Complejidad del Taller 4

1. Suma de los elementos de un arreglo

1.1 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 Tamaño del problema (llamado también “n”)

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

1.3 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 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 Ecuación resuelta con Wolfram Alpha

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

T(n) = c_2*n + c_1

1.6 Notación O de la solución de la ecuación

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

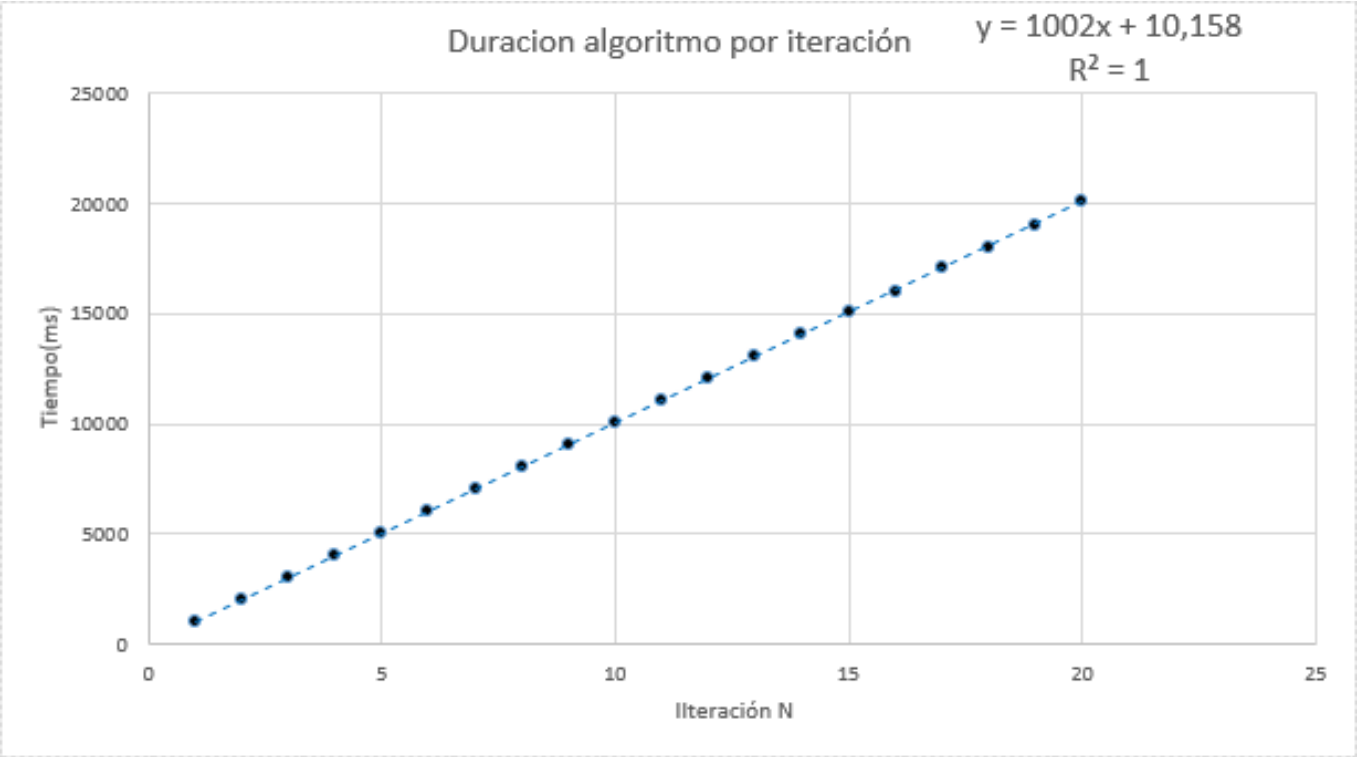
T(n) es O(c_2*n), por Regla de la Suma

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

1.7 Explicado 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 sumar los elementos de un arreglo recursivamente es O(n).

