Laboratorio Nro. 1
Recursión y complejidad.

John Esteban Castro Ramírez
Universidad Eafit
Medellín, Colombia
jecastror@eafit.edu.co

Carlos Gustavo Vélez Manco
Universidad Eafit
Medellín, Colombia
cgvelezm@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1

$$T(p) = \begin{cases} C1 & if\ (longitud\ de\ chain1 = 0\ or\ \ longitud\ de\ chain2 = 0) \\ C2 + T(p-2) & if\ chain1[len-1] = chain2[len-1] \\ C3 + T(p-1) + T(p-1) & otherwise \end{cases}$$

Donde p=n+m.
Y tomando el peor de los casos que sería el último, debido a que se deben de hacer dos llamados recursivos, la solución en WolframAlpha da como resultado

$$C3 * (2^p - 1) + C1 * (2^{p-1})$$

Y luego efectuamos los siguientes pasos:

1. $T(p) es\ \ O(C3 * (2^p - 1) + C1 * (2^{p-1}))$ **------>(Notación O)**
2. $O\big(C3 * (2^p - 1) + C1 * (2^{p-1})\big) = O((2^p - 1) + (2^{p-1}))$ **--->(Regla del producto)**
3. $O((2^p - 1) + (2^{p-1})) = O(2^p + 2^p)$ **----->(Regla de la suma)**
4. $O(2^p + 2^p) = \ O(2(2^p))$ **------>(Factor común)**
5. $O\big(2(2^p)\big) = O(2^p)$ **------>(Regla del producto)**

Entonces T(p) es O(2^p) donde p es la suma de las longitudes de las cadenas.

Así, podemos ver que este algoritmo al tener una complejidad exponencial para el peor de los casos es muy efectivo para pequeñas longitudes de ambas cadenas; pero al ejecutarlo con tamaños muy grandes, tardará un tiempo demasiado considerable.

3.2

Tamaño del problema (p)(Suma de las longitudes de ambas cadenas)	Tiempo de complejidad T(p) (En ms)
0	0
2	0
4	0
6	0
8	0
10	0.000997305
12	0.000522137
14	0.007563591
16	0.016020536
18	0.040824175
20	0.225693703
22	1.593865871
24	3.783751965
26	8.569449663
28	34.91284418
30	173.4508719
32	1350.43354
34	2207.359112
36	2902.453211

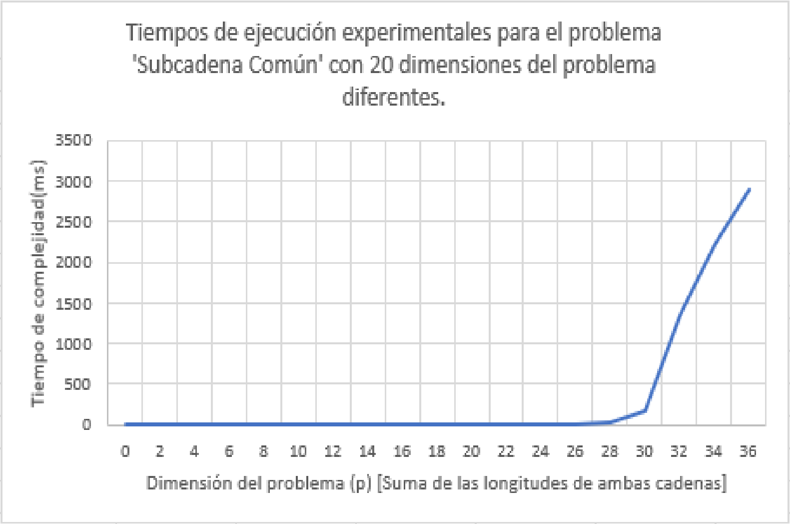


Figura 1. Gráfica en Excel de los tiempos de ejecución experimentales para el problema 'Subcadena Común' con 20 dimensiones del problema diferentes.

