## Assignment 2 – Computational Problem Solving

1. Find Missing Numbers in Array:

**Prompts Used:** There are two mains edge cases here: Empty array and array with duplicate numbers. This is how I handled empty arrays. How to deal with arrays with duplicates? Help me complete the code and tell me how to account for exceptions.

public static IList<int> FindMissingNumbers(int[] nums) //Method to find missing numbers

{

try

{

IList<int> missingNumbers = new List<int>(); // List of the missing numbers.

// Edge case 1: Check if the array is empty

if (nums.Length == 0)

{

return missingNumbers; // Return an empty

}

// Find the missing numbers

for (int k = 1; k <= nums.Length; k++)

{

missingNumbers.Add(k);

}

// Return the list of missing numbers

return missingNumbers;

}

catch (Exception)

{

throw;

}

}

**Relevant AI Suggestions**

// Edge case 2: Check if the array has duplicates

bool[] present = new bool[nums.Length + 1]; // Create a boolean array to track which numbers are present

// Mark the numbers as present

foreach (int num in nums)

{

if (num <= nums.Length) // Only consider numbers in the range 1 to n (inclusive)

{

present[num] = true; // Mark the number as present

}

}

// Find the missing numbers

for (int k = 1; k <= nums.Length; k++)

{

if (!present[k]) // If a number is missing (not marked as true in the `present` array)

{

missingNumbers.Add(k); // Add the missing number to the list

}

}

// Handle exceptions: ArgumentNullException and General Exceptions

catch (ArgumentNullException ex)

{

Console.WriteLine("Input array is null: " + ex.Message);

throw;

}

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

throw;

}

To handle duplicates, the AI suggested creating a Boolean array to track numbers and provided ways to catch specific exceptions like ArgumentNullException and general exceptions and provided specificl error messages.

**Implementation Details and Adjustments (if applicable)**

I included the response received to complete my code. I first added a line that checks if the array is empty and returns an empty list if so. Then, I incorporated the AI suggestion to create a boolean array to track which numbers from 1 to the array length are present. This iterated through the array to mark the numbers present as true, which also handles duplicates. The boolean array values that are false indicate missing numbers and they were added to the missing number list. I then included the AI suggestion on how to handle exceptions and included an error message.

1. Sort Array by Parity:

**Prompts Used:** There are two mains edge cases here: Empty array and array with no odd or no even numbers. This is how I handled them. Help me complete the code, I am not sure how to switch positions also for exceptions include message that says either numbers are all even or numbers are all odd if they are.

// Question 2: Sort Array by Parity

public static int[] SortArrayByParity(int[] nums) // Method to sort the array by parity

{

try

{

// Edge case 1: Check if array is empty

if (nums.Length == 0)

{

throw new InvalidOperationException("Empty Input Array");

}

// Edge case 2: Check if the array has no odd or no even numbers

bool allEven = true;

bool allOdd = true;

foreach (int num in nums)

{

{

if (num % 2 == 0) allOdd = false;

else allEven = false;

}

}

// Position in Array: "begin" as start at the beginning and "end" as starts at the end

int begin = 0;

int end = nums.Length - 1;

while (begin < end)

{

// If the first number is even, move to the next numbert

if (nums[begin] % 2 == 0) begin++;

// If the last number is odd, move to the number before it.

if (nums[end] % 2 == 1) end--;

// If the number from begin is odd and the number from end is even, switch their place

else

{

}

}

return nums;

}

// Handle exceptions

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

throw;

}

}

**Relevant AI Suggestions**

// Edge case 2: Check if the array has no odd or no even numbers

bool allEven = true;

bool allOdd = true;

foreach (int num in nums)

{

{

if (num % 2 == 0) allOdd = false;

else allEven = false;

if (!allEven && !allOdd) break; // Exit the loop if both conditions are met

}

if (allEven) throw new InvalidOperationException("All elements are even.");

if (allOdd) throw new InvalidOperationException("All elements are odd.");

}

// Position in Array: "begin" as start at the beginning and "end" as starts at the end

int begin = 0;

int end = nums.Length - 1;

while (begin < end)

{

// If the first number is even, move to the next number

if (nums[begin] % 2 == 0) begin++;

// If the last number is odd, move to the number before it.

if (nums[end] % 2 == 1) end--;

// If the number from begin is odd and the number from end is even, switch their place

else

{

int temp = nums[begin];

nums[begin] = nums[end];

nums[end] = temp;

begin++;

end--;

}

}

// Handle exceptions

catch (InvalidOperationException ex)

{

Console.WriteLine(ex.Message);

throw;

}

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

throw;

}

AI suggested a new line for edge case 2 that exists the loop if the array has a mix of both even and odd numbers. If that is not the case, there is a message that says all numbers are even or odd. I also did not know how to switch positions when an odd/even number was not in the right place. AI provided the code to do that and also included a new line in the exception handling section to catch InvalidOperationException.

