ESTRUCTURA DE DATOS 1
Código ST0245

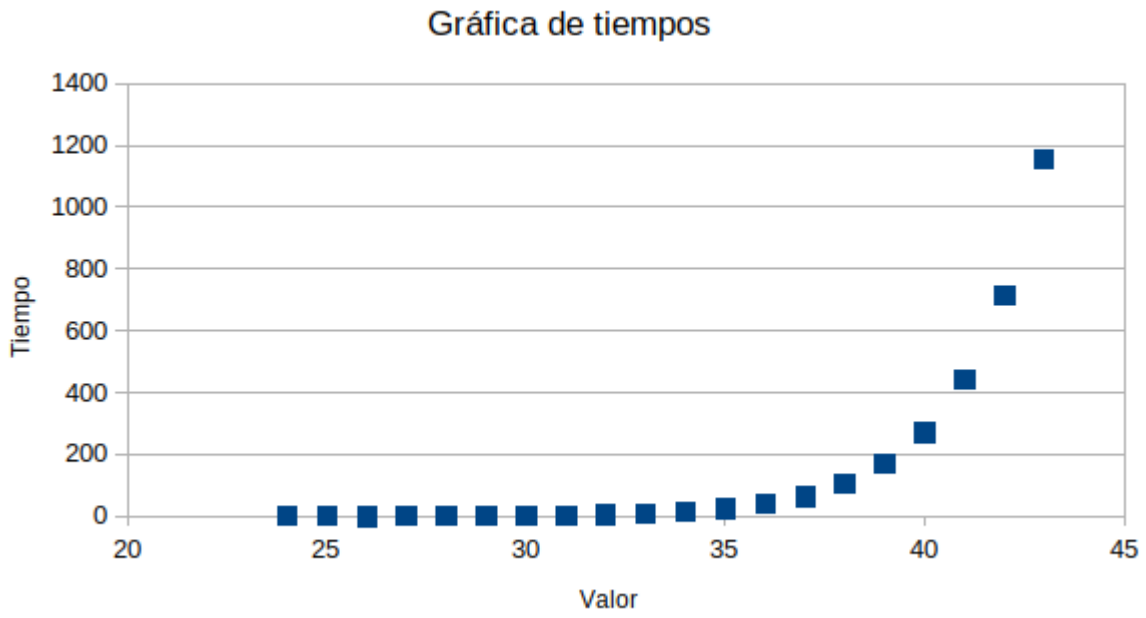
Laboratorio Nro. 1
Recursión

Daniel Alejandro Hincapié Sánchez
Universidad Eafit
Medellín, Colombia
dahincapis@eafit.edu.co

Anthony García Moncada
Universidad Eafit
Medellín, Colombia
agarciam@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1 T(n) = T(n-1)+T(n-2)+c
3.2



Analizando la gráfica podemos darnos cuenta de que la complejidad del algoritmo es de tipo exponencial, por lo que con un valor de entrada de 50, el tiempo que se tardaría sería casi eterno, lo que podría estimarse a la edad del universo.

3.3 La complejidad de este algoritmo lo hace inviable para usarse en Puerto Antioquia en 2020, puesto que ya con contenedores de 50 centímetros tenemos problemas, así que en caso de usarse con contenedores de miles de centímetros, el algoritmo no terminaría de ejecutarse jamás.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

3.4 En el ejercicio groupSum5 se nos pide realizar una suma de enteros que están en el interior de una arreglo, todo esto cumpliendo ciertas condiciones, como que en la suma se incluyan todos los dígitos divisibles por 5. Para dar solución a esto, realizamos varios llamados recursivos y finalmente una condición de parada, esta última nos dicta que dependiendo del valor del target se nos retornará el booleano correspondiente. En los llamados recursivos encontramos las siguientes posibilidades, en uno de ellos primero se decide si el número que recorre el arreglo es múltiplo de 5, para así efectuar el llamado recursivo y realizar una resta con el target y ese múltiplo de 5. En otro de los llamados verificamos por medio de restas entre los números del arreglo, cuales números se pueden sumar entre ellos.

3.5 **Factorial:**

```
//Este método se encarga de retornar el factorial de n de forma recursiva
public int factorial(int n) {
    if(n == 0) //C1
        return 1; //C2

    return factorial(n-1) * n; //C3+T(n-1)
}
```

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

T(n)= Cn+C1

T(n) es O(Cn+C1)
T(n) es O(Cn)
T(n) es O(n)

La variable n hace referencia al número dado para retornar su factorial.

bunnyEars:

```
//Este método se encarga de sumar los pares de orejas todos los conejos que tengamos, número definido por “bunnies”.
public int bunnyEars(int bunnies) {
    if(bunnies == 0) //C1
        return 0; //C2

    return bunnyEars(bunnies - 1) + 2; //C3 + T(n-1)
  }
```

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

T(n)= Cn+C1

T(n) es O(Cn+C1)
T(n) es O(Cn)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

T(n) es O(n)

La variable n representa el número de conejos cuyas orejas debemos sumar.

