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;

}

The complexity is n\*(n-1), i.e. n2-n, i.e. O(n2), because the first “for” statement is executed n times, and the inner “while” statement is executed n-1 times.

1. What is the expected running time of the following C# code? Explain why.

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 complexity is (n - a + m \* a), where “a” is the number of odd numbers contained in the matrix and “n” and “m” are the matrix dimensions. In the worst case scenario (when all the numbers are odd, aka a = n), the complexity is O(n-n + m\*n), i.e. O(m\*n).

The first for statement is executed n times, the inner statement is executed m\*a times, because it is only executed only when we have an odd numbers => “a” number of times.

1. What is the expected running time of the following C# code? Explain why.

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));

The complexity is n\*(n-m) => O(n2-n\*m). This is because the first “for” statement is executed “n” times and then recursively the method is called again, with complexity of (n-m).