Laboratorio Nro. 1: Recursión

Alejandro Murillo González
Universidad Eafit
Medellín, Colombia
amurillog@eafit.edu.co

Juan Pablo Vidal Correa
Universidad Eafit
Medellín, Colombia
Jpvidalc@ eafit.edu.co

2) *Ejercicios en línea sin documentación HTML en GitHub:*

3. Expliquen con sus propias palabras cómo funciona el ejercicio *GroupSum5:*

En el ejercicio groupSum5 al obtener el arreglo se debe verificar si en este se encuentra algún múltiplo de 5, y acto seguido se verifica si este múltiplo de 5 es un 1, para que se ignore al momento de calcular la suma del target pedido, esto por medio de if anidados que harían las veces de ciclos anidados, porque hacen llamadas recursiva dentro de otra llama recursiva; después teniendo en cuenta la condición de los múltiplos se suman los números del arreglo para obtener el target , esto sumando los números recursivamente hasta que no queden números, evaluando si la suma puede dar con todos los números, o descartando a los números para que se pueda obtener la suma, la cual de no ser el caso retorna un "false".

4. Calculen la complejidad de los Ejercicios en Línea de los numerales 2.1 y 2.2:

Recursión 1:

```
//BunnyEars2:

public int bunnyEars2(int bunnies){
    if (bunnies == 0) {  //C1
      return 0; //C2
   }
   else {
      return bunnies + bunnies + (bunnies / 2); // C3 + T(n/2)
   }
}
```

Complejidad:
T(n) = C1 + C2, si n == 0
T(n) = C3 + T(n/2), de lo contrario

$$T(n) = c_1 + \frac{C\log(n)}{\log(2)}$$ (c_1 is an arbitrary parameter)

T(n) = O(log(n)/log(2)) == T(n) es O(log(n))

//sumDigits:

```
public int sumDigits(int n){
   if (n == 0){  //C1
      return 0;
   }
   else{
      return n % 10 + sumDigits(n / 10); //C2 + T(n/10)
   }
}
```

Complejidad:
T(n) = C1 , si n == 0
T(n) = C2 + T(n/10), de lo contrario

$$T(n) = c_1 + \frac{C\log(n)}{\log(10)}$$ (c_1 is an arbitrary parameter)

T(n) = O(log(n)/log(10)) == T(n) es O(log(n))

//powerN :

```
public int powerN(int base, int n){
   if ( n == 1){  //C1
      return base;
   }
   else if ( n == 0){ //C2
      return 1;
   }
   else{
      return base * powerN( base, n - 1);  //C3 + T(n-1)
   }
}
```

Complejidad:

T(n) = C1, si n == 1
T(n) = C2, si n == 0
T(n) = C3 + T(n-1), de lo contrario

$T(n) = c_1 + C\,n$ (c_1 is an arbitrary parameter)

T(n) = O(C*n) == T(n) es O(n)

//array6:

```
public boolean array6(int[] nums, int index){
    if (index >= nums.length){  //C1
       return false;
   }
    else if (nums[index] == 6){  //C2
       return true;
   }
    else{
       return array6(nums, index + 1); //C3 + T(n+1)
   }
}
```

Complejidad:
T(n) = C1, si n >= m.length
T(n) = C2, si m == 0
T(n) = C3 + T(n-1), de lo contrario

$T(n) = c_1 - C\,n$ (c_1 is an arbitrary parameter)

T(n) = O(C*n) == T(n) es O(n)

//countPairs :

```
public int countPairs(String str) {
   int contador = 0; // C1
   if(str.length() < 3) { //C2
       return 0;
   }
   else if (str.charAt(0) == str.charAt(2)) { //C3
       contador += 1;
   }
   return contador + countPairs(str.substring(1)); //C4 + T(n+1)
}
```

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

Complejidad:
T(n) = C1, si m = 0
T(n) = C2, si n.lenght < 3
T(n) = C3, si nr.charAt(0) == nr.charAt(2)
T(n) = C4 + T(n+1), de lo contrario

