# 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;

}

Running time is **n\*n/2** **+ constants for each operation**, because in the inner cycle we will reduce the n by one on each turn of the while cycle. As with big numbers the constant becomes less significant we can say its O(**n2)**

# **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;

}

Running time is **n\*m** **+ constants for each operation**, because in the worst case we will have **matrix[row, 0] % 2** always equal 0. As with big numbers the constant becomes less significant we can say its O(**n\*m)**

# \* 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));

Running time is **n\*m + constants for each operation**, because we recursively call tge CalcSum method for each row ( this is essentially like having 2 for cycles nested ) . As with big numbers the constant becomes less significant we can say its O(n2).