1.8 Gráfica del problema



1.9 Datos obtenidos

Iteración N	Tiempo (ms)	Resultado
1	1017	0
2	2004	1
3	3002	3
4	4018	6
5	5040	10
6	6011	15
7	7038	21
8	8043	28
9	9020	36
10	10013	45
11	11029	55
12	12049	66
13	13046	78
14	14038	91
15	15052	105
16	16026	120
17	17038	136
18	18022	153
19	19033	171
20	20084	190

1.10 Análisis de los resultados

De acuerdo a los resultados obtenidos en la gráfica se puede concluir que la notación asintótica se relaciona con la grafica, puesto que ambas tienen una ecuación lineal, la cual platea la complejidad del ejercicio. Por esto se puede observar que a la hora compilar un programa con altos indices de datos el tiempo variaría de manera constante.

2. GroupSum

2.1 Código en Word

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

2.2 Tamaño del problema (llamado también “n”)

El tamaño del problema son los elementos que me faltan por evaluar respecto al target.

2.3 Cuánto se demora cada línea

```
private static boolean sumaGrupo(int start, int[] nums, int target) {
        if (start >= nums.length) return target == 0 ; //constante
        else{
          return sumaGrupo(start+1, nums, target - nums[start])|| sumaGrupo(start+1, nums, target);
//T(n-1) + T(n-1)
        }
 }
```

2.4 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 Ecuación resuelta con Wolfram Alpha

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

T(n) = c_2*2^(n-1)

2.6 Notación O de la solución de la ecuación

T(n) es O(c_2*2^(n-1)), por definición de O

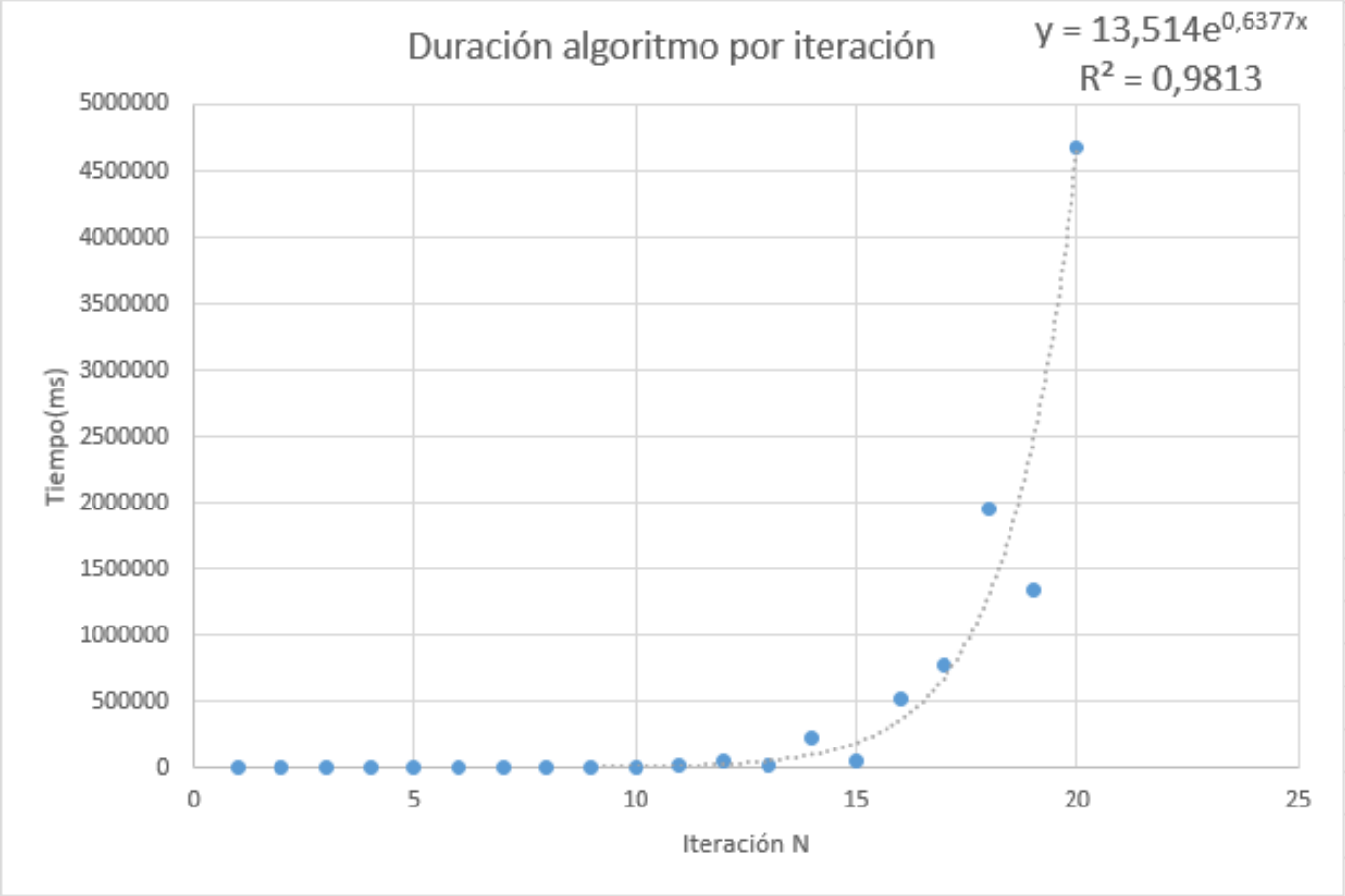
T(n) es O(2^(n-1)) , por Regla del Producto