**Implementation Details and Adjustments (if applicable)**

I included the relevant suggestions to complete my code. First, I added the line that exits from the loop in Edge Case 2 which I had missed. First my code check if the array is empty to return the empty array. It also checks if the array contains only even or only odd numbers using bool which trigger an exception to indicate this issue. There two pointers, one starting at the beginning of the array and the other at the end. The code moves these pointers and switch the numbers depending on whether the numbers are even or odd. At the end, we check for suggested exceptions. The only change I made was in the error message that “All elements are…” I changed them to “all numbers” to make it more intuitive for me.

1. *Two Sum*

**Prompts Used:** To solve this computational problem that returns the indices of the two numbers from nums such that they add up to a target, help me complete the iteration. There are multiple edge cases here: if array is empty, if array contains only one number or if no two numbers add up to the target. This is what I have so far:

public static int[] TwoSum(int[] nums, int target)

{

try

{

// Edge case 1: Check if array is empty

if (nums.Length == 0)

{

throw new InvalidOperationException("Empty Input Array");

}

// Edge case 2: Check if array contains only one number

if (nums.Length == 1)

{

throw new InvalidOperationException("No two sum solution");

}

// Declare Dictionary to store the indices of the numbers

Dictionary<int, int> numDict = new Dictionary<int, int>();

// Iterate through the array and find the right pairs for the target

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

{

}

}

// Edge case 3: Check if array contains only one number

throw new InvalidOperationException("No two sum solution");

catch (InvalidOperationException ex)

{

Console.WriteLine(ex.Message);

throw;

}

catch (Exception ex)

{

Console.WriteLine("An error occurred: " + ex.Message);

throw;

}

}

**Relevant AI Suggestions**

// Iterate through the array

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

{

int complement = target - nums[k]; // Calculate the complement

// Check if the complement exists in the dictionary

if (numDict.ContainsKey(complement))

{

return new int[] { numDict[complement], k }; // Return the indices of the complement and the current number

}

// Store the current number and its index in the dictionary

numDict[nums[k]] = k;

}

The AI suggestion gave me a solution to address the iteration in a way that the code loops through the array and calculates the sum of pairs (complement) for each element. Then, it checks if the sum is already in a dictionary that stores previous numbers and their indices. If the pair sum is found, it returns the indices of the two numbers that add up to the target.

**Implementation Details and Adjustments (if applicable)**

I incorporated the AI suggestion in my original code and as you can see, the AI referred to the sum of pairs as complement and it was not a name that was intuitive for me so I changed it to “pairSum”.

1. *Find Maximum Product of Three Numbers*

**Prompts Used:** Here, I already have an array called “nums” with numbers and I want create a code that can calculate the maximum product of three Numbers. I already sorted the number in ascending order to separate largest numbers from the smallest and facilitate computation. Complete this:

// Sort the array in ascending order

Array.Sort(nums);

// Calculate the maximum product of three numbers

int k = nums.Length;

int maxProduct = ?

// Return the maximum product

return maxProduct;

**Relevant AI Suggestions**

// Calculate the maximum product of three numbers

int n = nums.Length;

int maxProduct = Math.Max(nums[k - 1] \* nums[k - 2] \* nums[k - 3], nums[0] \* nums[1] \* nums[k - 1]);

**Implementation Details and Adjustments (if applicable)**

I used the AI suggestion to complete the problem solution. This code calculates two possible products: one is the product of the three largest numbers in the array and the other is the product of the two smallest numbers and the largest numbers. Then, it returns the larger of the two products. I modified it because the product of the three largest numbers is usually the biggest, so the final code I used is: int maxProduct = nums[k - 1] \* nums[k - 2] \* nums[k - 3];

1. *Decimal to Binary Conversion*

**Prompts Used:** This is what I used to convert a decimal number into binary. Is there a better way to do it with less lines?

// Store the binary number

string binary = string.Empty;

// Convert the decimal number to binary

// Calculate the remainder, add it to the binary and continue with the next decimal number until it becomes 0

while (decimalNumber > 0)

{

int remainder = decimalNumber % 2;

binary = remainder + binary;

decimalNumber /= 2;

}

// Return the binary string

return binary;

**Relevant AI Suggestions**

// Use StringBuilder to store the binary representation

var binary = new System.Text.StringBuilder();

// Convert the decimal number to binary

while (decimalNumber > 0)

{

binary.Insert(0, decimalNumber % 2);

decimalNumber /= 2;

}

// Return the binary string

return binary.ToString();

**Implementation Details and Adjustments (if applicable)**

I did not implement the AI suggestion. It seems more complicated than the lines of codes I had originally.

