

## Ejercicio 2:

$O(n)$

```
public static int ArraymaxCalc(int[] nums, int index){  
    int max = nums[index]; // Const  
    if(index == 0){ //Const  
        max = nums[0]; //Const  
    }  
    if(index != 0){ //Const  
        int temporal = arrayMax.ArraymaxCalc(nums, index-1); //T(n-1)  
        if(temporal > max){ //Const  
            max = temporal; //Const  
        }  
    }  
    return max; //Const  
}
```

### 1.2 Identificar el tamaño del n:

El tamaño del arreglo es el número de elementos que me faltan por chequear no sea más grande que el anterior.

Handwritten notes on graph paper showing the derivation of linear complexity for a recursive array maximum function.

Recurrence relation:

$$F(n) = \begin{cases} C_1 & n = 0 \\ C_2 + F(n-1) & n \neq 0 \end{cases}$$

Time complexity derivation:

$$T(n) = C_2(n) + C_1$$
$$O(C_2(n) + C_1)$$
$$O(C_2(n)) \quad // \text{ se cancela porque la verdad no importa cuando } n \text{ es grande}$$
$$O(n) \quad // \text{ se ignora el coeficiente porque cuando } n \text{ es grande es muy poco importante}$$

↓  
Lineal

Conclusions:

- la complejidad tiene un crecimiento lineal.
- Aquí no hay un peor caso porque igual el algo siempre recorre el total de los elementos del Arreglo.
- El tamaño del problema o  $n$  es igual a el número de elementos en el Arreglo.

## Ejercicio 2

$O(2^n)$

```
16 public static boolean subVolumen(int[] objetos, int total, int index) {  
17     if (index >= objetos.length) { C17 = 2 Instrucciones = O(1)  
18         return total == 0; C18 = 2 Instrucciones = O(1)  
19     }  
20     return subVolumen(objetos, total - objetos[index], index + 1) || subVolumen(objetos, total, index + 1); C20 n*n instrucciones  
21 }  
22 }  
F(n) = F(n-1)*F(n-1) + 4
```

### 2.2 Identificar el tamaño del n:

El tamaño d n es el número de elementos en el arreglo.

$$F(n) = \begin{cases} C, & \text{si } n=1 \\ T(n-1) + T(n-1) & \text{si } n > 1 \end{cases}$$

$$T(n) = C \cdot 2^{n-1}$$

$$\begin{matrix} O(C \cdot 2^{n-1}) \\ O(C \cdot 2^n) \\ O(2^n) \end{matrix} \left. \vphantom{\begin{matrix} O(C \cdot 2^{n-1}) \\ O(C \cdot 2^n) \\ O(2^n) \end{matrix}} \right\} \begin{array}{l} \text{eliminamos esos terminos} \\ \text{porque son poco importantes} \\ \text{cuando } n \text{ es} \\ \text{muy grande} \end{array}$$

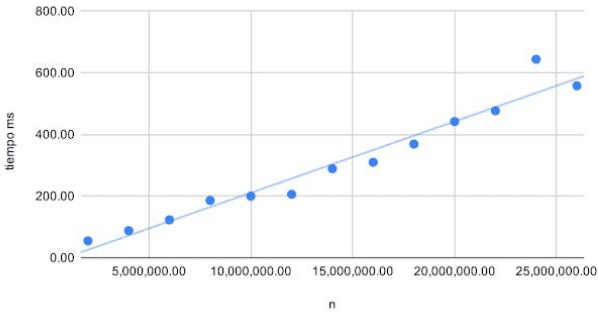
↓  
exponencial

La complejidad en el peor de los casos la complejidad es representada por  $O(2^n)$

Graficas

n	tiempo ms
2,000,000.00	55.00
4,000,000.00	88.00
6,000,000.00	123.00
8,000,000.00	186.00
10,000,000.00	200.00
12,000,000.00	206.00
14,000,000.00	289.00
16,000,000.00	310.00
18,000,000.00	369.00
20,000,000.00	442.00
22,000,000.00	477.00
24,000,000.00	644.00
26,000,000.00	558.00

Taller 4, ejercicio 1



n	ms
16	2
17	3
18	3
19	5
20	7
21	11
22	16
23	32
24	52
25	118
26	203
27	459
28	787
29	1906
30	3162
31	7421
32	12706
33	31420
34	50712
35	118352
36	221356

Taller 4, ejercicio 2

