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.

> Reviewing assignment 1 (Question 3) submitted by Jyoti Narang - https://github.com/drop2jyoti/algorithms_and_data_structures/pull/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.

> Validate the existence of missing values in a giving list (it is possible that the list is unordered, also nothing is mentioned about duplicity of the list values)
>
> If the list have any missing values we should create a new list with those missing values, other wise we should return the value -1

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

My example

> input: [10, 5, 2, 3, 2, 8, 3]
>
> output: [0, 1, 4, 6, 7, 9]

Review of my partner's example

> Input : [0,-2,4,3,1] output : [-1,2]
>
> Steps:
>
> > ```
> > Check whether the list contains something or not -> it does.
> > Instantiate a set based from the input list.
> > Find the bounds of the set -> (-2, 4).
> > Create a list with the missing values using the bounds to iterate -> [-1,2]
> > Check whether the missing values list is empty or not:
> >
> >   If the list is not empty return it -> returning [-1,2]
> >   Else return [-1]
> ```

- Copy the solution your partner wrote.

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

def missingNumber(nums: List[int]) -> List[int]:
    if not nums:  # Handle empty list
        return [-1]

    nums_set = set(nums)
    lower_range = min(nums_set)
    upper_range = max(nums_set)

    # Create a list of all numbers in the range that are not in nums_set
    missing_numbers = [number for number in range(lower_range, upper_range + 1) if number not in nums_set]

    # Return missing numbers or [-1] if none are found
    return missing_numbers if missing_numbers else [-1]
```

- Explain why their solution works in your own words.

The solution lacks of certain rules needed to work according to the problem statement as follows:

- The problem states: _You are given a list containing n integers in the range __[0, n]___
  - Then the programmer should have started to analyze the range from **0** rather than finding a lower bound
  - The line *[number for number in range(lower_range, upper_range + 1) if number not in nums_set]* should've been [number for number in **range(upper_range + 1)** if number not in nums_set] if and only if the upper range is a non-negative integer.
- Due to the previous issue if the input is a list with a single value (either size 1 or several repetitions of the same number) it'll fail.
- Also since the problem states that the lower bound should be kept at 0 it will produces wrong inputs when negative values are used in the input list.

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

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

# def missingNumber(nums: List[int]) -> List[int]:
#     if not nums:  # Handle empty list      -> T.C O(1)
#         return [-1]                        -> T.C O(1)

#     nums_set = set(nums)                   -> T.C O(n)
#     lower_range = min(nums_set)            -> T.C O(n)
#     upper_range = max(nums_set)            -> T.C O(n)

#     # Create a list of all numbers in the range that are not in nums_set
#     missing_numbers = [number for number in range(lower_range, upper_range + 1) if number not in nums_set] -> T.C O(n+1) -> -> T.C O(n)

#     # Return missing numbers or [-1] if none are found
#     return missing_numbers if missing_numbers else [-1] -> T.C O(1)
```

The time complexity of my partner's solution is **O(n)** then explained as:

- (O(1) + O(1) + O(n) + O(n) + O(n) + O(n+1) + O(1))
- (O(1 * 3) + O(n * 2) + O(n + 1)) -> (O(1 * 3) + O(n * 3))
- (O(1) + O(n))
- (O(n))

Since we're not saving any data then the space complexity will remain as **O(1)**

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

I'll be reviewing my partner's code as they explained it in assignment 1, complementing with my own recomendations.

- In the first step of the explanation it is said that the code will for check for empty list or list with only one item - return [1] without processing further, but in order to do that it needs to be:
  - `if len(nums) <= 1:  # Handle empty list or list with a single member.`
  - However, doing this could also introduce problems due to the possibility of an input with a single value greater than 0, in this case the method will be incorrect again. i.e. input = [2] produces output = [-1] but the expected output should be [0,1]
- I think the creation of a set is a very smart solution since it's reduces the problem complexity due to the multiple times we have to search in the input list and also removes a possible problem while dealing with duplicated values.
- Finding the minimum value in the input has been proven unnecessary and even prone to create bugs.
- Finding the max value in the input was (in my point of view) the way to go in order to stop the looping mechanism that the developer find fit to solve this problem.
- As mentioned before instead of ranging from min to max value they should loop from 0 to max.
- Doing a search operation in a set has an O(1) complexity in a set of instead of an O(n) in a list so this is where the algorithm becomes efficient!
- The last step of checking whether or not the resultant list was empty was also a very important step in order to secure accuracy in the response.

Also, regarding to code style I think this code is highly maintainable and self explanatory as well as using comments to explain the code is a very good practice.

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 with my partner's solution in the missing numbers problem was a very nice exercise since I could have the opportunity to review the algorithm design of another developer as well as view the problem not as a solver but as a jury (like in a hackaton) which is really nice since it also helps you to develop critical thinking.

This exercise is a clear demonstration on how important it is to fully understand the problem statement requirements and acceptance criteria, since it can create production bugs even thought the code looked good and demonstrated high skills from the programmer's side. In the acceptance criteria it is also important to generate edge cases since they can provide important insights to the programmer and also to the person/team in charge of doing Q.A. to the solution given.

There's a practice in coding called defensive programming which will command the programmer with the task of thinking that the users are a malicious actor and will provide wrong or harmful input, this practice is often a lifesaver for edge cases and will be very helpful to catch early errors in the code lifecycle rather than waiting for a program failure.

The complexity of the solution provided was efficient since it was kept in O(n) which from my perspective is the best solution for the problem statement.

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_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.