$T(n) = c_1 - C\,n$ (c_1 is an arbitrary parameter)

T(n) = O(C*n) == T(n) es O(n)

Recursion 2:

//groupSum6:

```
public boolean groupSum6(int start, int[] nums, int target){
   if(start >= nums.length){ //C1
      return target == 0;
   }
   else if(nums[start] == 6) {
      return groupSum6(start + 1, nums, target - nums[start]); //C2 + T(n-1)
   }
   else if(groupSum6(start + 1, nums, target - nums[start])){ //C3 + T(n-1)
      return true;
   }
   else{
      return groupSum6(start + 1, nums, target); //C4
   }
}
```

Complejidad:
T(n) = C1 , si m >= y.length
T(n) = C2 + T(n-1), si y.lenght == 6
T(n) = C3 + T(n-1)
T(n) = C4, de lo contrario

$T(n) = c_1\,2^{n-1} + C\left(2^n - 1\right)$ (c_1 is an arbitrary parameter)

T(n) = O(C*2^n - C) == T(n) es O(2^n)

//groupNoAdj:

```
public boolean groupNoAdj(int start, int[] nums, int target){
   if (start >= nums.length){   //C1
```

```
    return (target == 0);
  }
  else if (groupNoAdj(start + 2, nums, target - nums[start])){  // C2 + T(n-1)
    return true;
  }
  return(groupNoAdj(start + 1, nums, target)); //C3
}
```

Complejidad:
T(n) = C1, si m >= y.length
T(n) = C2+ T(n-1)
T(n) = C3, de lo contrario

$T(n) = c_1 + C\,n$ (c_1 is an arbitrary parameter)

T(n) = O(C*n) == T(n) es O(n)

//groupSum5:

```
public boolean groupSum5(int start, int[] nums, int target){
  if (start >= nums.length){ //C1
    return (target == 0);
  }
  else if(nums[start] % 5 == 0){ //C2
     if(start < nums.length - 1 && nums[start+1] == 1){ //C3
        return groupSum5(start + 2, nums, target - nums[start]);
     }
     return groupSum5(start + 1, nums, target - nums[start]); //C4  + T(n-1)
  }
  else if(groupSum5(start + 1, nums, target - nums[start])){ //C5  + T(n-1)
    return true;
  }
  return groupSum5(start + 1, nums, target); //C6
}
```

Complejidad:
T(n) = C1, si m >= y.length
T(n) = C2+ T(n-1), si y % 5 == 0
T(n) = C3 + T(n-1) , si m < y.length -1 && y [m+1]== 1
T(n) = C4 + T(n-1), de lo contrario
T(n) = C5 + T(n-1)
T(n) = C5 + T(n), de lo contrario (linea 5)

$T(n) = c_1\,2^{n-1} + C\,(2^n - 1)$ (c_1 is an arbitrary parameter)

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

T(n) = O(C*2^n - C) == T(n) es O(2^n)

//groupSumClump (con un ciclo):

```
public boolean groupSumClump(int start, int[] nums, int target){
   if(start>=nums.length){ //C1
      return target==0;
   }
   int i = start; //C2
   for(i = start;  i < nums.length && nums[start] == nums[i]; i++){  //C3 * m
      if(groupSumClump(i, nums, target - ((i - start) * nums[start]))){ // C4  + T(n-1)*m
         return true;
      }
   }
    return groupSumClump(i, nums, target);   //C5
}
```

Complejidad:
T(n) = C1, si y >= m.length
T(n) = C2
T(n) = C3 *m
T(n) = C4 + T(n-1)*m
T(n) = C5 , de lo contrario (línea 2)

$$T(n) = c_1\, m^{n-1} + \frac{C\,(m^n - 1)}{m - 1}$$ (c_1 is an arbitrary parameter)

T(n) = O(C*m^n - C) == T(n) es O(m^n)