Fibonnacci:

```
//Este método retorna el enésimo número de la secuencia de Fibonnacci
public int fibonacci(int n) {
    if(n<=1) //C1
        return n; //C2

    return fibonacci(n-1) + fibonacci(n-2); //T(n-1) + T(n-2) + C3
}
```

T(n)= C1+C2, si n<=1.
T(n)= C3+T(n-1)+T(n-2), de lo contrario.

T(n)= C*2^n+C1

T(n) es O(C*2^n+C1)
T(n) es O(C*2^n)
T(n) es O(2^n)

La variable n se refiere al número de la secuencia que quiere hallarse

bunnyEars2:

```
// Este metodo se encarga de sumar las orejas de los conejos, con la condición de que cada conejo representado por un número par, no tiene dos orejas, sino tres de estas
public int bunnyEars2(int bunnies) {
    if(bunnies <= 1) //C1
        return bunnies*2; //C2

    return (3 - bunnies % 2) + bunnyEars2(bunnies - 1); //C3+T(n-1)
  }
```

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

T(n)= Cn+C1

T(n) es O(Cn+C1)
T(n) es O(Cn)
T(n) es O(n)

La variable n se refiere al número de conejos cuyas orejas deben sumarse.

triangle:

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
/*Este método se encarga de hallar el número de bloques que hay un triángulo de acuerdo al número de filas que éste tenga
*/
public int triangle(int rows) {
    if(rows <= 1) //C1
        return rows; //C2

    return triangle(rows - 1) + rows; //C3 + T(n-1)
}
```

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

T(n)=Cn+C1

T(n) es O(Cn+C1)
T(n) es O(Cn)
T(n) es O(n)

La variable n se refiere al número de filas con las que cuenta el triángulo de bloques.

splitArray:

```
/*Este método se encarga de hallar si es posible dividir un arreglo de enteros dado en dos, de tal forma que la suma de cada uno de los grupos resultantes sea la misma, teniendo en cuenta que cada elemento del arreglo debe incluirse en uno de los grupos
*/
public boolean splitArray(int[] nums) {
    int i=0;
    int sum1=0;
    int sum2=0;
    return splitArray(nums, i, sum1, sum2);
}
private boolean splitArray(int[] nums, int i, int sum1, int sum2){
    if(i>=nums.length){   //C1
      return sum1==sum2; //C2
    }
    return
splitArray(nums,i+1,sum1+nums[i],sum2)||splitArray(nums,i+1,sum1,sum2+nums[i]);
//C3+2T(n-1)
```

T(n)= C1+C2, si i >= nums.length.
T(n)= C3+2T(n-1), de lo contrario.

T(n)=C((2^n)-1)+C1(2^(n-1))

T(n) es O(C((2^n)-1)+C1(2^(n-1)))
T(n) es O(C((2^n)-1))

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

T(n) es O((2^n)-1)
T(n) es O(2^n)

La variable n se refiere al tamaño del arreglo.

splitOdd10:

```
/*Este método se encarga de hallar la posibilidad de dividir un arreglo en dos grupos de tal forma que la suma de los elementos de uno de los grupos se múltiplo de 10, y la suma del otro sea un número impar, teniendo en cuenta que todos los elementos del arreglo deben ubicarse en alguno de los dos grupos
*/
public boolean splitOdd10(int[] nums) {
    int i=0;
    int sum1=0;
    int sum2=0;
    return splitOdd10(nums,i,sum1,sum2);
}
private boolean splitOdd10(int[] nums, int i, int sum1, int sum2){
    if(i>=nums.length){  //C1
            return (sum1%10==0&&sum2%2!=0);  //C2
    }
    return splitOdd10(nums, i+1, sum1+nums[i], sum2)||splitOdd10(nums, i+1, sum1, sum2+nums[i]);  //C3+2T(n-1)
}
```

T(n)= C1+C2, si i >= nums.length.
T(n)= C3+2T(n-1), de lo contrario.

T(n)=C((2^n)-1)+C1(2^(n-1))

T(n) es O(C((2^n)-1)+C1(2^(n-1)))
T(n) es O(C((2^n)-1))
T(n) es O((2^n)-1)
T(n) es O(2^n)

La variable n se refiere al tamaño del arreglo.

split53:

```
/* Este método se encarga de hallar si es posible dividir un arreglo en dos grupos de tal forma que la suma de ambos grupos sea la misma, teniendo en cuenta que todos los elementos del arreglo deben incluirse en uno de los dos grupos, además, todos los múltiplos de 5 deben estar en un grupo, y los múltiplos de 3 (que no sean múltiplos de 5), en el otro grupo.
*/
public boolean split53(int[] nums) {
    int i=0;
    int sum1=0;
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
    int sum2=0;
    return split53(nums,i,sum1,sum2);
}
private boolean split53(int[] nums, int i, int sum1, int sum2){
    if(i>=nums.length){  //C1
            return (sum1==sum2);  //C2
    }
    int value= nums[i]; //C3
    if (value%5==0){  //C4
            return split53(nums, i+1, sum1+nums[i], sum2);  //C5+T(n-1)
    }
    if (value%3==0){  //C6
            return split53(nums, i+1, sum1, sum2+nums[i]);  //C7+T(n-1)
    }
    return split53(nums, i+1, sum1+nums[i], sum2)||split53(nums, i+1, sum1, sum2+nums[i]);
//C8+2T(n-1)
}
```

T(n)= C1+C2, si i >= n
T(n)= C1+C3+C4+C5+T(n-1), si nums[i] es múltiplo de 5
T(n)= C1+C3+C4+C6+C7+T(n-1), si nums[i] es múltiplo de 3
T(n)= C1+C3+C4+C6+C8+2T(n-1), de lo contrario.

T(n)=C((2^n)-1)+C1(2^(n-1))

T(n) es O(C((2^n)-1)+C1(2^(n-1)))
T(n) es O(C((2^n)-1))
T(n) es O((2^n)-1)
T(n) es O(2^n)

La variable n se refiere al tamaño del arreglo.

groupNoAdj:

```
/*Este método evalúa la posibilidad de tomar un grupo de números de un arreglo dado de tal forma que su suma alcance un número “target”, cumpliendo la condición de que si tomo un número, no puedo tomar el número que le sigue en el arreglo
*/
public boolean groupNoAdj(int start, int[] nums, int target) {
    if(start>=nums.length){                                                    //C1
      return target==0;                                                        //C2
    }
    return groupNoAdj(start+2, nums, target-nums[start])||groupNoAdj(start+1, nums, target);                                                    //C3+T(n-1)+T(n-2)
}
```

T(n)= C1+C2, si start >= n.
T(n)= C3+T(n-1)+T(n-2), de lo contrario.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

T(n)= C*2^n+C1

T(n) es O(C*2^n+C1)
T(n) es O(C*2^n)
T(n) es O(2^n)

La variable n se refiere al tamaño del arreglo.

groupSumClump:

```
public boolean groupSumClump(int start, int[] nums, int target) {
    if(start>=nums.length){                                          //C1
      return target==0;                                              //C2
    }
    int rep=1;                                                       //C3
    for(int i=start+1;i<=nums.length-1;i++){                         //C4*m
      if(nums[i]==nums[start]){                                      //C5*m
        rep++;                                                       //C6*m
      }else{
        break;                                                       //C7
      }
    }
    return groupSumClump(start+rep, nums, target)||groupSumClump(start+rep, nums,
target-(nums[start]*rep));                                           //C8+2T(n-m)
}
```

T(n)= C1+C2, si start>=n.
T(n)= C1+C3+m*(C4+C5+C6)+C7+C8+

groupSum5:

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

  if(target == 0)
     return true;
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
        return false;
      }
    }
```

4) Simulacro de Parcial

4.1. *start+1, nums, target.*

4.2. *a.*

4.3. *Línea 4: (n-a, b, c, a)*
Línea 5: (res , Math.max(solucionar(n-a, b, a, c)+1, Math.max(solucionar(n-a, c, a, c)+1, solucionar(n-b, a, c, b)+1)));
Línea 6: (res , Math.max(solucionar(n-a, a, b, c)+1, Math.max(solucionar(n-b, a, b, c)+1, solucionar(n-c, a, b, c)+1)));

4.4. *e*

4.5. *1) Línea 2: return n.*
Línea 3: n-1.
Línea 4: n-2.
2) b.

4.6. *Línea 10: sumaAux(n, i+2).*
Línea 12: sumaAux(n, i+1).

4.7. *Línea 9: S, i+1, t-S[i].*
Línea 10: S, i+1, t.

4.8. *Línea 9: return 0.*
Línea 13: ni+nj.

4.9. *c*

4.10. *b*

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473