**Prompts Given:**

1. **Find Missing Numbers in Array**  
   Prompt: Write a function to find missing numbers in an array where numbers should range from 1 to n.
2. **Sort Array by Parity**  
   Prompt: Write a function to sort an array such that all even numbers come before all odd numbers while maintaining relative order.
3. **Two Sum**  
   Prompt: Write a function that returns the indices of two numbers in an array that add up to a target value.
4. **Maximum Product of Three Numbers**  
   Prompt: Write a function to find the maximum product of any three numbers in an array.
5. **Decimal to Binary Conversion**  
   Prompt: Write a function that converts a decimal number to its binary representation.
6. **Find Minimum in Rotated Sorted Array**  
   Prompt: Write a function to find the minimum value in a rotated sorted array.
7. **Palindrome Number**  
   Prompt: Write a function to check if a given number is a palindrome.
8. **Fibonacci Number**  
   Prompt: Write a function to calculate the nth Fibonacci number.

**Responses Received:**

**Requested**: Explain the error and how to fix my code?

**Response Received**: You got this error because the function **FindMissingNumbers** was not returning the correct list of missing numbers. This might be happening because the placeholder return statement was not replaced with the actual list of missing numbers.

Here's how I fixed the code: I replaced the placeholder return statement with the correct return statement that returns the list of missing numbers. This ensures that the function correctly identifies and returns the missing numbers in the array.

**Requested**: How can I optimize the **TwoSum** function for better performance?

**Response Received**: Here's how I fixed the code: I added a check to avoid overwriting existing indices in the dictionary. This ensures that the first occurrence of each number is stored, which is important for returning the correct pair of indices. This optimization leverages the dictionary's average O(1) time complexity for lookups and insertions, making the function more efficient.

**Requested**: How can I handle edge cases when calculating the maximum product of three numbers in an array?

**Response Received**: To handle edge cases when calculating the maximum product of three numbers in an array, you need to consider the following scenarios:

1. The array has fewer than three elements.
2. The array contains both positive and negative numbers.
3. The array contains zeros.

**Requested**: What is the time complexity of the matrix exponentiation method for calculating Fibonacci numbers?

**Response**: The matrix exponentiation method leverages the properties of matrix multiplication to compute Fibonacci numbers efficiently.

* The time complexity of this method is O(log n) due to the exponentiation by squaring technique used in matrix multiplication.
* This method is particularly useful for calculating large Fibonacci numbers efficiently.

**Implementation Details:**

Prompt: Explain the error and how to fix my code.

**Implementation**: I applied the suggestion as it was. I replaced the placeholder with a correct return statement that returns the list of missing numbers, and this fixed the issue.

Prompt: How can I optimize the TwoSum function for better performance?

**Implementation**: I applied this optimization without modification. The dictionary now ensures that the first occurrence of each number is stored, improving the overall time complexity of the solution to O(n).

Prompt: How can I handle edge cases when calculating the maximum product of three numbers in an array?

**Implementation**: I modified the code to include a check for arrays with fewer than three elements, ensuring the function returns 0 if there aren’t enough elements to calculate the product. I also considered cases with negative numbers and zeros.

Prompt: What is the time complexity of the matrix exponentiation method for calculating Fibonacci numbers?

**Implementation**: This was just an informative response. No modifications were needed, but I understood that the matrix exponentiation method would allow me to calculate Fibonacci numbers in logarithmic time complexity.

**Adjustments:**

**Find Missing Numbers in Array**  
**Adjustment:** Replaced the placeholder return statement with the correct one to return the missing numbers.

**TwoSum Function Optimization**  
**Adjustment:** Added a check to avoid overwriting indices in the dictionary to ensure the correct pair of indices is returned

**Maximum Product of Three Numbers**  
**Adjustment:** Added checks for arrays with fewer than three elements and considered both negative numbers and zeros.

**Time Complexity of Matrix Exponentiation for Fibonacci Numbers**  
**Adjustment:** No modification to the code, confirmed that matrix exponentiation has O(log n) time complexity for Fibonacci numbers.