GitHub Copilot Interaction Documentation – Assignment 2

# Question 1: Find Missing Numbers in Array

## Prompt Used

Given an array of numbers that should contain all values from 1 to n, identify which numbers are missing, even if there are duplicates in the array.

## Response Copilot Gave "Sure! I created a function that loops from 1 to n and checks which numbers aren’t in the array. To handle duplicates efficiently, I stored the input in a HashSet. That way, the lookup is fast, and we skip repeated values."

public static IList<int> FindMissingNumbers(int[] nums)  
{  
 try  
 {  
 int n = nums.Length;  
 HashSet<int> numSet = new HashSet<int>(nums);  
 List<int> missingNumbers = new List<int>();  
 for (int i = 1; i <= n; i++)  
 {  
 if (!numSet.Contains(i))  
 {  
 missingNumbers.Add(i);  
 }  
 }  
 return missingNumbers;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: [4, 3, 2, 7, 8, 2, 3, 1] → Output: [5, 6]

- Input: [1, 1] → Output: [2]

- Handles empty and duplicate inputs

# Question 2: Sort Array by Parity

## Prompt Used

Reorder the given array such that all even numbers come before any odd numbers. The function should handle zeros, single-element arrays, and preserve stability where possible.

## Response Copilot Gave

"Of course! I used a two-pointer approach where one pointer fills in even numbers from the start, and the other fills in odd numbers from the end. This keeps the implementation simple and ensures that even numbers appear first regardless of the original order."

public static int[] SortArrayByParity(int[] nums)  
{  
 try  
 {  
 int[] result = new int[nums.Length];  
 int evenIndex = 0;  
 int oddIndex = nums.Length - 1;  
 foreach (int num in nums)  
 {  
 if (num % 2 == 0)  
 {  
 result[evenIndex++] = num;  
 }  
 else  
 {  
 result[oddIndex--] = num;  
 }  
 }  
 return result;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: [3, 1, 2, 4] → Output: [2, 4, 3, 1] or similar

- Input: [0, 1, 2] → Output: [0, 2, 1]

- Reverses odd order when placed at the end

# Question 3: Two Sum

## Prompt Used

Implement logic that, given an integer array and a target sum, returns the indices of the two elements whose sum equals the target. Assume that each input has exactly one solution.

## Response Copilot Gave

"Got it! I used a dictionary to store the numbers and their indices. As I loop through the array, I check if the complement (target - current value) already exists in the dictionary. If it does, I return the indices. It’s efficient and works in O(n) time."

public static int[] TwoSum(int[] nums, int target)  
{  
 try  
 {  
 Dictionary<int, int> numDict = new Dictionary<int, int>();  
 for (int i = 0; i < nums.Length; i++)  
 {  
 int complement = target - nums[i];  
 if (numDict.ContainsKey(complement))  
 {  
 return new int[] { numDict[complement], i };  
 }  
 numDict[nums[i]] = i;  
 }  
 throw new ArgumentException("No two sum solution");  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: [2, 7, 11, 15], Target: 9 → Output: [0, 1]

- Input: [3, 2, 4], Target: 6 → Output: [1, 2]

- Efficient with O(n) complexity

# Question 4: Find Maximum Product of Three Numbers

## Prompt Used

Given an array of integers (which may include negative values), determine the maximum product that can be achieved by multiplying any three of them.

## Response Copilot Gave

"Here’s what I did: I sorted the array and calculated two possible products — the product of the three largest numbers, and the product of the two smallest (most negative) numbers and the largest one. The maximum of these two gives the correct answer."

public static int MaximumProduct(int[] nums)  
{  
 try  
 {  
 if (nums.Length < 3)  
 {  
 throw new ArgumentException("Array must contain at least three elements.");  
 }  
 Array.Sort(nums);  
 int n = nums.Length;  
 int maxProduct = Math.Max(nums[n - 1] \* nums[n - 2] \* nums[n - 3], nums[0] \* nums[1] \* nums[n - 1]);  
 return maxProduct;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: [1, 2, 3, 4] → Output: 24

- Handles negative values for maximum product correctly

# Question 5: Decimal to Binary Conversion

## Prompt Used

Convert a non-negative decimal integer to its binary string representation without using built-in binary formatting

## Response Copilot Gave

"Sure thing! I used a loop to repeatedly divide the number by 2 and prepend the remainder to a string. This builds the binary representation manually. I also included a special check for the number 0, since that’s a unique case."

public static string DecimalToBinary(int decimalNumber)  
{  
 try  
 {  
 if (decimalNumber == 0)  
 {  
 return "0";  
 }  
 string binary = string.Empty;  
 while (decimalNumber > 0)  
 {  
 binary = (decimalNumber % 2) + binary;  
 decimalNumber /= 2;  
 }  
 return binary;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: 42 → Output: 101010

- Input: 10 → Output: 1010

- Handles zero case

# Question 6: Find Minimum in Rotated Sorted Array

## Prompt Used

Given a sorted array that has been rotated at some unknown pivot, implement a function to find the smallest element efficiently using binary search.

