Laboratorio Nro. 1: Recursión

Isabela Muriel Roldan
Universidad Eafit
Medellín, Colombia
imurielr@eafit.edu.co

Mateo Flórez Restrepo
Universidad Eafit
Medellín, Colombia
mflorezr@eafit.edu.co

2) Ejercicios en línea sin documentación HTML en GitHub

2.3. Explicación groupSum5

El ejercicio groupSum5 busca determinar si se puede tomar un grupo de enteros de un arreglo de tal forma que la suma de los números sea igual al valor de “target cumpliendo las condiciones dadas. Para lograrlo, lo primero que hace el método es comprobar si la longitud del arreglo es 0 y a partir de esto determinar si se puede o no cumplir la condición, esto depende de si target es 0 o no. Luego se comprueba si el número en la posición en la que se encuentra es múltiplo de 5, y si lo es, revisar si el número siguiente es igual a 1, si esto se cumple se llama al método recursivamente, utilizando el número que se encuentra 2 posiciones más adelante y restándole a target el múltiplo de 5. Si es número siguiente al múltiplo de 5 no es 1 entonces al llamar al método se toma este número.

2.4. Complejidad ejercicios en línea

-Recursión 1

```
Public static int(int n){
    Int count=0;                    //C1
    If(n%10==7){                    //C2
        Count++;                //C3
    }
    if(n/10==0){                    //C4
        return count;               //C5
    }
    return count+count7(n/10);          //C6+T(n/10)
}
```

```
T(n)=O(C6+T(n/10))
T(n)=O(T(n/10))
T(n)=O(T(log/10))
T(n)=O(log n)

public static int triangle(int rows){
    if(rows==0 || rows==1){              //C1
        return rows;                     //C2
       }
    return rows+triangle(rows-1);        //C3+T(n-1)
}

T(n)=O(C3+T(n-1))
T(n)=O(T(n-1))
T(n)=O(n)

Public static int bunnyEars(int bunny){
    If(bunny==0){                        //C1
        Return bunny;                    //C2
    }
    return 2+bunnyEars(b-1);             //C3+T(n-1)
}

T(n)=O(C3+T(n-1))
T(n)=O(T(n-1))
T(n)=O(n)

Public static boolean array6(int[] nums, int index){
    If(index==nums.length){              //C1
        Return false;                    //C2
    }
    if(nums[index]==6){                  //C3
        return true;                //C4
    }
    return array6(nums, index+1);        //T(n+1)+n
}

T(n)=O(T(n+1)+n
T(n)=O(n)
```

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

```
Public static boolean array220(int[] nums, int index){
    If(nums.length-1<=index){            //C1
        Return false;                    //C2
    }
    if(nums[index]*10==nums[index+1]){   //C3
        return true;                     //C4
    }
    return array220(nums, index+1);      //T(n+1)+n
}
```

T(n)=O(T(n+1)+n)
T(n)=O(n)

-Recursión 2

```
public static boolean groupSum5(int start, int[] nums, int target){
      if(start>=nums.length){
         if(target==0){
            return true;
         }
         return false;
      }
      if(nums[start]%5==0){
         if(start<nums.length-1 && nums[start+1]==1){
            return groupSum5(start+2, nums, target-nums[start]);
         }
         return groupSum5(start+1, nums, target-nums[start]);
      }
      if(groupSum5(start+1, nums, target-nums[start])){
         return true;
      }
      return groupSum5(start+1, nums, target);
   }
```

T(n)=O(2^n)

```
public static boolean groupSum6(int start, int[] nums, int target){
      if(start==nums.length){
         if(target==0){
            return true;
         }
```

```
        return false;
    }
    if(nums[start]==6){
        return groupSum6(start+1, nums, target-nums[start]);
    }
    if(groupSum6(start+1, nums, target-nums[start])){
        return true;
    }
    return groupSum6(start+1, nums, target);
}
```

T(n)=O(2^n)

```
public static boolean groupNoAdj(int start, int[] nums, int target){
    if(target==0){
        return true;
    }
    if(start>=nums.length){
        return false;
    }
    if(groupNoAdj(start+2, nums, target-nums[start])){
        return true;
    }
    return groupNoAdj(start+1, nums, target);
}
```

T(n)=O(2^n)

```
public static boolean groupSumClump(int start, int[] nums, int target){
    if(start>=nums.length){
        return target==0;
    }
    if(start<nums.length-1 && nums[start]==nums[start+1]){
        return groupSumClump(start+2, nums, target);
    }
    if(groupSumClump(start+1, nums, target)){
        return true;
    }
    if(groupSumClump(start+1, nums, target-nums[start])){
        return true;
    }
```

```
        return false;
    }
```

