Complejidad de los ejercicios del Taller 4

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.

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)

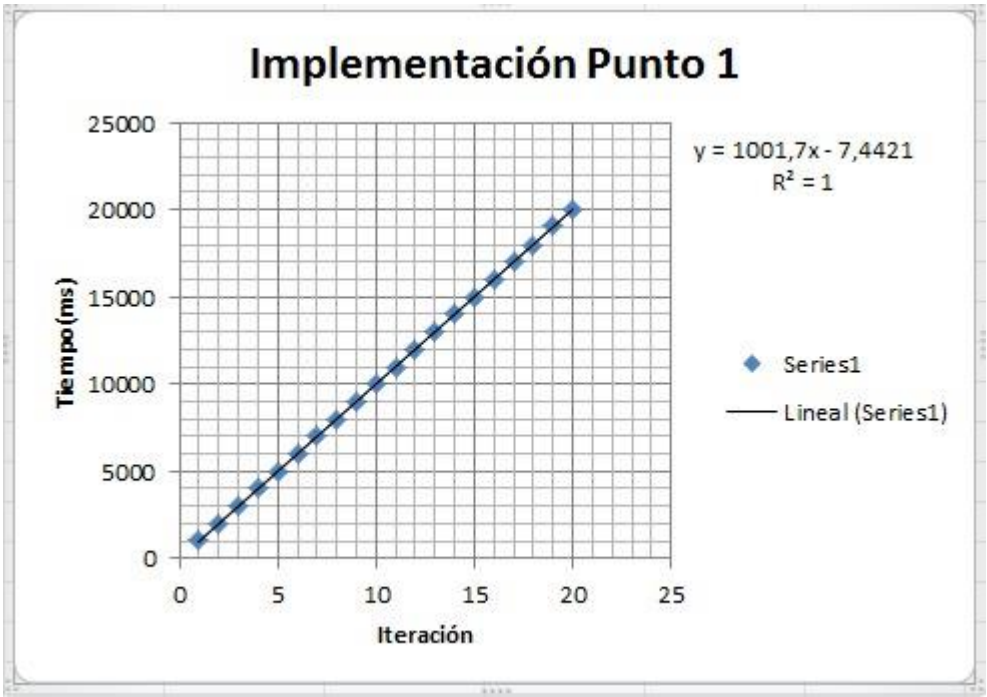
T(n) = c_2*n + c_1

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 equivalente a la encontrada de manera experimental en el algoritmo, los datos tomados se ajustaron perfectamente al modelo lineal encontrado por la notación O.

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

2.1 Copiar el código en Word

```
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”)

El tamaño del problema es el número de elementos del arreglo.

2.3 Etiquetar cuánto se demora cada línea

```
public static boolean SumaGrupo(int start, int[] nums, int target){
if(start>=nums.length) return target==0; //constante
return SumaGrupo (start+1,nums,target-nums[start]) || //constante + T(n-1)
        Suma Grupo(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