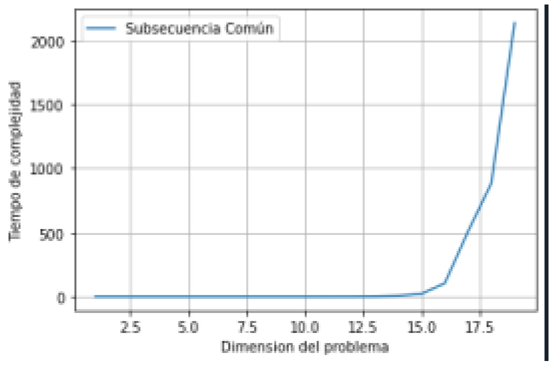


Figura 2. Gráfica en Python con 20 tamaños de problema diferentes.

Como se puede ver la gráfica experimental coincide con la complejidad calculada de manera teórica, (2^p). Además, el algoritmo para tamaños pequeños de cadenas funciona muy bien, pero a medida que las cadenas son más grandes el tiempo va a ser

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

considerablemente extenso, ya que su tiempo de complejidad lo modela una función exponencial.
El tiempo estimado que se demorara para encontrar la longitud de la subsecuencia común más larga entre dos ADNs mitocondriales que tienen alrededor de 300000 caracteres cada uno, es decir, en este caso p=300000+300000=600000, entonces en el peor de los casos, el tiempo que tarda es de 2^(600000), lo cual es aproximadamente:

9940274755017732308941916648000732363805000596148498825803720094675126660
6863050719339624340948809626687301624793165989296999201279522312346618451
8984381892156769367157420437232952082971877007817355388947760524209677465
6916273429522025988651761514473524842848677043328826562349255660335044043
8588442301685461646366553417437554881129667882576190646828538076295591008
3328210388951406167628182540751734880734136409213430134222957037382109235
9640024941900032612393215920912375795765294332910872774215815959403232674
1012369505814210234717588440851773138523647948060605332426103057166076811
9106211343048441730145685149245449633961694871618370837452723354826623308
7180561361873548974019035559 (....) **ms**

Lo cual como podemos ver, es un tiempo bastante significativo.

3.3

No, la complejidad del algoritmo no es apropiada para encontrar la longitud de la subsecuencia común más larga entre dos cadenas de ADNs mitocondriales como los que se encuentran en la carpeta de Datasets, ya que por ejemplo el archivo ‘*Acipenser transmontanus mitocondrial DNA.txt*’ tiene un total de 16693 caracteres y el archivo ‘Anoplogaster cornuta mitochondrial DNA.txt’ tiene un total de 16501 caracteres, según un contador de caracteres online, entonces en este caso p sería p=16693+16501= 33194, y dado que su complejidad para el peor de los casos es de 2^p, es decir, a medida que aumenta la dimensión del problema, el algoritmo duplica su complejidad; entonces este algoritmo tardaría 2^33194 ms, lo cual es aproximadamente:

2452878689992092512980740107446975902732410134737685506301652446565000592
6417444388482821600284106670946580200580091359022717238822453370617391804
3946269777358306373459033953318104872316912670502832817596021054035788801
8929813812885722756677046299963174596631211460052905335113405436720463888
8643514672857299431048772463978927817690875608673951976580169990848178712
3975938636183756800830334471348925255276237201697663569936775686484938020
9453906326793843755501589397810441058584734527744742804667202139274816167
8612426848322961824120783182446600208800077251407142909376270904778336639
0812417627230434571638457983612995054157005038786840936404680640371554161
4733359159314571251975429929411864943723648870936224413232601787720714945
9942263682196837394500877827587662650381997265592971983102103577677351643
1680964637046218489509128453 (...) **ms**

