ESTRUCTURA DE DATOS Y ALGORITMOS I **2020-?**

SOLUCIÓN DE EJERCICIOS

EJERCICIO VIRTUAL 3

- **¿Qué tienen en común los 3 problemas que las computadoras jamás podrán resolver, mostrados en el video?**

Jóvenes no sean así… Aquí es donde nos damos cuenta quiénes están poniendo atención a los videos y quiénes nada más los ponen de fondo musical. Casi al principio del video mencionamos que los problemas que no se pueden resolver por una computadora normalmente es porque no tienen una solución desde el punto de vista matemático y tal es el caso de los mostrados en el video.

Recuerden que las computadoras se basan en el modelo de la máquina de Turing y hablamos de que eso está basado en *Entscheidungs Problem*, demostrado por el cálculo lambda (mostradas por Alonzo Church) y por la misma máquina de Turing etc.

Bastaba con que pusieran que no tienen una forma de resolverse matemáticamente y eso era una respuesta válida.

Varios de ustedes pusieron que se tiene que ejecutar una serie de pasos infinitos en un tiempo finito, pero eso no necesariamente aplica para los problemas que aparecen en el video tal vez para el del rompecabezas Wang Tiles pero para los otros dos no estoy seguro de que eso apliqué.

Ordenar un conjunto de valores

En este problema, algunos de ustedes tuvieron la confusión referida que los datos que se van a ordenar son muchos o son pocos pero el mejor y peor caso **no** dependen de la cantidad de datos que vamos a ordenar o de la cantidad de datos en los que vamos a buscar, más bien depende de la cantidad de operaciones que se deben hacer para ordenarlos o buscarlos.

El **mejor caso** de este problema sería aquel donde los elementos que vamos a ordenar ya se encuentran ordenados desde un principio.

Recuerden que el mejor caso es donde hacemos menos operaciones, por lo tanto, si los datos que nos dan desde un principio ya se encuentran ordenados pues no tenemos que hacer absolutamente nada más que tal vez hacer un recorrido para verificar el criterio de ordenamiento.

El **peor caso** sería que los datos estén ordenados exactamente al revés de como nos lo piden, por ejemplo si se requiere ordenar de menor a mayor, el peor caso sería que los datos que nos encontramos estén ordenados de mayor a menor.

Esto pudiera no parecer el peor caso porque en ciertos casos en la vida real, como por ejemplo, los exámenes de los alumnos ordenados por número de lista de manera inversa, pues simplemente las invertimos físicamente y ya estarían ordenadas “correctamente”. Algorítmicamente no funciona así porque no se tiene un panorama externo, por lo tanto tendría tendrían que hacer todas las operaciones posibles para hacer comparaciones e intercambios de los datos.

El caso promedio es todas las otras posibles situaciones, puede ser algunos datos ordenados parcialmente, algunos muy desordenados, ahí entra la campana de distribución gaussiana, en dónde pudiera ver listas casi ordenadas o listas casi desordenadas, de cualquier modo, todo eso entra en el en el caso promedio.

Búsqueda en una lista desordenada

En esta situación el **mejor caso** sería que el valor que vamos a buscar sea **el primero** o se encuentre en la primera posición de la lista, porque así ya no tenemos que recorrer o avanzar nada más, y de manera inmediata es posible determinar que el dato si se encuentra. (esto sería O(1))

El **peor caso** sería que el valor que buscamos se encuentre hasta la última posición o que no esté en la lista porque en este caso realizaríamos el recorrido completo y si la lista tiene mil, diez mil, "n" elementos, sería más tardado y posiblemente no encontremos el valor que buscamos.

El caso promedio es que si se encuentre el valor pero en cualquier posición intermedia y aquí nuevamente tenemos distribución gaussiana donde por probabilidad podría estar en medio al principio o al final en cualquier parte de la lista

Ejemplos de compromisos espacio tiempo

Aquí la idea era que pensaran o que en algún momento dado investigarán posibles casos computacionales donde se involucrara el concepto de compromiso espacio tiempo.

El objetivo **NO** era que volvieran a poner los mismos que mencionamos en el video, como peor ajuste, mejor ajuste, o archivos comprimidos pues esos eran tal cual los que venían ahí.