T(n)=O(2^n)

```
public static boolean splitArray(int[] nums){
        return helper(nums, 0, 0);
    }
    public static boolean helper(int[] nums, int i, int sum){
        if(i==nums.length){
            return sum==0;
        }
        if(helper(nums, i+1, sum+nums[i])){
            return true;
        }
        return helper(nums, i+1, sum-nums[i]);
    }
```

T(n)=O(2^n)

2.5. Explicación de las variables

T(n) se refiere al tiempo de ejecución en el peor caso, es decir en el caso en el que se ejecuta el mayor número de operaciones. Las variables n y m se refieren al número de instrucciones ejecutadas, normalmente se utiliza la n para algoritmos lineales, pero en el caso de algoritmos aún más complejos que estos, como lo son los cuadráticos, se utiliza también la m para representar otra operación, esto equivale a tener n^2.

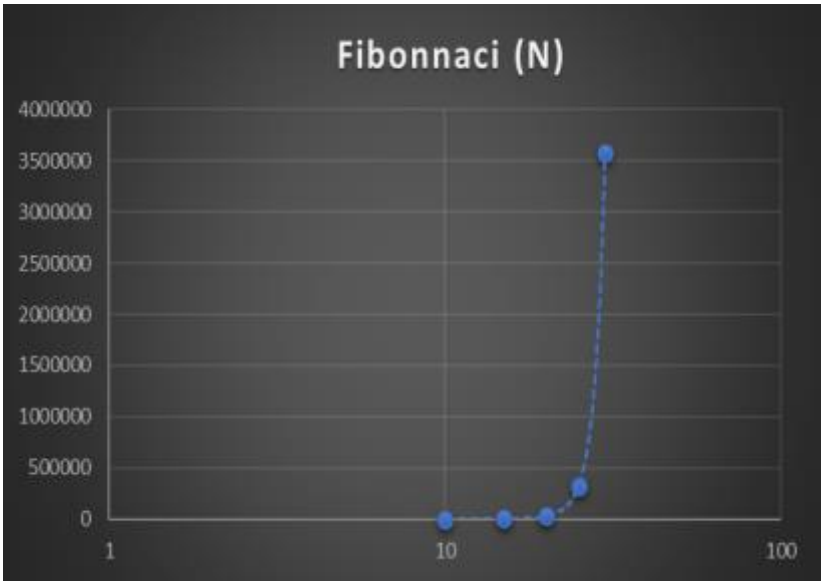
3) Simulacro de preguntas de sustentación de Proyectos

3.1

	N=100.000	N=1’000.000	N=10’000.000	N=100’000.000
R Array sum	Mas de 1 min	Mas de 1 min	Mas de 1 min	Mas de 1 min
R Array maximun	Mas de 1 min	Mas de 1 min	Mas de 1 min	Mas de 1 min
Fibonacci	Mas de 1 min	Mas de 1 min	Mas de 1 min	Mas de 1 min