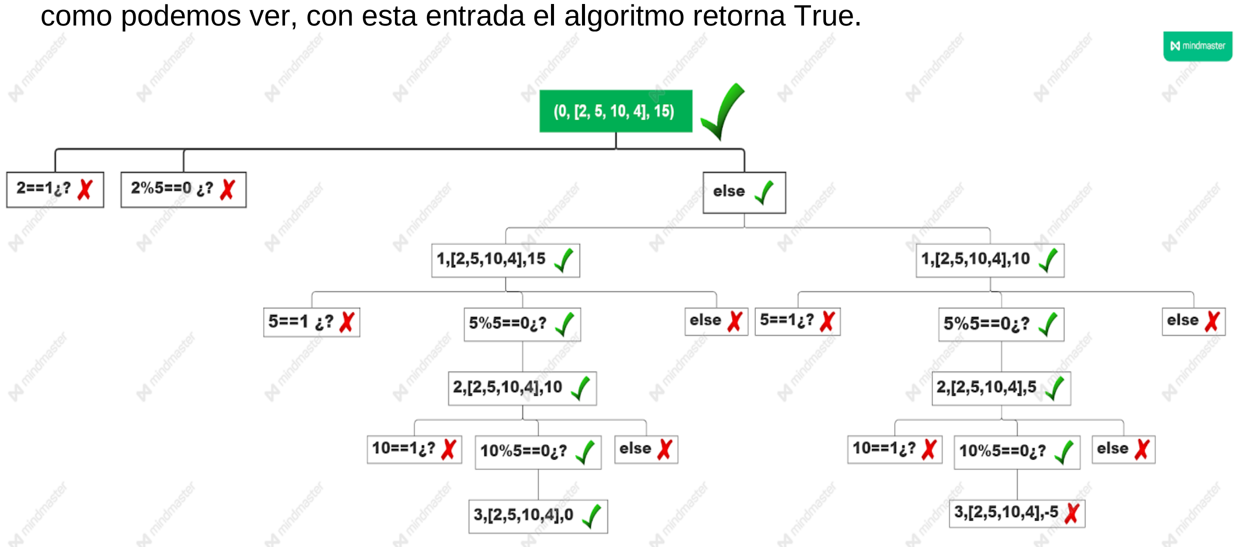
Que como podemos ver es un tiempo bastante considerable y para nada factible.

3.4

El algoritmo GroupSum5 funciona de la siguiente manera, lo que pretende el algoritmo es que dado un arreglo de enteros y un “objetivo”, arroje si existe o no alguna posibilidad de que al sumar números del arreglo dé como resultado exacto el valor del objetivo, pero con 2 restricciones: Todos los números múltiplos de 5 del arreglo se deben de incluir en la suma y además, si el número en la posición consecutiva al múltiplo de 5 es un 1, no se puede incluir en la suma.

Para su ejecución, se toma una implementación similar al algoritmo ‘GroupSum’ pero con ciertas modificaciones; la condición de parada o base del algoritmo recursivo se ejecuta cuando el parámetro ‘start’ es mayor o igual a la longitud del arreglo, es decir, cuando ya ha recorrido todas las posiciones del arreglo evaluando todos los casos; y la condición recursiva tiene varios caminos, primero toma la condición de que si el número en la posición que estamos evaluando es 1, el antecesor es un múltiplo de 5 y el parámetro start es mayor a 0 (esto para evitar salirnos de los límites del arreglo, suponiendo que el 1 puede estar en la primera posición del arreglo), entonces simplemente aumentaremos el parámetro start sin restar el número 1; luego evalúa la condición de que si el número en x posición es múltiplo de 5 obligatoriamente lo tendrá que restar y en caso de que no cumpla ninguna de estas condiciones optará por una disyunción, una restando el número al arreglo y otra sin restarlo, abarcando así todos los casos posibles. Ya al llegar a la condición de parada, si algún resultado de estas restas da 0, el algoritmo nos arrojará un True, a causa de las disyunciones, indicando que si se puede realizar la suma con los números del arreglo de tal forma que se alcance el objetivo, en caso contrario arrojará False.

A continuación se presenta el proceso paso a paso del algoritmo con la entrada (0,[2,5,10,4],15), los iconos verdes representan el retorno True y las X el retorno False, como podemos ver, con esta entrada el algoritmo retorna 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

3.5
RECURSIÓN 1.

- **BunnyEars:**

$$T(n) = \begin{cases} C1 & if \;\; n = 0 \\ C2 + T(n-1) & if \; n > 0 \end{cases}$$

Y tomando el peor de los casos, que sería cuando n>0, solucionándola en WolframAlpha da como resultado C2n+C1 y paso seguido efectuamos los siguientes pasos:

1. T(n) es O(C2n+C1) ----(Notación O)
2. O(C2n+C1) =O(C2n) ----(Por regla de la suma)
3. O(C2n) =O(n) ------(Por regla del producto)

Entonces T(n) es O(n), donde n es el número de conejos.
Así podemos ver que el método recursivo en general es medianamente bueno para grandes volúmenes de datos ya que no tarda tanto tiempo.

