**Algorithmic Thinking in Problem Solving Framework**

**Dilan Ramirez & Andres Silva**

* List the steps one should take when tackling a coding problem during a coding interview. Be very specific. Your task is to create a framework/problem-solving strategy.
  1. Listen carefully to the question
  2. Read the question back to the interviewer
  3. Identify key components of the question
     1. Input
     2. Output
  4. Ask probing questions about those components
     1. Are we getting the input directly or do we have to read and parse from somewhere?
     2. If we have an array can it be empty, will it always contain what we are looking for?
     3. If not, what are some edge cases?
     4. If we have linked list as input, is it single linked, it is a loop?
     5. If input is a string, what language is it in? do we worry about special characters?
     6. How will the output be returned? In a data structure? Text file?
  5. State any assumptions about the question
     1. Can you assume the data structure class is already implemented?
  6. Take a minute or two to come up with an algorithm
     1. If nothing is coming to you, say it and perhaps start thinking about a brute force approach and worry about improving it later
  7. Communicate your thought out loud to the interviewer
     1. If you need a moment to think, say it but share your thought process after you are done.
  8. Once an algorithm is roughly in place, explain it.
     1. It is fine if some details are missing, this can be polished during the coding phase.
     2. If the interviewer asks about something, pay attention! It might be a hint.
     3. Address any concerns in the algorithm if the interviewer pointed them out.
  9. If the algorithm was green lit, you might begin start coding.
     1. If we are coding on a whiteboard, leave enough space between lines in case we need to go back
     2. have edge cases in mind while coding
     3. If you want to use a library, make sure to mention it and ask for permission.
        1. Try not to do import \* if possible.
     4. If you forget any syntax, acknowledge it, and keep going.
  10. After the code is in place, trace it using a concrete that has a good input that perhaps includes elements of the edge cases.
      1. If time allows, do another relevant and different test case.
      2. Visually writing the values of key variables on the side really helps you and the interviewer to follow the example.
      3. This is to verify that the code is correct and to catch any errors/ missing behavior.
  11. Analyze time and space complexity
      1. It is fine if you do not 100% nail this part but is very important to explain how you came up with these values.
  12. If time allows, state any improvements you can see.
      1. If there are not improvements, stating the weaknesses of the code also can help, this is just to acknowledge that given more time, you can always do better or something. Also, do not sell yourself short!
      2. Hint: you might be able to squeeze-in a dictionary and take off a whole “n” out of a nested for loop.
  13. Breathe because the hard part of the interview is almost over, you got this.
* What do you think should be done when someone “gets stuck” when solving a coding problem? List the steps. Be very specific. Feel free to use examples to illustrate your points.
  1. Remain calm, everybody gets stuck. It is normal and you can recover!
  2. Revisit algorithm design
  3. Try to trace your current solution carefully to identify what went wrong.
     1. What should be the correct behavior?
     2. What is happening?
     3. How do we get to the correct behavior from our current state?
  4. If you feel like the interviewer is there to help you and you think is adequate, you might ask for a hint. Most of the time they want to see you succeed.
* Imagine you are being interviewed and you are asked to use a whiteboard to problem solve. How would you use the whiteboard? Draw a diagram that shows the different sections that you would have and justify your decisions.

**Optimize**

Provide if possible, an optimized solution by marking on the whiteboard the parts to optimize.

**Walkthrough**

Trace the code with the provided use case.

Test it with more use cases.

**Write Solution Draft**

Write a draft of the possible solution.

Write the pseudo code solution.

**Evaluate Situation**

Describe how it can be solved by just talking and drawing.

Draw a simulation of the problem.

* Create a checklist for assessing your problem-solving process when tackling a coding problem. For example, one of you items in your checklist could be “Ask Clarifying Questions” or “Test your code”
  1. Listen to the interviewer
  2. Ask for clarifying questions if needed
  3. Come up with more examples for problem understanding
  4. Analyze and find the best suitable data structure to use
  5. Work on the solution
  6. Ask more questions if needed
  7. Think out loud my solving processing
  8. Solve the problem
     1. If cannot solve, ask for help
  9. Walkthrough and explaining the solution
  10. Use more test cases for better development
  11. Optimize if possible
  12. Ask for feedback
* Come up with a list of DOs and DON’Ts that people can use in the context of coding interviews.

|  |  |
| --- | --- |
| Do | Do not |
| * Read about the company before the interview * Practice coding problems * Know basic interview etiquette * Stay hydrated * Practice writing on a whiteboard/ Google doc * Have an introduction about yourself * Have good answers for behavioral questions prepared * Polish and study your resume * Prepare a stable and quiet interview environment * Review your syntax knowledge * Brush up on data structures * Practice interviews * Send a thank you note * Think out loud | * Forget to sleep * Panic * Slack * Come unprepared * Be underdressed * Discuss salary/benefits during the interview * Jump straight into coding |

* Use your proposed framework/strategy, whiteboard section distribution, and checklist to tackle the following problems – If you had to refine anything based on your experience tackling the problems, state what you changed and why.

Problem A: Given a collection of **distinct** integers, return all possible permutations.

**Example:**

**Input:** [1,2,3]

**Output:**

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

Problem B: Given an array of integers nums and a positive integer k, find whether it's possible to divide this array into k non-empty subsets whose sums are all equal.

**Example 1:**

**Input:** nums = [4, 3, 2, 3, 5, 2, 1], k = 4

**Output:** True

**Explanation:** It's possible to divide it into 4 subsets (5), (1, 4), (2,3), (2,3) with equal sums.

**Note:**

* 1 <= k <= len(nums) <= 16.
* 0 < nums[i] < 10000.

a. Listen to the interviewer XX

b. Ask for clarifying questions if needed XX

c. Come up with more examples for problem understanding XX

d. Analyze and find the best suitable data structure to use XX

e. Work on the solution XX

f. Ask more questions if needed XX

g. Think out loud my solving processing XX

h. Solve the problem

i. If cannot solve, ask for help XX

i. Walkthrough and explaining the solution

j. Use more test cases for better development XX

k. Optimize if possible

l. Ask for feedback XX