Laboratorio Nro. 1: Recursión

Alejandro Cano Munera
Universidad Eafit
Medellín, Colombia
acanom@eafit.edu.co

Jorge Luis Herrera Chamat
Universidad Eafit
Medellín, Colombia
jlherrerac@eafit.edu.co

2.3) GroupSum5: Este algoritmo funciona similar al algoritmo groupSum, cada vez que se ejecuta se cuenta un elemento menos del arreglo y se tiene en cuenta o no un elemento más de este, hasta que el parámetro start haya llegado al final del arreglo. En este algoritmo, cuando el parámetro start sea múltiplo de 5 y el siguiente elemento en el arreglo sea 1 se invoca la recursión saltando dos elementos del arreglo con el parámetro start+2, ya que no se tiene en cuenta este último, de lo contrario se invoca la recursión con start+1, este algoritmo retorna el valor booleano true si es posible formar con la suma de un grupo de algunos números del arreglo, incluyendo todos los múltiplos de 5, el parámetro target, de lo contrario se retornara false.

2.4)

2.1)

1. Array11: T(n) = T(n-1) + C es O(n)
2. BunnyEars2: T(n) = T(n-1) + C es O(n)
3. Count7: T(n) = T(n/10) + C es O(log(n))
4. PowerN: T(n) = T(n-1) + C es O(n)
5. SumDigits: T(n) = T(n/10) + C es O(log(n))

2.2)

1. SplitOdd10: T(n) = 2 * T(n-1) + C es O(2^n)
2. GroupSum6: T(n) = T(n-1) + T(n-1) + C es O(2^n)
3. Split53: T(n) = T(n-1) + T(n-1) + C es O(2^n)
4. GroupNoAdj: T(n) = T(n-1) + C es O(n)
5. GroupSumClump: T(n) = T(n-m) + T(n-1) + Cn + C es O(2^n)

2.5)

2.1)

1. Array11: n significa el número de elementos en el arreglo.

2. BunnyEars2: n significa el número de “Conejos” que debemos analizar, es decir, n significa el número de elementos.

3. Count7: n significa el digito que se va a evaluar.

4. PowerN: n significa el exponente de la expresión.

5. SumDigitis: n significa el digito que se va a evaluar.

2.2)

1. SplitOdd10: En este algoritmo, n es el número de elementos del arreglo, que se recorren con el parámetro i se le suma 1 en cada arreglo y se usa para crear la variable que se suma a cada grupo.

2. GroupSum6: En este algoritmo, n es el número de elementos del arreglo, que se recorren con el parámetro start al que se le suma 1 con cada llamado.

3. Split53: En este algoritmo, n es el número de elementos del arreglo, que se recorren con el parámetro index se le suma 1 en cada llamado y se usa para crear la variable que determina a cuál suma va cada elemento.

4. GroupNoAdj: En este algoritmo, n es el número de elementos del arreglo, que se recorren con el parámetro start se le suma 2 si se le resta el elemento de esa posición al número al que se quiere llegar, si no, se suma 1.

5. GroupSumClump: En este algoritmo, n es el número de elementos del arreglo, que se recorren con el parámetro start al que se le suma el número de veces que se repite el número o se le suma 1, m es el número de veces que se repite el número.

3.1) Para una mayor exactitud y un mejor manejo de los intervalos de tiempos, se midieron los procedimientos en nanosegundos.

ArrayMax							
N	1000	5000	10000	15000	20000	25000	30000
Tiempo	261269	62321	132175	204084	268803	330781	470490

*N, tamaño del arreglo.

Fibonacci								
N	1	10	15	20	25	35	40	45
Tiempo	285	1425	9119	38186	462231	60200686	551475422	6200943386

*N, enésimo termino en la serie Fibonacci.

GroupSum							
N	1	5	10	15	20	25	30
Tiempo	3082	5821	185251	6760465	8460940	212524753	12565983794

*N, tamaño del arreglo.

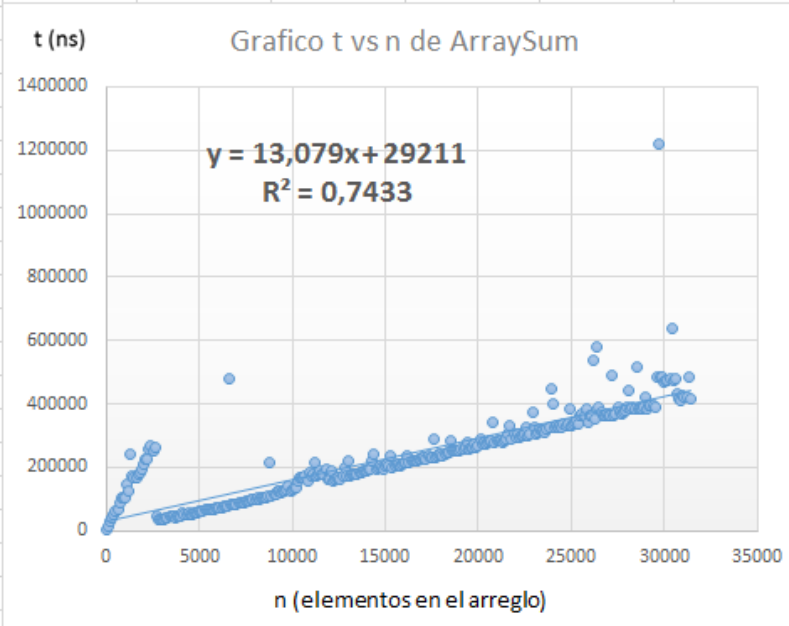
3.2) Graficas:

ArrayMax:

Laboratorio 1.

