**Laboratorio Nro. 1  
laboratorio recursión**

|  |  |
| --- | --- |
| **Juan José Zuluaga Bedoya**  Universidad Eafit  Medellín, Colombia  jjzuluagab@eafit.edu.co | **Juan José Wilches Rivas**  Universidad Eafit  Medellín, Colombia  jjwilchesr@eafit.edu.co |

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

**3.1**

**T(m,n) = T(m,n-1)+T(m-1,n)**

**3.2**

|  |  |
| --- | --- |
| n | t(n) |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |
| 8 | 2 |
| 9 | 10 |
| 10 | 6 |
| 11 | 15 |
| 12 | 60 |
| 13 | 216 |
| 14 | 943 |
| 15 | 3634 |
| 16 | 13835 |
| 17 | 58269 |
| 18 | 219541 |
| 19 | 884880 |

**3.3** no, como podemos ver en el ejercicio anterior al tener características de una función exponencial el programa se demoraría demasiado para cadenas de caracteres de 300.000.

**3.5**

**groupSum6:**

C1 = 5 = T (0)

C2 = 7

**Mejor caso:** T(n) = T(n-1) + C2

T(n) = C2\*n + C1

**Pero caso**: T(n) = T(n-1) + T(n-1) + C2

T(n) = C2(2^n-1) +C1\*2^(n-1)

n es el número de espacios restantes para terminar el arreglo

**groupSumClump:**

C1 = 6 = T (0)

C2 = 10

C3 = 14

C4 = 31

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

T(n)=C3(2^n-1) +C1\*2^(n-1) (mejor caso)

T(n) = T(n-2) + T(n-2) + C\_3

T(n)=2^ (1/2) \*(C2(-1) ^(n)+ C1) - C3 (peor caso)

n es el número de espacios restantes para terminar el arreglo

**groupSum5**

C1 = 6

C2 = 10

C3 = 13

C4 = 16

C5= 18

**Caso intermedia**: T(n) = T(n-2) + C\_3

T(n) = -(1/4) \*C3\*((-1) ^(2n)-2n) + C2\*(-1) ^(n) +C1

**mejor caso:** T(n) = T(n-1) + C\_4

T(n) = C4\*n + C1

**Peor caso:** T(n) = T(n-1) + T(n-1) + C\_5

T(n)=C5(2^n-1) +C1\*2^(n-1)

n es el número de espacios restantes para terminar el arreglo

**groupNoAdj**

C1 = 4

C2 = 9

C3 = 18

C4 = 18

**Mejor caso:** T(n) = T(n-2) + T(n-1) + C\_3

T(n) = -C3+ C1\*F(n) + C2\*L(n)

with F(n) the nth Fibonacci number and L(n) the nth Lucas number

**Peor caso:** T(n) = T(n-1) + T(n-1) + C4

T(n)= C4(2^n-1) +C1\*2^(n-1)

n es el número de espacios restantes para terminar el arreglo

**splitArray**

C1 = 7

C= 16

T(n) = T(n-1) + T(n-1) +C\_2

T(n)= C2(2^n-1) +C1\*2^(n-1)

n es el número de espacios restantes para terminar el arreglo

**Factorial**

C1=3

Cuando n= factorial

T(n)= c\_2 + T(n-1)

T(n) = c\_2 n + c\_1

RSolve[T[n] == Subscript[c, 2] + T[-1 + n], {T[n]}, n]

**bunnyEars**

C1=3

Cuando n= numero de filas

T(n)= c1 + T(n-1) 🡪 T(n) = c1 n + c\_1

RSolve[T[n] == c1 + T[-1 + n], {T[n]}, n],

**bunnyEars2**

C1= 3

C2=13

Cuando n= bunnies

T(n) = -c\_2 + c\_1 F\_n + c\_2 L\_n

RSolve[T[n] == Subscript[c, 2] + T[-2 + n] + T[-1 + n], {T[n]}, n]

**Triangle**

C1= 2

C2=3

Cuando n= numero de filas

T(n)= T(n-1) + c\_2--> T(n) = c\_2 n + c\_1

RSolve[T[n] == Subscript[c, 2] + T[-1 + n], {T[n]}, n]

**CountHi**

C1=4

C2=8

C3=9

Siendo n= tamaño de arreglo

T(n)=c\_3 +T(n-1) -->T(n) = c\_3 n + c\_1

RSolve[T[n] == Subscript[c, 3] + T[-1 + n], {T[n]}, n]

***4) Simulacro de Parcial***



1. A

2. A

3. A



1. A) Verdadero
2. A) Falso

B)Falso

C) Falso

***1. B***

int lucas(int n){

        if(n == 1)

        return 2;

        if(n == 2)

        return 1;

        return lucas(n-1) +lucas(n-2);

         }

* 1. C

static boolean isPal(String s) {

02 if(s.length() == 0 || s.length() == 1)

03 return true;

04 if(s.charAt(0)== s.charAt(s.length())

05 return isPal(s.substring(1, s.length()-1));

06 //else

07 return false;

08}