1. How can I improve findmissingnumbers? Suggest code for any improvements

Copilot response:

To improve the FindMissingNumbers method, we can optimize the sorting process and simplify the logic for finding missing numbers.

The current implementation uses a bubble sort, which is inefficient for large arrays.

We can replace it with a more efficient sorting algorithm like Array.Sort. Additionally, we can streamline the process of finding missing numbers by using a HashSet to track the numbers present in the array.

Here is the improved code:

// Question 1: Find Missing Numbers in Array

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

{

try

{

// Sort the array using Array.Sort for better performance

Array.Sort(nums);

List<int> missingNumbers = new List<int>();

HashSet<int> numSet = new HashSet<int>(nums);

int expectedNumber = 1;

// Find missing numbers

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

{

// Skip duplicate values

if (i > 0 && nums[i] == nums[i - 1])

continue;

// Add missing numbers

while (expectedNumber < nums[i])

{

if (!numSet.Contains(expectedNumber))

{

missingNumbers.Add(expectedNumber);

}

expectedNumber++;

}

expectedNumber++;

}

// Add remaining missing numbers till n

while (expectedNumber <= nums.Length)

{

if (!numSet.Contains(expectedNumber))

{

missingNumbers.Add(expectedNumber);

}

expectedNumber++;

}

return missingNumbers;

}

catch (Exception)

{

throw;

}

}

Implementation and adjustment: Modify the code by replacing bubble sort with array sort to make it clearer and more efficient for the large arrays.

2. How can I improve SortArrayByParity?

To improve the SortArrayByParity method, we can optimize the code by using a single pass approach with two pointers.

This will reduce the space complexity and improve the performance. Here is the improved code:

// Question 2: Sort Array by Parity

public static int[] SortArrayByParity(int[] nums)

{

try

{

int left = 0;

int right = nums.Length - 1;

while (left < right)

{

// If the left element is even, move the left pointer to the right

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

{

left++;

}

// If the right element is odd, move the right pointer to the left

else if (nums[right] % 2 != 0)

{

right--;

}

// Swap the left odd element with the right even element

else

{

int temp = nums[left];

nums[left] = nums[right];

nums[right] = temp;

left++;

right--;

}

}

return nums;

}

catch (Exception)

{

throw;

}

}