- **BunnyEars2:**

$$T(n) = \begin{cases} C1 & if \; n = 0 \\ C2 + T(n-1) & if \; n\%2 = 0 \\ C3 + T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería cuando n es diferente de 0, al solucionarla en WolframAlpha da como resultado C3n+C1 y luego, efectuamos los siguientes pasos:

1. T(n) es O(C3n+C1) ---(Notación O)
2. O(C3n+C1) =O(C3n) ---(Por regla de la suma)
3. O(C3n) =O(n) -----(Por regla del producto)

Entonces T(n) es O(n), donde n es el número de conejos.
Así podemos ver que el método recursivo en general es medianamente bueno para grandes volúmenes de datos ya que no tarda tanto tiempo.

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

- **Triangles:**

$$T(n) = \begin{cases} C1 & if\ n = 0 \\ C2 + T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería cuando n es diferente de 0, al solucionarla en WolframAlpha da como resultado C2n+C1 y luego, efectuamos los siguientes pasos:

1. T(n) es O(C2n+C1)--->(Notación O)
2. O(C2n+C1) =O(C2n) --->(Por regla de la suma)
3. O(C2n) =O(n) ----->(Por regla del producto)

Entonces T(n) es O(n), donde n es el número de filas del triángulo.
Así podemos ver que el método recursivo en general es medianamente bueno para grandes volúmenes de datos ya que no tarda tanto tiempo.

- **SumDigits:**

$$T(n) = \begin{cases} C1 & if\ n = 0 \\ C2 + T\left(\frac{n}{10}\right) & otherwise \end{cases}$$

Y tomando el peor de los casos que sería cuando n es diferente de 0, al solucionarla en WolframAlpha da como resultado:

$$\frac{C2\log(n)}{log10} + C1$$

y luego efectuamos los siguientes pasos:

1. T(n)=c2log10(n)+C1--->(Cambio de base)
2. T(n) es O(c2log10(n)+C1)---->(Notación O)
3. O(c2log10(n)+C1)=O(c2log10(n))----->(Por regla de la suma)
4. O(c2log10(n))=O(log10(n))---->(Por regla del producto)

Entonces T(n) es O(log10(n)), donde n es el número que estamos dividiendo.
Así podemos ver que el método recursivo en general es muy bueno para grandes volúmenes de datos, es mejor que los casos pasados donde su complejidad era O(n) ya que tomará menos tiempo de ejecución y realización de operaciones.

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

- **PowerN:**

$$T(n)=\begin{cases} C1 & if\ n=1 \\ C2+T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería cuando n es diferente a 1, al solucionarla en WolframAlpha da como resultado C2n+C1 y luego, efectuamos los siguientes pasos:

1.T(n) es O(C2n+C1)--->(Notación O)
2.O(C2n+C1)=O(C2n)--->(Por regla de la suma)
3.O(C2n)=O(n)----->(Por regla del producto)

Entonces T(n) es O(n), donde n es el número al cual se eleva la base.
Así podemos ver que el método recursivo en general es medianamente bueno para grandes volúmenes de datos ya que no tarda tanto tiempo.

RECURSIÓN 2.

- **GroupSum5.**

$$T(n)=\begin{cases} C1 & if\ n \geq longitud\ del\ arreglo\ nums \\ C2+T(n-1) & if\ nums[n]=1\ and\ nums[n-1]\%5=0\ and\ n>0 \\ C3+T(n-1) & if\ nums[n]\%5=0 \\ C4+2*T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería el último, el del else debido a que se deben hacer dos llamados recursivos a diferencia de los otros casos; así al ingresar esta ecuación a WolframAlpha da como resultado: $C4*(2^n-1)+C1*(2^{n-1})$
Y luego al efectuar los siguientes pasos:

1. $T(n)es\ \ O(C4*(2^n-1)+C1*(2^{n-1}))$ **------>(Notación O)**
2. $O\left(C4*(2^n-1)+C1*(2^{n-1})\right)=O((2^n-1)+(2^{n-1}))$ **--->(Por regla del producto)**
3. $O((2^n-1)+(2^{n-1}))=O(2^n+2^n)$ **--->(Por regla de la suma)**
4. $O(2^n+2^n)=\ O(2(2^n))$ **------>(Factor común)**
5. $O\left(2(2^n)\right)=O(2^n)$ **------>(Regla del producto)**

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

Entonces T(n) es O(2^n), donde n es es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro ‘start’ hasta el final del arreglo.
Así podemos ver que el método recursivo es bueno solo para dimensiones del problema pequeños, pero para grandes dimensiones tomará tiempos muy grandes y será poco eficaz.

