

	<b>UNIVERSIDAD EAFIT</b> <b>ESCUELA DE INGENIERÍA</b> <b>DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS</b>	<b>Código: ST245</b>
		<b>Estructura de Datos 1</b>

### Laboratorio Nro. 1: Recursión

**Laura María Giraldo Castrillón**

Universidad Eafit  
Medellín, Colombia  
lgiraldo1@eafit.edu.co

**Andrés Felipe Oquendo Usma**

Universidad Eafit  
Medellín, Colombia  
afoquendou@eafit.edu.co

### 3) Simulacro de preguntas de sustentación de Proyectos

1. El algoritmo groupSum5 recibe como parámetros: start(índice que recorre el arreglo), target(Suma objetivo) y nums(arreglo de números enteros).

La instrucción de parada se da cuando el índice es mayor o igual a el tamaño del arreglo, y retorna verdadero si el target es igual a cero o falso si el target es diferente de cero, es decir, si se consiguió o no la suma objetivo, con algún subconjunto del arreglo (subconjunto que llamaremos conjunto solución o S).

El problema groupSum5 se diferencia del groupSum, ya que, en este, nos obligan a agregar todo múltiplo de cinco a el conjunto solución y, además, si el numero en la siguiente posición del arreglo a este múltiplo es 1, entonces no se tendrá en cuenta como parte de dicho conjunto. Esto define un condicional de la siguiente manera: Si es múltiplo de cinco compruebe si el arreglo tiene mínimo dos números más con respecto a la posición de start(para prevenir un ArrayIndexOutOfBoundsException) y, además, si el numero a continuación de la posición start es uno entonces llama recursivamente el método groupSum5 en la posición start mas 2(ya que el número a continuación es uno y debe omitirse), con el target menos el número múltiplo de 5(ya que es obligatorio añadirlo a S) y el arreglo de números enteros. Si no se cumple lo anterior, entonces, llama recursivamente el método groupSum5 con start en la posición inmediatamente siguiente (ya que no es uno), el target menos el número múltiplo de 5 y el arreglo de números enteros.

Si el número no es múltiplo de cinco, se llama recursivamente el método groupSum5 con start más uno (en la siguiente posición del arreglo) y target

**DOCENTE MAURICIO TORO BERMÚDEZ**

**Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627**

**Correo: mtorobe@eafit.edu.co**

menos el número, si al hacer ese llamado retorna falso, es que no se puede lograr la suma objetivo con ese número perteneciendo a el conjunto solución, por lo que entra en funcionamiento la disyunción y se llama de nuevo a groupSum5, pero sin incluir el número en la solución del problema.

## 2. Recursión 1:

**Factorial:**

$$T(n) = T(n - 1) + c$$

$$T(n) = O(n)$$

**bunnyEars:**

$$T(n) = T(n - 1) + c$$

$$T(n) = O(n)$$

**Fibonacci:**

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

$$T(n) = O(2^n)$$

**bunnyEars2:**

$$T(n) = T(n - 1) + c$$

$$T(n) = O(n)$$

**sumDigits:**

$$T(n) = 2T(n/10)$$

$$T(n) = O(n)$$

**Recursión 2:**

**groupSum6:**

$$T(n) = T(n - 1) + T(n - 1) + c$$

$$T(n) = O(2^n)$$

**groupNoAdj:**

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

**Resultado de wólffram:**

**Ln = enésimo número de Lucas.**

$F_n$  = enésimo número de Fibonacci.

$$T(n) = c_1 F_n + c_2 L_n + c$$

$$T(n) = O(2^n)$$

groupSum5:

