using your favorite tools.

To bone up on your debugging skills, learn the common commands for your prefered debugger. This should be the ***generally preferred*** debugger for your language. Don't use some niche tool and expect access to that in your interview. Understand how to get stack traces, view variable values, set breakpoints, and other key techniques.

Also spend time refreshing your memory of how to do these same things manually, in case you don't have a debugger or they present you with one that is unfamiliar. This article covers many of the basic techniques that you should have in your back pocket.

Once you have the debugging stuff down, practice reading code. Find an open source project in your language that interests you and start reading. Try to understand exactly what is going on in the code. Here's a good article on getting up to speed with new code.

The ability to read code quickly and accurately will make it way easier for you to pinpoint issues. I also highly encourage you to try running the code and adding print statements so that you can more clearly see the control flow.

If you're looking for a good open source project, this is a good place to start for those with little open source experience.

Behavioral Interview Questions

Behavioral interview questions may be the easiest of all the question types listed here. They may also be the hardest. It kinda depends on how comfortable you are talking about yourself. I personally love talking about myself, but if you don't, you may want to spend some extra time here.

These are the only interview questions on this list that are completely nontechnical, but that doesn't make them any less important. This is where your interviewer gets to probe into the deep recesses of your past experiences.

Imagine it like a first date where your date only cares about your work history. They're probably not going to ask you about how you like to sing in the shower or your trip to the Bahamas. This is strictly work talk.

In reality, behavioral interviews are what most people would consider normal interviews. They are certainly the most "traditional" interview questions you will be asked and are often asked by a recruiter or someone from HR. If you have a recruiter reach out to you asking to chat about a potential job, expect these sorts of questions.

Although it's come under criticism for being used discriminatorily, many companies still take culture fit very seriously. Behavioral questions are often used as a way to judge culture fit.

About 50% of culture fit is "do we like you?" This is something that is evaluated throughout the entire interview process. The other 50%, though, is evaluated directly by behavioral interviews. The interviewer is fundamentally trying to answer the question "will you work well with us?"

Behavioral interview questions primarily ask about your past experience working with teams to determine how you might work with their team. Essentially, interviewers use your past experience as an indicator of future behavior.

Like System Design and Conceptual interviews, these interviews are also testing how well you communicate. Good communication is critical to an effective organization and interviewers want to see that you are able to communicate your thoughts clearly. Do you know what is most important? Are you able to focus the conversation so that everyone learns exactly what they need to know? This is a great skill to have within an organization.

How to prepare

With behavioral interview questions, this is the closest I'll ever get to recommending that you memorize your answers. Simply put, there are a limited number of questions that you may be asked, so it's good to have answers prepared for all of those.

These questions generally look like "What was the hardest challenge you faced with project X?" "What would you do differently with project Y?" Gayle McDowell outlines a great way to prepare for these questions in Cracking the Coding Interview [Affiliate Link].

As shown on page 6 of this presentation Gayle recommends creating a grid with your three best/biggest/most recent projects on your resume along the top and common behavioral questions along the side. Creating this table is super valuable, because as you fill it out, you can find relevant stories for each project you may be asked about.

Doing advanced prep like this will help your interview go much more smoothly. Rather than fumbling around and trying to remember the details of past projects, you've taken the time to think clearly about them already and have your answers ready to go.

To take this one step further, you can even practice your exact responses to different questions. You may or may not be asked those exact questions, but it is valuable to practice answering out loud. It can be hard to frame your thoughts clearly, and a little practice goes a long way.

Finally, while you should never lie in your interview, don't say things that are unflattering about yourself. If there are projects that didn't go well or you didn't handle properly, frame them as a learning experience. "Here's how I do things differently now because of what happened."

This post is not an exhaustive list of every question you could be asked. However, the techniques and frameworks here should be able to help you answer pretty much any question you could be asked.

While I know it's not a lot of fun, preparing for these different types of questions is critical to doing well in your interview. You are a performer, and you'd better know your damn lines. Maybe you're improvising, but even improv troupes practice regularly to get the skills under their belts.

