Laboratorio Nro. I: Recursión

Juan Camilo Guerrero Alarcón
Universidad Eafit
Medellín, Colombia
jcguerrera@eafit.edu.co

Santiago Pulgarín Vasquez
Universidad Eafit
Medellín, Colombia
spulgarinv@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1)

El ejercicio groupSum5 se trata a cerca de encontrar un subgrupo dentro de un arreglo que sume algún puntaje (este puntaje es dado por el usuario), además tiene una condición extra, esta es: que todo múltiplo de 5 debe estar en el subgrupo y si el valor siguiente a un múltiplo de 5 es 1, este no se debe tomar en cuenta para formar el subgrupo.

Dicho esto, el algoritmo funciona de esta manera:

El algoritmo recibe por parámetro tres valores de tipo entero, el primero es start (contador que generalmente comienza en 0), el segundo es un arreglo (de este arreglo se sacara el subgrupo para la suma) y por último esta "target" (es el valor que debemos lograr con la suma del subgrupo). Primero realiza una condición para saber si ya sobrepaso el tamaño del arreglo, de ser cierta retorna el valor booleano de la expresión (target == 0) cuando este es verdadero está diciendo que si hay un subgrupo que cumple todas las condiciones, y cuando es falso sigue su ejecución con otro condicional esta vez pregunta si el número del arreglo en la posición "start" es múltiplo de 5, si es cierto llega a otro condicional que pregunta si start es menor que el tamaño del arreglo y si la posición del arreglo siguiente es igual a 1, de ser cierto aplica el primer llamado recursivo en el cual cambia el valor de "start" por "start + 2" esto debido a que se debe saltar al número 1, luego pasa el arreglo "nums" y al "target" le resta el valor del arreglo en la posición "start". Por otro lado, si el condicional es falso sigue la ejecución y llega a otro llamado recursivo, en el cual se sabe que el valor siguiente del arreglo en la posición "start" no es 1, entonces cambia "start" por "start + 1", pasa el arreglo "nums" y al "target" le resta el valor del arreglo en la posición "start".

Al final si la condición de ser múltiplo de 5 no fue cumplida hace dos llamados recursivos con un (||) en el primero cambia "start" por "start + 1", pasa el arreglo "nums" y al "target" le resta el valor del arreglo en la posición "start" y en el segundo cambia "start" por "start + 1", pasa el arreglo "nums" y el "target".

Por último, en caso de que en la primera ejecución del programa no se cumpla (target == 0), el programa comienza a devolverse de sus llamados recursivos y no empieza en (start == 0) sino en (start == 1) y repite todo lo anterior.

3.2)

Recursión 1

SumDigits:

```
public int sumDigits (int n) {
      if (n < 10)                                              // Contante
         return n;                                            // Constante
      return sumDigits (n % 10) + sumDigits(n / 10);   // T(n-1) + T(n-1)
      }
```

Complejidad

$$T(n) = \begin{cases} c_1 & \text{if } n < 10 \\ T(n-1) + T(n-1) + c_2 & \text{if } n > 0 \end{cases}$$

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

Por definición de O

T(n) es O (C * ((2^n) – 1) + C_1 * (2^n-1)) Por definición de O.
T(n) es O (C_1 * (2^n-1)) Por regla de la suma.
T(n) es O (2^n-1) Por regla del producto.
T(n) es O (2^n) Por regla de la suma.

Count7

```
public int count7(int n) {
      if (n == 0)                  //Constante
         return 0;               //Constante
      if (n % 10 == 7)            //Constante
         return 1 + count7(n/10);  //T(n-1)

      return count7(n/10);        //T(n-1)
}
```

Complejidad:

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ 2 * t(n-1) + c_2 & \text{if } n/10 = 7 \end{cases}$$

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

Por definición de O

T(n) es O (C * ((2^n) – 1) + C_1 * (2^n-1)) Por definición de O.
T(n) es O (C_1 * (2^n-1)) Por regla de la suma.
T(n) es O (2^n-1) Por regla del producto.
T(n) es O (2^n) Por regla de la suma.

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

Factorial

```
public int factorial (int n) {
        if (n <=2)                  // Constante
           return n;                //Constante
       return n * factorial(n-1); // T(n-1)
   }
```

Complejidad

$$T(n) = \begin{cases} c_1 & \text{if } n <= 2 \\ t(n-1) + c & \text{if } n > 2 \end{cases}$$

$t(n) = c\,n + c_1$ (c_1 is an arbitrary parameter)

Por definición de O

T(n) es O (C * n + C_1) Por definición de O.
T(n) es O (C * (n)) Por regla de la suma.
T(n) es O (n) Por regla del producto.

CountX

```
public int countX(String str) {
        if(str.length() == 0)                   //Constante
            return 0;                           //Constante
       if (str.charAt(0)== 'x')                 //Constante
            return 1 + countX(str.substring(1)); // t(n-1)

       return countX(str.substring(1));         //t(n-1)
   }
```

Complejidad

$$T(n) = \begin{cases} c_1 & \text{if } str.length() = 0 \\ 2 * t(n-1) + c_2 & \text{if } str.charAt =' x' \end{cases}$$

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

Por definición de O

T(n) es O (C * ((2^n) – 1) + C_1 * (2^n-1)) Por definición de O.
T(n) es O (C_1 * (2^n-1)) Por regla de la suma.