1. *Find Minimum in Rotated Sorted Array*

**Prompts Used:** what’s the best way to approach a problem that involves finding the minimum in rotated sorted array where the pointers are set as begin and end.

**Relevant AI Suggestions**

Here is a step-by-step approach:

1. Initialize Pointers: Set two pointers, begin and end, to the start and end of the array, respectively.
2. Binary Search Loop: Use a while loop to perform binary search until begin is less than end.
3. Calculate Midpoint: Calculate the midpoint of the current range.
4. Compare Midpoint with End:
   * If the element at the midpoint is greater than the element at end, the minimum is in the right half, so move the begin pointer to mid + 1.
   * Otherwise, the minimum is in the left half (including the midpoint), so move the end pointer to mid.
5. Return Minimum: When the loop exits, the begin pointer will point to the minimum element.

Here is the implementation in C#:

// Initialize pointers

int begin = 0;

int end = nums.Length - 1;

// Binary search to find the minimum element

while (begin < end)

{

int mid = begin + (end - begin) / 2;

// If the middle element is greater than the rightmost element,

// the minimum is in the right half

if (nums[mid] > nums[end])

{

begin = mid + 1;

}

// Otherwise, the minimum is in the left half (including mid)

else

{

end = mid;

}

}

return nums[begin];

**Implementation Details and Adjustments (if applicable)**

I included the AI suggestion into my solution and simplified the comments to make them easier for me to understand. The code uses binary search to find the smallest number in a rotated sorted array. It adjusts the pointers based on comparing the middle number with the last number which narrows down the search for the minimum. When the loop ends, the begin pointer will point to the smallest number. I found this method great and did not make any changes.

1. *Palindrome Number*

**Prompts Used:** Without converting a number into a string, how can I check if a number is a palindrome? I have already ensured that if is not a negative number.

**Relevant AI Suggestions**

To reverse a number without converting it to a string and check if it is a palindrome, follow these steps:

* + 1. Handle Edge Cases: If the number is negative, it is not a palindrome. If the number is a single digit, it is a palindrome.
    2. Reverse the Number: Initialize a variable to store the reversed number. Use a loop to extract each digit from the original number and build the reversed number.
    3. Compare: Compare the original number with the reversed number.

Here is the code snippet:

// Store the original number

int original = x;

int reversed = 0;

// Reverse the number

while (x > 0)

{

int digit = x % 10;

reversed = reversed \* 10 + digit;

x /= 10;

}

// Check if the original number is equal to the reversed number

return original == reversed;

**Implementation Details and Adjustments (if applicable)**

I included the AI suggestion into my solution. Here, to check if a number is a palindrome, the digits of the number are reversed using a while loop. Afterward, the original number is compared to the reversed number. If they are the same, it means the number is a palindrome. This approach seems pretty simple so I did not make any adjustment.

1. *Fibonacci Number*

**Prompt Used:** What are the steps to compute the Fibonacci number of an integer

**Answer:**

To compute the Fibonacci number of an integer, you can follow these steps:

1. Check if the input integer n is 0 or 1. If so, return n directly because F(0) = 0 and F(1) = 1.
2. Initialize Variables: Initialize two variables to store the first two Fibonacci numbers, F(0) and F(1).
3. Iterative Calculation: Use a loop to calculate the Fibonacci number iteratively from 2 to n.
4. Return Result: Return the nth Fibonacci number.

**Prompt Used**: I have set up the first part of the code as follows. Complete it by filling in the loop

// Set the first two numbers in the sequence

int k = 0;

int s = 1;

// Calculate the Fibonacci number using a loop starting at n = 2

for (int j = 2; j <= n; j++)

{

}

// Return the nth Fibonacci number

**Relevant AI Suggestions**

// Set the first two numbers in the sequence

int k = 0;

int s = 1;

// Calculate the Fibonacci number using a loop starting at n = 2

for (int j = 2; j <= n; j++)

{

int temp = k + s;

k = s;

s = temp;

}

// Return the nth Fibonacci number

return s;

**Implementation Details and Adjustments (if applicable)**

I included the AI suggestion into my solution. After setting the first two numbers as k for F(0) = 0 and s for F(1) = 1, the code uses a loop to calculate the next Fibonacci numbers, update the two numbers until you reach the final number n. Then, it returns the Fibonacci number for n. I did not make any changes to the AI suggestions