**Data Structures Algorithms and Complexity Homework**

**Task 1. What is the expected running time of the following C# code? Explain why. Assume the array's size is n.**

**long Compute(int[] arr)**

**{**

**long count = 0;**

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

**{**

**int start = 0, end = arr.Length-1;**

**while (start < end)**

**if (arr[start] < arr[end])**

**{ start++; count++; }**

**else**

**end--;**

**}**

**return count;**

**}**

**EXPLANATION:**

**First cycle runs n times. Each time the inner ‘while’ cycle runs another n times, cause either ‘start’ grows by 1 each step until it reaches ‘end’ or ‘end’ diminishes by 1.**

**RESULT is:**

**Quadratic time => O(n\*n)**

**Approximate steps => n \* n**

**Task 2. What is the expected running time of the following C# code? Explain why. Assume the input matrix has size of n \* m.**

**long CalcCount(int[,] matrix)**

**{**

**long count = 0;**

**for (int row=0; row<matrix.GetLength(0); row++)**

**if (matrix[row, 0] % 2 == 0)**

**for (int col=0; col<matrix.GetLength(1); col++)**

**if (matrix[row,col] > 0)**

**count++;**

**return count;**

**}**

**EXPLANATION:**

**First cycle runs n times – the number of rows in the matrix. Then the inner cycle depends on the if statement, so it depends on the elements of the matrix themselves. So we have 3 cases**

**RESULT is:**

**WORST CASE => All elements in the matrix are divisible by 2. Then second cycle runs m times – the number of columns in the matrix, for each element.**

**Quadratic time => O(n\*m)**

**Approximate steps => n \* m**

**AVERAGE CASE => Half of the elements in the matrix are divisible by 2. Then second cycle runs m/2 times.**

**Quadratic time => O(n\*m)**

**Approximate steps => n \* m/2**

**BEST CASE => None of the elements in the matrix is divisible by 2, so inner loop wont run at all.**

**Quadratic time => O(n)**

**Approximate steps => n**

**Task 3\*. What is the expected running time of the following C# code? Explain why. Assume the input matrix has size of n \* m.**

**long CalcSum(int[,] matrix, int row)**

**{**

**long sum = 0;**

**for (int col = 0; col < matrix.GetLength(0); col++)**

**sum += matrix[row, col];**

**if (row + 1 < matrix.GetLength(1))**

**sum += CalcSum(matrix, row + 1);**

**return sum;**

**}**

**Console.WriteLine(CalcSum(matrix, 0));**

**EXPLANATION:**

**This C# code throws IndexOutOfRangeException if the number of rows and columns are different. So in order to do it right we swap the places of matrix.GetLength(0) and matrix.GetLength(1). Now we have working code.**

**The first loop will go m times – the number of columns in the matrix. Inside it, the recursion will be called n times – the number of rows, because at start the function is called starting from row 0 and the ‘if’ statement stops it, when row + 1 < number of rows.**

**RESULT is:**

**Quadratic time => O(n\*m)**

**Approximate steps => m \* n**