# Data Structures, Algorithms and Complexity Homework

***Task 1:***

**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;**

**}**

**Solution:**

For calculating the algorithm complexity of the method we consider *for* and *while* loops. The *for* loop will be executed **c1\*n** times, where **c1** is a constant that does not depend on **n**. The *while* loop will be executed **c2\*(n-1),** where **c2** does not depend on **n.** The *while* loop is nested in the *for* loop so we have complexity: ~ **c1\*n \*c2\*(n-1)** = **c1\*c2\* (n2 – n)**. Constants are ignored and the first degree of **n** also and we can say the algorithm complexity is **O(n2) – quadratic time**.   
NOTE: Here **n** is the length of the array that is passed to the method.

***Task 2:***

**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;**

**}**

**The input matrix has size n \* m.**

**Solution:**

For calculating the algorithm complexity of the method we consider the two *for* loops. The first *for* loop will be executed **c1\*n** times, where **c1** is a constant that does not depend on **n**. The conditional statement’s body in it will be executed **k** times. Here we take **k ~ n/2**, so the second for loop will be executed ~ **c2\*n\*m/2** times. So for algorithm complexity we have ~ **c1n/2 + c2n\*m/2** and we can say the algorithm complexity here is **O(n \* m).**

***Task 3:***

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

**{**

**long sum = 0;**

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

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

**if (row + 1 < matrix.GetLength(0))**

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

**return sum;**

**}**

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

**------------------------------------------------------------------------**

**Assume the input matrix has size of n \* m.**

**Solution:**

The provided code is performing a sum operation for all the cells of a matrix of size n \* m and uses recursion for it. The same effect can be produced with 2 nested loops (one for iterating over the rows and another for iterating over the columns in the current row). So the first loop will be executed **c1n** times and the second one will be executed **c1n\*c2m** times. The algorithm complexity of the given code then will be **O(n \* m)**.