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

Part 1:

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

Gibran's assignment-1

Fredy's assignment-1

Part 2:

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.

You have a list of numbers. These numbers should include every number from 0 up to the length of the list, but some numbers might be missing. In addition, the list can have repeated numbers.

The goal of this task is to find and return the missing numbers.

- If no numbers are missing, return -1.

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

NEW EXAMPLE: We have the following list of numbers:

- Input List: [4, 0, 1, 3, 8, 6, 10]
- Expected Output: [2, 5, 7, 9]

Gibran's (my classmate's) Example

- Input List: [6, 8, 2, 3, 5, 7, 0, 1, 10]
- Expected Output [4, 9]

Trace/Walkthrough/Explanation of Gibran's Code

First, Gibran's code sorts the list using bubble sort to get the largest number at the end of the list.

```
Initial list: [6, 8, 2, 3, 5, 7, 0, 1, 10]
After sorting: [0, 1, 2, 3, 5, 6, 7, 8, 10]

The largest number is now in the last position: 10
```

Next, Gibran's code creates a complete_list containing all numbers from 0 to the largest number in the sorted list.

```
complete_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Finally, Gibran's code compares the complete_list with the original list to find the missing numbers.

```
Iterate through complete_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Compare each element with the sorted lst = [0, 1, 2, 3, 5, 6, 7, 8, 10]
Missing numbers: 4 and 9 (since they are not in the sorted lst)
```

The function returns the missing_list, which contains [4, 9].

- Copy the solution your partner wrote.

In [1]:
```python
# Gibran's Code

def missing_num(i, n, lst) -> int:

#ordering list with bubble sort to obtain the largest number wich will be in the last position of the list --> lst[-1]
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]

#Making a list to compare to from 0 to the largest number on the ist
        complete_list = []
    for i in range(lst[-1]):
        complete_list.append(i)

#Making the missing_list out of the numbers in the complete_list that are not on lst
    missing_list = []
    for i in complete_list:
        if i not in lst:
            if i not in missing_list:
                missing_list.append(i)

    return missing_list
```

- Explain why their solution works in your own words.

Gibran's code identifies any missing numbers in a given input list by generating a range of expected numbers and then comparing it with the input list.

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

Gibran's algorithm time complexity is O(n). The main operations are bubble sort, list creation, and missing number detection, each with a complexity of O(m). In the worst case, when the input list contains numbers from 0 to n with some missing, the overall time complexity can be approximated to O(n^2).

As for space complexity, the primary contributors are the complete_list and missing_list, each with a complexity of O(m). Therefore, the overall space complexity is O(m), where m is the largest number in the input list.

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

Gibran's code identifies any missing numbers in a given input list by generating a range of expected numbers and then comparing it with the input list.

The method of sorting the list, creating a range of numbers, and then finding the missing numbers is reliable. The code is easy to follow and understand, which is great for maintainability.

Gibran might consider using a more efficient sorting algorithm like lst.sort (Python's built-in sort function) which has a time complexity of $O(n \log n)$ instead of Bubble sort for larger lists. Also, when creating a complete list, adjust the range to include the largest number using range(lst[-1] + 1).

In [2]:
```python
from typing import List

def missing_num(i, n, lst) -> List[int]:
    # Use Python's built-in sort function
    n = len(lst)
    lst.sort()

    # Create a complete list to compare from 0 to the largest number inclusive
    complete_list = []
    for i in range(lst[-1] + 1):  # Fix: include the largest number
        complete_list.append(i)

    # Use a set to find missing numbers
    num_set = set(lst)
    missing_list = []
    for i in complete_list:
        if i not in num_set:
            missing_list.append(i)

    return missing_list

# Test example for the given scenario
lst = [6, 8, 2, 3, 5, 7, 0, 1, 10]
print(missing_num(0, len(lst), lst))  # Output: [4, 9]
```

```
[4, 9]
```

Part 3:

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

Working on Assignment 1 and 2 for the Algorithms and Data Structures course was engaging. The task involved creating a function to find all root-to-leaf paths in a binary tree. I started by understanding the problem and writing a clear plan. The input for the first test scenario was a tree with values `[1, 2, 2, 3, 5, 6, 7]`, and the output was paths like `[1, 2, 3]`. Implementing the function using Depth-First Search (DFS) was straightforward, and it handled different tree structures effectively, as seen in the second scenario with input `[10, 9, 7, 8]`.

Creating two new examples helped solidify my understanding of the problem. We also simplified and optimized the code for Question 2. The simplified version explained how to traverse the tree, while the optimized version demonstrated the importance of efficient code.

Reviewing Gibran's code for Question 3 was insightful. His approach to finding missing numbers in a list was good; However, I identified two areas for improvement: Gibran's code had a a minor bug and could be more efficient.

- The bug could be fixed by correctly including the largest number in the range.
- The code could be optimized by using Python's built-in `sort()` function and sets for faster lookups.

This process emphasized the importance of clarity and correctness in coding.

Throughout these exercises, the significance of data structures became evident. Understanding how to effectively use binary trees, sorting algorithms, and efficient search techniques is crucial. These data structures form the backbone of efficient algorithm design and problem-solving in data science.

Overall, this assignment highlighted the value of testing and code review, ensuring effective and efficient solutions. Understanding the time and space complexity of our algorithms was crucial in making these improvements and ensuring our code was up to standards and scalable.

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: `11:59 PM - 21/07/2024`
- 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/fredylrincon/algorithms_and_data_structures/pull/<pr_id>`
  - 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-3-help` . Our Technical Facilitators and Learning Support staff are here to help you navigate any challenges.