**AI Assistance Documentation for ISM6225 Assignment 2**

**Find Missing Numbers in Array:**

* **Prompt Used:** “How can I find missing numbers in an array from 1 to n where there are duplicate values?”
* **Copilot’s Input:** Copilot suggested an approach where the numbers are marked as negative based on their indices, so we can identify the missing numbers by looking for unmarked indices.
* **My Approach:** While I came across a similar idea from Copilot, I decided to implement it on my own after thinking about how to handle duplicate values. I marked the numbers using their indices and adjusted the approach to account for possible duplicates, ensuring that the solution would work for all edge cases.
* **Adjustments:** I didn’t directly use Copilot’s code. Instead, I adapted the index-based method, adding logic to handle duplicates and ensuring that I didn’t miss any numbers.

**Function 2: Sort Array by Parity**

* **Prompt Used:** “How do I sort an array such that all even numbers come before odd numbers using a two-pointer technique?”
* **Copilot’s Input:** Copilot suggested the two-pointer technique, which seemed like a valid approach. The idea is to swap elements when the left pointer points to an odd number and the right pointer points to an even number.
* **My Approach:** I took the idea from Copilot’s suggestion and made some modifications to ensure that the code works for edge cases, like when the array has only one element or when all elements are either even or odd. I also made sure the code was efficient for such cases, but the core logic came from my own understanding.
* **Adjustments:** I added additional checks for arrays with one element or arrays where all numbers are of the same parity, ensuring that the solution is robust for all scenarios.

**Function 3: Two Sum (Find Two Numbers that Add to Target):**

* **Prompt Used**: Can you write code to solve the Two Sum problem using a hash map to find the indices of two numbers that sum up to a target value?”
* **Copilot’s Input:** Copilot suggested using a dictionary to store the numbers and their indices while iterating over the array. This is an optimal approach, so I was familiar with it.
* **My Approach:** Even though Copilot provided a standard solution for the Two Sum problem, I preferred to code it myself based on my understanding. I used the hash map (dictionary) approach and handled the edge case where the same element cannot be used twice. The code is simple and efficient with O(n) time complexity, and I didn’t need to rely on Copilot’s exact solution.
* **Adjustments:**I didn't modify Copilot's suggestion directly since I wrote the logic on my own. The main adjustment was ensuring that no element is reused, which is essential for correctness.

**Function 4: Maximum Product of Three Numbers**

* **Prompt** **Used :** “How can I find the maximum product of three numbers in an array, considering that negative numbers may also be involved?”
* **Copilot’s Input:**Copilot provided the idea of sorting the array and then comparing the product of the three largest numbers and the product of the two smallest numbers with the largest number (to account for negative values).
* **My Approach:**This approach was very similar to what I had in mind. I implemented the solution by sorting the array and comparing the two possible maximum products. However, I coded this entirely on my own, with Copilot's suggestion simply confirming that I was on the right track.
* **Adjustments:**No major changes were needed. I followed my understanding of the problem and used the sorted array approach with some minor refinements.

**Function 5: Decimal to Binary Conversion**

* **Prompt Used**: “How do I convert a decimal number to binary?”
* **Response Received**: Copilot suggested using the standard method of dividing the decimal number by 2 and appending the remainder until the number becomes 0.
* **Implementation Details**: I implemented the solution as suggested by Copilot. I ensured that the number was divided by 2 iteratively, and the remainders were stored in a string to build the binary representation.
* **Adjustments**: I added an edge case for the **decimal number 0**, returning the string "0" since the binary representation of 0 is "0".

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

* **Prompt Used**: “How do I find the minimum element in a rotated sorted array using binary search?”
* **Response Received**: Copilot suggested using binary search to locate the pivot point where the rotation occurs. The minimum element would be found at the pivot point.
* **Implementation Details**: I implemented the binary search as suggested by Copilot. This method efficiently finds the minimum element even in rotated arrays. I ensured it works for sorted arrays and arrays with a single element.
* **Adjustments**: I added handling for cases where the array is already sorted and cases with a single element, as these were not explicitly mentioned by Copilot.

**Function 7: Palindrome Number**

* **Prompt Used**: “How can I check if a number is a palindrome?”
* **Response Received**: Copilot suggested reversing the digits of the number and comparing the reversed number with the original number.
* **Implementation Details**: I implemented the solution as suggested by Copilot. I reversed the digits of the number and compared the original number to the reversed one.
* **Adjustments**: I added an edge case for **negative numbers**, as they cannot be palindromes.

**Function 8: Fibonacci Number**

* **Prompt Used**: “How do I compute the nth Fibonacci number iteratively?”
* **Response Received**: Copilot suggested using an iterative approach where two variables are used to keep track of the previous two Fibonacci numbers.
* **Implementation Details**: I implemented the iterative solution as suggested by Copilot. The solution efficiently computes Fibonacci numbers in **O(n)** time complexity.
* **Adjustments**: No adjustments were made. Copilot’s solution was fully applicable.