ISM6225 Spring25 Assignment 2

Mauricio Bermudez

04/04/2025

# Question 1:

**Me:** I am looking to review a way to check if I am missing a number in my counting array. Here is the code I currently have. Would this be correct?

int[] tempArray = new int[nums.Length];   
for (int i = 0; i < nums.Length; i++)  
{   
 if (nums[i] > 0 && nums[i] <= nums.Length)  
 {  
 tempArray[nums[i] - 1] = 1;  
 }  
}

**Copilot:** Your code snippet is a good start for identifying missing numbers in an array. However, it currently only marks the presence of numbers in the tempArray but does not return the missing numbers. To complete the functionality, you need to iterate through the tempArray and collect the indices where the value is still 0 (indicating the missing numbers). Here’s how you can modify your code to achieve this:

int[] tempArray = new int[nums.Length]; for (int i = 0; i < nums.Length; i++) { if (nums[i] > 0 && nums[i] <= nums.Length) { tempArray[nums[i] - 1] = 1; } }  
List<int> missingNumbers = new List<int>(); for (int i = 0; i < nums.Length; i++) { if (tempArray[i] == 0) { missingNumbers.Add(i + 1); } }  
return missingNumbers;

# Question 2:

**Me:** Hi there, I am trying to take an array of numbers, check if it’s odd or even, assign it to an even array if it is even and an odd array if it’s odd. Then I’d like to combine the 2 arrays, where the even array is first and the odd array is second. Here is my code, I was getting an exception as well. Would you be able to verify if I’m doing this in the right fashion, and that my code is not out of scope?

int[] evenIndex = new int[nums.Length/2];  
int[] oddIndex = new int[nums.Length/2];  
int[] sortedArray = new int[nums.Length];  
for (int i = 0; i < nums.Length; i++)  
{  
 if (nums[i] % 2 == 0)  
 {  
 evenIndex[i] = nums[i];  
 }  
 else  
 {  
 oddIndex[i] = nums[i];  
 }  
}  
sortedArray = evenIndex.Concat(oddIndex).ToArray();  
return sortedArray;

**Copilot:** Your code has a few issues that could lead to exceptions or incorrect behavior:

Array Index Out of Bounds:

You are using evenIndex[i] and oddIndex[i] directly, but i is iterating over the entire nums array. Since evenIndex and oddIndex are initialized with a size of nums.Length / 2, this will cause an IndexOutOfRangeException when i exceeds the size of these arrays. Incorrect Array Initialization:

The size of evenIndex and oddIndex assumes that exactly half of the numbers will be even and half odd, which may not always be true. This assumption can lead to incorrect results or exceptions. Unused Indices for Even and Odd Arrays:

You are not keeping track of the current index for evenIndex and oddIndex, so values are being placed at incorrect positions. Corrected Code Here is a corrected version of your method:

List<int> evenIndex = new List<int>();  
List<int> oddIndex = new List<int>()  
for (int i = 0; i < nums.Length; i++)  
{  
 if (nums[i] % 2 == 0)  
 {  
 evenIndex.Add(nums[i]);  
 }  
 else  
 {  
 oddIndex.Add(nums[i]);  
 }  
}  
return evenIndex.Concat(oddIndex).ToArray();

# Question 4:

**Me:** Would you be able to review my code? I’m trying to iterate through the loop and get the 3 maximum product values and then return the product itself. Below is the code I wrote but I seem to be receiving an error of unusued variables. Am I on the right path? My thought process is start at the end of the array, and compare that index to the one before it. If the value is higher, it assigns to variable. If not, it’ll step to the next lowest index. These loops are repeated until we find the 3 highest values and then return the multiplied value.

int x,y,z;  
for (int i = -1; i >= 0; i--)  
{  
 if (nums[i] > nums[i - 1])  
 {  
 x = nums[i];  
 for (int j = -1; j >= 0; j--)  
 {  
 if (nums[j] > nums[j - 1] && nums[j] != x)  
 {  
 y = nums[j];  
 for (int k = -1; k >= 0; k--)  
 {  
 if (nums[k] > nums[k - 1] && nums[k] != x && nums[k] != y)  
 {  
 z = nums[k];  
 }  
 }  
 }  
 }  
 }  
}  
return x \* y \* z;

**Copilot:** Your code has several issues that need to be addressed:

Issues: Negative Indexing: In C#, arrays do not support negative indexing. Using nums[-1] or iterating with i = -1 will throw an IndexOutOfRangeException. Uninitialized Variables: The variables x, y, and z are declared but not initialized. If the conditions in the loops are never met, they will remain uninitialized, causing a compilation error when used in return x \* y \* z. Inefficient Logic: The nested loops are unnecessarily complex for finding the three largest numbers in an array. This can be simplified significantly. Edge Cases: The code does not handle cases where the array has fewer than three elements. Corrected Approach: To find the three largest numbers in an array and return their product, you can use a more efficient approach:

Sort the array in descending order and pick the first three elements. Alternatively, iterate through the array once to find the three largest numbers. Here’s the corrected code:

int FindMaxProduct(int[] nums)  
{  
 if (nums.Length < 3)  
 {  
 throw new ArgumentException("Array must contain at least three elements.");  
 }  
  
 // Initialize the three largest numbers  
 int max1 = int.MinValue, max2 = int.MinValue, max3 = int.MinValue;  
  
 foreach (int num in nums)  
 {  
 if (num > max1)  
 {  
 max3 = max2;  
 max2 = max1;  
 max1 = num;  
 }  
 else if (num > max2)  
 {  
 max3 = max2;  
 max2 = num;  
 }  
 else if (num > max3)  
 {  
 max3 = num;  
 }  
 }  
  
 return max1 \* max2 \* max3;  
}

Explanation: Edge Case Handling: The function checks if the array has fewer than three elements and throws an exception if true. Efficient Iteration: The array is traversed once (O(n) complexity) to find the three largest numbers. Variable Updates: The variables max1, max2, and max3 are updated dynamically as larger numbers are found.