ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 1
Escribir el tema del laboratorio

Santiago Ochoa Castaño
Universidad Eafit
Medellín, Colombia
sochoac1@eafit.edu.co

Miguel Ángel Zapata Jiménez
Universidad Eafit
Medellín, Colombia
mazapataj@eafit.edu.co

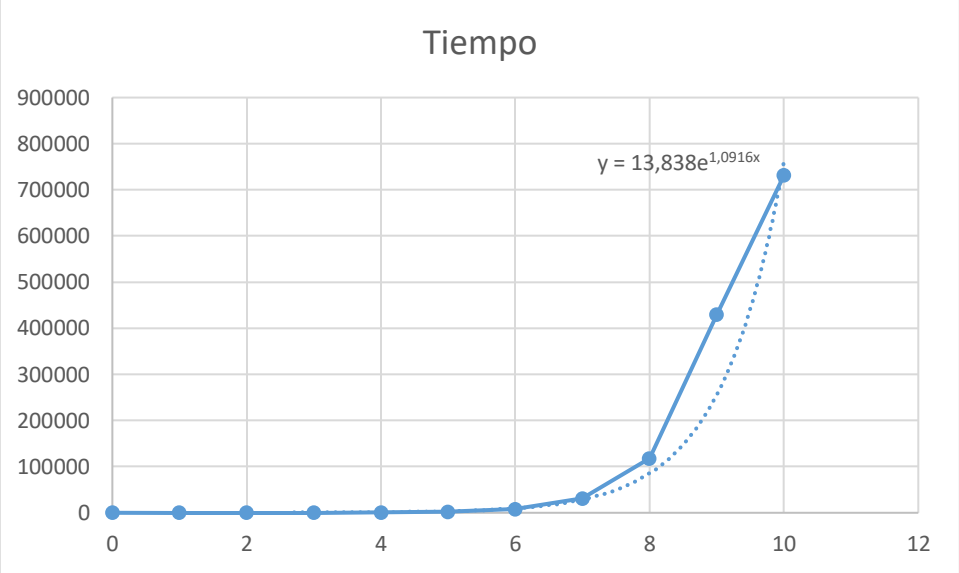
3) Simulacro de preguntas de sustentación de Proyectos

3.1 Complejidad asintótica para el peor de los casos

- **Ecuación de complejidad:**

T(n, m) = C_3 + T(n-1, m) + T(n, m-1) / De esta manera no es posible solucionar la ecuación de recurrencia por lo que hay que realizar la siguiente: **p = n + m**
T(p) = C_3 + T(p-1) + T(p-1) / En notación O: O(2n)
$T(p) = C_3\,(2^p - 1) + c_1\,2^{p-1}$ **(solución)**

3.2 Grafica de complejidad del punto 1.1



Como se puede observar en la gráfica el tiempo de ejecución de los datos crece de manera exponencial siendo esto una de las complejidades más inadecuadas a la hora de analizar

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

una gran cantidad de datos. En el caso que se plantea para este laboratorio cada String tiene una cantidad de 300.000 caracteres lo cual a la hora de ser ejecutado consume mucho tiempo, ya que el tiempo que se demora en ejecutar cada operación crece de manera exponencial como se mencionó anteriormente. Podemos llegar a la conclusión de que el algoritmo implementado es un poco ineficiente por su alta complejidad.

3.3 ¿La complejidad del algoritmo del ejercicio 1.1 es apropiada para encontrar subsecuencia común más larga entre ADN mitocondriales como los de los dataset?

La ecuación que se obtuvo en el ejercicio 1.1 mostro una complejidad exponencial en el crecimiento del tiempo con respecto a cada dato proporcionado. Al analizar se comprende que un problema que sea de complejidad exponencial es un problema que tarda mucho en solucionarse y más si los datos proporcionados son muy grandes como lo es en este caso, por lo que a la hora de encontrar la subsecuencia común más larga entre dos String de 300.000 caracteres cada uno, el algoritmo tardaría un tiempo muy largo para dar una posible solución al problema. En conclusión, si se posee un algoritmo con complejidad exponencial respecto a su tiempo de ejecución, se estaría hablando de un programa muy inadecuado para solucionar problemas como el que se planteó.

3.5 CodingBat ecuaciones de complejidad

Recursión 1

- **Fibonacci:** T(n) = C_3 + T(n-1) + T(n-2) / En notación O: O(2^n)
  $T(n) = -C_3 + c_1 F_n + c_2 L_n$ **(Solución)**
- **CountHi:** T(n) = C_4 + T(n-1) / En notación O: O(n)
  $T(n) = C_4 n + c_1$ **(Solucion)**
- **Triangle:** T(n) = C_2 + T(n-1) / En notación O: O(n)
  $T(n) = C_2 n + c_1$ **(Solucion)**
- **StringClean:** T(n) = C_4 + T(n-1)/ En notación O: O(n)
  $T(n) = C_4 n + c_1$ **(Solucion)**
- **Array 11:** T(n) = C_3 + T(n-1)/ En notación O: O(n)
  $T(n) = C_3 n + c_1$ **(Solucion)**

