AI Assistance Documentation

# Q1: Find Missing Numbers in Array

**Prompt Used:**  
Copilot: Generate function to find missing numbers from array where numbers range from 1 to n

**Response Received:**  
Copilot suggested using an in-place marking technique where array values are used to indicate visited indices by negating them. This allows detection of missing indices in a second pass. The suggestion included a concise C# function that modifies the array in-place and returns the missing numbers.

**Implementation Details:**  
The AI-generated solution was directly applied with minor formatting adjustments. The logic involved two passes:

1. Mark the presence of numbers by negating the element at the index corresponding to each value.
2. In the second pass, indices with positive values indicate missing numbers.

**Adjustments:**  
No major modifications were needed.

# Q2: Sort Array by Parity

**Prompt Used:**  
Copilot: Write a function to sort an array by moving even numbers to the front and odd numbers to the back

**Response Received:**  
Copilot suggested a compact approach using a List<int> to separate even and odd numbers. It recommended inserting even numbers at the beginning of the list using Insert(0, n) and appending odd numbers to the end using Add(n). This results in even numbers coming first in the final array.

**Implementation Details:**  
Used Copilot’s code mostly unchanged. It used a foreach loop to iterate through the input array, checked for parity using n % 2, and modified the result list accordingly. Finally, the list was converted to an array and returned.

**Adjustments:**  
No changes were made to the core logic. Included edge case for all even or all odd inputs.

# Q3: Two Sum

**Prompt Used:**  
Copilot: Find two indices in an array whose values sum up to a given target

**Response Received:**  
Copilot suggested using a single-pass hash map (dictionary) approach where for each element in the array, we check if the complement (i.e., target - current number) exists in the map. If it does, the indices are returned. Otherwise, the current number and its index are added to the map. The solution was concise and optimized for time complexity..

**Implementation Details:**  
The dictionary-based solution was implemented exactly as suggested. A loop iterates through the array while maintaining a dictionary of seen values and their indices. When a matching complement is found, the function returns the corresponding indices.

**Adjustments:**  
The return statement was simplified to new[] { map[diff], i } for brevity. Also, Array.Empty<int>() was used instead of new int[0] for a modern C# approach.

# Q4: Maximum Product of Three Numbers

**Prompt Used:**  
Copilot: Find the maximum product of any three numbers in an array

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

1. The product of the three largest numbers.
2. The product of the two smallest numbers (possibly negative) and the largest number.  
   This ensures both positive and negative number cases are correctly handled.

**Implementation Details:**  
The implementation directly applied the AI-suggested logic using Array.Sort(nums) followed by a single return statement using Math.Max() to compute both product options. The C# index-from-end (^) operator was used for clarity and brevity

**Adjustments:**  
No changes were made to the core logic.

# Q5: Decimal to Binary Conversion

**Prompt Used:**  
Copilot: Convert a given decimal number to binary string representation

**Response Received:**  
Copilot recommended using Convert.ToString(decimalNumber, 2) as a concise and built-in way to convert a decimal number to its binary string representation. It explained that the second argument 2 specifies the base (binary)

**Implementation Details:**  
This approach leverages .NET's Convert class to handle the conversion efficiently without any manual logic.

**Adjustments:**  
No modifications were made since the built-in method was already optimized.

# Q6: Find Minimum in Rotated Sorted Array

**Prompt Used:**  
Copilot: Find the minimum element in a rotated sorted array

**Response Received:**  
The Copilot recommended using a binary search approach to efficiently locate the minimum element in a rotated sorted array. The key insight was comparing the middle element with the rightmost element to decide whether to search in the left or right half. The suggestion was optimized for O(log n) time complexity and used minimal code lines.

**Implementation Details:**  
The suggested binary search solution was directly implemented . The midpoint was recalculated in each iteration, and comparisons were made between nums[mid] and nums[right] to adjust the search boundaries accordingly. The loop ends when left points to the minimum value, which is then returned.

**Adjustments:**  
No changes were made to the logic itself.

# Q7: Palindrome Number

**Prompt Used:**  
Copilot: Check if a given integer is a palindrome number

**Response Received:**  
The Copilot suggested checking for palindromes by converting the number to a string, reversing it, and comparing it to the original string. It also noted that negative numbers are not palindromes by definition and should be filtered out.

**Implementation Details:**  
Used Copilot’s logic. Wrapped with an initial check to return false for negative integers. The ToString(), Reverse(), and new string() functions were used to achieve a compact and efficient string-based reversal for comparison.

**Adjustments:**  
None.

# Q8: Fibonacci Number

**Prompt Used:**  
Copilot: Write a function to return the nth Fibonacci number

**Response Received:**  
The Copilot suggested using an iterative approach to calculate the nth Fibonacci number. The logic initializes the first two Fibonacci numbers and iteratively computes the rest up to n, which avoids the performance issues of recursion.

**Implementation Details:**  
Two variables (a and b) were used to hold the last two Fibonacci values. A loop was used to update the values and compute the final result, which is returned after n iterations. This approach is efficient with O(n) time and O(1) space complexity.

**Adjustments:**  
No changes were made to the core logic.