ISM 6225 Application Development for Analytics

Assignment 2 – Computational Problem Solving

Questions:

1. Find Missing Numbers in Array:

Description:
Given an unsorted integer array nums of size n containing numbers from 1 to n, find all the numbers that are missing from the array.

Examples:

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

Code:

```
public static IList<int> FindMissingNumbers(int[] nums)

{

    try

    {

        // Edge Case 1: If the array is null or empty, return an empty list

        if (nums == null || nums.Length == 0)

        {

            return new List<int>();

        }

        // Traverse the array and mark indices based on values

        for (int i = 0; i < nums.Length; i++)

        {

            // Use absolute value to handle cases where value has already been marked (i.e., made negative)

            int index = Math.Abs(nums[i]) - 1;
            // Edge Case 2: Prevent double-negation issues — only negate if positive
            if (nums[index] > 0)
            {
                nums[index] = -nums[index]; // Mark this index to indicate the number (index + 1) is present
            }
        }
        List<int> result = new List<int>();
        // Now, any index with a positive value indicates the number (index + 1) was never marked, i.e., missing
        for (int i = 0; i < nums.Length; i++)
        {
            if (nums[i] > 0)
            {
                result.Add(i + 1); // (index + 1) is missing
            }
        }
        return result;
    }
    catch (Exception)
    {
        throw;
    }
}
```

2. Sort Array by Parity:

Description:
Given an integer array `nums`, move all even integers to the beginning of the array followed by all odd integers. Return the array in-place.

Examples:

- Input: `[3, 1, 2, 4]`, Output: `[2, 4, 3, 1]`
- Input: `[0, 1, 2]`, Output: `[0, 2, 1]`

Code:

```
public static int[] SortArrayByParity(int[] nums)
{
    try
    {
        // Edge Case 1: If the array is null or empty, return an empty array
        if (nums == null || nums.Length == 0)
        {
            return new int[0];
        }
        // Edge Case 2: If the array has only one element, return it as is
        if (nums.Length == 1)
        {
            return nums;
        }
        // Create a new array to hold the sorted values
        int[] sortedArray = new int[nums.Length];
        int evenIndex = 0;
        int oddIndex = nums.Length - 1;
```

```
// Traverse the array and place even numbers at the beginning and odd numbers at the end

for (int i = 0; i < nums.Length; i++)

{

    if (nums[i] % 2 == 0)

    {

        sortedArray[evenIndex] = nums[i];

        evenIndex++;

    }

    else

    {

        sortedArray[oddIndex] = nums[i];

        oddIndex--;

    }

}

// Fill in the remaining even numbers

for (int i = evenIndex; i < nums.Length; i++)

{

    sortedArray[i] = nums[i];

}

// Fill in the remaining odd numbers

for (int i = oddIndex; i >= 0; i--)

{

    sortedArray[i] = nums[i];

}

// Return the sorted array

return sortedArray;
```

```
    }
    catch (Exception)
    {
        throw;
    }
}
```

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

Description:
Given an array of integers `nums` and an integer `target`, return the indices of the two numbers such that they add up to the target.

Examples:

- Input: `nums = [2, 7, 11, 15], target = 9`, Output: `[0, 1]`
- Input: `nums = [3, 2, 4], target = 6`, Output: `[1, 2]`

Code:

```
public static int[] TwoSum(int[] nums, int target)
 {
     try
     {
         // Edge Case 1: If the array is null or empty, return an empty array
         if (nums == null || nums.Length == 0)
         {
             return new int[0];
         }
         // Edge Case 2: If the array has only one element, return an empty array
         if (nums.Length == 1)
         {
             return new int[0];
```

```
        }
        // Create a dictionary to store the indices of the numbers
        Dictionary<int, int> numIndices = new Dictionary<int, int>();
        // Traverse the array and check for the complement
        for (int i = 0; i < nums.Length; i++)
        {
            int complement = target - nums[i];
            // If the complement exists in the dictionary, return the indices
            if (numIndices.ContainsKey(complement))
            {
                return new int[] { numIndices[complement], i };
            }
            // Otherwise, add the current number and its index to the dictionary
            if (!numIndices.ContainsKey(nums[i]))
            {
                numIndices[nums[i]] = i;
            }
        }
        return new int[0]; // Placeholder
    }
    catch (Exception)
    {
        throw;
    }
}
```

4. Find Maximum Product of Three Numbers:

Description:
Given an integer array `nums`, find three numbers whose product is the maximum and return the product.