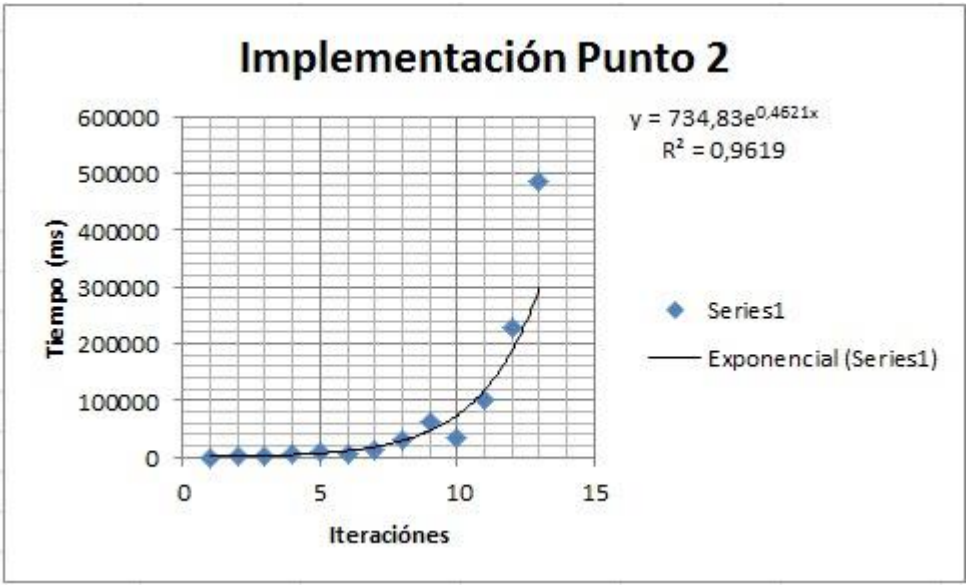
T(n) = T(n-1) + T(n-1) + c2

T(n) = 2 ^(n-1) + c2

2.6 Aplicar la notación O a la solución de la ecuación

T(n) es O(2^(n-1) +c2), por definición de O

T(n) es O(2^(n-1)), por Regla de la suma
T(n) es O(2^n), por Regla de la suma



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) 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), este a su vez muestra una relación con los datos experimentales y aunque, los datos no se inscriben por completo en la función, esta si muestra una tendencia de crecimientos exponencial.

*Solo tomamos doce valores experimentales pues el tiempo de ejecución del algoritmo no eran óptimos.

3. Sucesión de Fibonacci

3.1 Copiar el código en Word

```
public static long Fibonacci (long n) {
if (n<=1) return n;
return Fibonacci(n-1) + Fibonacci(n-2);
}
```

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

El tamaño del problema es el término enésimo.

3.3 Etiquetar cuánto se demora cada línea

```
public static long Fibonacci (long n) {
if (n<=1) return n; // constante
return Fibonacci(n-1) + Fibonacci(n-2); //T(n-1) + T(n-2)
}
```

3.4 Escribir la ecuación de recurrencia

$$T(n) = \begin{cases} c1 & if \quad n <= 1 \\ T(n-1) + T(n-2) & if \quad n > 1 \end{cases}$$

3.5 Resolver la ecuación con Wolfram Alpha

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

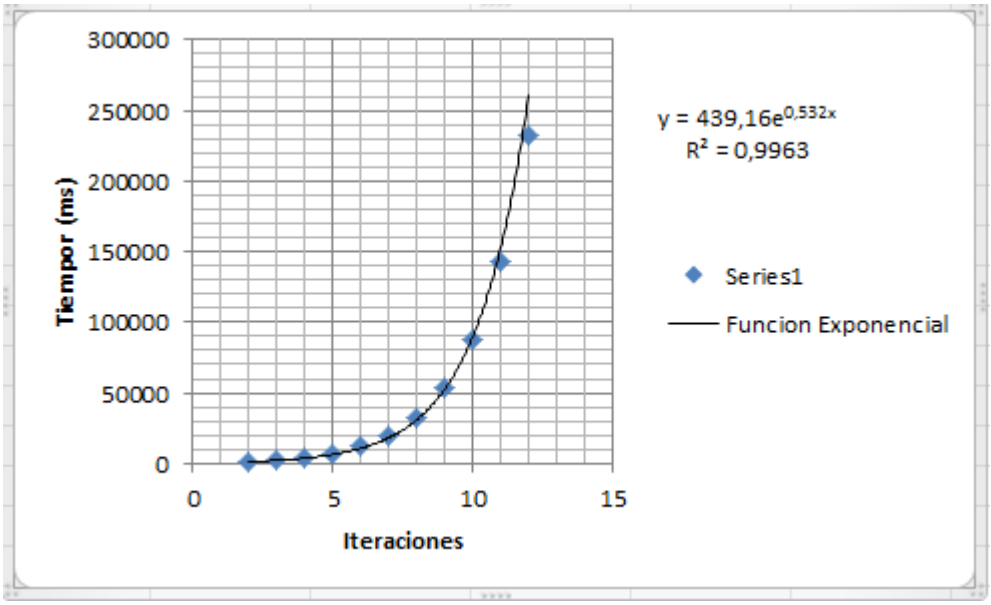
T(n) = c1*Fn + c2*Ln, donde Fn es el número enésimo de Fibonacci y Ln de la sucesión de Lucas

T(n) = c1*2^(n-2)

3.6 Aplicar la notación O a la solución de la ecuación

T(n) es O (c1*2 ^(n-2)), por definición de O

T(n) es O (2^(n-2)), por Regla del producto
T(n) es O (2^n), por Regla de la suma



3.7 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) para el algoritmo que calcula el enésimo número de Fibonacci recursivamente es O(2^n).
*Para este punto trabajamos con menos de las 20 muestras experimentales y además tuvimos que prescindir del primer datos puesto que este no era aceptado para la función exponencial, la cual se ajustaba muchos mejor que las demás.