T(n) es O(2^n), por Regla del Producto

2.7 Explicado en palabras

La complejidad asintótica para el peor de los casos del algoritmo de “hallar si hay un subgrupo de volumenes cuya suma sea igual al total” es O(2^n).

2.8 Gráfica del problema



2.9 Datos obtenidos

Iteración N	Tiempo (ms)	Resultado
1	16	false
2	47	false
3	109	false
4	235	false
5	203	true
6	939	true
7	777	true
8	3004	true
9	3430	true
10	11001	true
11	20040	true
12	58138	true
13	25952	true
14	225413	true
15	55303	true
16	517402	true
17	770638	true
18	1955076	true
19	1334106	true
20	4674936	true

2.10 Análisis de los resultados

De acuerdo a los resultados obtenidos en la gráfica se puede concluir que la notación asintótica se relaciona con la grafica puesto que ambas tienen una ecuación exponencial, la cual platea la complejidad del ejercicio. Por lo tanto, el proceso del logaritmo aumenta constantemente.

3. Fibonacci

3.1 Código en Word

```
public  int fibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

(Código completo en Git)

3.2 Tamaño del problema (llamado también “n”)

El tamaño del problema son los elementos que me faltan por sumar respecto a la iteración.

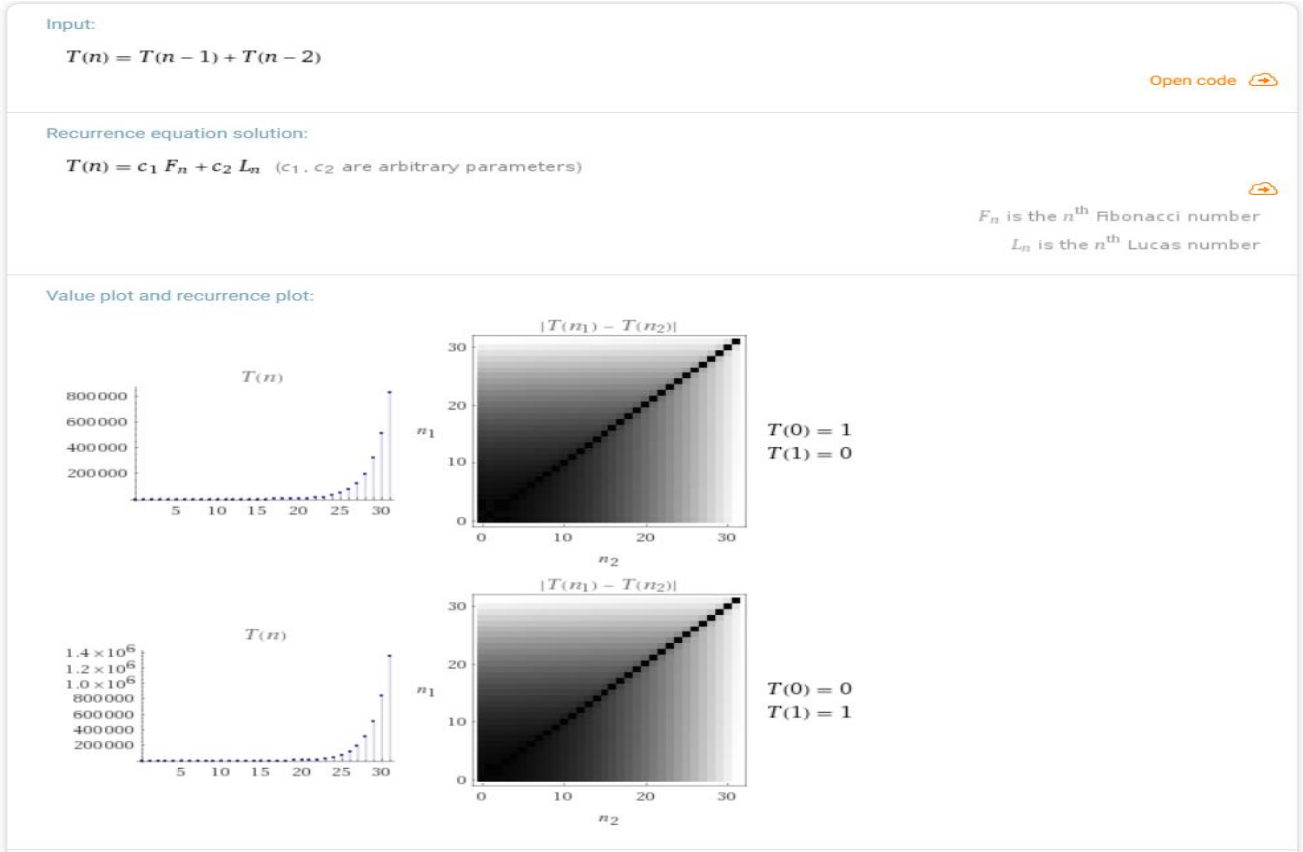
3.3 Cuánto se demora cada línea

