**Starting Prompt:**

* Help me solving the coding problem in C#. Don’t change the method declarations, just write your logic inside as it is (same parameters, same return types).
* Add inline comments wherever it is necessary, so that it will be easy to understand
* Able to handle errors through try and catch blocks
* Your solution should handle all kinds of test cases like empty inputs, invalid values, negatives, and duplicates.
* Make sure the algorithm you provide is efficient and covers the time and space complexity constraints.
* Implement right data structures or techniques wherever needed like hash maps, sorting, binary search according to the problem.

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

* **Prompt Used:**   
  Find missing numbers in an unsorted integer array nums of size n containing numbers from 1 to n, find all the numbers that are missing from the array.
* **Response Received:**  
  Copilot suggested using a boolean array to keep track of which numbers from 1 to n are in the input. For each number, if it’s valid, we mark it as found. Then we just go through the list and pick out the numbers that weren’t marked those are the missing ones.
* **Implementation Details:**  
  Implemented a boolean array of size n + 1 to track valid numbers. Implemented loop through the array and mark true for each valid number (1 to n). At the end, collected all indices that are still false.
* **Edge Cases Covered:**  
  Empty array, duplicates, negatives, values greater than n, and all values present.

**Question 2: Sort Array by Parity**

* **Prompt Used:**  
  Solve Sort Array by Parity, move all even integers to the beginning of the array followed by all odd integers.
* **Response Received:**  
  Copilot recommended a two-pointer solution and confirmed that parity (even/odd) logic applies to both positive and negative numbers, including zero.
* **Implementation Details:**  
  Used two-pointer approach. First initialized one pointer at start and one at end. Swapped elements based on parity, moved left pointer for even, right pointer for odd.
* **Edge Cases Covered:**  
  Negative numbers, 0, all even or all odd, mixed inputs, and already sorted parity.

**Question 3: Two Sum**

* **Prompt Used:**  
  Given an array of integers nums and an integer target, return the indices of the two numbers such that they add up to the target.
* **Response Received:**  
  Copilot suggested using a dictionary for complement checks and allowed duplicates as long as indices differ.
* **Implementation Details:**  
  Iterated through the array, checking if the complement of each number exists in the dictionary. If found, returned both indices. Otherwise, stored the number and its index.
* **Edge Cases Covered:**  
  Duplicates (like [3,3]), empty array, single element, target not reachable, negatives.

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

* **Prompt Used:**  
  Find maximum product of three numbers in integer array
* **Response Received:** Copilot suggested sorting the array and checking two possible products: the product of the three biggest numbers, and the product of the two smallest (possibly negative) numbers with the largest one. The higher of the two is our answer.
* **Implementation Details:**  
  Sorted the array and calculated both arr[n-1] \* arr[n-2] \* arr[n-3] and arr[0] \* arr[1] \* arr[n-1]. Returned the greater of the two values.
* **Edge Cases Covered:**  
  Less than 3 elements, zeros, all negatives, mixed signs.

**Question 5: Decimal to Binary**

* **Prompt Used:**  
  how to convert decimal to binary representation
* **Response Received:**  
  Copilot recommended dividing the number by 2 repeatedly and storing the remainder each time. It build the binary string by adding each remainder to the front. It also mentioned that if the input is 0, we just return "0".
* **Implementation Details:**  
  Used a loop to divide the number by 2, storing each remainder at the front of a string. Special case: returned "0" if the input was 0.
* **Edge Cases Covered:**  
  Zero, one, large numbers.

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

* **Prompt Used:**Given a sorted array that has been rotated, find the minimum element.
* **Response Received:**  
  Copilot explained that we can use binary search by comparing the middle element with the rightmost one to decide which half of the array contains the minimum, and continue this process until we narrow it down to the smallest element.
* **Implementation Details:**  
  Used binary search: compared mid and right values to adjust the search window. Continued until left and right pointers converged on the minimum.
* **Edge Cases Covered:**  
  Array with one element, not rotated, rotated at different positions, small arrays.

**Question 7: Palindrome Number**

* **Prompt Used:**  
  c# Program to check if number is palindrome with overflow and negative cases
* **Response Received:**  
  Copilot suggested reversing the number by taking digits from the end and building a new number. Before reversing, we check for negatives. At the end, we compare the reversed number to the original to see if it’s a palindrome.
* **Implementation Details:**  
  Checked if the number is negative. Reversed the digits by extracting the last digit and building the reverse using multiplication and addition.
* **Edge Cases Covered:**  
  Negative numbers, 0, overflow risk, single-digit numbers.

**Question 8: Fibonacci Number**

* **Prompt Used:**  
  Implement fibonacci in c# with input constraint between 0 and 30
* **Response Received:**  
  Copilot suggested using a loop instead of recursion to calculate the Fibonacci number. We start with 0 and 1, and add them up until we reach the nth number. It also recommended checking if the input is within the valid range (0 to 30).
* **Implementation Details:**  
  Used a loop to build the Fibonacci series up to the given number. Returned 0 for n=0 and 1 for n=1. Used iteration for efficiency.
* **Edge Cases Covered:**  
  n = -1, n = 31, n = 0, n = 1, upper bound check.

**Adjustments:**

In all the problems, I incorporated try-catch blocks to catch errors in a safe manner and made sure code is processed for edge cases like empty inputs, negatives, and inputs outside of expected ranges. I tried with invalid inputs, accepted duplicates when necessary, and used appropriate data structures like dictionaries and boolean arrays so that everything remained efficient. For overflow scenarios with potential limit or special scenarios such as reversing a number or calculating the Fibonacci, I included a safety check. I also added comments where ever necessary