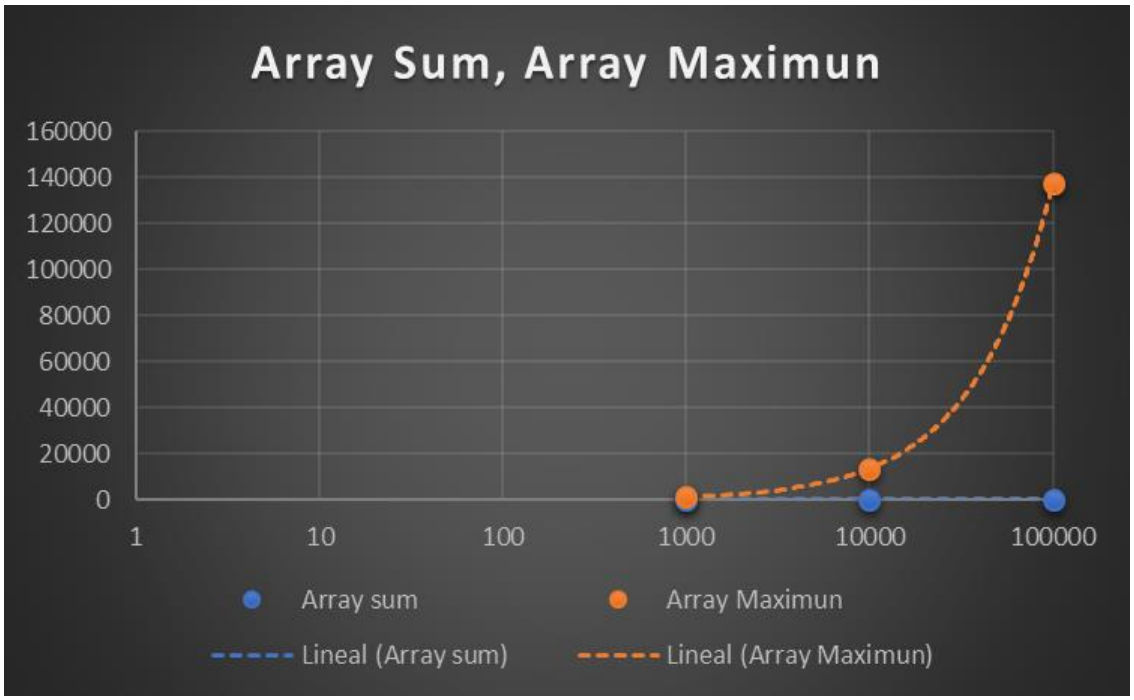
3.2

Fibonacci	
N	**Tiempo**
10	**241**
15	**2623**
20	**29861**
25	**321240**
30	**3576500**



N	1000	10000	100000
Array Sum	0	0	0
Array Maximun	1272	13377	137242

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



3.3 Datos

De acuerdo con los resultados del tiempo obtenidos en el laboratorio, se puede ver como las gráficas de estos tienen coherencia con lo que supone la complejidad de los algoritmos. Fibonacci cuya complejidad es O(2^n) concuerda con la gráfica que representa la expresión, al igual que Array Maximun con complejidad de O(n), sin embargo, al evaluar los resultados de Array Sum se generan desacuerdos, pues su complejidad es también O(n), pero su grafica representa la de una constante, y la expresión de su complejidad debería ser totalmente diferente. El problema de coherencia entre los resultados de Array sum es debido a que en los valores que lo evaluamos siempre dio 0, suponemos que estos resultados no variaron de 0 ya que los valores quizá eran pequeños, con valores más grandes podría variar el resultado. Concluimos que entre más grandes sean los valores evaluados, es más notable la complejidad de los tiempos de ejecución.

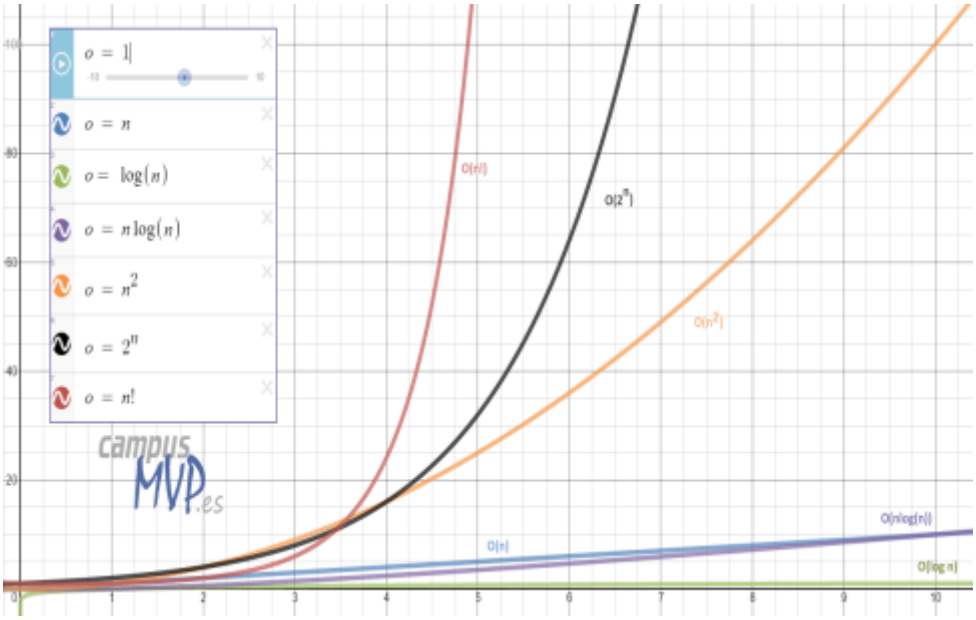


Figura 1: Graficas de la complejidad de un algoritmo

3.4 StackOverFlow error

Cada línea creada en un programa de java tiene un espacio de pila, y cada pila correspondiente a un programa tiene una cantidad ilimitada, y esta cantidad determina que tanto podemos hacer dentro del programa. Este error se produce cuando se excede la cantidad de esa pila.
Este suele ser un error muy común sobre todo cuando trabajamos con algoritmos recursivos, ya que, al hacer constantemente llamados entre métodos, una mala codificación puede generar una cantidad de llamados excesivos y sobrepasar la cantidad de la pila, haciendo que nuestra aplicación falle.

