Practice Interview

Objective

The partner assignment aims to provide participants with the opportunity to practice coding in an interview context. You will analyze your partner's Assignment 1. Moreover, code reviews are common practice in a software development team. This assignment should give you a taste of the code review process.

Group Size

Each group should have 2 people. You will be assigned a partner

Parts:

- Part 1: Complete 1 of 3 questions
- Part 2: Review your partner's Assignment 1 submission
- Part 3: Perform code review of your partner's assignment 1 by answering the questions below
- Part 3: Reflect on Assignment 1 and Assignment 2

Part 1:

You will be assigned one of three problems based of your first name. Enter your first name, in all lower case, execute the code below, and that will tell you your assigned problem. Include the output as part of your submission (do not clear the output). The problems are based-off problems from Leetcode.

In [1]:
```
import hashlib

def hash_to_range(input_string: str) -> int:
     hash_object = hashlib.sha256(input_string.encode())
     hash_int = int(hash_object.hexdigest(), 16)
     return (hash_int % 3) + 1
input_string = "cristian"
result = hash_to_range(input_string)
print(result)
```

```
1
```

▶ Question 1

Starter Code for Question 1

```
In [ ]: # Definition for a binary tree node.
        class TreeNode(object):
            def __init__(self, val = 0, left = None, right = None):
                self.val = val
                self.left = left
                self.right = right

        # Example inputs for testing the is_duplicate function
        input1 = [1, 2, 3, 4, 5, 6, 7]
        input2 = [10, 9, 8, 7]
        input3 = [1, 10, 2, 3, 10, 12, 12]
        input4 = [1, 2, 2, 3, 5, 6, 7]

        # Uses the class TreeNode to create a binary tree from a list input.
        def create_binary_tree(input, index = 0) -> TreeNode:
            if not input:
                return None
            root = TreeNode(input[index])
            root.left = create_binary_tree(input, 2*index + 1) if 2*index + 1 < len(
            root.right = create_binary_tree(input, 2*index + 2) if 2*index + 2 < len
            return root

        # Function to check if a binary tree has duplicate values.
        def is_duplicate(root: TreeNode, seen = None) -> int:
          if seen is None:
            seen = set()
          if not root:
            return -1
          elif root.val in seen:
            return root.val
          else:
            seen.add(root.val)
            left = is_duplicate(root.left, seen)
            right = is_duplicate(root.right, seen)
            return left if left != -1 else right

        print(is_duplicate(create_binary_tree(input1)))
        print(is_duplicate(create_binary_tree(input2)))
        print(is_duplicate(create_binary_tree(input3)))
        print(is_duplicate(create_binary_tree(input4)))

        # As Per chatgpt, my implementation of the function is_duplicate is not corr
        # It's not returning the duplicate closest to the root, but rather the first
        # This algorithm could be improved by using a breadth-first search (BFS) app
```

```
-1
-1
10
2
```

▶ Question 2

Starter Code for Question 2

In [ ]:
```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val = 0, left = None, right = None):
#         self.val = val
#         self.left = left
#         self.right = right
def bt_path(root: TreeNode) -> List[List[int]]:
  # TODO
```

▶ Question 3

Starter Code for Question 3

In [ ]:
```
def missing_num(nums: List) -> int:
  # TODO
```

Part 2:

You and your partner must share each other's Assignment 1 submission.

Part 3:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

In [ ]:
```
# Your answer here
```

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

In [ ]:
```
# Your answer here
```

- Copy the solution your partner wrote.

In [ ]:
```
# Your answer here
```

- Explain why their solution works in your own words.

In [ ]:
```
# Your answer here
```

- Explain the problem’s time and space complexity in your own words.

In [ ]:
```
# Your answer here
```

- Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

In [ ]:
```
# Your answer here
```

Part 4:

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

Reflection

In Assignment 1, I was assigned Question 3, moving zeros to the end of a list. The problem was pretty easy to understand, starting with paraphrasing that emphasized the core requirement: segregating zeros while preserving non-zero element order.

I designed test cases that demonstrated different scenarios, including lists with multiple consecutive zeros and alternating patterns. This helped me understand what to do before coding.

My solution documentation was methodical. I explained each step clearly: iterating through the list, building a separate non-zero list, counting zeros, and concatenating results. I did my best analizing the complexities, both time (O(n)) and space (O(n)) considerations.

The most valuable aspect was documenting the alternative approach. I walked through an in-place solution step-by-step with a concrete example, showing how to use an index pointer to achieve O(1) space complexity. All thanks to Copilot and ChatGPT for providing me improvements to my code that I haven't noticed.

Overall, it was a pretty good experience, it honestly reignited my desire to improve my coding skills, bringing myself to a competitive programming level.

Evaluation Criteria

We are looking for the similar points as Assignment 1

- Problem is accurately stated
- New example is correct and easily understandable

- Correctness, time, and space complexity of the coding solution
- Clarity in explaining why the solution works, its time and space complexity
- Quality of critique of your partner's assignment, if necessary

Submission Information

🚨 **Please review our Assignment Submission Guide** 🚨 for detailed instructions on how to format, branch, and submit your work. Following these guidelines is crucial for your submissions to be evaluated correctly.

Submission Parameters:

- Submission Due Date: `HH:MM AM/PM - DD/MM/YYYY`
- The branch name for your repo should be: `assignment-2`
- What to submit for this assignment:
  - This Jupyter Notebook (assignment_2.ipynb) should be populated and should be the only change in your pull request.
- What the pull request link should look like for this assignment: `https://github.com/<your_github_username>/algorithms_and_data_struc`
  - Open a private window in your browser. Copy and paste the link to your pull request into the address bar. Make sure you can see your pull request properly. This helps the technical facilitator and learning support staff review your submission easily.

Checklist:

- [ ] Created a branch with the correct naming convention.
- [ ] Ensured that the repository is public.
- [ ] Reviewed the PR description guidelines and adhered to them.
- [ ] Verify that the link is accessible in a private browser window.

If you encounter any difficulties or have questions, please don't hesitate to reach out to our team via our Slack at `#cohort-6-help`. Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.

```
In [ ]:
```