# Expected Running Time

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;

}

Сложността на алгоритъма е n2 . Start и end не се променят. За всяко число правим обхождане с общ брой стъпки равни на дължината на масива.

# 02.Expected Running Time

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;

}

Сложността на алгоритъма е . Обхождаме всички редове в матрицата(n), за четните(половината) n/2 обхождаме всички колони(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;

}

Методът изчислява сумата на клетките в редовете по-големи и равни на row. В най-лошият случай, когато row = 0 имаме O(n\*m). По принцип е .