Recursion 2

- **GroupSum5:** T(n) = C_4 + 2*T(n-1) /En notación O: O(2^n)
  $T(n) = C_4 (2^n - 1) + c_1 2^{n-1}$ **(Solucion)**
- **GroupSum6:** T(n) = C_3 + 2*T(n-1) /En notación O: O(2^n)
  $T(n) = C_3 (2^n - 1) + c_1 2^{n-1}$ **(Solucion)**
- **SplitOdd10:** T(n) = C 2 + 2*T(n-1) /En notación O: O(2^n)
  $T(n) = C_2 (2^n - 1) + c_1 2^{n-1}$ **(Solucion)**

- **SplitArray:** T(n) = C_2 + 2*T(n-1) / En notación O: (2^n)

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

$T(n) = C_2\,(2^n - 1) + c_1\,2^{n-1}$ **(Solucion)**

- **GroupNoAdj:** T(n) = C_2 + T(n-1) + T(n-2)/ En notación O: (2^n)
  $T(n) = -C_2 + c_1\,F_n + c_2\,L_n$ **(Solucion)**

3.6 Explique con sus palabras las variables del calculo de complejidad del ejercicio anterior

- **Fibonacci:** En el cálculo la variable n sería el equivalente del número que deseamos encontrar en Fibonacci. Se encuentra la presencia de 3 constantes en donde cada una se asocia a una serie de operaciones realizadas dentro de un condicional "if". Se puede apreciar que la complejidad de este algoritmo está dada por $O(2^n)$ lo que indica un consumo de tiempo exponencial.

- **CountHi:** En el cálculo la variable n representaría el tamaño del String que es ingresado en el algoritmo. La C está asociada a 4 constantes que están dadas por unas operaciones realizadas dentro de unos condicionales "if". O(n) nos permite comprender que el algoritmo tiene un consumo de tiempo lineal.

- **Triangle:** En el cálculo la variable n representa el número de filas que posee un triángulo dado. La C nos habla de que existen 2 constantes que están asociadas a las operaciones realizadas dentro de los condicionales "if". O(n) nos habla de que el algoritmo tiene un consumo de tiempo lineal.

- **StringClean:** En el cálculo la variable n representa el tamaño del String que se le brinda al algoritmo. La C se refiere 4 constantes que están asociadas a las operaciones presentes dentro de los condicionales "if". El O(n) Indica que el algoritmo presenta un consumo de tiempo lineal.

- **Array11:** En el cálculo la variable n representa el tamaño del arreglo int que se ingresa y la C representa 3 constantes que están ligadas a las operaciones realizadas dentro de los condicionales "if". El O(n) indica que el algoritmo posee un consumo de tiempo lineal.

- **GroupSum5:** En el cálculo la variable n representa el tamaño del arreglo que se ingresa y la C representa 4 constantes presentes en el algoritmo y en los condicionales "if". Se puede apreciar que en el algoritmo existen 2 casos de recurrencia que actúan de manera simultánea por lo que la complejidad $O(2^n)$ indica que el algoritmo posee un consumo de tiempo exponencial.

- **GroupSum6:** En el cálculo la variable n representa el tamaño del arreglo que se ingresa y la C representa 3 constantes presentes en el algoritmo y en las operaciones de los condicionales "if". Se puede apreciar que en el algoritmo existen 2 casos de recurrencia que actúan de manera simultánea por lo que la complejidad dada $O(2^n)$ nos ayuda a comprender que el consumo de tiempo es exponencial.

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

- **SplitOdd10:** En el cálculo la variable n representa el tamaño del arreglo int que se ingresa y la C representa 2 constantes presentes en el algoritmo y en las operaciones dadas por los condicionales "if". También se puede apreciar que en el algoritmo existen 2 casos de recurrencia que actúan de manera simultánea por lo que la complejidad que posee este algoritmo es $O(2^n)$ lo que indica que es un consumo de tiempo exponencial.

- **SpliArray:** En el cálculo la variable n representa el tamaño del arreglo int que se ingresa y la C representa 2 constantes presentes en el algoritmo y en las operaciones dadas por los condicionales "if". También se puede apreciar que en el algoritmo existen 2 casos de recurrencia que actúan de manera simultánea por lo que la complejidad que posee este algoritmo es $O(2^n)$ lo que indica que es un consumo de tiempo exponencial.

- **GroupNoAdj:** En el cálculo la variable n representa el tamaño del arreglo int que se ingresa y la C representa 2 constantes presentes en el algoritmo y en las operaciones dadas por los condicionales "if". También se puede apreciar que en el algoritmo existen 2 casos de recurrencia que actúan de manera simultánea uno en el que se usa los números de Fibonacci y otro en el que se usa los números de lucas, por lo que la complejidad que posee este algoritmo es $O(2^n)$ lo que indica que es un consumo de tiempo exponencial.

4) Simulacro de Parcial

4.1 *1.A) / 2.C) / 3.A)*
4.2 *1.A) / 2.A) y C)*
4.3 *B)*
4.4 *C)*
4.5 *1.A) / 2.B)*
4.6 *Linea 10: return 0 + sumaAux(n,i+1);*
Linea 12: return (n.charAt(i) - '0') + sumaAux(n,i+1);

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