Interview Cheat Sheet

From Andrei Neagoie's Master The Coding Interview: Data Structures + Algorithms

The 3 pillars of good code:

1. Readable
2. Time Complexity
3. Space Complexity

What skills interviewer is looking for:

- Analytic Skills - How can you think through problems and analyze things?
- Coding Skills - Do you code well, by writing clean, simple, organized, readable code?
- Technical knowledge - Do you know the fundamentals of the job you're applying for?
- Communication skills: Does your personality match the companies' culture?

Step By Step through a problem:

1. When the interviewer says the question, write down the key points at the top (i.e. sorted array). Make sure you have all the details. Show how organized you are.

2. Make sure you double check: What are the inputs? What are the outputs?

3. What is the most important value of the problem? Do you have time, and space and memory, etc.. What is the main goal?

4. Don't be annoying and ask too many questions.

5. Start with the naive/brute force approach. First thing that comes into mind. It shows that you're able to think well and critically (you don't need to write this code, just speak about it).

6. Tell them why this approach is not the best (i.e. O(n^2) or higher, not readable, etc...)

7. Walk through your approach, comment things and see where you may be able to break things. Any repetition, bottlenecks like O(N^2), or unnecessary work? Did you use all the information the interviewer gave you? Bottleneck is the part of the code with the biggest Big O. Focus on that. Sometimes this occurs with repeated work as well.

8. Before you start coding, walk through your code and write down the steps you are going to follow.

9. Modularize your code from the very beginning. Break up your code into beautiful small pieces and add just comments if you need to.

10. Start actually writing your code now. Keep in mind that the more you prepare and understand what you need to code, the better the whiteboard will go. So never start a whiteboard interview not being sure of how things are going to work out. That is a recipe for disaster. Keep in mind: A lot of interviews ask questions that you won't be able to fully answer on time. So think: What can I show in order to show that I can do this and I am better than other coders. Break things up in Functions (if you can't remember a method, just make up a function and you will at least have it there. Write something, and start with the easy part.

11. Think about error checks and how you can break this code. Never make assumptions about the input. Assume people are trying to break your code and that Darth Vader is using your function. How will you safeguard it? Always check for false inputs that you don't want. Here is a trick: Comment in the code, the checks that you want to do… write the function, then tell the interviewer that you would write tests now to make your function fail (but you won't need to actually write the tests).

12. Don't use bad/confusing names like i and j. Write code that reads well.

13. Test your code: Check for no params, 0, undefined, null, massive arrays, async code, etc… Ask the interviewer if we can make assumption about the code. Can you make the answer return an error? Poke holes into your solution. Are you repeating yourself?

14. Finally talk to the interviewer where you would improve the code. Does it work? Are there different approaches? Is it readable? What would you google to improve? How can performance be improved? Possibly: Ask the interviewer what was the most interesting solution you have seen to this problem

15. If your interviewer is happy with the solution, the interview usually ends here. It is also common that the interviewer asks you extension questions, such as how you would handle the problem if the whole input is too large to fit into memory, or if the input arrives as a stream. This is a common follow-up question at Google, where they care a lot about scale. The answer is usually a divide-and-conquer approach—perform distributed processing of the data and only read certain chunks of the input from disk into memory, write the output back to disk and combine them later.

Good code checklist:

[]It works
[]Good use of data structures
[]Code Re-use/ Do Not Repeat Yourself
[]Modular - makes code more readable, maintainable and testable
[]Less than O(N^2). We want to avoid nested loops if we can since they are expensive. Two separate loops are better than 2 nested loops
[]Low Space Complexity --> Recursion can cause stack overflow, copying of large arrays may exceed memory of machine

Heurestics to ace the question:

[]Hash Maps are usually the answer to improve Time Complexity
[]If it's a sorted array, use Binary tree to achieve O(log N). Divide and Conquer - Divide a data set into smaller chunks and then repeating a process with a subset of data. Binary search is a great example of this
[]Try Sorting your input
[]Hash tables and precomputed information (i.e. sorted) are some of the best ways to optimize your code
[]Look at the Time vs Space tradeoff. Sometimes storing extra state in memory can help the time. (Runtime)
[]If the interviewer is giving you advice/tips/hints. Follow them
[]Space time tradeoffs: Hastables usually solve this a lot of the times. You use more space, but you can get a time optimization to the process. In programming, you often times can use up a little bit more space to get faster time

---

And always remember: Communicate your thought process as much as possible. Don’t worry about finishing it fast. Every part of the interview matters.