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

T(n) es O (2^n-1) Por regla del producto.
T(n) es O (2^n) Por regla de la suma.

BunnyEars

```
public int bunnyEars(int bunnies) {
      if (bunnies == 0) return 0;              //Constante

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

Complejidad

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ t(n-1) + c_2 & \text{if } n > 0 \end{cases}$$

$t(n) = c\,n + c_1$ (c_1 is an arbitrary parameter)

Por definición de O

T(n) es O (C * n + C_1) Por definición de O.
T(n) es O (C * (n)) Por regla de la suma.
T(n) es O (n) Por regla del producto.

Recursión 2

groupSum5

```
public boolean groupSum5(int start, int [] nums, int target) {
      if (start >= nums.length) {              // Constante
         return target == 0;           // Constante
      }
      if (nums[start] % 5 == 0) {                                    // Constante
         if (start < nums.length - 1 && nums[start + 1] == 1  ) {              // Constante
            return groupSum5(start + 2, nums, target - nums[start]);        // T(n-1)
         } else {
            return groupSum5(start + 1, nums, target - nums[start]);        // T(n-1)
         }
      }
      return groupSum5(start + 1, nums, target - nums[start]) ||   groupSum5(start + 1, nums, target);
      //2*T(n-1) + Constante
   }
```

Complejidad

$$T(n) = \begin{cases} c1 & \text{if } start >= nums.length \\ T(n-1) + T(n-1) + T(n-1) + T(n-1) + c2 & \text{if } start < nums.length \end{cases}$$

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

$$T(n) = \frac{1}{3} c^2 (4^n - 1) + c_1 4^{n-1}$$

Por definición de O
T(n) es O((1/3*c^2*((4^n)-1)) + (c1*4^n-1)), por definición de O
T(n) es O(((4^n)-1) + (4^n-1)), por regla del producto
T(n) es O((4^n) + (4^n-1)), por regla de la suma
T(n) es O(4^n)

groupSum6

```
public boolean groupSum6(int start, int[] nums, int target) {
  if (start >= nums. length) {                  // Constante
    return target == 0;                         // Constante
  }
  if (nums[start] == 6) {                       // Constante
    return groupSum6(start + 1, nums, target - nums[start]);   // T(n-1)
  }
return groupSum6(start + 1, nums, target - nums[start]) || groupSum6(start + 1, nums, target); // 2*T(n-1) +
Constante
}
```

Complejidad

$$T(n) = \begin{cases} c1 & \text{if } n >= nums.length \\ T(n-1) + T(n-1) + T(n-1) + c2 & \text{if } n < nums.length \end{cases}$$

$$T(n) = c_1 3^{n-1} + \frac{1}{2} c_2 (3^n - 1)$$

Por definición de O
T(n) es O((c1*3^n-1) +(1/2 * c2*((3^n)-1))), por definición de O
T(n) es O((3^n-1) + ((3^n)-1)), por regla del producto
T(n) es O((3^n-1) + (3^n)), por regla de la suma
T(n) es O(3^n)

groupNoAdj

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

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

```
    return false;                    // Constante
  }
```

Complejidad

$$T(n) = \begin{cases} c1 & \text{if } start >= nums.length \\ T(n-1) + T(n-1) + c2 & \text{if } start < nums.length \end{cases}$$

$$T(n) = c^2\left(2^n - 1\right) + c_1\, 2^{n-1}$$

Por definición de O

T(n) es O((c^2 * ((2^n) - 1)) + (c1 * (2^n-1))), por definición de O
T(n) es O(((2^n)-1) + (2^n-1)), por definición de producto
T(n) es O((2^n) + (2^n-1)), por definición de suma
T(n) es O(2^n)

Split53

```
public boolean split53(int[] nums) {
  return helper(0, nums, 0, 0);
}

public boolean helper (int start, int [] nums, int suma1, int suma2) {
  if (start >= nums.length) {                    // Constante
    return suma1 == suma2;                       // Constante
  }
  if (nums[start] % 5 == 0) {                    // Constante
    return helper(start + 1, nums, suma1 + nums[start], suma2);  // T(n-1)
  }
  if (nums[start] % 3 == 0) {                    // Constante
    return helper(start + 1, nums, suma1, suma2 + nums[start]);          // T(n-1)
  }
  return helper(start + 1, nums, suma1 + nums[start], suma2) || helper(start + 1, nums, suma1, suma2 +
nums[start]);   // 2*T(n-1) + Constante
}
```

Complejidad

$$T(n) = \begin{cases} c1 & \text{if } start >= nums.length \\ T(n-1) + T(n-1) + T(n-1) + T(n-1) + c2 & \text{if } start < nums.length \end{cases}$$

$$T(n) = \frac{1}{3}\, c^2\left(4^n - 1\right) + c_1\, 4^{n-1}$$

Por definición de O

T(n) es O((1/3*c^2*((4^n)-1)) + (c1*4^n-1)), por definición de O
T(n) es O(((4^n)-1) + (4^n-1)), por regla del producto
T(n) es O((4^n) + (4^n-1)), por regla de la suma
T(n) es O(4^n)

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

splitOdd10