```
public  int fibonacci(int n) {
       if (n == 0) { // constante
          return 0;
       } else if (n == 1) { //constante
          return 1;
       }
       return fibonacci(n - 1) + fibonacci(n - 2); //T(n-1) + T(n-2)
   }
```

3.4 Ecuación de recurrencia:

$$T(n) = \begin{cases} c_1 & \text{if } n \geq 0 \wedge n \leq 1 \\ T(n-1) + T(n-2) & \text{if } n > 1 \end{cases}$$

3.5 Resultado arrojado por Wolfram Alpha:

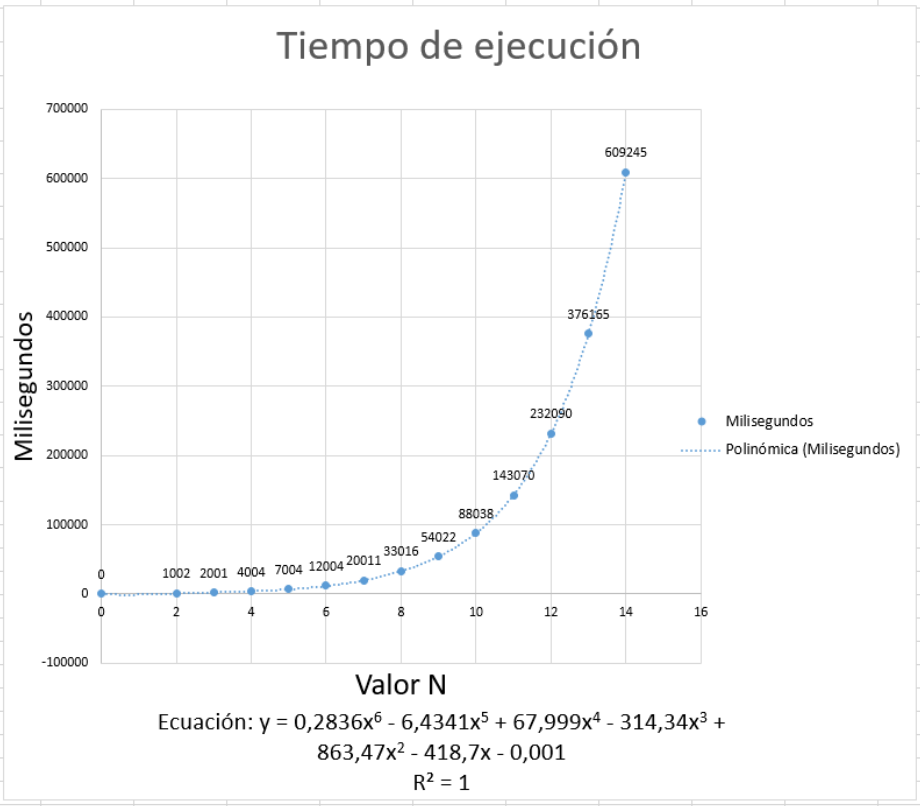


(Comentario: para este resultado del wólfram no tenemos muy claro cómo sacar la complejidad (aunque sabemos que es 2^n), por esto la dejaremos para preguntarla la próxima clase.)

3.6 Gráficas y análisis:

Gráfica del tiempo de ejecución:

Valor N	Milisegundos
0	0
2	1002
3	2001
4	4004
5	7004
6	12004
7	20011
8	33016
9	54022
10	88038
11	143070
12	232090
13	376165
14	609245



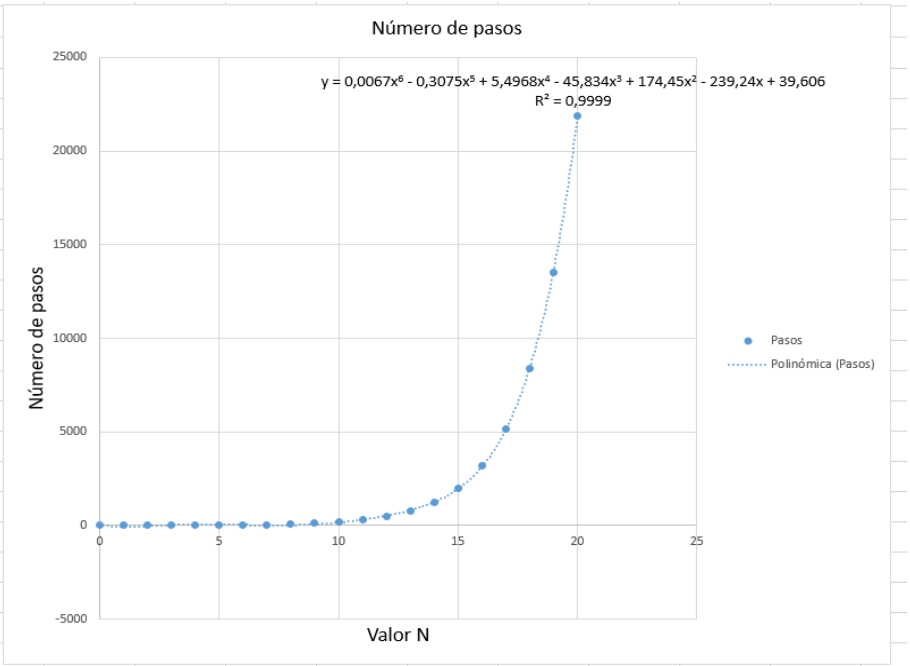
(Comentario: No se analizaron los 20 datos ya que el tiempo de ejecución se elevaba demasiado a partir de 14)

