Taller 4

Integrantes: Juan Camilo Guerrero Alarcón && Santiago Pulgarin Vasquez.

1. Suma de los elementos de un arreglo

1.1 Copiar el código en Word

```
private static int suma(int[] a, int i){
    if (i == a.length)
      return 0;
    else
      return a[i] + suma(a,i+1);
}
```

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

El tamaño del problema son los elementos que me falta por sumar en el arreglo “a”.

1.3 Etiquetar cuánto se demora cada línea

```
private static int suma(int[] a, int i){
    if (i == a.length)  // constante
      return 0;  // constante
    else
      return a[i] + suma(a,i+1); //constante + T(n-1)
}
```

1.4 Escribir la ecuación de recurrencia

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ c_2 + T(n-1) & \text{if } n > 1 \end{cases}$$

1.5 Resolver la ecuación con Wolfram Alpha

T(n) = c_2 + T(n-1)

T(n) = c_2*n + c_1

1.6 Aplicar la notación O a la solución de la ecuación

T(n) es O(c_2*n + c_1), por definición de O

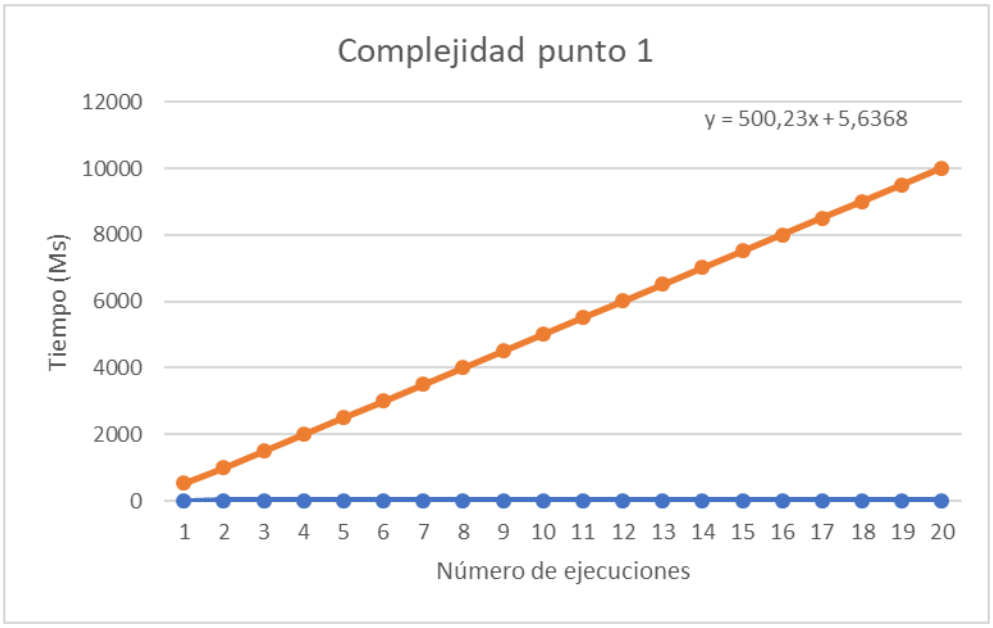
T(n) es O(c_2*n), por Regla de la Suma

T(n) es O(n), por Regla del Producto

1.7 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 unas más operaciones) para el algoritmo de sumar los elementos de un arreglo recursivamente es O(n).

1.8 Grafica de los datos



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

2.1 Copiar el Código en Word

```
public static boolean SumaSubgrupo (int start, int [] nums, int target) {
        if (start >= nums.length) {
            return target == 0;
        } else {
            return SumaSubgrupo(start + 1, nums, target - nums[start]) || SumaSubgrupo(start + 1, nums,
target);
        }
    }
```

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

El tamaño del problema es el número de elementos que contiene el arreglo “nums”.

2.3 Etiquetar cuánto se demora cada línea

```
public static boolean SumaSubgrupo (int start, int [] nums, int target) {
        if (start >= nums.length) {  // Constante
            return target == 0;   // Constante
        } else {
```

```
        return SumaSubgrupo(start + 1, nums, target - nums[start]) ||  // Constante + T(n-1)
SumaSubgrupo(start + 1, nums, target);  // Constante + T(n-1)
    }
  }
```

2.4 Escribir la ecuación de recurrencia

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ T(n-1) + T(n-1) + c_2 & \text{if } n > 0 \end{cases}$$

2.5 Resolver la ecuación con Wolfram Alpha

T(n) = T(n-1) + T(n-1) + c2

T(n) = 2*T(n-1) + c

2.6 Aplicar la notación O a la solución de la ecuación

T(n) es O(2*T(n-1) + c), por definición de O

T(n) es O(T^(n-1) + c), por Regla del producto

T(n) es O(T^(n-1)), por Regla de la suma

T(n) es O(2^n),

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 unas más operaciones) para el algoritmo de ver si es posible obtener un valor objetivo entre los elementos, al sumarlos, de un arreglo, recursivamente, es O(2^n).