Algunos de los ejemplos más interesantes que ustedes compartieron fueron

- **Tablas hash o tablas de dispersión**. Esa es una estructura de datos que van a ver más adelante; la idea es que se asocia una clave única con un valor cualquiera y ahí lo que se optimiza es el tiempo porque tener una clave única da un acceso más rápido al valor, pero se sacrifica memoria porque se necesita más espacio para almacenar todas esas claves únicas.
- **Procesamiento de video o imágenes.** En general los aspectos que se refieren a la calidad del video en una transmisión o la calidad de una imagen al almacenarla. Si se tiene una mayor calidad se requiere un mayor ancho de banda lo cual implica un mayor tamaño y un uso amplio de memoria y si se reduce la calidad se puede acelerar la transmisión o reducir el tamaño del archivo.
- **Ecuaciones en latex en sitios web.** Varios de ustedes pusieron este ejemplo, y efectivamente, si las ecuaciones se almacenan utilizando código eso permite que se abran más rápido al momento de cargar el sitio web.
- **Algoritmo de ordenamiento "merge sort"**. También lo verán en eda 2, merge sort es un ejemplo de compromiso espacio tiempo porque en cuestión de algoritmos de ordenamiento es el más rápido (o uno de los más rápidos) pero también es el que utiliza más memoria, ya que en su forma de ordenar, se basa en división de datos y esta división utiliza memoria adicional para almacenar los datos de manera temporal mientras se hace el proceso de ordenamiento.
- **Arreglos vs Memoria ligada** (qué es el tema que tenemos pendiente) También es un buen ejemplo; en el caso de los arreglos, se utiliza más memoria debido a que las localidades se encuentran consecutivamente, pero acceder a un elemento es más rápido porque se puede hacer mediante un índice. Mientras que en el caso de memoria ligada, el uso de memoria es más eficiente porque se puede optimizar con el uso de nodos pero el acceso es más lento debido a que es necesario recorrer esos nodos para encontrar un dato.
- **Archivos en la nube.** Otra perspectiva relacionada con archivos, en lugar de que sean comprimidos, es almacenarlos en la nube, pues ahorra la memoria de nuestra computadora pero cuando queremos acceder a nuestros archivos necesitamos una conexión forzosamente y eso implica aunque sea poco pero un mayor tiempo.

- **Copy-Paste.** Un ejemplo sencillo pero que también es válido, es el del comando copiar y pegar, cuando usamos el comando copiar, ahorramos tiempo porque no tenemos que volver a escribir una o varias palabras, pero se utiliza memoria de la computadora y aunque el uso es reducido y tal vez no se considere como significativo, requiere localidades de memoria que no se usarían si solo se escribiera nuevamente el texto.

Ejercicio virtual 4

Complejidad de los códigos

Algunos de ustedes no pusieron la justificación de su respuesta, recuerden que la deben incluir en el examen porque de esta manera se nota si realmente comprendieron o si le atinaron de churro a la respuesta

Código 1

```
a)   algoritmo1(){
         for(cnt1=0, i=1; i<=n;i++){       // n
               for(j=1; j<=n; j++){         // n
                     cnt1++;               // c
               }
         }
     }
```

Este código era prácticamente el mismo que venía en los ejemplos mostrados en la clase virtual y se trata de un ciclo anidado dentro de otro, y vemos que ambos ciclos el crecimiento se comporta en función de n y el incremento se hace de uno en uno por lo tanto claramente se trataba de una complejidad cuadrática. **$O(n^2)$**

Para cada una de las iteraciones del ciclo de afuera se ejecutan todas las iteraciones del ciclo de adentro

Código 3 (recursivo)

```
c)   algoritmo3(int n){
          if(n<=0)
                return 1;
          else
                return 1 + algoritmo3(n-8)
     }
```

En el caso de la función recursiva, la duda que pudo haber surgido es que en lugar de restar *n - 1* en la llamada recursiva se iba restando de 8 en 8. Pero nuevamente pensemos en valores de *n* suficientemente grandes; suponiendo en comienza su ejecución en cien mil, un millón, etc. Al ir restando de 8 en 8 pues en realidad no genera un impacto en la reducción del número de operaciones o el número de sumas que se van a realizar en la nueva llamada recursiva, por lo tanto sigue siendo una complejidad lineal. **O(n)**

Código 2 (final inesperado, giro en la trama)

```
b)   algoritmo2(){
          for(cnt3=0, i=1; i<=n; i*=2){
               for(j=1; j<=n; j++){
                    cnt3++; int a=0; int b=10;
               }
          }
     }
```

Y bueno el plot twist de este ejercicio radicaba en el segundo código. La clave está en el primer ciclo for ya que, en este caso, a pesar de que el alcance del ciclo llega hasta *n*, la diferencia fundamental con respecto a los otros ciclos que habíamos visto hasta el momento es que **el incremento de i es multiplicar por dos**, veamos como crece el número de operaciones:

Si n vale 10, las iteraciones de i serían 1,2, 4 y 8

Si n vale 20, las iteraciones de i serían 1,2, 4, 8, 16 (solo es una operación mas con respecto a n=10)

Si n vale 100, las iteraciones de i serían 1,2, 4, 8,16,32,64, (solo son dos operaciones más con respecto a n= 20)

Por lo tanto, si se observa el incremento en n es muy grande y el incremento de operaciones no lo es tanto, eso significa que la complejidad del primer for es **logarítmica,** y dado que el ciclo for de adentro si se incrementa en función de *n* para cada una de las veces del ciclo de afuera se ejecutan todas las del ciclo de adentro.

Finalmente, al tener "afuera" una complejidad logarítmica y adentro una complejidad lineal, al multiplicarlo entonces finalmente la complejidad de este código sería **O (n log n)**