

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.

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.

Your answer here

==> I've been assigned Sehroz Khan's problem, which is number 3. His code file:

https://github.com/sehroz/algorithms_and_data_structures/blob/c1159b8ffbf1ddbd4352583

==> This question-3 problem is to find out all the missing numbers from a list of numbers given. Basically, as an input, we've been given a list of numbers between 0 and n. However, some numbers could be missing. Objective is to find these missing numbers and return them as output. If there are no numbers identified as missing, then return -1. Some numbers in the list could be repeating themselves.

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

Your answer here

A) One New Example:

Input: lst: [3, 0, 1, 4, 1, 5]

Range: [0, 5]

Output: [2]

=> The range [0, 5] contains the numbers 0, 1, 2, 3, 4, 5 even though 1 is repeated twice in the list. As can be seen, the number 2 is missing, so the output that should be returned is [2].

B) Sehroz's Example Walkthrough:

Input: nums = [0, 1, 3, 6, 7]

Range: [0, 7] (as the largest number is 7, so we consider all numbers from 0 to 7)

Output: [2, 4, 5]

=> We looked at all numbers within the range of 0 to 7 and found that 2, 4, and 5 are missing from the list, so we can say the output is [2, 4, 5].

- Copy the solution your partner wrote.

```
In [ ]: # Your answer here
from typing import List
def missing_num(nums: List[int]) -> int:

    if not nums:
        return -1

    n = max(nums) ## Largest num in list

    full_range = set(range(n + 1)) ## Creates a set for all ints between 0 & n
    nums_set = set(nums) ## Converts list into set

    missing = list((full_range - nums_set)) ## Removes ints from nums_set from full_range

    return missing

nums = [0, 3, 8, 20]
print(missing_num(nums))
```

[1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

- Explain why their solution works in your own words.

Your answer here

==> The solution creates a function called `missing_num` to run the logic. When a list of integers are passed to this function, it will first try to determine whether the list is empty or not. If it is empty, then the function returns - 1

==> If the list is NOT empty, then the code identifies the largest number in the list

==> Thereafter, it creates a full range starting from 0 to the largest number identified above

==> The code further then converts the input list into a set object

==> Then it removes this converted set object from the entire full range set object to identify missing numbers, which are then converted into a list

==> Finally, that missing number list is returned back as an output.

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

Your answer here

Time complexity = $O(n + k)$, here n is $\max(\text{nums})$ and k is the size of the inputted list

A) Finding the highest number from the list ($\max(\text{nums})$) = $O(n)$

B) Creating a Full Range ($\text{set}(\text{range}(n + 1))$) = $O(n)$

C) Converting the Input List to a Set ($\text{set}(\text{nums})$) = $O(k)$

D) Finding missing numbers from the set ($\text{full_range} - \text{nums_set}$) = $O(n)$

E) Converting the Result to a List = $O(m)$ time, where m is the number of missing numbers

Major factors are creating a full range and finding missing numbers from the set, both are $O(n)$

Space Complexity = $O(n + k)$ (due to storing full range and the input set)

- A) Full Range Set ($\text{set}(\text{range}(n + 1)) = O(n)$)
- B) Input List Set ($\text{set}(\text{nums}) = O(k)$)
- C) Missing Numbers List = $O(m)$ space, where m is the number of missing numbers

Conclusion: The solution could become inefficient when $\text{max}(\text{nums})$ becomes larger or when the input list itself is too big.

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

Your answer here

Some of the code limitations that can be identified:

- A) Problems with large ranges: So, basically, if $\text{max}(\text{nums})$ is large, creating the full range ($\text{range}(n + 1)$) consumes a lot of memory and time. This can be avoided by using $\text{min}(\text{nums})$ and $\text{max}(\text{nums})$ to avoid unnecessary calculations for numbers outside the actual range of consideration.
- B) The function assumes that the list contains valid integers only. If an invalid input (e.g., strings, floats) is given, the function will create an error. We can add checks in the function to ensure all elements in `nums` are integers and handle edge cases like negative numbers or invalid data types.
- C) Problem with Disjoint or Non-Sequential Inputs: If `nums` is highly sparse (e.g., `[1000, 2000, 3000]`), the solution will compute the range `[0, 3000]`, which may be unnecessary. This can be avoided by iterating over the range `[0, $\text{max}(\text{nums})$]` and checking the missing number directly to reduce memory overhead.

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

Your answer here

To tackle the question-1 of finding duplicates in a binary tree of Assignment-1, I first analyzed and understood the requirements clearly: I needed to identify duplicate values and return the one closest to the root. I began by considering tree traversal methods. I found BFS (Breadth-First Search) to be well-suited because it explores nodes level by level, which naturally ensures the closest duplicates to the root are found first.

I then decided to use a set to store values as I traversed the tree. This would help quickly identify duplicates since checking if an element exists in a set is efficient. During traversal, if a value was already in the set, I would return it immediately as the first duplicate. If no duplicates were found, the function would return -1.

Overall, this approach ensured optimal time and space complexity, both being $O(n)$, where n is the number of nodes in the tree.

For Assignment-2, I first tried to understand the challenge of missing number problem. I started reviewing my partner's code and tried to understand it as thoroughly as possible line by line. The comments made by Sehroz in his code was much helpful in understanding the logic and what's the overall thought process while building this function. There were some challenges in understanding the time and space complexities of the code written, but eventually I was able to understand and could find some limitations. The approach worked well for small inputs, it could become inefficient for larger ranges, consuming significant memory and time. The code also requires a way to handle disjoint or sparse inputs. The entire process helped me understand the problem and code from a different perspective and also to review and troubleshoot, which further gives a boost to my coding skills.



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/<pull_request_number>`
 - 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.