David Gómez, Simón Correa

Taller 5

Calculo de Complejidades y toma de tiempos

1. Código

```
    int aux;
        int cont1;
        int cont2;
        for(cont1=1; cont1 <array.length; cont1++) {       //c_1
c_2             T(n) = c_1 + c_2*n
            aux = array[cont1];                                 //c_3*n

            for(cont2 = cont1-1;cont2>=0 && array[cont2] > aux;
cont2--) {  //c_4 = 2 c_5= 5    T(n) = (c_4 + c_5*(n-1))*n
                array[cont2+1] =
array[cont2];
                array[cont2] =
aux;                                                //c_6 =
6?    T(n) = c_6(n-1)n

            }
        }
        return
array;           //c_7                                        T(n)= c_1
+ c_2*n+c_3*n+(c_4 + c_5*(n-1))*n + c_6(n-1)(n) + c_7
```

Calculando la ecuación de la complejidad asintótica del código tenemos que:

T(n) = c_5 n^2 + c_6 n^2 + c_2 n + c_3 n + c_4 n - c_5 n - c_6 n + c_1 + c_7

O(n) = c_5 n^2 + c_6 n^2 + c_2 n + c_3 n + c_4 n - c_5 n - c_6 n + c_1 + c_7 (por def. de O)

O(n) = c_5 n^2 + c_6 n^2 + c_2 n + c_3 n + c_4 n - c_5 n - c_6 n (por regla de la suma)

O(n) = n^2(c_5 + c_6) + n(c_2 + c_3+ c_4 - c_5 - c_6)

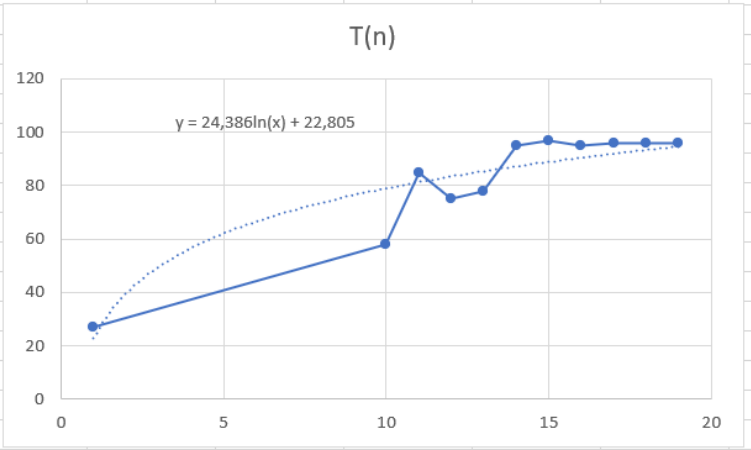
O(n) = n^2 + n (por regla de la multiplicación)

O(n) = n^2 (sacando el maximo de las 2)

Observando la complejidad hallada teóricamente por medio de la Big O, en contraste con la medición de tiempos experimentales, se pasa de una complejidad cuadrática a una complejidad con tendencia logarítmica. En particular, incluso tomando arreglos de grandes dimensiones, el

tiempo del algoritmo era optimo, por lo que consideramos que resulta conveniente incluso para ordenar grandes volúmenes de datos.

n*10^6	T(n)
1	27
10	58
11	85
12	75
13	78
14	95
15	97
16	95
17	96
18	96
19	96



2. Código:

```
int suma = 0;                                          //c_1
        for (int i = 0; i < array.length; i++)
{//c_2
T(n) = c_2 + c_3*n
            suma+= array[i];                    //c_4            T(n)
= c_4*n
        }

        return suma; //c_5    T(n) = c_1 + c_2 + (c_3 + c_4)*n +c_5
```

Calculando la ecuación de la complejidad asintótica del código tenemos que:

T(n) = c_1 + c_2 + (c_3 + c_4)*n + c_5

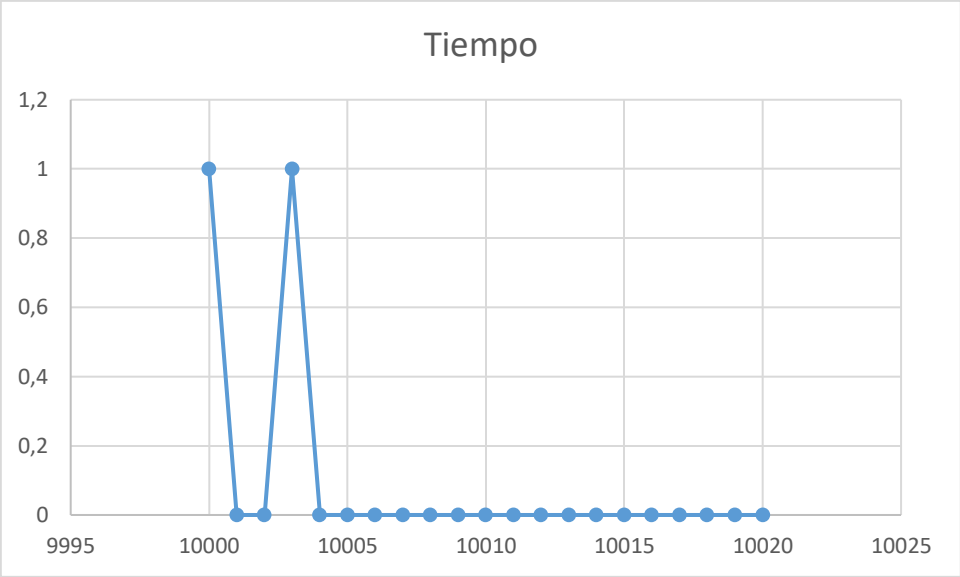
O (c_1 + c_2 + (c_3 + c_4)*n + c_5) (por definición de O)

O ((c_3 + c_4)*n) (por regla de la suma)

O (n) (por regla del producto)

Tenemos como resultado una complejidad lineal, es decir que el consumo de tiempo va aumentando de manera constante, esto dependiendo del tamaño del arreglo. Por otra parte, realizando la implementación del algoritmo de forma recursiva encontramos que la complejidad es la misma, (O(n)) por lo que independiente de cómo se implemente el algoritmo (con ciclos o

recursividad) es igualmente eficiente. Esto último se presenta en particular para este caso, dado que, incluso resolviéndolo con recursividad, no son necesarios dos llamados recursivos; en esos casos o en otros, la elección de uno u otro método puede variar según el caso.



Podemos observar que los tiempos que toma el algoritmo en sumar los elementos de un arreglo con tamaños entre 10000 y 10020 son 0 segundos en la mayoría de los casos, esto pues ya que el procesador alcanza a realizar muchas operaciones por segundo y la suma toma un tiempo constante.

Tamaño	Tiempo
10000	1
10001	0
10002	0
10003	1
10004	0
10005	0
10006	0
10007	0
10008	0
10009	0
10010	0
10011	0
10012	0
10013	0
10014	0
10015	0
10016	0
10017	0
10018	0
10019	0

10020 0