ArrayMax

Tabla de valores	
n	t (ns)
1	3424
1001	261269
3001	34928
5001	62321
6001	69512
9001	111973
10001	132175
12001	164021
15001	204084
18001	240723
20001	268803
21001	285239
24001	397896
25001	330781
27001	363311
30001	470490
31401	417071



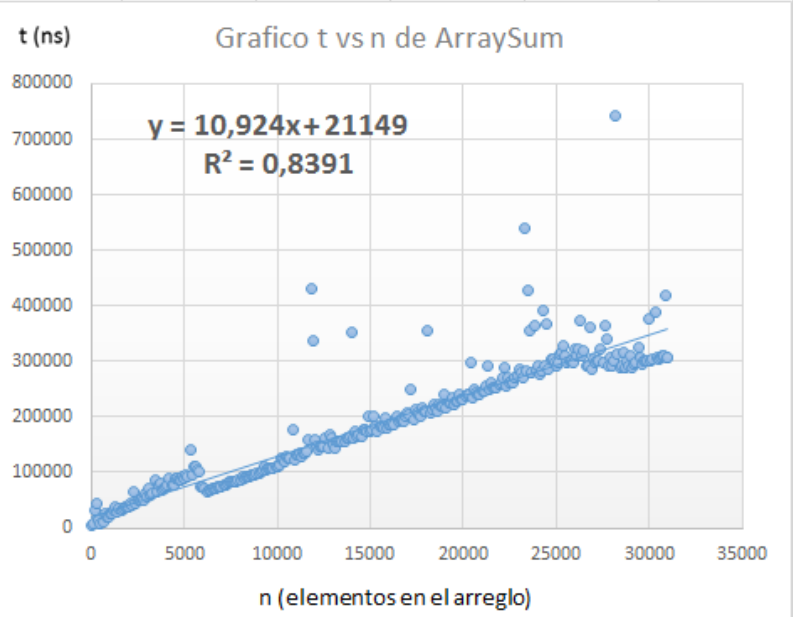
*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

ArraySum:

Laboratorio 1.

ArraySum

Tabla de valores	
n	t (ns)
1	4794
1001	24654
3001	54789
5001	88689
6001	72965
9001	99647
10001	110947
12001	157517
15001	172926
18001	18001
20001	230112
21001	246206
24001	293461
25001	291749
27001	304761
30001	377013
31001	307843



*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

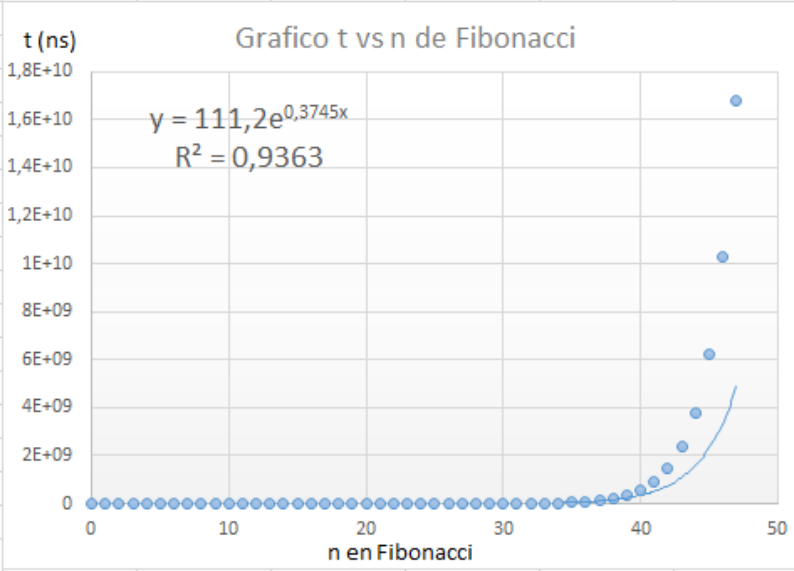
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Fibonacci:

Laboratorio 1.