## Response Copilot Gave

"I implemented a binary search to identify the pivot point where the rotation happened. If the middle element is greater than the rightmost one, the minimum is in the right half. Otherwise, it’s in the left. This makes the solution fast and clean."

public static int FindMin(int[] nums)  
{  
 try  
 {  
 if (nums == null || nums.Length == 0)  
 {  
 throw new ArgumentException("Array must not be null or empty.");  
 }  
 int left = 0;  
 int right = nums.Length - 1;  
 while (left < right)  
 {  
 int mid = left + (right - left) / 2;  
 if (nums[mid] > nums[right])  
 {  
 left = mid + 1;  
 }  
 else  
 {  
 right = mid;  
 }  
 }  
 return nums[left];  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: [3, 4, 5, 1, 2] → Output: 1

- Input: [4, 5, 6, 7, 0, 1, 2] → Output: 0

- Efficient with binary search

# Question 7: Palindrome Number

## Prompt Used

Create a function that checks whether a given non-negative integer reads the same backward as forward, without converting it to a string.

## Response Copilot Gave

"Absolutely! I reversed the number digit by digit using modulo and integer division. Then I compared the reversed number to the original one to determine if it’s a palindrome. This avoids string conversion and handles edge cases like negatives."

public static bool IsPalindrome(int x)  
{  
 try  
 {  
 if (x < 0)  
 {  
 return false;  
 }  
 int original = x;  
 int reversed = 0;  
 while (x > 0)  
 {  
 int digit = x % 10;  
 reversed = reversed \* 10 + digit;  
 x /= 10;  
 }  
 return original == reversed;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: 121 → Output: true

- Input: 10 → Output: false

- Handles negative inputs

# Question 8: Fibonacci Number

## Prompt Used

Implement an efficient method to compute the nth number in the Fibonacci sequence using iteration, handling inputs from 0 up to at least 30.

## Response Copilot Gave

"Here you go! I handled the base cases (0 and 1) directly, then used a loop to calculate each Fibonacci number up to n. This avoids recursion and stack overflows, and runs efficiently even for larger values like 30."

public static int Fibonacci(int n)  
{  
 try  
 {  
 if (n < 0)  
 {  
 throw new ArgumentException("Input must be a non-negative integer.");  
 }  
 if (n == 0) return 0;  
 if (n == 1) return 1;  
 int a = 0;  
 int b = 1;  
 for (int i = 2; i <= n; i++)  
 {  
 int temp = a + b;  
 a = b;  
 b = temp;  
 }  
 return b;  
 }  
 catch (Exception)  
 {  
 throw;  
 }  
}

## Implementation Details

Code was generated using GitHub Copilot and integrated directly. Try-catch blocks are used for safety.

## Edge Cases Covered

- Input: 2 → Output: 1

- Input: 3 → Output: 2

- Input: 4 → Output: 3

**Copilot's Response:**  
"Sure! I created a function that loops from 1 to n and checks which numbers aren’t in the array. To handle duplicates efficiently, I stored the input in a HashSet. That way, the lookup is fast, and we skip repeated values."

### ✅ ****Question 2: Sort Array by Parity****

**Prompt:**  
Reorder the given array such that all even numbers come before any odd numbers. The function should handle zeros, single-element arrays, and preserve stability where possible.

**Copilot's Response:**  
"Of course! I used a two-pointer approach where one pointer fills in even numbers from the start, and the other fills in odd numbers from the end. This keeps the implementation simple and ensures that even numbers appear first regardless of the original order."

### ✅ ****Question 3: Two Sum****

**Prompt:**  
Implement logic that, given an integer array and a target sum, returns the indices of the two elements whose sum equals the target. Assume that each input has exactly one solution.

**Copilot's Response:**  
"Got it! I used a dictionary to store the numbers and their indices. As I loop through the array, I check if the complement (target - current value) already exists in the dictionary. If it does, I return the indices. It’s efficient and works in O(n) time."

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

**Prompt:**  
Given an array of integers (which may include negative values), determine the maximum product that can be achieved by multiplying any three of them.

**Copilot's Response:**  
"Here’s what I did: I sorted the array and calculated two possible products — the product of the three largest numbers, and the product of the two smallest (most negative) numbers and the largest one. The maximum of these two gives the correct answer."

### ✅ ****Question 5: Decimal to Binary Conversion****

**Prompt:**  
Convert a non-negative decimal integer to its binary string representation without using built-in binary formatting.

**Copilot's Response:**  
"Sure thing! I used a loop to repeatedly divide the number by 2 and prepend the remainder to a string. This builds the binary representation manually. I also included a special check for the number 0, since that’s a unique case."

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

**Prompt:**  
Given a sorted array that has been rotated at some unknown pivot, implement a function to find the smallest element efficiently using binary search.

**Copilot's Response:**  
"I implemented a binary search to identify the pivot point where the rotation happened. If the middle element is greater than the rightmost one, the minimum is in the right half. Otherwise, it’s in the left. This makes the solution fast and clean."

### ✅ ****Question 7: Palindrome Number****

**Prompt:**  
Create a function that checks whether a given non-negative integer reads the same backward as forward, without converting it to a string.

**Copilot's Response:**  
"Absolutely! I reversed the number digit by digit using modulo and integer division. Then I compared the reversed number to the original one to determine if it’s a palindrome. This avoids string conversion and handles edge cases like negatives."

### ✅ ****Question 8: Fibonacci Number****

**Prompt:**  
Implement an efficient method to compute the nth number in the Fibonacci sequence using iteration, handling inputs from 0 up to at least 30.

**Copilot's Response:**  
"Here you go! I handled the base cases (0 and 1) directly, then used a loop to calculate each Fibonacci number up to n. This avoids recursion and stack overflows, and runs efficiently even for larger values like 30."