**ASSIGNMENT 2**

**COMPUTATIONAL PROBLEM SOLVING**

**GOPI DODDA (U70850116)**

All assignment questions were completed independently, after which additional optimization insights were sought from Copilot to review the code for potential improvements in efficiency, algorithm design, and edge case handling. The prompt given to the AI tool and the corresponding suggestions received for each question have been documented below.

**PROMPT:**

Optimize the code to meet time and space complexity requirements while implementing any recommended data structures or algorithms. Identify any overlooked edge cases, and document both the edge cases found, and the specific changes made, through clear comments in the code.

**QUESTIONS:**

1. **Finding Missing Numbers in Array**

**Initial code:** Used in-place marking by negating values at specific indices to identify missing numbers.

**Response to prompt:** Copilot responded that the code was already optimal and did not require changes. However, it recommended adding a check for edge cases like an empty array.

**Changes made:** No changes were made to the logic. Added a check for the edge case of an empty array input.

A black screen with colorful text

AI-generated content may be incorrect.

**FIGURE 1:** Addition of a check for empty array input

1. **Sort Array by Parity**

**Initial code:** Used two separate lists to store even and odd numbers, then combined them into a result array.

**Response to prompt:** Copilot recommended optimizing the code by reducing space complexity from O(n) to O(1) using an in-place sorting technique with a two-pointer approach.

**Changes made:** Replaced the list-based approach with a two-pointer technique to sort the array in-place, reducing space complexity to O(1). Added a check for the edge case of an empty array input.

A computer screen with text on it

AI-generated content may be incorrect.

**FIGURE 2:** Addition of a check for empty array input

1. **Two Sum (Find Two Numbers that Add to Target)**

**Initial code:** Used nested loops to find two numbers that sum to the target. This resulted an O(n²) time complexity.

A screen shot of a computer code

AI-generated content may be incorrect.

**FIGURE 3:** Initial logic

**Response to prompt:** Copilot suggested replacing the nested loops with a dictionary-based approach to improve time complexity to O(n).

**Changes made:** Replaced the nested loops with a dictionary (hash map) to achieve O(n) time complexity. This approach efficiently finds the complement of each number. Added comments for clarity.

1. **Find Maximum Product of Three Numbers**

**Initial code:** Sorted the array and calculated the product of the three largest numbers and the product of two smallest numbers with the largest number, followed by the comparison of the two products.

A computer screen with numbers and symbols

AI-generated content may be incorrect.

**FIGURE 4:** Initial logic for Question-4

**Response to prompt:** Copilot recommended avoiding sorting and instead using a linear scan to find the required numbers, reducing time complexity to O(n).

**Changes made:** Replaced sorting with a single linear scan to find the three largest and two smallest numbers, reducing time complexity from O(n log n) to O(n).

1. **Decimal to Binary Conversion**

**Initial code:** Used a loop to repeatedly divide the number by 2 and prepend the remainder to a string.

**Response to prompt:** Copilot suggested that the code was already optimal and did not require changes.

**Changes made:** No changes were made. Added comments for the base case “0”.

1. **Find Maximum in Rotated Sorted Array**

**Initial code:** Used binary search to find the minimum element, in O(log n) time complexity.

**Response to prompt:** Copilot recommended no changes as the code was already optimal.

**Changes made:** No changes were made as the implementation is already efficient. Added comments for clarity.

1. **Palindrome Number**

**Initial code:** Reversed the number using a while loop and compared it with the original number.

**Response to prompt:** Copilot suggested that the code was already optimal and did not require changes.

**Changes made:** Made a quick fix by using a compound assignment. This didn’t affect anything as the logic was already efficient with O(log n) time complexity. Added comments for edge cases.

A screenshot of a computer

AI-generated content may be incorrect. A black screen with white text

AI-generated content may be incorrect.

**FIGURE 5 & FIGURE 6:** Using compound assignment with Quick Fix prompt (Before & After)

1. **Fibonacci Number**

**Initial code:** Used an iterative approach to calculate the Fibonacci number. The implementation already achieved optimal O(n) time and O(1) space complexity.

**Response to prompt:** Copilot recommended no changes as the code was already optimal.

**Changes made:** No changes were made. Added comments for the base cases.