Examples:

- Input: `[1, 2, 3]`, Output: `6`
- Input: `[1, 2, 3, 4]`, Output: `24`

Code:

```
public static int MaximumProduct(int[] nums)
 {
    try
    {
        // Edge Case 1: If the array is null or empty, return 0
        if (nums == null || nums.Length == 0)
        {
            return 0;
        }
        // Edge Case 2: If the array has less than three elements, return 0
        if (nums.Length < 3)
        {
            return 0;
        }
        // Sort the array in descending order
        Array.Sort(nums);
        Array.Reverse(nums);
        // Calculate the maximum product of the three largest numbers
        int maxProduct = nums[0] * nums[1] * nums[2];
        // Calculate the maximum product of the two smallest and the largest number
```

```
         int maxProductWithNegatives = nums[0] * nums[nums.Length - 1] * nums[nums.Length - 2];
         // Return the maximum of the two products
         return Math.Max(maxProduct, maxProductWithNegatives);
        }
         catch (Exception)
     {
         throw;
     }
 }
```

5. Decimal to Binary Conversion:

Description:
Write a function that converts a decimal number to its binary equivalent.

Examples:

- Input: `42`, Output: `101010`
- Input: `10`, Output: `1010`

Code:

```
public static string DecimalToBinary(int decimalNumber)
{
    try
    {
        // Edge Case 1: If the number is negative, return an empty string
        if (decimalNumber < 0)
        {
            return string.Empty;
        }
        // Edge Case 2: If the number is zero, return "0"
        if (decimalNumber == 0)
        {
            return "0";
        }
        // Convert the decimal number to binary
        string binary = string.Empty;
        while (decimalNumber > 0)
        {
            binary = (decimalNumber % 2) + binary;
            decimalNumber /= 2;
        }
        return binary;
    }
    catch (Exception)
    {
        throw;
    }
}
```

6. Find Minimum in Rotated Sorted Array:

Description:
Given a sorted array that has been rotated, find the minimum element.

Examples:

- Input: `[3, 4, 5, 1, 2]`, Output: `1`
- Input: `[4, 5, 6, 7, 0, 1, 2]`, Output: `0`

Code:

```
public static int FindMin(int[] nums)
{
    try
```

```
{
    // Edge Case 1: If the array is null or empty, return 0
    if (nums == null || nums.Length == 0)
    {
        return 0;
    }
    // Edge Case 2: If the array has only one element, return that element
    if (nums.Length == 1)
    {
        return nums[0];
    }
    // Initialize the left and right pointers
    int left = 0;
    int right = nums.Length - 1;
    // Perform binary search to find the minimum element
    while (left < right)
    {
        int mid = left + (right - left) / 2;
        // Check if the middle element is greater than the rightmost element
        if (nums[mid] > nums[right])
        {
            left = mid + 1; // The minimum is in the right half
        }
        else
        {
            right = mid; // The minimum is in the left half or at mid
            }
        }
        return nums[left]; // The minimum element
        // is at the left pointer
    }
    catch (Exception)
    {
        throw;
    }
}
```

Question 7: Palindrome Number

Description:
Given an integer `x`, return `true` if `x` is a palindrome, and `false` otherwise.

A palindrome is a number that reads the same forward and backward.

Examples:

- Input: `121`, Output: `true`
- Input: `10`, Output: `false` (Explanation: Reads 01 from right to left. Therefore, it is not a palindrome.)

