**AI ASSISTANCE DOCUMENTATION**

**Prompts Used:**

1. "How to efficiently find missing numbers in an array in C#?"
2. "Best way to sort an array by parity in C#?"
3. "How to implement Two Sum in C# using a dictionary?"
4. "How to find the maximum product of three numbers in an array in C#?"
5. "Convert decimal number to binary string in C#?"
6. "Find the minimum value in a rotated sorted array in C#?"
7. "Check if a number is a palindrome in C# without converting it to a string?"
8. "Efficient Fibonacci sequence implementation in C#?"

**Responses Received:**

1. **Find Missing Numbers:** Suggested using a HashSet to track existing numbers and iterate over the range to find missing ones.
2. **Sort Array by Parity:** Recommended using OrderBy(x => x % 2).ToArray() for simplicity and efficiency.
3. **Two Sum:** Proposed using a dictionary to store seen numbers and find the complement efficiently in one pass.
4. **Maximum Product:** Suggested sorting the array and considering both the top three and the lowest two with the highest positive number.
5. **Decimal to Binary:** Advised using Convert.ToString(decimalNumber, 2).
6. **Find Minimum in Rotated Array:** Recommended a binary search approach to find the minimum efficiently.
7. **Palindrome Check:** Suggested reversing the number mathematically rather than converting to a string.
8. **Fibonacci Number:** Proposed using an iterative approach to optimize space complexity over recursion.

**Implementation Details:**

* Implemented the HashSet approach for missing numbers to ensure O(n) complexity.
* Used OrderBy(x => x % 2) for concise and readable parity sorting.
* Utilized a dictionary for Two Sum, ensuring an O(n) time complexity solution.
* Considered edge cases in the Maximum Product function by handling negatives and large numbers.
* Used Convert.ToString(decimalNumber, 2) for a simple and effective decimal-to-binary conversion.
* Applied binary search for FindMin, optimizing the solution to O(log n).
* Reversed integers mathematically for palindrome check to avoid extra space usage.
* Used an iterative method for Fibonacci to ensure O(n) time and O(1) space complexity.

**Adjustments:**

* Added try-catch blocks to handle unexpected input errors and edge cases.
* Ensured solutions handled edge cases such as empty arrays, single-element arrays, and large values.
* Used throw; in the catch blocks to maintain proper exception propagation.
* Optimized sorting and searching logic to meet time complexity constraints.

This document records AI-generated suggestions, their application, and necessary modifications made to align with constraints and requirements.