**Fundaments of Algorithms – Practice 1: Efficiency of Algorithms**

**Names and surnames of the two members of the team**:

- Ramón Costales de Ledesma

- José Ignacio Daguerre Garrido

**Number of the team**:

Given a sequence of non-negative integer numbers X = (x1, …., xn), the “odd-counting” of this sequence is defined as the sequence Y = (y1, …., yn), where every element of Y is defined for each i-position as yi= (#: 0<= j <= i: odd(xi)), where # is the count operator (see chapter about formal specification), and odd(x) is true only if x is odd. In other words, each element counts the number of odd numbers up to the current position.

Examples:

|  |  |
| --- | --- |
| X | Odd-counting sequence of X |
| 7 | 1 |
| 2 7 20 | 0 1 1 |
| 7 4 6 5 2 21 31 | 1 1 1 2 2 3 4 |

The following function returns the “odd-counting” sequence of a given input sequence. Both sequences are represented as arrays, using a sentinel value of -1 to indicate the end of sequences. The output sequence is assumed to have enough room for the calculated sequence.

void calculateOddCounting**(**const int inputSequence**[],** int oddCounting**[])** **{**

int i **=** 0**;**

**while** **(**inputSequence**[**i**]** **!=** **-**1**)** **{**

oddCounting**[**i**]** **=** 0**;**

**for** **(**int j **=** 0**;** j **<=** i**;** j**++)** **{**

if((inputSequence**[**j**]%**2) ==1){

oddCounting**[**i**]** = oddCounting**[**i**]**+1**;**

**}**

i**++;**

**}**

oddCounting**[**i**]** **=** **-**1**;**

**}**

We consider the size of the problem **n** as the number of elements of the input sequence, excluding the sentinel value.

1. For the analysis of time difference, ¿is there any difference between the worst case and best case? Please, explain your answer.

|  |
| --- |
| There is no difference because the best and worst cases only differ in the if condition inside the for loop which doesn´t imply more iterations:  if ((inputSequence[j] % 2) == 1)  oddCounting[i] = oddCounting[i] + 1; |

1. How many iterations does the “*for”* loop perform to calculate the first element of the odd-sequence? And how many for the second element? And for the third? And for calculating the element in the i-position according to i?

|  |  |
| --- | --- |
| To calculate the first element | 1 |
| To calculate the 2nd element | 2 |
| To calculate the 3rd element | 3 |
| To calculate the i-position element | i + 1 |

1. Please, indicate the number of the following operations that the “***for****”* loop does according to “*i*” for calculating the **i-position** element of the odd-counting sequence in the worst-case scenario:

|  |  |
| --- | --- |
| assignments | i + 2 |
| comparisons | 2i + 1 |
| increments | 2i |
| Remainder operations | i + 1 |
| Retrieving elements from arrays | 2i + 2 |

1. How many times the “**while**” loop iterate given an input sequence of size **n**?

|  |
| --- |
| n |

1. Determine the number of the following operations performed by the “*calculateOddCounting*” function in the worst-case scenario:

|  |
| --- |
| Assignments |
| n·(n+1)/2 + 2n + 2 |
| Comparisons |
| 3n + 2 |
| Increments |
| 3n |
| Remainder |
| n + 1 |
| Retrieving elements from arrays |
| 4n + 3 |

1. Assuming the following execution time of each kind of instructions:

|  |  |
| --- | --- |
| assignments | tassign |
| comparisons | tcomp |
| increments | tinc |
| Remainder operations | trem |
| Retrieving elements from arrays | tretriev |

Determine the t(**n**) function that calculates the execution time of the “*calculateOddCounting*” function according to the **n** size problem in the worst case:

|  |
| --- |
| tassign·(n·(n+1)/2 + 2n + 2) + tcomp· (3n + 2) + tinc· (3n) + trem· (n + 1) + tretriev·(4n + 3) |

1. Indicate the exact order of complexity Θ (t(n)) for the function “t(n)” calculated in the previous section.

|  |
| --- |
| Θ (n^2) |

1. Write the algorithm in C++ programming language for a new version of the function “*calculateOddCounting”* that solves the same problem but in linear time Θ (n). Please, make sure that it works by executing test cases in a C++ environment, and include here only the code of the tested function:

|  |
| --- |
| void calculateOddCounting2(const int inputSequence[], int oddCounting[]) {  int i = 0;  while (inputSequence[i] != -1) {  oddCounting[i] = 0;  int j = 0;  while (j <= i) {  if ((inputSequence[j] % 2) == 1)  oddCounting[i] = oddCounting[i] + 1;  j++;  }  i++;  }  //oddCounting[i] = -1;  } |

1. Determine the time function t(**n**) according to the **n** size problem for your new algorithm:

|  |
| --- |
| void calculateOddCounting**(**const int inputSequence**[],** int sum)**{**  int i **=** 0**;**  **sum = 0;**  **while** **(**inputSequence**[**i**]** **!=** **-**1**)** **{**  if((inputSequence**[**i++**]%**2) == 1)  sum++**;**  **}**  **}** |

1. Prove that the execution time has an exact complexity order of linear, i.e. Θ(n):

|  |
| --- |
| There`s only a while loop, which implies the array is traversed only once. That means a linear complexity. |