Tomas Duque Giraldo

Juan Felipe Oriz Salgado

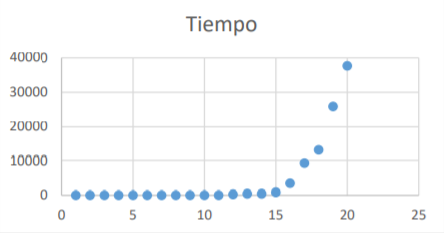
**3) Simulacro de preguntas de sustentación de Proyectos**

**3.1**

T (n, m) = C3 + T(n-1, m) + T(n, m-1), no tiene ecuación de recurrencia, no es posible solucionarla

La complejidad de este algoritmo es de O(2n), lo que hace que crezca exponencialmente

**3.2**



Como podemos observar, la gráfica crece exponencialmente a medida que se van presentando los datos, este tipo de gráficas muestras una de las complejidades más extensas de desarrollar ya que si tenemos una cantidad de 300.000 datos para este laboratorio, a la hora de ser ejecutado tomará mucho tiempo ya que a medida de que van pasando los datos su tiempo de respuesta crece exponencialmente. En lo que podríamos afirmar que es poco práctico debido a su complejidad y su tiempo de ejecución.

**3.3**

Como se pudo ver en los puntos anteriores, este algoritmo tiene una complejidad de O(2n) por lo cual lo hace una complejidad exponencial, al cargar tantos datos lo que haría seria un caos ya que demoraría demasiado tiempo en ejecutarse y no sería para nada eficiente implementarlo con los datasets.

**3.4**

**GroupSum5** tiene tres condiciones. la primera verifica si el número real es múltiplo de cinco y luego si el siguiente número es uno. suma el múltiplo de cinco y omite el uno La otra condición comprueba si el número real es múltiplo de cinco y si es cierto, suma el múltiplo de cinco y pasa al siguiente número. Y finalmente tiene una condición que verifica si sumando el número real a la suma logrará el objetivo o no sumando y en ambos casos continuar con el siguiente número

**3.5 Calculen la complejidad de los Ejercicios en Línea de los numerales 2.1 y 2.2 y agréguenla al informe PDF**

**Recursión 1**

**Recursion-1 > Factorial**

public int factorial(int n) {

if(n == 1 || n == 2) return n; // C1

else return factorial(n-1)\*n; // C2\*T(n-1)

}

T(n) = C1 + C2\*T(n-1) => resuelto con WolphramAlpha

T(n) = C1\*n + C1 => O reglas de notación, suma y multiplicación O (n)

**Recursion-1 > Bunny ears**

public int bunnyEars(int bunnies) {

if(bunnies == 0) return 0; // C1

else return bunnyEars(bunnies-1) + 2; // C2\*T(n-1) + C3 }

T(n) = C1 + T(n-1) + C3 => resuelto con WolphramAlpha

T(n) = (C1 + C2)\*n + C1 => O reglas de notación, suma y multiplicación O (n)

**Recursion-1 > fibonacci**

public int fibonacci(int n) {

if(n == 0) return 0; // C1

if(n == 1 || n == 2) return 1; // C2

else return fibonacci(n-2) + fibonacci(n-1); // C3\*T(n-1) + C3\*T(n-2)

}

T(n) = C1 + T(n-1) + T(n-2) => resuelto con la práctica de laboratorio No. 1: “Recursión" punto 4.4.1 O(2^n)

**Recursion-1 > bunnyEars2**

public int bunnyEars2(int bunnies) {

if(bunnies == 0) return 0; // C1

if(bunnies % 2 == 0) return bunnyEars2(bunnies - 1) + 3; // C2\*T(n - 1) + C3

else return bunnyEars2(bunnies - 1) + 2; // C2\*T(n - 1) + C4 }

T(n) = C1 + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1\*n + C1 =>) O reglas de notación, suma y multiplicación O (n)

**Recursion-1 > triangle**

public int triangle(int rows) {

if (rows == 0) return 0; // C1

if(rows == 1) return 1; // C1

else return triangle(rows - 1) + rows; // C2\*T(n - 1) + C3

}

T(n) = C1 + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1\*n + C1 => O reglas de notación, suma y multiplicación O (n)

**Recursion 2**

**Recursion-2 > groupSum**

public boolean groupSum(int start, int[] nums, int target) {

if(start == nums.length) return target == 0; // C1

else return (groupSum(start + 1, nums, target - nums[start]) || // C2\*T(n - 1)

groupSum(start + 1, nums, target)); // C2\*T(n - 1)

}

T(n) = C1 + T(n-1) + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1(2^n-1) + C1 2^(n-1) =>O reglas de notación, suma y multiplicación O (2^n)