//splitArray :

```
public boolean splitArray(int[] nums){
  return split( nums, 0, 0);    // C1
}
public boolean split( int[] nums, int side1, int side2){
   if(side1 == nums.length){  //C2
      return (side2 == 0);
   }
   if(split(nums, side1 + 1, side2 + nums[side1])){  //C3 + T(n+1)
      return true;
   }
   return split(nums, side1 + 1, side2 - nums[side1]); //C4  + T(n-1)
}
```

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

Complejidad:
T(n) = C1
T(n) = C2, si y = m.lenght
T(n) = C3 + T(n+1)
T(n) = C4 + T(n-1)

$T(n) = c_1 - C\,n$ (c_1 is an arbitrary parameter)

T(n) = O(C*n) == T(n) es O(n)

5. Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del ejercicio 2.4:

Las variables "n" , "m" y/o "y" son las variables de entradas de los ejercicios en línea de codingBat, tomando un ejemplo, en el problema powerN las variables de entrada son (int base, int n), las cuales int base pasa a ser "m" y n es la misma variable, esto con el objetivo de facilitar los cálculos de complejidad.

3) Simulacro de preguntas de sustentación de Proyectos

1. De acuerdo a lo realizado en el numeral 1.1, completen la siguiente tabla con tiempos en milisegundos:

Nota: En python el límite de recursión está fijado en 999, pero con la función sys.setrecursionlimit() ,este límite se puede expandir hasta donde el computador en donde se ejecuta lo permita. En el caso de Fibonacci como el máximo al que se llego fue 997 se trabajó con rangos distintos. También se trabajaron valores diferentes para group sum porque es un algoritmo de complejidad O (2^n).

	N = 100.000	N = 1'000.000	N = 10'000.000	N = 100'000.000
ArrayMax	0.44141475458 ms	5.54782934483473 ms	81.18749375834 ms	952.284743937 ms

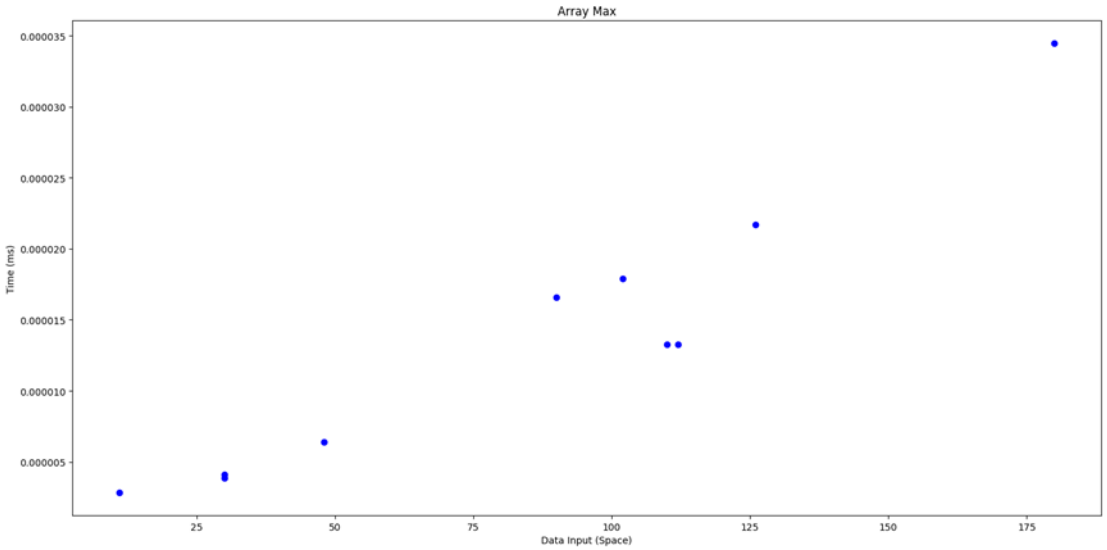
	N= 300	N= 500	N= 700	N= 997
GroupSum	0.0036997351090835018 ms	0.001108821930353443 ms	0.0047523494761454455 ms	0.006160859914400962 ms
Fibonacci	0.0003359679351184752 ms	0.004003000461923811 ms	0.0007853729525120201 ms	0.008067062776694334 ms

