**Laboratorio Nro. 1  
Recursión**

|  |  |
| --- | --- |
| **Thomas Nieto Valencia**  Universidad Eafit  Medellín, Colombia  tnietov@eafit.edu.co | **Miguel Ángel Echavarría Ocampo**  Universidad Eafit  Medellín, Colombia  maechavaro@eafit.edu.co |

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

**3.1** La complejidad figura en que el peor de los casos sería cuando ninguno de los subconjuntos es igual entonces debe recorrer la totalidad de caracteres de ambos Strings entonces se harían 2 llamados recursivos y en cada llamado recursivo se disminuye n y m que respectivamente son los tamaños de cada String y al ser 2 variables es donde empleamos m que sería la suma de n y m entonces al reducir las ambas p sería tomaría el valor de -2, en los condicionales que serían mejores casos y solo se reduciría una de las 2 variables a p se le restaría -1

Complejidad: T(p) =

**3.2**

|  |  |
| --- | --- |
| **Caracteres de cada String** | **Tiempo(milisegundos)** |
| 9 | 0 |
| 11 | 0 |
| 13 | 1 |
| 14 | 2 |
| 15 | 3 |
| 16 | 6 |
| 17 | 11 |
| 18 | 11 |
| 19 | 102 |
| 20 | 124 |
| 25 | 1088 |
| 26 | 70573 |
| 27 | 144598 |
| 28 | 150581 |
| 29 | 170059 |
| 30 | 483517 |

Cuando calcule una cadena de 300000 caracteres no terminaría, la tendencia sería al infinito

**3.3** No sería apropiado pues estas cadenas suelen ser muy largas y el programa no terminaría de ejecutar el programa

**3.5 I. Recursion-1**

**Countevens:** t(n)=n^2-(c+n^-1)

**No14:** t(n) = n

**Matchup:** T(n)=|n|

**Sum13:** t (n)=n! N no mayor igual a 13

**Fizzbuzz:** t(n)=n

**II. Recursion-2**

**groupSum6:** T(n) = C_4 (2^n - 1) + c_1 2^(n - 1) (where c_1 is an arbitrary parameter)

## groupNoAdj:T(n) = -C_4 + c_1 F_n + c_2 L_n (where c_1 and c_2 are arbitrary parameters)

## groupSum5: T(n) = C_4 (2^n - 1) + c_1 2^(n - 1) (where c_1 is an arbitrary parameter)

## groupSumClump:T(n) = C_4 (2^n - 1) + c_1 2^(n - 1) (where c_1 is an arbitrary parameter)

## splitArray: T(n) = C_4 (2^n - 1) + c_1 2^(n - 1) (where c_1 is an arbitrary parameter)

**3.6**

**I. Recursion-1**

**Countevens:** n depende de la longitud del arreglo, se recorre todo el arreglo en el peor de los casos.

**No14:** n depende de la longitud del arreglo, en el peor de los casos recorre todo el arreglo.

**Matchup:** n depende de la longitud de los dos arrays, en el peor de los casos, recorre todo el array y ningún elemento difiere.

**Sum13:** n depende de la longitud del array, en el peor de los casos recorre todo el array y todos los elementos son mayores o iguales a 13.

**Fizzbuzz:** n depende de la longitud del array, en el peor de los casos recorre todo el array para asignar e indexar

**II. Recursion-2** En todos los ejercicios se trataba de arreglos, por consiguiente, en cada uno de estos la complejidad del peor de los casos depende de la cantidad de posiciones que tiene cada uno de estos pues el peor de los casos sería recorrer cada una de estas complejidades.

**groupSum6:** n depende de la longitud del arreglo, se hacen dos llamados recursivos n veces en el peor de los casos.

**groupNoAdj:** n depende de la longitud del arreglo, se hacen dos llamados recursivos uno de estos n veces y el otro n/2 en el peor de los casos. Por otra parte, al resolver la ecuación en Wolfram Alpha nos devuelve L y F los cuales son el número de Lucas y el de Fibonacci Respectivamente de los cuales el subíndice n es la posición de cada uno.

**groupSum5:** n depende de la longitud del arreglo, se hacen dos llamados recursivos n veces en el peor de los casos.

**groupSumClump:** n depende de la longitud del arreglo, se hacen dos llamados recursivos n veces en el peor de los casos.

**SplitArray:** n depende de la longitud del arreglo, se hacen dos llamados recursivos n veces en el peor de los casos.

***4) Simulacro de Parcial***

* 1. 1. B
     2. C
     3. A
     4. A
     5. a) verdadera

b) falsa

c) verdadera

d) falsa

* 1. *B*
  2. 1. C
  3. 1. A
     2. A
  4. A