$$T(n) = T(n-1) + T(n-1) + c$$

$$T(n) = O(2^n)$$

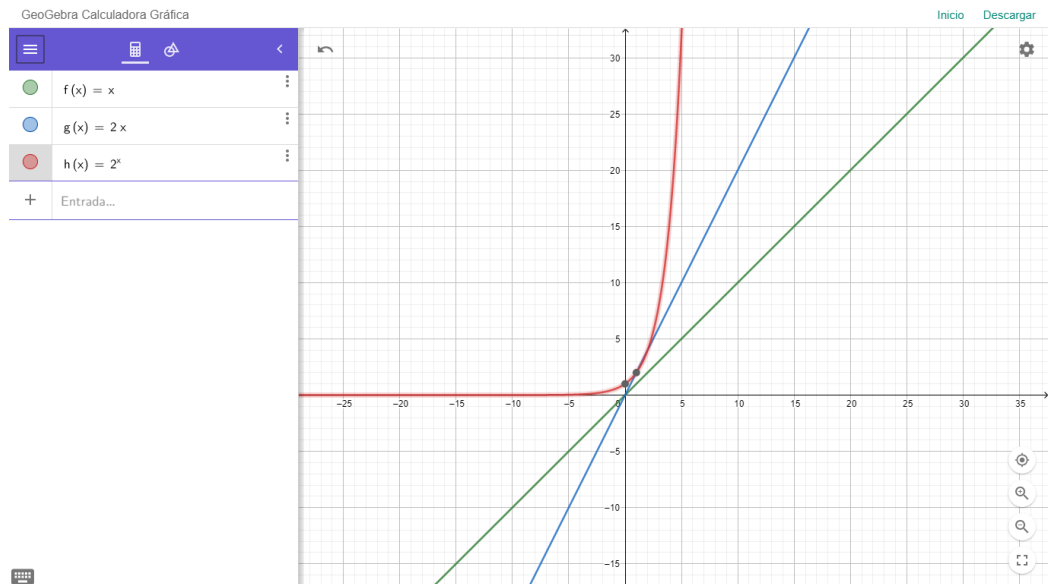
splitArray:

$$T(n) = O(2^n)$$

groupSumClump:

$$T(n) = O(2^n)$$

3. 3.3 En el cálculo de las complejidades anteriores,  $n$  es la variable que rige cada uno de los algoritmos y el tiempo que conlleva realizarlos.
- 3.4 Stack Overflow sucede cuando almacenamos variables u objetos en una pila (una lista de datos almacenados) y al hacer un proceso ésta se agota, comúnmente sucede en una recursión con llamado infinito, por lo tanto, la solución a esto es verificar si es correcta la condición de parada, en caso de que el llamado recursivo este correcto se puede ampliar el tamaño de la pila.
- 3.5 Una de las razones por las cuales Fibonacci no funciona para números grandes es porque el computador trabaja con bits y llega un momento en el que al sumarle un número más este se empieza a tornar negativo, para esto lo ideal es utilizar variables de tipo long. En este caso el algoritmo resulta ser más efectivo y rápido al realizarlo con ciclos, ya que internamente lo que hace el llamado recursivo es buscar el Fibonacci del número inmediatamente anterior y el de dos puestos antes de él y esto pasa sucesivamente hasta llegar a 0 o 1 por lo tanto el proceso se hace más extenso y lento. Luego de  $n=50$  el rango de ejecución del programa para dar la respuesta es entre un minuto o más.
- 3.6 En recursión 1 la complejidad era de  $n$  o  $2n$  para recursión 2 todas son  $2^n$  esto gráficamente se vería así:



Esto significa que los problemas en recursion 2 tardaran mas tiempo en ejecutarse.

#### **4) Simulacro de Parcial**

1.  $start+1$ ,  $nums$ ,  $target$

2.  $b$

3.

Línea 4.  $(n-1, a, b, c)$

Línea 5.  $(a, b)$

Línea 6.  $(res, c)$

4.: $c$

5.1:

Línea 2. Return  $n$ ;

Línea 3. Return  $formas(n-1)$

Línea 4. Return  $formas(n-2)$

5.2:

b)  $T(n)=T(n-1) + T(n-2) + c$

**6.**

**6.1** Completen la línea 10. sumaAux (n, i+2);

**6.2** Completen la línea 12. sumaAux (n, i+1);

**7.** Complete Línea 9 S, i+1, t-S[i];

Complete Línea 10 S, i+1, t;

**8.**

**8.1.** Complete Línea 9 return 0;

**8.2.** Complete Línea 13 return ni+nj;