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

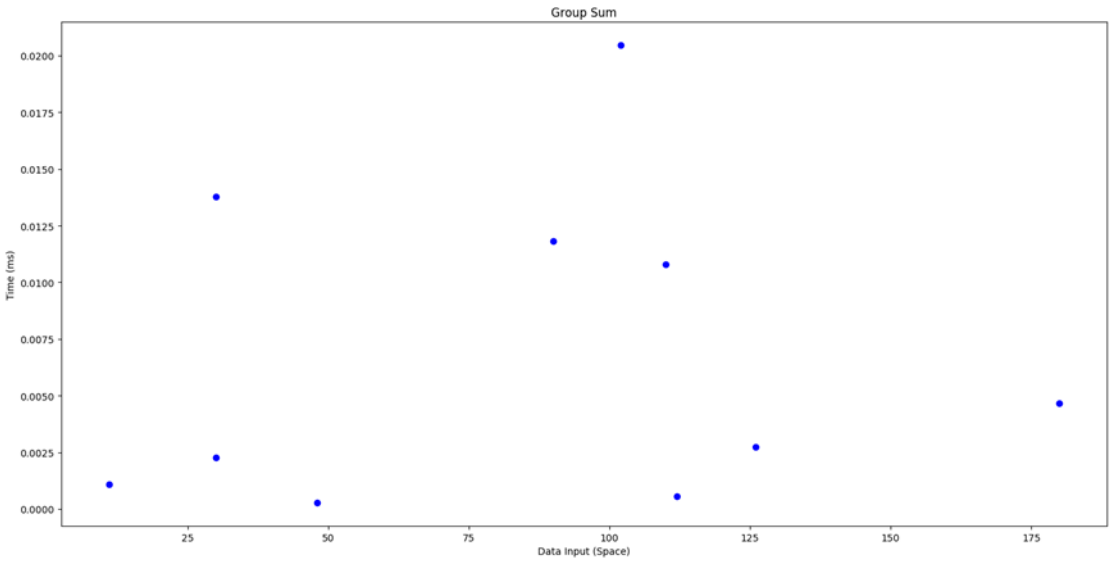
2. Grafiquen los tiempos que tomó en ejecutarse Array Sum, Array Maximum y Fibonacci recursivo, para entradas de diferentes tamaños:

Se trabajaron con los siguientes valores (n): 11, 30, 30, 48, 90,102, 126, 112, 180,110.

ArrayMax:

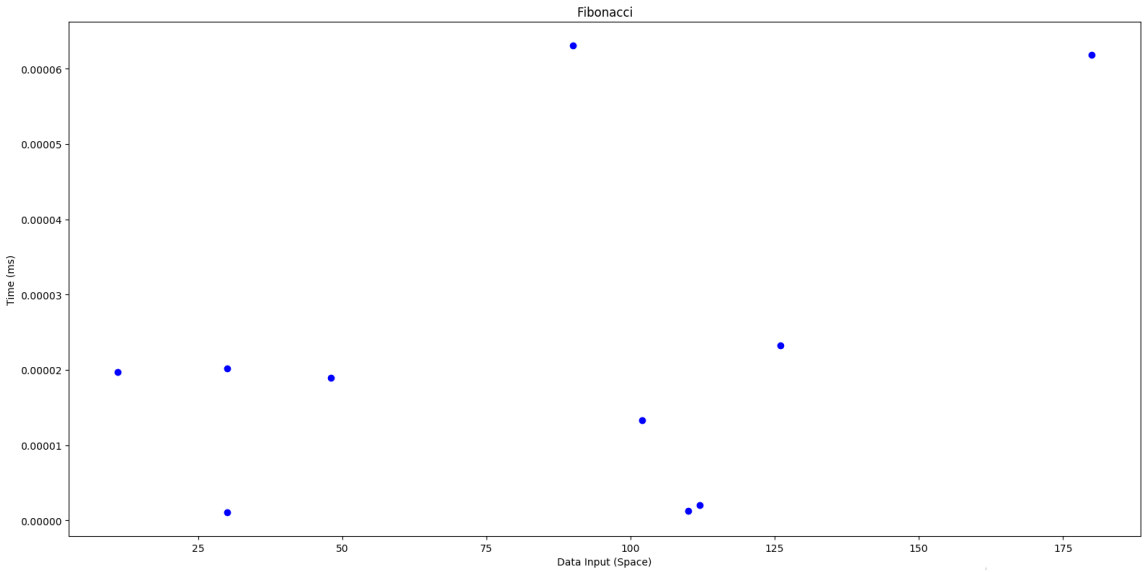


groupSum:



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

Fibonacci:



3. ¿Qué concluyen respecto a los tiempos obtenidos en el laboratorio y los resultados teóricos?

En la práctica, las gráficas indican que arrayMax es una función lineal, lo que es correcto porque su complejidad es O(n), ya que solo hace un llamado; por otro lado, la gráfica de GroupSum nos dio algo dispersa por que los valores eran aleatorios (véase punto 3.2) pero se puede identificar una función exponencial, lo que es válido ya que su complejidad es O(2^n), el caso anterior es muy similar a lo que ocurrió con Fibonacci, ya que ambos tienen una complejidad de O(2^n), esto se puede evidenciar al ver la similitud de las gráficas.

4. ¿Qué aprendieron sobre *Stack Overflow*?

El stack overflow es un problema aritmético que hace referencia al exceso de flujo de datos almacenados en el stack de una función, lo cual impide que se generen más subtrinas anidadas, debido a que se ha copado el espacio en el stack por el exceso de referencias, ya que cada subtrina trae su referencia y la de la que la llamo. Stack es la ubicación donde se almacena esta información.

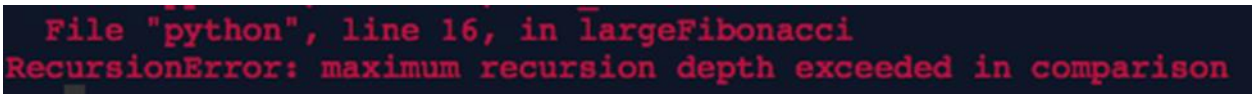
5. **¿Cuál es el valor más grande qué pudo calcular para Fibonnacci? ¿Por qué? ¿Por qué no se puede ejecutar Fibonacci con 1 millón?**

El valor más grande que se pudo computar recursivamente de la serie Fibonacci fue el correspondiente a la posición 997, cuyo valor es:

1026106236203326233660492672924522213266855812060212427776462290569940798254671148827285946888745795908773311924256407785074365766118082732679853917775891982813511440749936979646564952426675539110499009912037 7

Para todos los valores de n entre 1 y 997, logramos obtener la respuesta en menos de un segundo.

Al momento de ejecutar valores tales como Fibonacci de un millón, obtenemos un error debido a que se sobrepasa un límite de 995 llamadas recursivas del compilador.



6. **¿Cómo se puede hacer para calcular el Fibonacci de valores grandes?**

Para calcular el valor de la serie de Fibonacci más rápidamente, nos aproximamos al problema con una perspectiva de reutilización/economía de trabajo. Para nuestro caso, trabajo se refiere a tareas/computaciones que son llevadas a cabo por el computador en tiempo de ejecución.

