**Question 1:**

**Prompts Used:**

"How to find all numbers missing from an array where elements are in range 1 to n and some elements may be duplicated in C#?"

**Responses Received:**

1. Iterate through the array and mark the presence of each number by negating the value at the index corresponding to the number.
2. Iterate through the array again and collect the indices of the positive values, which represent the missing numbers.

**Implementation Details:**

* Loop 1: For each number, use Math.Abs(nums[i]) - 1 to find the index, and negate the value at that index to mark presence.
* Loop 2: Find indices with positive values (meaning that index + 1 is missing).

Initially, the method returned new List<int>() as a placeholder to satisfy the return type requirement. This was replaced after implementing the logic suggested by Copilot, which constructs the correct result based on in-place negation marking.

Edge Case 1: **Array with all numbers present**

**IP:** int[] nums = { 1, 2, 3, 4, 5 };

**OP:** If the input array contains all numbers from 1 to n, there should be no missing numbers, and the result should be an empty list [].

**Question 2:**

Didn’t use copilot.

**Edge cases:**

1. Array with Only Even Numbers:  
   IP : {2, 4, 6, 8}  
   OP: {2, 4, 6, 8}
2. Array with Only Odd Numbers:

IP: {1, 3, 5, 7}

OP: {1, 3, 5, 7}

**Question 3:**

Didn’t use copilot.

For this I am using bruteforce approach here to keep things simple but has O(n^2) time complexity.

**Edge cases:**

1. Array with only one element:  
   Input: {3}, Target: 6  
   There are no pairs, so the output should be an empty array ([])

**Question 4:**

**Copilot Prompt:**

1. in built function for array sort

Answer:

Array.Sort

1. I want to compare the product of the three largest numbers with the product of the two smallest numbers and the largest number, and then pick the larger one

Answer:

To compare the product of the three largest numbers with the product of the two smallest numbers and the largest number, and then pick the larger one, you can use the **Math.Max** method as shown in the MaximumProduct method.

So, I used this to first sort and then compare the results of best possible two scenarios(when negatives are present)

**Edge Cases:**

**Array with All Negative Numbers**:

IP: int[] nums4 = { -10, -10, -1, -3, -76 };

OP: -30

**Array with a Mix of Negative and Positive Numbers**:

IP: { -10, -10, 5, 2 }

OP: 500 (since -10 \* -10 \* 5 = 500)

**Question 5:**

Copilot is not used

**Edgecase:**

Input One

Output: "1"

**Question 6:**

**Copilot Prompt:**

Can you help me write a function to find the minimum element in a rotated sorted array? The array might have duplicate values.

**Answer:**

Certainly! To find the minimum element in a rotated sorted array that might have duplicate values, you can use a modified binary search algorithm. The key is to handle the case where the middle element is equal to the rightmost element, which requires adjusting the search range.

A screenshot of a computer program

AI-generated content may be incorrect.

By implementing binary search and strategically adjusting the right pointer to bypass duplicate values, the solution effectively avoids search stagnation and accurately finds the minimum element in all possible rotated sorted array configurations.

**Edge Cases:**

Array with Only One Element.

IP: [1]

OP: 1

Array with All Elements the Same

IP: [2, 2, 2, 2]

OP:2

Array with Duplicates:

IP: [2, 2, 2, 0, 1, 2]

OP: 0

**Question 7**

Copilot not used

**EdgeCases**

Negative Numbers

IP: -2

OP: False

Single-Digit Numbers

IP: 7

OP: True

**Question 8**

Copilot not used

**EdgeCases**

Fibonacci of 0

IP: 0

OP: 0

Fibonacci of 1

IP: 1

OP: 1