Fibonacci

Tabla de valores	
n	t (ns)
1	285
5	855
10	1425
15	9119
20	38186
25	462231
30	4909573
35	60200686
40	551475422
45	6200943386



*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

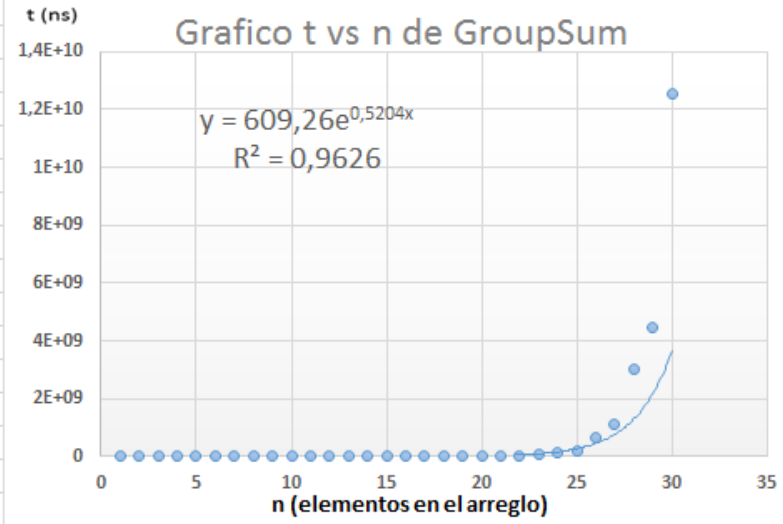
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

GroupSum:

Laboratorio 1.

GroupSum

Tabla de valores	
n	t (ns)
1	3082
5	5821
10	185251
15	6760465
20	8460940
25	212524753
30	12565983794



*Se tomaron mas valores para verificar la tendencia en la grafica Los mas representativos aparecen en la tabla

3.3) Lo que concluimos respecto a los tiempos obtenidos en el laboratorio y los resultados teóricos es que los resultados fueron iguales, pues en ambos casos las líneas de tendencia de los algoritmos, es decir, el comportamiento de estos respecto al tiempo (notación O) presentaron el mismo comportamiento.

3.4) Lo que aprendimos sobre Stack Overflow es que es un error muy común cuando se manejan algoritmos que realizan muchos llamados recursivos pues los datos allí utilizados ocupan todo el espacio en la pila del computador desbordándola. Para prevenir este error es necesario conocer la cantidad de datos con la que se va a trabajar y realizar modificaciones en nuestro código tales como ampliar la capacidad de la pila o limitar la cantidad con los que se trabajara.

3.5) El termino más grande que pudimos calcular para la serie de Fibonacci fue de 53, con un tiempo de ejecución de 5 minutos aproximadamente (solo para este término), este fue el valor máximo que logramos encontrar ya que el programa llevaba ya mucho tiempo en ejecución y decidimos detenerlo. Para saber cuál sería el fibonacci de 1 millón necesitaríamos conocer el valor de fibonacci de 999.999 y de 999.998, es decir, necesitaríamos conocer necesariamente todos los términos

fibonacci hasta llegar allí lo cual tardaría demasiado tiempo, además se nos podrían presentar otros problemas tales como el desbordamiento de la memoria o de la pila de nuestro computador.

3.6) Para calcular el fibonacci de valores grandes se podría utilizar la programación dinámica. Básicamente un término fibonacci se calcula conociendo sus dos antecesores en la serie, pero si ya conocemos algunos valores de esta serie calculados con anterioridad no necesitaremos empezar a calcular el fibonacci desde cero, sino que podríamos emplear estos valores para iniciar los cálculos varios pasos más adelante; lo que se necesita para llevar a cabo esto es guardar los términos que se calculan para utilizarlos posteriormente en nuevos cálculos.

3.7) Lo que concluimos sobre la complejidad entre los ejercicios de CodingBat recursión 1 y recursión 2, es que la complejidad de los ejercicios de recursión 2 es mucho mayor, puesto que proponen una lógica y un análisis algorítmico más elevado, en gran parte porque en estos ejercicios de recursión 2 hay que hacer dos llamados recursivos para lograr lo propuesto y en los ejercicios de recursión 1 solo uno.

4) Simulacro de Parcial

1. Start+1, nums, target
2. A. T(n) = T(n/2) + C
3.1 int res = solucionar(n-a, a,b,c) + 1;
3.2 res = Math.max(res,solucionar(n-b, a,b,c)+1);
3.3 res = Math.max(res,solucionar(n-c, a,b,c)+1);
4. E. La suma de los elementos del arreglo a y es O(n)

5. Lectura recomendada

a) Título: Notación asintótica.
b) Informe: La notación asintótica es una notación que nos permite representar y reflejar la complejidad de un algoritmo al ingresar como entrada el peor de los casos en los que puede funcionar el algoritmo, a esto se le llama tasa de cambio en tiempo de ejecución. La notación asintótica o notación Big-O, se representa siempre con la letra O (Mayúscula) y hay diferentes tipos de notación para los algoritmos Notación asintótica omega Esta notación se utiliza cuando se quiere representar una cantidad de tiempo mínima que tarda un algoritmo antes de dar una cota superior ya que acota el crecimiento del tiempo de ejecución por abajo para entradas de tamaños suficientemente grandes.

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co