Santiago Ochoa Castaño -sochoac1@eafit.edu.co

Miguel Ángel Zapata Jiménez -mazapataj@eafit.edu.co

1. Determinar el máximo de los elementos de un arreglo

1.1 Codigo

```
public class maximun {
    public static int maxF(int index, int[] nums){
        if(nums.length==0){
            return Integer.MIN_VALUE;
        }else if(index==nums.length){
            return nums[index-1];
        }else{
            return Math.max(nums[index], maxF(index+1,nums));
        }
    }
```

1.2 Identificar quién es el tamaño el problema (llamado también "n")

El tamaño del problema son los elementos restantes del tamaño del arreglo en cada llamado recursivo.

1.3 Etiquetas cuantas operaciones ejecuta cada línea

```
public class maximun {
    public static int maxF(int index, int[] nums){
        if(nums.length==0){ //  C_1 = 2
            return Integer.MIN_VALUE; // C_2=1/ T(n)=C_1+C_2= 3
        }else if(index==nums.length){ // C_3 = 2
            return nums[index-1]; //  C_4 = 1 / T(n) = C_2 + C_4 = 3
        }else{
            return Math.max(nums[index], maxF(index+1,nums)); // T(n) = C_5 + T(n-1)
        }
    }
```

1.4 Escribir la ecuación de recurrencia

$$T(n) = \begin{cases} C_2, & if\ n = 0 \\ C_4, & if\ n = 1 \\ C_5 + T(n-1), & if\ n > 1 \end{cases}$$

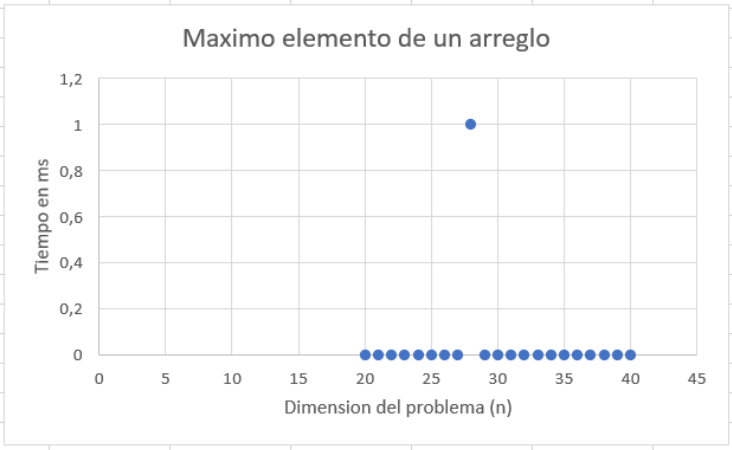
1.5 Resolver la ecuación con Wolfram Alpha

C_5 + T(n – 1)

= T(n) = C_5(n-1) + C_4

1.6 Grafica

n	f(n)
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	1
29	0
30	0
31	0
32	0
33	0
34	0
35	0
36	0
37	0
38	0
39	0
40	0



1.6 Explicar en palabras

La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace más operaciones) para el algoritmo de encontrar el maximo elemento dentro de un arreglo recursivamente es T(n)=C_5(n-1) +C_4

2. Suma para ver si es posible alcanzar un valor objetivo

2.1 Código

```
public static boolean SumaGrupo(int start, int[] nums, int target){

If(start>=nums.length){

 return target==0;

}

return SumaGrupo (start+1,nums,target-nums[start]) ||

Suma Grupo(start+1,nums,target);
```

2.2 Identificar quién es el tamaño del problema (llamado también “n”)

En número de instrucciones depende del tamaño del arreglo

2.3 Etiquetar cuántas operaciones ejecuta cada línea

```
public static boolean SumaGrupo(int start, int[] nums, int target){

If(start>=nums.length) return target==0; //  C_1 =  4

return SumaGrupo (start+1,nums,target-nums[start]) ||
```

Suma Grupo(start+1,nums,target); // C_2+2T(n-1)

2.4 Ecuación de complejidad

$$T(n) = \begin{cases} C_1, & if\ n \leq 1 \\ C_2 + 2T(n-1), & if\ n > 1 \end{cases}$$

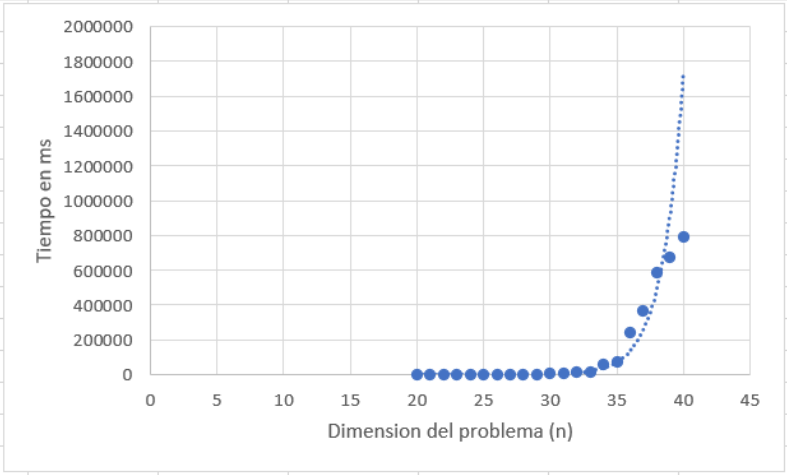
2.5 Solución de ecuación de complejidad

C_2 + 2*T(n-1)

= $T(n) = C_2\,(2^n - 1) + c_1\,2^{n-1}$

2.6 Grafica

n	f(n)
20	8
21	7
22	18
23	43
24	92
25	90
26	264
27	304
28	1018
29	1316
30	4926
31	4869
32	14981
33	18405
34	58806
35	74100
36	240446
37	368908
38	587940
39	678390
40	789099



2.7 Explicar en palabras

La complejidad asintótica (es decir, para valores grandes de n) para el peor de los casos (es decir, en el que el algoritmo hace más operaciones) donde se implementa un algoritmo para encontrar si existe un subgrupo de volúmenes cuya suma sea igual a un volumen máximo constante.

$T(n) = C_2\,(2^n - 1) + c_1\,2^{n-1}$