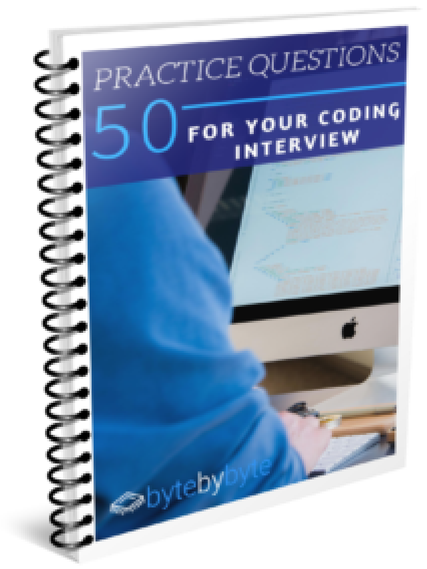
Take the time now, and once you get an offer for your dream job you can kick back and relax. Well at least until work starts.

Think I missed something? Experience any particularly odd interview questions? Let me know in the comments below!

Don't do another coding interview...

...Until you've mastered these 50 questions!

GET YOUR FREE GUIDE



Sam Gavis-Hughson

Sam, founder of Byte by Byte, helps software engineers successfully interview for jobs at top tech companies.

Sam has helped thousands of students through his blog and free content -- as well as 400+ paying students -- land jobs at companies such as Google, Amazon, Microsoft, Bloomberg, Uber, and more.

← How to get time off work for job interviews

Your goals are probably unrealistic...but here's what you can do about it →

2 Comments **Byte By Byte** 1 **Login**

Recommend Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Chris Nyles • 2 years ago

This is a great document and would remain a reference for me for all the future interviews. For system design interviews I have found following two Quora answers quite helpful:

https://www.quora.com/How-c...
https://www.quora.com/What-...

Also do read this course, it has discussed a good set of design questions:
https://www.educative.io/co...

• Reply • Share ›

Sam Gavis-Hughson Chris Nyles • 2 years ago

Thanks for adding those resources. There definitely looks to be a lot of info there that people could find useful!

• Reply • Share ›

ALSO ON **BYTE BY BYTE**

The reason why you should always find a brute force solution first

2 comments • 3 years ago

Avatar **Sam Gavis-Hughson** — That's certainly true that it doesn't always help you find the optimal solution, but I think that in that

Coding Interview Question: Median of Arrays

10 comments • 2 years ago

Avatar **Peter** — There's no chance in hell I could figure this out in an interview. I should just cancel the interview now if this is

Coding Interview Question: Consecutive Array

7 comments • 3 years ago

Avatar **Hitesh** — #include "QuickSort.hpp"#include

Private: How to use Cracking the Coding Interview effectively

2 comments • a year ago

Avatar **Sam Gavis-Hughson** — 6.

Search …

DYNAMIC PROGRAMMING CRASH COURSE FOR NON-GENIUSES

Download my free guide to learn:

- How to finally "get" what Dynamic Programming really is – no Ph.D required
- The not-so-obvious way you can solve any dynamic programming problem fast – and not freeze up during your interview
- The only 10% of information you need to know to ace your interview – forget all the useless fluff

Enter your email below and get instant access to your free Dynamic Programming guide.

GET THE FREE GUIDE

RECENT POSTS

- How To Nail the Amazon Interview: A Practical Guide
- The Ultimate Guide to Dynamic Programming
- Behavioral Interviews for Software Engineers
- Acing the Google Interview: The Ultimate Guide

INTERVIEW CAKE

Interview Cake is an awesome resource for more practice interview questions. Get 50% off for a limited time.

CRACKING THE CODING INTERVIEW

CRACKING the CODING INTERVIEW
189 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6TH EDITION

Check out my hands down favorite resource for coding interview prep here.

Made in NYC

sam@byte-by-byte.com

Youtube

Sam Gavis-Hughson is a software engineer based in New York City. Through Byte by Byte, he publishes regular coding interview question videos, demonstrating proper interview techniques. He also helps many students by offering practice coding interviews to help them get jobs at Google, Facebook, and other exciting tech companies.