```
public boolean splitOdd10(int[] nums) {
  return helper(0, nums, 0, 0);
}

public boolean helper (int start, int [] nums, int suma1, int suma2) {
  if (start >= nums.length) {                    // Constante
    return suma1 % 10 == 0 && suma2 % 2 == 1 || suma1 % 2 == 1 && suma2 % 10 == 0;
   // 2*T(n-1) + Constante
  }
  return helper(start + 1, nums, suma1 + nums[start], suma2) || helper(start + 1, nums, suma1, suma2 +
nums[start]);  // 2*T(n-1) + Constante
}
```

Complejidad

$$T(n) = \begin{cases} c1 & \text{if } start >= nums.length \\ T(n-1) + T(n-1) + T(n-1) + T(n-1) + c2 & \text{if } start < nums.length \end{cases}$$

$$T(n) = \frac{1}{3} c^2 \left(4^n - 1\right) + c_1 \, 4^{n-1}$$

Por definición de O
T(n) es O((1/3*c^2*((4^n)-1)) + (c1*4^n-1)), por definición de O
T(n) es O(((4^n)-1) + (4^n-1)), por regla del producto
T(n) es O((4^n) + (4^n-1)), por regla de la suma
T(n) es O(4^n)

3.3)

Recursión 1

- SumDigits: En este caso n expresa el valor de un numero entero pero su llamada recursiva va hasta determinado termino

- Count7: La n en este caso también va a este el enésimo termino ya que en este caso depende la cantidad de operaciones que realiza para ir comprobando y evaluando.

- Factorial: La variable n en este caso en lineal ya que va aumentado conforme al valor n que se desee calcular, entre más grande sea el valor n aumentara de acuerdo al caso.

- CountX: En este ejercicio expresa O(2^n) donde n es el tamaño del arreglo de caracteres conforme se desea evaluar en el ejercicio.

- BunnyEars: En este ejercicio la variable n se refiere al numero de orejas de conejos que queramos calcular y así como en el ejercicio de factorial también su notación es O(n) lo que quiere decir que también es lineal.

Recursión 2

- En cálculo de complejidad del ejercicio groupSum5, la variable "n" representa el tamaño del arreglo.
- En cálculo de complejidad del ejercicio groupSum6, la variable "n" representa el tamaño del arreglo.
- En cálculo de complejidad del ejercicio groupNoAdj, la variable "n" representa el tamaño del arreglo.
- En cálculo de complejidad del ejercicio split53, la variable "n" representa el tamaño del arreglo.
- En cálculo de complejidad del ejercicio splitOdd10, la variable "n" representa el tamaño del arreglo.

3.4)

En nuestro grupo de trabajo deducimos que el error Stack Overflow ocurre básicamente por no determinar una apropiada condición de parada a nuestro algoritmo recursivo, a su vez este error en si se genera por un desbordamiento en la pila, lo que no nos permite seguir haciendo nuestro llamado recursivo y este no seguirá ejecutándose ya que en dicha pila ya no hay más espacio.

3.5)

El valor mas grande que pudimos calcular de Fibonacci fue de 100, nos dimos cuenta de esto, ya que , el computador tardo mucho en mostrar el resultado, y el ¿Por qué no se puede ejecutar el Fibonacci de 1 millón? Es muy relativo, ya que, depende de la capacidad de procesamiento de cada máquina.

3.7)

Lo que pudimos concluir acerca de la complejidad entre los ejercicios de recursión 1 y recursión 2, es que si bien en algunos casos de recursión 1, el orden era de 2^n y en otros era simplemente de n, en recursión 2 el orden de la complejidad alcanzo 4^n, esto lo que nos dice es que el algoritmo tiene mas llamados recursivos, y cuando esto sucede el nivel de complejidad se eleva lo que hace que el tiempo de ejecución sea mas grande en comparación por ejemplo de un orden "n".

4) Simulacro de Parcial

1. ***Start + 1; nums; target***
2. **a)** T(n) = T(n/2) +C
3. **solucionar (**int n – a, int a, int b, int c**);**
   res, solucionar (int n - b, int a, int b, int c**),**
   res, solucionar (int n - c, int a, int b, int c**);**
4. **e)** La suma de los elementos del arreglo a y es O(n)
5. return n;
   formas(n-1) +
   formas(n-2);
   b) T(n) = T(n-1) + T(n-2) + C

6. sumaAux(n, i+2);

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

n.charAt(i) – ‘0’ + sumaAux(n, i+1);

7. comb(s, i + 1, t – s[i]) ||
comb(s, i + 1, t)

8. return 0;
suma = ni + nj

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