Tarea extraclase #3

Alumnos:

Fabián Fallas

Kevin Rodríguez

Complejidad computacional, estudia el orden de complejidad de un algoritmo que resuelve un problema computacional.

Para ello, considera los dos tipos de recursos requeridos durante el cómputo para resolver un problema, que son tiempo y memoria.

Tiempo, número de pasos base de ejecución de un algoritmo para resolver un problema.

Memoria, Cantidad de memoria utilizada para resolver un problema.

La complejidad de un algoritmo se expresa como función del tamaño de la entrada del problema, n.

Órdenes de crecimiento a analizar.

-Logarítmico $log\ (n)$

-Lineal N

-Lineal logarítmico $N\ log\ (n)$

-Cuadrático N^2

-Cúbico N^3

-Exponencial 2^N

-Factorial $N!$

1. $log\ (n)$.

Árbol binario de búsqueda.

¿por qué calza en esta complejidad?

Porque en este algoritmo no se compara de uno en uno, sino que se divide a la mitad los elementos y así hace la búsqueda más eficiente. Por ejemplo, buscar un número de teléfono en una guía telefónica, es búsqueda se comporta como $log\ (n)$, porque no se hace una búsqueda uno por uno, sino que se puede hacer una búsqueda más eficiente, que es buscar alfabéticamente lo que nos permite descartar gran parte de la información con una sola comparación.

En el caso del árbol binario de búsqueda en cada comparación se puede descartar la mitad de la información.

Característica para determinar si un algoritmo cae en esta complejidad:

El algoritmo debe dividir los elementos a la mitad y así sucesivamente en orden de menor a mayor para lograr llegar con mayor facilidad al resultado.

2. N

Búsqueda secuencial.

¿por qué calza en esta complejidad?

Calza en la complejidad porque su patrón de comportamiento al comparar es lineal, pues lo va haciendo de uno en uno sin causar que en algún momento pierda su linealidad, eso sí entre más grande sea la lista más dura, sin embargo, sigue siendo lineal/secuencial.
Característica para determinar si un algoritmo cae en esta complejidad:
Al momento de probar el algoritmo este debe de realizar las comparaciones de manera lineal sin saltarse o alterar el orden.

3. $N\ log\ (n)$

Quicksort.

¿por qué calza en esta complejidad?

Es como unir las dos anteriores, analizando el Quicksort paso a paso, este hace particiones y luego simplemente comparaciones secuenciales, entonces entendiendo que las particiones en la lista de elementos para comparar generan un comportamiento con la función $log\ (n)$, y que luego hace procesos de comparaciones secuenciales de la forma N.

Es decir, el Quicksort hace procesos algorítmicos al dividir los elementos, y lineales luego al comparar, por lo que $N\ log\ (n)$ puede verse como la combinación de ambos.
Característica para determinar si un algoritmo cae en esta complejidad:
para saber si un algoritmo cae en esta complejidad debemos revisar dicho algoritmo revisando que cumpla con las características de que primero divide los datos a la mitad y luego lo hace de manera lineal

4. N^2

Bubble Sort

¿por qué calza en esta complejidad?

Se tiene que un algoritmo de orden dos, N^2 es la multiplicación de N*N, ósea dos procesos lineales, esto lo podemos ver en algoritmos que tienen un bucle dentro de otro bucle, como lo es el caso del Bubble Sort, que tiene dos ciclos o bucles anidados.
Característica para determinar si un algoritmo cae en esta complejidad:
En este proceso el algoritmos debe de realizar dos ciclos a la vez , es decir; mientras está realizando un ciclo este debe estar dentro de otro

5. N^3

Recorrer una matriz tridimensional

¿por qué calza en esta complejidad?

Para recorrer una matriz bidimensional se necesitan dos ciclos para recorrer las filas y columnas, lo que haría a ese algoritmo caer en el orden anterior N^2, sin embargo, al tener una matriz tridimensional lo que pasa es que para recorrerla se necesitan tres bucles anidados lo que nos provocaría algo como N^2* N = N^3.
Característica para determinar si un algoritmo cae en esta complejidad:
El algoritmo con esta caracteristicas debe de realizar tres ciclos a la vez, es decir un ciclo se va a realizar dentro de otro.

6. 2^N

Problema del viajante

¿por qué calza en esta complejidad?

El problema del viajante responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?, el algoritmo Dijkstra vino a ser una solución menos compleja.

Lo que el algoritmo hace es encontrar la ruta más corta, para logra eso lo que hace es explorar cada combinación de rutas posibles, generando todas las combinaciones posibles, es convierte a es algoritmo en uno con complejidad exponencial, pues la cantidad máxima de combinaciones es una exponencial del número de ciudades que se tenga, es decir que entre más ciudades haya más crece el número de combinaciones, y dicho crecimiento lo hace de forma exponencial.

Característica para determinar si un algoritmo cae en esta complejidad:

En este caso el algoritmo debe de realizar todas las comparaciones posibles para dar el resultado

7. $N!$

Generar todas las permutaciones de una lista

¿por qué calza en esta complejidad?

Generar todas las permutaciones de una lista es un algoritmo factorial de la forma $N!$, la diferencia clave entre este algoritmo y el anterior, es que en el anterior se buscan todas las combinaciones posibles y en este caso se busca todas las permutaciones, en las combinaciones se puede repetir, en las permutaciones no. Ejemplo, si hay que ordenar 16 bolas de billar sin repetir, cuantas posibilidades hay de hacerlos. Para escoger la primera bola hay 16 posibilidades y para la segunda hay 15 y así sucesivamente, matemáticamente eso se resuelve así 16*15*14*13*12*11*9*8*7*6*5*4*3*2*1= 20,922,789,888,000, esto es 16!, lo que conocemos como factorial, y bueno esto se logra más que todo con recursión, en el caso de todas las permutaciones de una matriz es la misma funcionalidad, entonces por esto es que este algoritmo encaja en esta complejidad.

Característica para determinar si un algoritmo cae en esta complejidad:

Los algoritmos con esta característica deben de realizar todas las permutación posibles para dar el resultado