- **GroupSum6:**

$$T(n)=\begin{cases} C1 & if\ n \geq longitud\ del\ arreglo\ nums \\ C2 + T(n-1) & if\ nums[n] = 6 \\ C3 + 2 * T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería el último, el del else debido a que se deben hacer dos llamados recursivos a diferencia de los otros casos; así, al ingresar esta ecuación a WolframAlpha da como resultado:

$$C3 * (2^n - 1) + C1 * (2^{n-1})$$

Y luego al efectuar los siguientes pasos:

1. $T(n) es\ \ O(C3 * (2^n - 1) + C1 * (2^{n-1}))$ **------>(Notación O)**
2. $O\left(C3 * (2^n - 1) + C1 * (2^{n-1})\right) = O((2^n - 1) + (2^{n-1}))$ **--->(Regla del producto)**
3. $O((2^n - 1) + (2^{n-1})) = O(2^n + 2^n)$ **----->(Regla de la suma)**
4. $O(2^n + 2^n) = O(2(2^n))$ **------->(Factor común)**
5. $O\left(2(2^n)\right) = O(2^n)$ **------>(Regla del producto)**

Entonces T(n) es O(2^n), donde n es es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro ‘start’ hasta el final del arreglo.
Así podemos ver que el método recursivo es bueno solo para dimensiones del problema pequeños, pero para grandes dimensiones tomará tiempos muy grandes y será poco eficaz.

- **SplitArray:**

$$T(n) = \begin{cases} C1 & if\ n \geq longitud\ del\ arreglo\ nums \\ C2 + 2 * T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería el último, el del else debido a que se deben hacer dos llamados recursivos a diferencia de los otros casos; así, al ingresar esta ecuación a WolframAlpha da como resultado:

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

$$C2 * (2^n - 1) + C1 * (2^{n-1})$$

Y luego al efectuar los siguientes pasos:

1. $T(n) es \;\; O(C2 * (2^n - 1) + C1 * (2^{n-1}))$ **---->(Notación O)**
2. $O\left(C2 * (2^n - 1) + C1 * (2^{n-1})\right) = O((2^n - 1) + (2^{n-1}))$ **---->(Regla del producto)**
3. $O((2^n - 1) + (2^{n-1})) = O(2^n + 2^n)$ **----->(Regla de la suma)**
4. $O(2^n + 2^n) = \; O(2(2^n))$ **---->(Factor común)**
5. $O\left(2(2^n)\right) = O(2^n)$ **----->(Regla del producto)**

Entonces T(n) es O(2^n), donde n es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'i' hasta el final del arreglo.
Así podemos ver que el método recursivo es bueno solo para dimensiones del problema pequeños, pero para grandes dimensiones tomará tiempos muy grandes y será poco eficaz.

- **SplitOdd10:**

$$T(n) = \begin{cases} C1 & if\ n \geq longitud\ del\ arreglo\ nums \\ C2 + 2 * T(n-1) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería el último, el del else debido a que se deben hacer dos llamados recursivos a diferencia de los otros casos; así, al ingresar esta ecuación a WolframAlpha da como resultado:

$$C2 * (2^n - 1) + C1 * (2^{n-1})$$

Y luego al efectuar los siguientes pasos:

1. $T(n) es \;\; O(C2 * (2^n - 1) + C1 * (2^{n-1}))$ **---->(Notación O)**
2. $O\left(C2 * (2^n - 1) + C1 * (2^{n-1})\right) = O((2^n - 1) + (2^{n-1}))$ **---->(Regla del producto)**
3. $O((2^n - 1) + (2^{n-1})) = O(2^n + 2^n)$ **----->(Regla de la suma)**
4. $O(2^n + 2^n) = \; O(2(2^n))$ **---->(Factor común)**
5. $O\left(2(2^n)\right) = O(2^n)$ **----->(Regla del producto)**

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

Entonces T(n) es O(2^n), donde n es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'i' hasta el final del arreglo.
Así podemos ver que el método recursivo es bueno solo para dimensiones del problema pequeños, pero para grandes dimensiones tomará tiempos muy grandes y será poco eficaz.