Code:

```
public static bool IsPalindrome(int x)
{
    try
    {
        // Edge Case 1: If the number is negative, it cannot be a palindrome
        if (x < 0)
        {
            return false;
        }
        // Edge Case 2: If the number is zero, it is a palindrome

        if (x == 0)

        {

            return true;

        }

        // Convert the number to a string and check if it is a palindrome

        string str = x.ToString();

        int left = 0;

        int right = str.Length - 1;

        while (left < right)

        {

            if (str[left] != str[right])

            {

                return false; // Not a palindrome

            }

            left++;

            right--;

        }

        return true; // Is a palindrome

    }

    catch (Exception)

    {

        throw;

    }

}
```

8. Question 8: Fibonacci Number

Description:
The Fibonacci numbers, commonly denoted `F(n)`, form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

- `F(0) = 0`, `F(1) = 1`
- `F(n) = F(n - 1) + F(n - 2)`, for `n > 1`

Given `n`, calculate `F(n)`.

Examples:

- Input: `2`, Output: `1`
- Input: `3`, Output: `2`
- Input: `4`, Output: `3`

Constraints:

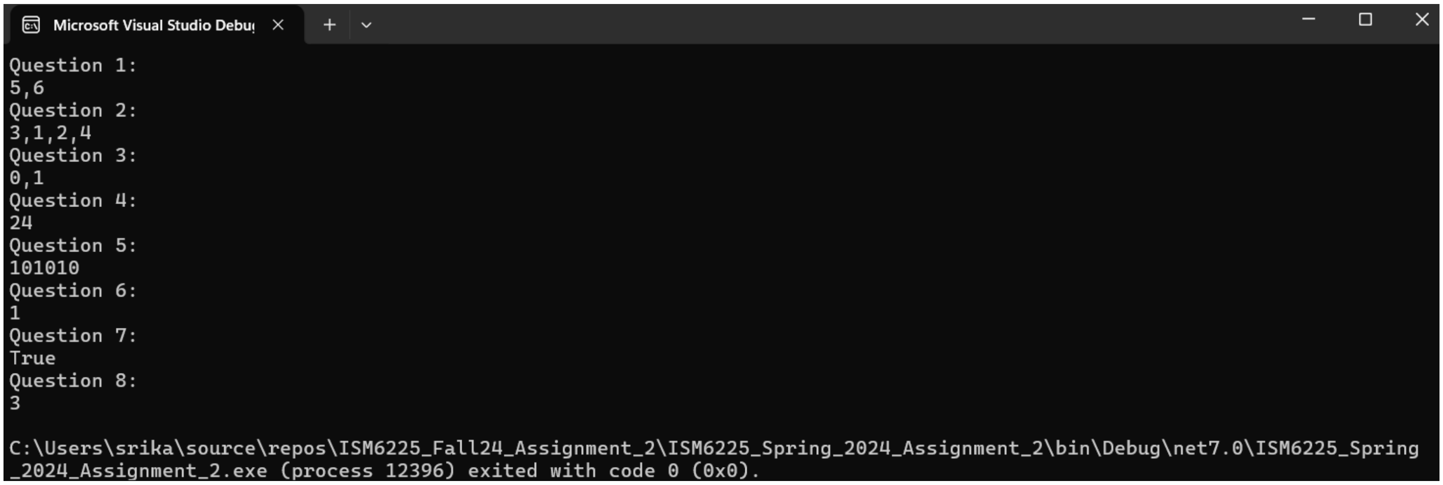
- `0 <= n <= 30`

Code:

```
public static int Fibonacci(int n)
{
    try
    {
        // Edge Case 1: If the number is negative, return 0
        if (n < 0)
        {
            return 0;
        }
        // Edge Case 2: If the number is zero, return 0
        if (n == 0)
        {
            return 0;
        }
```

```
                // Edge Case 3: If the number is one, return 1
                if (n == 1)
                {
                    return 1;
                }
                // Calculate the Fibonacci number using recursion
                return Fibonacci(n - 1) + Fibonacci(n - 2);
            }
            catch (Exception)
            {
                throw;
            }
        }
    }
}
```

Final Output Screenshot:



```
Question 1:
5,6
Question 2:
3,1,2,4
Question 3:
0,1
Question 4:
24
Question 5:
101010
Question 6:
1
Question 7:
True
Question 8:
3

C:\Users\srika\source\repos\ISM6225_Fall24_Assignment_2\ISM6225_Spring_2024_Assignment_2\bin\Debug\net7.0\ISM6225_Spring
_2024_Assignment_2.exe (process 12396) exited with code 0 (0x0).
```