**Recursion-2 > groupNoAdj**

public boolean groupNoAdj(int start, int[] nums, int target) {

if(start >= nums.length) return target == 0; // C1

else return (groupNoAdj(start + 2, nums, target - nums[start]) || // C3\*T(n - 2) groupNoAdj(start + 1, nums, target)); // C4\*T(n - 1)

}

T(n) = C1 + T(n-1) + T(n-2) => resuelto con la práctica de laboratorio No. 1: “Recursión" punto 4.4.1 O(2^n)

**Recursion-2 > groupSum5**

public boolean groupSum5(int start, int[] nums, int target) {

if(start == nums.length){ // C1 return target == 0; }

if(nums[start] % 5 == 0 && start + 1 < nums.length && nums[start + 1] == 1)

return groupSum5(start + 2, nums, target - nums[start]); // C2\*T(n - 2)

if(nums[start] % 5 == 0) return groupSum5(start + 1, nums, target - nums[start]); // C3\*T(n - 1)

else return (groupSum5(start +1, nums, target -nums[start]) || // C4\*T(n - 1)

groupSum5(start +1, nums, target)); // C5\*T(n - 1)

}

T(n) = C1 + T(n-1) + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1(2^n-1) + C1 2^(n-1)=> O reglas de notación, suma y multiplicación O (2^n)

**Recursión-2 > splitArray**

public boolean splitArrayHelper(int start, int sum1, int sum2, int[] nums){

if(start == nums.length) return sum1 == sum2; // C1

else{

return (splitArrayHelper(start + 1, sum1 + nums[start], sum2, nums) || // C2\*T(n -1)

splitArrayHelper(start + 1, sum1, sum2 + nums[start], nums)); // C3\*T(n -1)

}

}

T(n) = C1 + T(n-1) + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1(2^n-1) + C1 2^(n-1)=> O reglas de notación, suma y multiplicación O (2^n)

**Recursion-2 > groupSumClump**

public boolean groupSumClump(int start, int[] nums, int target) {

if(start == nums.length) return target == 0; // C1

if(start + 1 < nums.length && nums[start] == nums[start + 1] && nums[start] != nums[start - 1]){

int j = 0;

for(int i = 0; i + start < nums.length; i++){

if(start + i + 1 < nums.length && nums[start + i] == nums[start + i + 1]){

// C2\*m j++;

}

}

return (groupSumClump(start + j + 1, nums, target - nums[start]\*(j + 1)) || // C3\*T(n - m)

groupSumClump(start + j + 1, nums, target)); // C4\*T(n - m)

}else if(start >= 1 && nums[start] == nums[start - 1]) return groupSumClump(start + 1, nums, target); // C5\*T(n - 1)

else return (groupSumClump(start + 1, nums, target - nums[start]) || // C6\*T(n - 1)

groupSumClump(start + 1, nums, target)); // C7\*T(n - 1)

}

T(n) = C1 + T(n-1) + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1(2^n-1) + C1 2^(n-1)=>O reglas de notación, suma y multiplicación O (2^n) en este algoritmo, la “m” en el “C4\*T(n – m)

**Recursion-2 > groupSum6**

public boolean groupSum6(int start, int[] nums, int target) {

if(start == nums.length) return target == 0; // C1

else{ if(nums[start] == 6)

return groupSum6(start + 1, nums, target - nums[start]);//C2\*T(n-1)

else return (groupSum6(start + 1, nums, target - nums[start]) || // C3\*T(n - 1)

groupSum6(start + 1, nums, target)); // C3\*T(n - 1)

}

}

T(n) = C1 + T(n-1) + T(n-1) => Resuelto con WolphramAlpha

T(n) = C1(2^n-1) + C1 2^(n-1) =>O reglas de notación, suma y multiplicación O (2^n)

**3.6**

**N:** el tamaño de la matriz que recibe el algoritmo.

**M:** el tamaño de la matriz (nums.length) menos la posición real en la matriz (inicio).

**4. SIMULACRO PARCIAL**

* 4.2.1 A.
* 4.2.2 C.
* 4.3 A.
* 4.2.1 A.
* 4.2.2 A, C.
* 4.3 B.
* 4.4.1 C.
* 4.5.1 A.
* 4.5.2 B.
* 4.6 A.
* 4.7 E.
* 4.8 B.
* 4.6.1 sumaAux(n, i + 2)
* 4.6.2 sumaAux(n, i + 1)
* 4.7.1 comb(S, i + 1, t – S[i]
* 4.7.2 comb(S, i + 1, t)
* 4.8 C.
* 4.9 A.
* 4.10 C