Por lo anterior, decidimos crear una clase (usando Python) llamada Fibonacci, esta cuenta con un atributo privado llamado _knownAnswers. Este último es un diccionario que almacena respuestas para valores de la serie de Fibonacci calculados previamente. Mediante el uso de _knownAnswers, seguimos utilizando recursión, pero podemos hacer que esta llegue a un caso base más rápidamente dado que no necesita recalcular para todos los valores cada vez que se solicite una respuesta.

Esta aproximación nos da respuestas para valores de n entre 1 y 2150 en menos de 1 segundo; pero el tiempo crece exponencialmente, dado que para

calcular Fibonacci de 2500, el tiempo ronda los 5.2 segundos y para Fibonacci de 3000 el tiempo ronda los 49 segundos.

Para valores aún más grandes de 499, el algoritmo calcula primero (ascendentemente) el Fibonacci de valores múltiplos de 150 y n, de manera que se pueda contrarrestar el error: RecursionError: maximum recursion depth exceeded in comparison.

7. **¿Qué concluyen sobre la complejidad de los problemas de CodingBat Recursión 1 con respecto a los de Recursión 2?:**

   La complejidad de los problemas de recursión 1 se mantenía entre O(log(n)) y O(n), debido a que solamente se generaba un solo llamado recursivo, mientras que en los ejercicios de recursión 2, la complejidad era mucho mayor en la mayoría de ejercicios debido a que se tenían que hacer 2 llamados recursivos porque eran problemas que involucraban buscar en arreglos, esto se ve reflejado al tener una complejidad en su mayoría de O(2^n), por lo que teniendo en cuenta estos resultados a menor número de llamadas recursivas, menor complejidad, por lo que los cálculos serán mucho más rápidos.

4) *Simulacro de Parcial*

1. start+1, nums, target
2. a). T(n) = T(n/2) + C
3.
   3.1 (n - Math.min(a, b, c), a, b, c)
   3.2 (res, solucionar(n - Math.min(a, b, c), a, b, c))
   3.3 (res, solucionar(n - Math.min(a, b, c), a, b, c))

   ```
   def solve(n, a, b, c):
       if (n == 0) or (a < 0 and b < 0 and c < 0):
           return 0
       res = solve(n - min(a, b, c), a, b, c) + 1
       res = max(res, solve(n - min(a, b, c), a, b, c))
       res = max(res, solve(n - min(a, b, c), a, b, c))
       return res
   ```

4. e). La suma de los elementos del arreglo a y es O(n).

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

5) Lectura recomendada (opcional)

a) Otra Notación Asintótica.
b) Ideas principales:

Notación Omega (Ω):
Se usa para proporcionar umbrales inferiores acerca del tiempo de ejecución o la cantidad de recursos requeridos por un algoritmo. De igual modo, se usa para denotar la dificultad intrínseca de resolver ciertos problemas.

Aplica la regla de la dualidad:
$f(n) \in \Omega(f(n))$ sii $f(n) \in O(f(n))$

La regla de dualidad sirve para transformar la regla del limite, la regla del máximo y la regla del umbral en reglas acerca de la notación omega.

Algunos autores definen la notación omega como sigue: $f(n) \in \Omega(f(n))$ si existe un constante real y positiva d tal que $f(n) >= df(n)$ para un número finito de valores de n.

Notación Theta (Θ):
Es usada para demostrar que $t(n)$ está en el orden exacto de $f(n)$, es decir:
$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

6) Trabajo en Equipo y Progreso Gradual (Opcional)

a) Actas de reunión:

Integrante	Fecha	Hecho	Haciendo	Por Hacer
Murillo	7/08/2017	Implemento código ArrayMax	Código GroupSum y Fibonacci	Implementación HTML
Vidal	7/08/2017	Implemento código en línea recursión 1	Código en línea recursión 2	Calcular complejidad códigos en línea

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

Murillo	15/08/2017	Implemento gráficas y tablas del código del punto 1		Lectura y resumen opcional
Vidal	15/08/2017	Calculo complejidad y respondía pregunta numeral 2	Contestando preguntas numeral 3	Simulacro del parcial

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