Getting Started part2

1. Make a good self introduction at the start of the interview

- □ Introduce yourself in a few sentences under a minute or 2.

  > Follow our guide on how to make a good self introduction for software engineers

- □ Sound enthusiastic!

  > Speak with a smile and you will naturally sound more engaging.

- □ Do not spend too long on your self introduction as you will have less time left to code.

2. Upon receiving the question, make clarifications

Do not jump into coding right away. Coding questions tend to be vague and underspecified on purpose to allow the interviewer to gauge the candidate's attention to detail and carefulness. Ask at least 2-3 clarifying questions.

- □ Paraphrase and repeat the question back at the interviewer.

  > Make sure you understand exactly what they are asking.

- □ Clarify assumptions (Refer to algorithms cheatsheets for common assumptions)
  - > A tree-like diagram could very well be a graph that allows for cycles and a naive recursive solution would not work. Clarify if the given diagram is a tree or a graph.

    - > Can you modify the original array / graph / data structure in any way?
    - > How is the input stored?
    - > If you are given a dictionary of words, is it a list of strings or a Trie?
    - > Is the input array sorted? (e.g. for deciding between binary / linear search)

- ▯ Clarify input value range.
  > Inputs: how big and what is the range?
- ▯ Clarify input value format
  > Values: Negative? Floating points? Empty? Null? Duplicates? Extremely large?
- ▯ Work through a simplified example to ensure you understood the question.
  > E.g., you are asked to write a palindrome checker, before coding, come up with simple test cases like "KAYAK" => true, "MOUSE" => false, then check with the interviewer if those example cases are in line with their expectations
- ▯ Do not jump into coding right away or before the interviewer gives you the green light to do so.

3. Work out and optimize your approach with the interviewer

The worst thing you can do next is jump straight to coding - interviewers expect there to be some time for a 2-way discussion on the correct approach to take for the question, including analysis of the time and space complexity.

This discussion can range from a few minutes to up to 5-10 minutes depending on the complexity of the question. This also gives interviewers a chance to provide you with hints to guide you towards an acceptable solution.

- □ If you get stuck on the approach or optimization, use this structured way to jog your memory / find a good approach
- □ Explain a few approaches that you could take at a high level (don't go too much into implementation details). Discuss the tradeoffs of each approach with your interviewer as if the interviewer was your coworker and you all are collaborating on a problem.

  > For algorithmic questions, space/time is a common tradeoff. Let's take the famous Two Sum question for example. There are two common solutions
  >
  > 1. Use nested for loops. This would be O(n^2) in terms of time complexity and O(1) in terms of space.
  > 2. In one pass of the array, you would hash a value to its index into a hash table. For subsequent values, look up the hash table to see if you can find an existing value that can sum up to the target. This approach is O(N) in terms of both time and space. Discuss both solutions, mention the tradeoffs and conclude on which solution is better (typically the one with lower time complexity)

- □ State and explain the time and space complexity of your proposed approach(es).

  > Mention the Big O complexity for time and explain why (e.g O(n^2) for time because there are nested for loops, O(n) for space because an extra array is created). Master all the time and space complexity using the algorithm optimization techniques.

- □ Agree on the most ideal approach and optimize it. Identify repeated/duplicated/overlapping computations and reduce them via caching. Refer to the page on optimizing your solution.
- □ Do not jump into coding right away or before the interviewer gives you the green light to do so.
- □ Do not ignore any piece of information given.
- □ Do not appear unsure about your approach or analysis.

4. Code out your solution while talking through it

- □ Only start coding after you have explained your approach and the interviewer has given you the green light.
- □ Explain what you are trying to achieve as you are coding / writing. Compare different coding approaches where relevant.

  > In so doing, demonstrate mastery of your chosen programming language.

- □ Code / write at a reasonable speed so you can talk through it - but not too slow.

  > You want to type slow enough so you can explain the code, but not too slow as you may run out of time to answer all questions

- □ Write actual compilable, working code where possible, not pseudocode.
- □ Write clean, straightforward and neat code with as few syntax errors / bugs as possible.

  > Always go for a clean, straightforward implementation than a complex, messy one. Ensure you adopt a neat coding style and good coding practices as per language paradigms and constructs. Syntax errors and bugs should be avoided as much as possible.

- □ Use variable names that explain your code.

  > Good variable names are important because you need to explain your code to the interviewer. It's better to use long variable names that explain themselves. Let's say you need to find the multiples of 3 in an array of numbers. Name your results array `multiplesOfThree` instead of array/numbers.

- □ Ask for permission to use trivial functions without having to implement them.

  > E.g. `reduce`, `filter`, `min`, `max` should all be ok to use

- □ Write in a modular fashion, going from higher-level functions and breaking them down into smaller helper functions.

  > Let's say you're asked to build a car. You can just write a few high level functions first: `gatherMaterials()`, `assemble()`. Then break down `assemble()` into smaller functions, `makeEngine()`, `polishWheels()`, `constructCarFrame()`. You could even ask the interviewer if it's ok to not code out some trivial helper functions.

- □ If you are cutting corners in your code, state that out loud to your interviewer and say what you would do in a non-interview setting (no time constraints).

  > E.g., "Under non-interview settings, I would write a regex to parse this string rather than using `split()` which may not cover certain edge cases."

- □ [Onsite / Whiteboarding] Practice whiteboard space management
- □ Do not interrupt your interviewer when they are talking. Usually if they speak, they are trying to give you hints or steer you in the right direction.
- □ Do not spend too much time writing comments.
- □ Do not repeat yourself
- □ Do not use bad variable names.
  - Do not use extremely verbose or single-character variable names, (unless they're common like `i`, `n`) variable names
- □ Do not copy and paste code without checking (e.g. some variables might need to be renamed after pasting).

5. After coding, check your code and add test cases

Once you are done coding, do not announce that you are done. Interviewers expect you to start scanning for mistakes and adding test cases to improve on your code.

- □ Scan through your code for mistakes - such as off-by-one errors.

  > Read through your code with a fresh pair of eyes - as if it's your first time seeing a piece of code written by someone else - and talk through your process of finding mistakes

- □ Brainstorm edge cases with the interviewer and add additional test cases. (Refer to algorithms cheatsheets for common corner cases)

  > Given test cases are usually simple by design. Brainstorm on possible edge cases such as large sized inputs, empty sets, single item sets, negative numbers.

- □ Step through your code with those test cases.
- □ Look out for places where you can refactor.
- □ Reiterate the time and space complexity of your code.

> This allows you to remind yourself to spot issues within your code that could deviate from the original time and space complexity.

- ▯ Explain trade-offs and how the code / approach can be improved if given more time.
- ▯ Do not immediately announce that you are done coding. Do the above first!
- ▯ Do not argue with the interviewer. They may be wrong but that is very unlikely given that they are familiar with the question.

6. At the end of the interview, leave a good impression

- ▯ Ask good final questions that are tailored to the company.

  > Read tips and sample final questions to ask.

- ▯ Thank the interviewer
- ▯ Do not end the interview without asking any questions.

Revision #1
Created 11 March 2024 07:14:23 by victor
Updated 11 March 2024 07:23:17 by victor