3.5 Fibonacci

El valor más grande que pudimos calcular con Fibonacci fue 30.
En la serie de Fibonacci recursivo es muy difícil calcular valores muy grandes, tarda demasiado tiempo en ejecutarse y dar un resultado.
Debido a que la recursión se encarga de resolver sub-problemas en un algoritmo y el algoritmo recursivo de la sucesión de Fibonacci tiene demasiados sub-problemas repetitivos e innecesarios, entonces al momento de resolver un n se vuelve ineficiente.
Complejidad del tiempo: T(n) = T(n-1) + T(n-2) + 1 = 2^n = O(2^n).

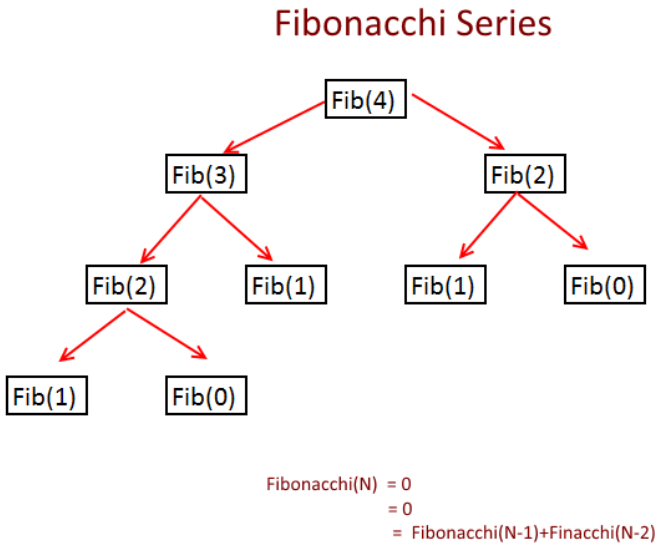


Figura 2: Fibonacci recursivo con n=4.

En la figura 1 podemos ver como este algoritmo recursivo resuelve muchos sub-problemas una y otra vez, aun cuando ya los había resuelto.

3.6. Fibonacci de valores grandes

Para calcular Fibonacci con valores muy grandes existe una solución: La programación dinámica (Dynamic Programming). La programación dinámica resuelve problemas recursivos de una manera más eficiente, un método que puede reducir el tiempo de ejecución de un algoritmo. En vez de repetir los sub-problemas como en la recursión, lo que hace la programación dinámica en este caso es usar "Memoizacion", la cual se encarga de que cuando hay sub-problemas repetidos, el algoritmo almacena la solución del primero para no tener que volver a calcularla, es decir, al momento de ejecutarse el algoritmo buscara si ya se encuentra almacenada la solución al sub-problema, en caso de que exista la usa, y si no, la resuelve y la guarda para ser usada después.
En este caso la complejidad del algoritmo es O(n).
Nota: Aparte de la memoizacion la programación dinámica tiene dos propiedades:
Sub-problemas superpuestos.
Sub-estructuras óptimas.

```java
public int fibDP(int x) {
        int fib[] = new int[x + 1];
        fib[0] = 0;
        fib[1] = 1;
        for (int i = 2; i < x + 1; i++) {
                fib[i] = fib[i - 1] + fib[i - 2];
        }
        return fib[x];
    }
```

fibDP.java hosted with ♥ by GitHub

Figura 3: Fibonacci con Programación dinámica.

3.7 Complejidad: Recursión 1 vs Recursión 2

La complejidad de los ejercicios de recursión 1 es O(n), y la de los ejercicios de recursión 2 es O(2^n). Concluimos que por jerarquía de complejidad los ejercicios de recursión 2 tienden a ser más complejos que los de recursión 1 en cuanto al tiempo de ejecución de cada algoritmo y por ende la recursión 2 es menos eficiente para calcular su complejidad con valores grandes.

4) Simulacro de Parcial

1. *Start+1, nums, target*
2. a
3. **3.1**(n-1,a-1,b-1,c-1)
   3.2(a, b)
   3.3(res, c)
4. *e*

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