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;

}

* Предвид факта, че масивът се състои от **n**-брой елементи, имаме два вложени цикъла, като респективно първият върти **n**-брой пъти, а вторият **n-1** – брой пъти. Сложността и бързината на алгоритъма е **n\*(n-1) = n2 – n** или **O(n2)**

1. **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;

}

* В случая имаме първи цикъл, който трябва да итерира **n**-брой пъти, но има ограничение (if (matrix[row, 0] % 2 == 0). Ако приемем, че ограничението е в сила **a**-брой пъти, то първият цикъл ще итерира **n-a** брой пъти. Вторият цикъл ще итерира **m-**брой пъти, когато влизаме в ограничението, тоест **m\*a**. Като цяло сложността изглежда така

**(n-a + m\*a).** Най-голяма сложност се постига, ако всеки път matrix[row,0] е четно число, така всеки път попадаме в ограничението, което означава, че **a = n**. В този случай, сложностт изглежда така **(n – n + m\*n)**, затова приемаме, че **O(m\*n)**.

1. **\* 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));

* В метода, цикълът представляващ колоните итерира **n**-брой пъти, имаме ограничение до **m**, което отговаря за итерациите на втория цикъл. Методът се повтаря рекурсивно всеки път, като сложността му е **(n-m)**. Сложността на целия алгоритъм е : **n\*(n-m) =O (n2 – n\*m)**