Dev.Ed
June 2022

Common Interview Pitfalls

…and how to avoid them!

Pitfall 1: *Writing code too early or without context.*
One thing that many people miss in interviews is explaining the solution they want to use before writing code.

Sometimes people do this because they aren't sure about the solution yet and want to poke around until they figure it out. Others do this because the answer is so obvious to them that they think we must know what they are going to write.

The problem with coding before explaining is that the interviewer is both trying to evaluate your code and guess what your goal is. If your goal is not going to work, we don't have a chance to warn you off and save you precious time. If your goal is going to work, we aren't certain of which of the 200 solutions you are going to use, so we can't help you if you miss an important condition or other operation/variable. The less we know, the more likely you are to lose time.

From an evaluation standpoint, if you don't tell us your plan, we assume it would be hard to pair with you on a project because we might be left in the dark while you are just coding away.

That doesn't mean that you have to wait until you are 100% certain of your solution before you write anything, just communicate. It's ok to say "I'm having a hard time with the solution structure, but I want to try some stuff until I figure it out. First, I want to toy with the idea of using a nested loop."

Pitfall 2: *Getting stuck in analysis paralysis.*
Many candidates are so concerned about looking smart or clever, that even when they know the brute force solution, they will not start coding, but will instead sit there and try to come up with the optimal solution until they run out of time.

Don't fall into this rabbit hole.
If you don't think of the optimal solution right away, just write the solution you know will work. Explain that it is brute force, mention that you will optimize it once you have had more time to think. If you do think of multiple solutions right away, describe each one in a short sentence and then tell them which one you plan to implement and why.

If you don't write any code, we are forced to assume you don't know how. So always write a solution if you have one in mind.

Pitfall 3: *Getting stuck when asked to optimize.*
This is a tough one, because sometimes the optimal solution is VERY hard to imagine and VERY different from the brute force's structure.

If you are asked "can you optimize this" and you don't immediately know the answer, you can stall the conversation by pointing out the line of code that is currently taking the Big O and saying, "I know that to optimize it, I need to change how I do X because X is currently the part which is bringing us to O(n^2)." Then you can move the conversation to things like "Let's brainstorm some ways I could approach that step differently."

IMPORTANT note about optimizing, it is almost always a known data structure, algorithm, or common solution pattern that optimizes the runtime. When you find the bottleneck in your code, ask yourself, "What does this look like?" and walk through a few different possibilities.

If you are REALLY stuck, walk through potential inputs. 9 times out of 10 when people are blocked, I can easily unblock them by just showing them a possible input.

Pitfall 4: *Getting blocked and shutting down.*
I often notice that candidates don't know what to do when they are stuck on part of the solution. Though just talking about it would be fine, often they just shut down, start blushing, and stop talking. If they do talk, they just say something like "I'm stuck".

The way I like to see people get out of this, is to recognize that I'm honestly just trying to see if we make a good team. They can ask for help. They just need to (1) ask the right question and (2) know that I'll help them come up with ideas, but I won't tell them how to do it – They get to keep their creative freedom.

So what should you say?
1. **Give context**: I have written ___, I believe that it currently does ___. I need it to be able to also do ___, but I'm having a hard time figuring out how to add that functionality.

2. **Prove that you tried**: ___ is what I have considered, but I'm getting stuck on ___.

3. **Make it clear you are asking for help**: Do you have any suggestions on how to proceed?

Pitfall 5: *Missing edge-cases and bugs.*
Many candidates rush into a solution without considering the edge cases. Before you write any code, try to write out a few meaningful test inputs. Clarify with the interviewer what should happen in ambiguous situations (like falsy input).

Once you have written that solution, tell the interviewer you want to test your solution and ask to perform a walk-through of some of your test inputs. Often these walk-throughs make bugs surface and give you back the points they took from you.

Pitfall 6: *Telling vague stories.*
Very often, candidates tell me stories that explain the outcome of a project in a very high level way, “I created a 2D platformer game with three levels which was used and evaluated by 30 testers before publishing it. I loved this project because…” This tells me nothing.

Did you also write the game engine, or did you use Unity? Did you make it all yourself, or did you use some existing libraries? Did you work alone, or on a team? What was your role on the team? Did you manage sprite animation, or did you just find audio clips while the rest of the team wrote code? Without detail, I am forced to assume the smallest level of involvement. I will ask for more detail, but the more I need to ask, the less value I will attribute to your storytelling skills and technical experience.

Note: that doesn’t mean that you are in trouble if they ask you questions – they might just be excited and curious.

Make sure that all your stories are focused on your exact role in the story, the actual work you personally did, how that work integrated with the rest of the project/team, and any important technologies you leveraged in your role.

And be sure to use the STAR answer method. Many people forget the 'R' or the 'S' so make sure you get those.