Chi Phuong Diep - #U32904573

**ASSIGNMENT 2**

**Question 1: Find Missing Numbers in Array**

***My Prompt:***

I was given an array nums of length n where each number is supposed to be in the range 1 to n. Some numbers appear once, some twice and some are missing. I need to find and return all the numbers that are missing and put them int a new list and maybe call it num1. For the FindMissingNumbers problem, my approach is to use HashSet. I want to loop through each number in the given array and if they do not exist, then add them to the HashSet. Is this approach correct? I was debating if a list is necessary or a HashSet alone is enough?

***Responses Received:***

Using a HashSet to track the numbers in the given array is a good approach for efficiently checking the presence of elements. However, to collect and return the missing numbers, you will need a list to store those missing numbers. The HashSet alone is used for quick lookups, but it does not provide a way to collect the missing numbers directly. Why do you need both?

1. HashSet for Tracking: Use a HashSet to store all the unique numbers from the input array. This allows for O(1) average time complexity for checking if a number is present in the array.
2. List for Collecting Missing Numbers: Use a List to collect the numbers that are missing from the array. This list will be returned as a result.

***Implementation Details:***

I decided to use both HashSet and List as I learned that HashSet alone does not provide a way to collect the missing numbers directly. I went through the logic I had initially to find the missing number. Using a list allowed me to store the numbers in order.

***Adjustment:***

No adjustments to the method or the approach.

**Question 2: Sort Array by Parity**

***My Prompt:***

For SortArrayByParity, my approach is to first set up a list. I want to loop through the nums array and for each loop, I want to check if the number in that array is even or odd. Then, I want to add those numbers to the new list in the order of even integers first and then the odd integers. Could you please check my approach and see if there is a way to optimize the solution to meet the time and space complexity?

***Responses Received:***

Your current approach for SortArrayByParity involves creating two separate lists for even and odd numbers and then combining them. This approach works, but it can be optimized to reduce space complexity by using an in-place algorithm. To optimize both time and space complexity, you can use a two-pointer technique to sort the array in place. This way, you avoid the extra space used by the two lists.

***Implementation Details****:* I realize that my initial approach is not satisfying the requirement in terms of returning the array in place. Therefore, I decided to revise my initial approach and make it satisfy the requirement by using the two-pointer approach. The two-pointer approach is honestly very confusing to me.

***Adjustment:***I made adjustment to the approach by setting up one pointer starts at the beginning and the other at the end of the array. As long as left < right, swap values when the left is odd, and the right is even. The pointers forward or backward are moved whenever their respective values are already in the correct group. This rearranges the array in place with all even numbers first, followed by all odd numbers.

**Question 3: Two Sum**

***My Prompt:***

For the TwoSum problem, my approach is to create a dictionary to store the input as key-value pair. The key will be the numbers in the array and the value will be their indices. I will initialize a remainder variable and create a loop to iterate over each value in the array. I plan to take the target and subtract it by each value we loop over until we find the other value that exists in the same array. Then, I will loop over the dictionary and return the values of those keys. Could you please check my approach and provide feedback?

***Responses Received:*** Your approach to the TwoSum problem is correct and aligns well with the standard solution for this problem. You can:

1. Ensure you handle edge cases such as null input, input arrays with fewer than two elements, and cases where no valid pair is found.
2. Consider how your approach handles duplicate values in the array. The dictionary should store the first occurrence of each number to ensure correct index retrieval.

***Implementation Details:***

I followed my original plan to build the code. I still receive syntax and logic errors along the way, but overall, my implementation was successful. I also have taken the edge cases into consideration along with time and space complexity.

***Adjustment:***

No adjustments to the method or the approach.

**Question 4: Find Maximum Product of Three Numbers**

***My Prompt:***

For the MaximumProduct problem, my approach was to sort the array in ascending order. By doing this, all the largest numbers will be placed at the end of the array. From there, I can get their product of the last three items in the array. What are some edge cases to consider when implementing the maximum product of three numbers algorithm?

***Responses Received****:*

When implementing the maximum product of three numbers algorithm, consider these edge cases:

* Array with fewer than 3 elements
* Array with negative numbers
* Array with zeros
* Array with duplicates
* Array with mixed positive and negative numbers

***Implementation Details:***

My solution did address the edge cases fully by sorting the array and considering both the largest three numbers and the combination of the two smallest numbers with the largest number.

***Adjustment:***

No adjustments to the method or the approach.

**Question 5: Decimal to Binary Conversion**

***My Prompt:***

For the DecimalToBinary question, my approach was to create a list to store the remainders after continuously dividing the remainder by 2. Then, I plan to reverse the list and return the new string by joining the list. What are some edge cases to consider when implementing the solution for decimal to binary conversion?

***Responses Received****:*

When implementing the solution for decimal to binary conversion, consider the following edge cases:

* Array with zero input
* Array with negative input
* Array with non-integer input

***Implementation Details:***

My solution did address the edge cases fully by handling zero input, throwing exceptions for negative inputs, and using a list to manage large numbers.

***Adjustment:***

No adjustments to the method or the approach.

**Question 6: Find Minimum in Rotated Sorted Array**

***My Prompt:***

What are some edge cases to consider when implementing the solution for finding minimum in rotated sort array?

***Responses Received****:*

When implementing the solution for finding the minimum in a rotated sorted array, consider these edge cases:

* Empty array
* Single element array
* Array not rotated
* All elements are the same
* Array with duplicates

***Implementation Details:***

My solution addressed the edge cases by using a binary search approach to find the minimum element in a rotated sorted array. It handles empty arrays, single-element arrays, non-rotated arrays, arrays rotated at different points, and arrays with all identical elements. Although my solution does not explicitly handle duplicates, the binary search approach will still correctly identify the minimum element even if there are duplicates.

***Adjustment:***

No adjustments to the method or the approach.

**Question 7: Palindrome Number**

***My Prompt:***

What are some edge cases to consider when implementing the solution for Palindrome Number?

***Responses Received****:*

When implementing the solution for checking if a number is a palindrome, consider these edge cases:

* Negative numbers
* Single digit numbers
* Numbers with even length
* Numbers with odd length
* Numbers ending with zero
* Large numbers

***Implementation Details:***

My solution addressed the edge cases by converting the number to a string, reversing it, and comparing it with the original string. It handles negative numbers, single-digit numbers, numbers with even and odd lengths, numbers ending with zero, and large numbers.

***Adjustment:***

No adjustments to the method or the approach.

**Question 8: Fibonacci Number**

***My Prompt:***

What are some edge cases to consider when implementing the solution for Fibonacci Number problem?

***Responses Received****:*

When implementing the solution for the Fibonacci Number problem, consider these cases:

* Negative input
* Zero input
* One input
* Large input

***Implementation Details:***

My solution covers the edge cases for negative inputs, zero, and one. However, to handle large inputs efficiently and avoid performance issues, it is recommended to use an iterative approach or memorization.

***Adjustment:***

The adjustment replaces the recursive approach to calculating the Fibonacci number with an iterative approach. This change improves performance and avoids stack overflow issues for large input values by using a loop to iteratively compute the Fibonacci sequence.