How to do a Technical Interview

By Emma Roudabush

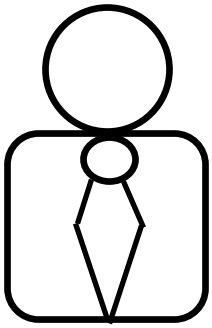
These instructions are designed for newcomers to the technology industry and therefore have never done a technical interview before. The purpose of this procedure to teach key components to be done in a typical technical interview (lasts 45 minutes to 1 hour). This guide requires at least an introductory level knowledge of programming, to a degree of basic algorithms and loops.

The high level steps to this process are (1) do your research, (2) go through the question and answer session, (3) code the technical question(s), (4) test code, and (5) finish with the interviewer.

Note: Not all "technical" interviews follow this exact format in this exact order (however it should be very similar) and not all technical companies conduct technical interviews.

Step 1: Do Your Research

There are a lot of considerations you should take into account before the day(s) of your interview(s) in order to be the most prepared and most effective during the interview.



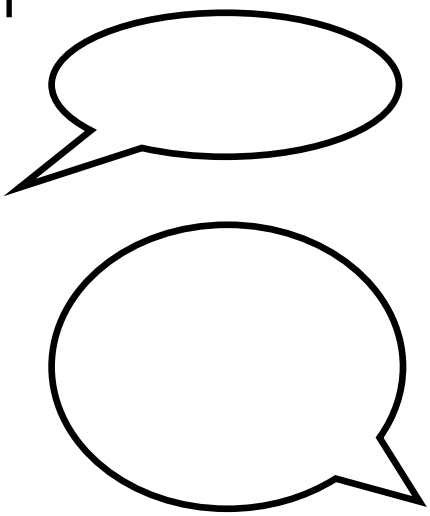
(a) Study: months before your interview, you should be reading up on common computer science concepts such as string manipulation, arrays, linked lists, trees, hashtables, and some degree of your preferred programming language syntax. They are typical areas you could be interviewed about when asked the technical question.

(b) Look at the dress code: Some companies do not have preferences as to your attire at the interview and encourage more casual wear, whereas other companies adhere to the more typical forms of business attire. If the company you are interviewing is the former, use your best judgment as to how you want to present yourself to your perspective employer.

(c) Bring necessary supplies: Some companies will ask for you to bring a copy of your resume for the interview for reference and it is most likely wise to bring a copy regardless, especially for the first part of the interview. It is also highly suggested that you bring a portfolio or at least a pad of paper in order to record any thoughts or help illustrate certain points to your interviewer.

(d) Research company: This is more recommended for lesser known company. This will make you seem knowledgeable to your interviewer, especially if you ask them questions about products their company makes.

Step 2: Preliminary Question and Answer Session



The interviewer will first ask you questions to assess your personality, previous projects or classes, your leadership ability, and/or your design/consumer focus.

For this step it is crucial that you know your resume. The interviewer will most likely have a copy of your resume in front of them and will ask you questions pertaining to certain aspects of your resume. Not knowing your resume will make you seem unprepared or like a liar and ultimately effects your interview negatively as it does not allow you to show your passion for technology, your skill in a practical setting, or your ability to work in a team.

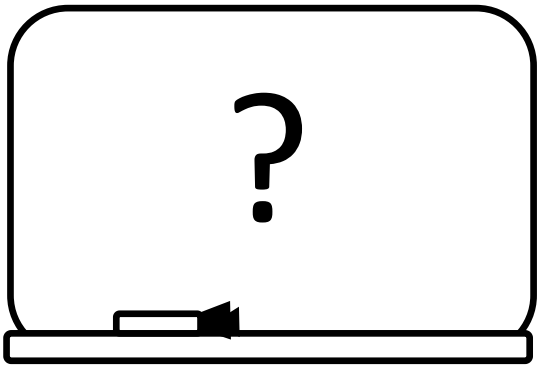
This section should take approximately 10-15 minutes.

Note: If an interviewer asks you about a project that you list on your resume, they may ask you to code a segment of it up.

Step 3: The Technical Question(s)

This is potentially the most important part of the technical interview. Your interviewer will ask anywhere between 1 to 3 questions to be solved programmatically. The question will typically entail coding a function to do a specific task. The following are steps you should take after a questions is asked. Be aware that during these steps you are fully expected to talk through your logic and thought processes with your interviewer. A general rule of thumb is that it is better to over-talk than to under-talk.