- **GroupNoAdj:**

$$T(n)=\begin{cases} C1 & if \quad n \geq longitud\ del\ arreglo\ nums \\ C2 + T(n-1) + T(n-2) & otherwise \end{cases}$$

Y tomando el peor de los casos, que sería el último, el del else debido a que se deben hacer dos llamados recursivos a diferencia de los otros casos; así, al ingresar esta ecuación a WolframAlpha da como resultado:

$$-C2 + C1Fn + C2Ln$$

Lo cual, da como resultado una complejidad de O(2^n), donde n es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'start' hasta el final del arreglo.
Así podemos ver que el método recursivo es bueno solo para dimensiones del problema pequeños, pero para grandes dimensiones tomará tiempos muy grandes y será poco eficaz.

3.6

Recursión 1.

- **BunnyEars:** 'n' o 'bunnies' es el número de conejos, ya que cada vez se llama a la recursión con un conejo menos.
- **BunnyEars2: '**n' o 'bunnies' es el número de conejos, ya que cada vez se llama a la recursión con un conejo menos.
- **Triangles:** 'n' o 'rows' es el número de filas del triángulo, ya que cada vez se llama al método recursivo con una fila menos para poder calcular el número total de triángulos.
- **SumDigits:** 'n' es el número, ya que cuando se llama al método recursivo dividiendo el número por 10 se le está reduciendo un dígito al mismo, hasta que su división por 10 arroje 0.
- **PowerN:** 'n' es el número al cual se está elevando la base, en otras palabras, es el exponente; por cada llamado recursivo se reduce el número de este exponente hasta que llegue a 1.

Recursión 2.

- **GroupSum5:** 'n' es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'start' hasta el final del arreglo. Se puede notar que por cada llamado recursivo se aumenta 'n' o 'start' en 1, lo que reduce la distancia desde start hasta el final del arreglo o la longitud de este.

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

- **GroupSum6:** Al igual que en 'GroupSum5', 'n' es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'start' hasta el final del arreglo. Se puede notar que por cada llamado recursivo se aumenta 'n' o 'start' en 1, lo que reduce la distancia desde start hasta el final del arreglo o por así decirlo, la longitud de este.
- **GroupNoAdj:** 'n' es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'start' hasta el final del arreglo. Se puede notar que por cada llamado recursivo se aumenta 'n' o 'start' en 1 o 2 según sea el caso,ya que no se pueden sumar números consecutivos respecto a la posición en el arreglo, lo que reduce la distancia desde start hasta el final del arreglo o por así decirlo, la longitud del mismo.
- **SplitArray:** 'n' es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'i' hasta el final del arreglo. Se puede notar que por cada llamado recursivo se aumenta 'n' o 'i' en 1, para sumarle el número de dicha posición del arreglo al parámetro 'a' o 'b' según sea el caso, lo que reduce la distancia desde 'n' o 'i' hasta el final del arreglo o por así decirlo, la longitud de este.
- **SplitOdd10:** 'n' al igual que en 'SplitArray', es la longitud del arreglo, o en otras palabras es la distancia en términos de posiciones que hay desde el parámetro 'i' hasta el final del arreglo. Se puede notar que por cada llamado recursivo se aumenta 'n' o 'i' en 1, para sumarle el número de dicha posición del arreglo al parámetro 'a' o 'b' según sea el caso, lo que reduce la distancia desde 'n' o 'i' hasta el final del arreglo o por así decirlo, la longitud de este.

4) Simulacro de Parcial

4.1

1. A
2. C
3. A

4.2

1. **Línea 9:** floodFillUtil(screen,x+1,y+1,prevC,newC,N,M)
   Línea 10: floodFillUtil(screen,x+1,y-1,prevC,newC,N,M)

2. **Línea 11:** floodFillUtil(screen, x-1,y+1, prevC,newC,N,M)
   Línea 12: floodFillUtil(screen,x-1,y-1,prevC,newC,N,M)

3. Para el peor de los casos, que sería el último, en donde se deben hacer 8 llamados recursivos, la complejidad sería:

   T(p)=T(p+2)+T(p)+T(p)+T(p-2)+T(p+1)+T(p-1)+T(p+1)+T(p-1)+C

   La cual no se puede solucionar en WolframAlpha, pero debido a que se hacen 8 llamados recursivos se podría predecir que su complejidad para el peor de los casos sería **O(8^n)**