La gráfica anterior fue hecha en Excel, con los datos de la izquierda. La tendencia de los datos es claramente exponencial, pero Excel no permitió generar una ecuación exponencial para la tendencia de los puntos, por lo tanto se usó una ecuación polinómica de grado 6, la cual dio un coeficiente de determinación igual a 1, lo que significa que se acopla muy bien a los puntos de la gráfica. El uso de esta ecuación polinómica de grado 6 deja en evidencia la alta complejidad del algoritmo que, en teoría, es 2^n.

Gráfica del número de pasos o llamados recursivos:

Esta es otra gráfica que se puede analizar del algoritmo. Esta gráfica está directamente relacionada con la gráfica del tiempo de ejecución, ya que a más llamados recursivos, más tiempo de ejecución.

Valor N	Pasos
0	0
1	0
2	2
3	4
4	8
5	14
6	24
7	40
8	66
9	108
10	176
11	286
12	464
13	752
14	1218
15	1972
16	3192
17	5166
18	8360
19	13528
20	21890



Como se puede observar, la cantidad de llamados recursivos que debe hacer el algoritmo para calcular un n-ésimo término de la sucesión de Fibonacci, es también exponencial, pero de nuevo Excel no permitió acoplar una ecuación exponencial, por lo que se usó una ecuación polinómica de grado 6, dando un coeficiente de determinación de 0,999.