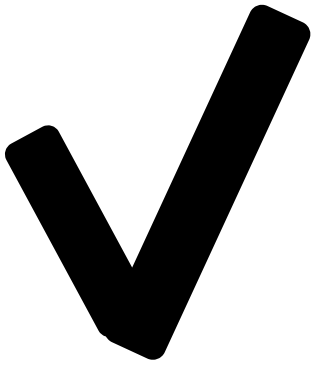
This step should take between 20-35 minutes.



(a) Ask follow-up questions: This should be the first thing you do before doing coding of any kind. Interviewers will give questions in a very vague way that will play into personal assumptions of how something is being described. Asking clarifying questions demonstrates that you are a person who gives consideration to a problem, wanting to know the full situation before solving the problem.

(b) Write pseudo-code or an outline of the program: Pseudo-code is when one writes a mixture of English and code, not adhering to many syntactic conventions. Overall, you should be writing a basic rundown of the program, paying more attention to the algorithm involved rather than the syntactic aspect. This conveys once again that you are one who gives thought into code and does not just run head first into a problem, which in the industry can lead to sloppy, inefficient code that wastes more time than it attempts to save.

(c) Write actual code: This should take the least amount of time, relatively speaking. After writing pseudo-code or an outline, you finally code the algorithm you have settled on. Most interviewers will allow you to code in whatever language you feel most comfortable in—unless you are interviewing for a specialized position—though you will want to confirm this early on. The trick to writing the code is to not too get hung up on a lot of syntactic intricacies for it can waste your time for nothing; an interviewer understands that many of your syntactic issues can be caught and solved by your development environment or online.



Step 4: Testing

After doing the coding question(s), the interviewer will ask you to provide test cases for your code. This step is to see how you would go about determining the quality of your code and your general testing aptitude. It is best when providing test cases to not only say them, but to write down the parameters that you would give for any given test case and the expected output based on the description of the program—as many software testers would think about when determining a testing regimen. You should not only list and talk about the simple test cases (functional testing) but also the less common or edge test cases—which you may have not initially considered when programming your solution.

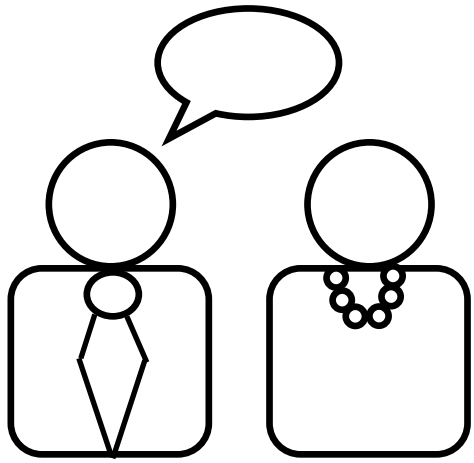
This part should take about 5 to 10 minutes.

Note: This may or may not be formally asked of you in an interview; however, it is highly suggested that you at least self test your code before you are done with a simple test case.

Step 5: Final Questions

This is the final part of the interview where the interview tries to wound down and come to an easy close. At this point you have the option to ask your interviewers questions and it is highly recommended that you ask him/her 2-3 questions. This is the point where you can learn about the company culture, your team, and potentially even your project should you get the job and accept. This is also an opportunity to show off your knowledge of the company and illustrate your passion for the company and position.

The time for this step fluctuates in terms of time depending on how long the previous parts take.



Conclusion:

A technical interview, while frightening to many, is very manageable once you learn the steps to excel and show your best self. The best practice for a technical interview is more technical interviews, either real or mock. Based on the interviewer's feedback, you can adapt and become even more proficient.

Tips:

• If you are irrefutably stuck once a technical problem is given, whatever you do, never just say "I don't know." In this case, you should explain to your interviewer where you are stuck *and* give some possible ideas that you think could help you move forward in the problem. An interviewer is likely to help and guide you on the right path, and this framing shows that you are not willing to give up quickly or entirely.

• Also, if you run out of time before finishing your code or even start your code, do not panic. It is entirely possible to still move on and/or get the offer. A technical interview's purpose is more about your process when thinking on the fly, your problem solving abilities, and the hidden characteristics needed for programming in a group setting—NOT just about how well you code.

• When first setting up your algorithm, it is better to go brute force and improve from there rather than take a long time contemplating the most efficient way to solve the problem. Overall, it is better to have a simplistic solution than no solution at all.