**Assignment 2 : Computational Problem Solving**

**Name: Aayushi Nirajkumar Desai  
U Number: U25625953  
Subject: Application Development for Analytics (ISM6225)**

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

**Prompt Used:**

* find missing numbers in array using in-place negation
* avoid IndexOutOfRangeException in array access c#

**Response Received:**

The suggested resposne was to use a two-pass algorithm:

1. In the first pass, it marks visited numbers by setting the value at index = abs(nums[i]) - 1 to negative.
2. In the second pass, indices with positive values represent missing numbers.

**Implementation Details:**

I implemented this approach and added a guard (if val >= 1 && val <= nums.Length) to prevent out-of-range errors. The logic was tested using normal and edge case inputs.

**Adjustments:**

While testing, I introduced the input {4, 4, 6, 7, 7, 8, 9} and encountered a System.IndexOutOfRangeException.

I recognized that values like 8 and 9 exceeded the array bounds (size = 7). This issue wasn’t handled in Copilot’s initial suggestion.

**Test Cases:**

{1,2,3,4} → Output: []

{4, 4, 6, 7, 7, 8, 9} → Output: [1, 2, 3, 5]

Duplicates and out-of-range values are handled gracefully using a guard clause.

**Question 2: Sort Array By Parity**

**Prompt Used:**

* sort array by parity using in c#
* maintain original order while separating even and odd numbers

**Response Received:**  
Copilot initially suggested using a two-pointer approach, which places even numbers before odd numbers by swapping values from both ends of the array. This method worked for grouping even and odd numbers but did not preserve their original order.

**Implementation Details:**  
I implemented the two-pointer method first, which grouped even and odd numbers correctly.

While reviewing the expected output provided, I realized the example preserved the relative order of the even and odd numbers, meaning a stable sort was required.

To address this, I rewrote the logic using two separate lists to collect even and odd numbers and then merged them.

**Adjustments:**  
The original Copilot-generated solution gave me an answer like [4, 2, 3, 1], but the output was [2, 4, 3, 1].

This difference helped me understand that the problem wasn’t just about grouping but also about maintaining input order.

**Test Cases:**

{2, 4, 6, 8} → Output [2, 4, 6, 8]

{1, 3, 5, 7} → Output [1, 3, 5, 7]

**Question 3 – Two Sum**

**Prompt Used:**

* two sum algorithm using dictionary in c#
* efficient lookup in c# array

**Response Received:**  
The suggested response was to use a dictionary to map each number to its index. For each number, it checked if the complement (target - current) was already seen.

**Implementation Details:**  
I used this efficient O(n) solution. It handles multiple inputs and avoids duplicates.

**Adjustments:**  
I added a check to avoid storing duplicate keys.

**Test Cases:**

{3, 2, 4} target: 6 → Output: {1, 2}

{3, 3} target: 6 → Output: {0, 1}

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

**Prompt Used:**  
maximum product of three numbers c# array

**Response Received:**  
Copilot suggested sorting the array and returning the maximum of:

product of last 3 numbers

product of 2 smallest + 1 largest value

**Implementation Details:**  
I applied the exact logic and added comments for clarity. It accounts for both negative and positive combinations. Used Array.Sort for simplicity and readability.

**Adjustments:**  
I added variable names product1 and product2 for clarity.

**Edge Cases:**

{-10, -10, 5, 2} → Output: 500

{-1, -2, -3} → Output: -6

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

**Prompt Used:**  
find minimum in rotated sorted array using binary search in c#

**Response Received:**  
Copilot suggested using a binary search algorithm:

* If nums[mid] > nums[right], move left pointer to mid + 1
* Else, move right pointer to mid

**Implementation Details:**  
I applied the exact logic from Copilot and added inline comments for clarity. The method stops when left == right, which is the index of the minimum.

**Adjustments:**  
None.

Edge Cases:

{3, 2, 3, 4, 10} → Output: 2

{2, 3, 4, 5, 1} → Output: 1

**Question 7 – Palindrome Number**

**Prompt Used:**  
check if number is palindrome without converting to string in c#

**Response Received:**  
Copilot recommended reversing the number mathematically (without converting to string) and comparing it to the original.

**Implementation Details:**  
I followed this logic, used a loop to reverse the number, and checked if the reversed value matches the original. I added a check for negative numbers at the beginning.

**Adjustments:**  
None.