4.3 *B*

4.4 return Lucas(n-1)+Lucas(n-2)

1. C

4.5

1. A
2. B

4.6 A

4.7

1. return sumaAux(n,i+2)
2. return (n.charAt(i)-‘0’)+sumaAux(n,i+1)

4.8

1. No encontramos una respuesta igual a las opciones de selección múltiple, ya que su complejidad para el peor de los casos sería:

   T(n)=T(n-3)+T(n-5)+T(n-7)+C

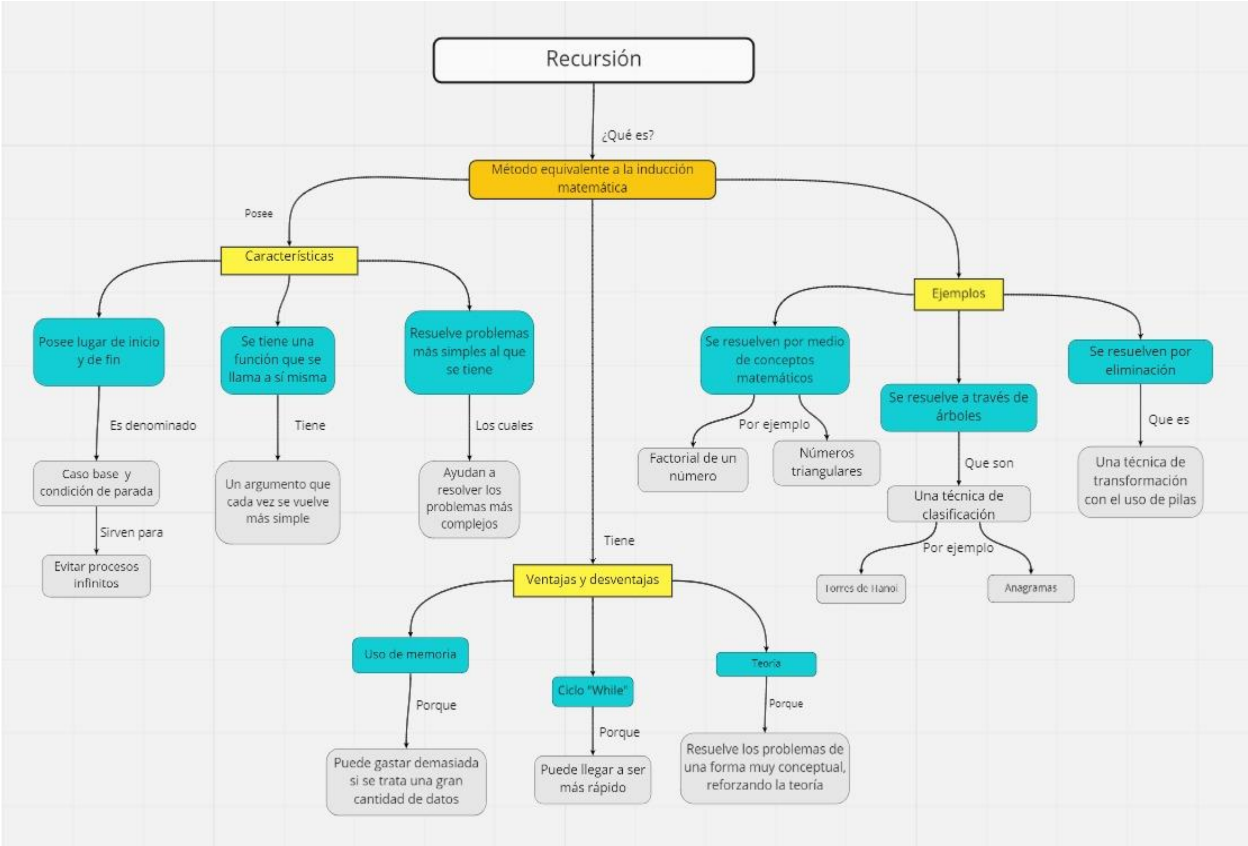
4.9 B

4.10 return Lucas(n-1)+Lucas(n-2)

1. C

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

5) *Lectura recomendada (opcional)*



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