Metodos de clasificacion

Clasificar -> ordenar en un momento determinado un conjunto de valores, que no se tiene ordenado en el tiempo. Para que queden ordenados en el tiempo hay que usar ***índices***.

Los registros que voy a ordenar están dados por:

Clave -> campo por el que voy a ordenar

Datos satélite-> resto de datos que acompañan al registro

Estabilidad -> un ordenamiento es estable cuando ante la igualdad de valores, el que ingreso primero toma la primer posición.

In situ -> ordenar in situ implica no pedir mas memoria de la que ocupan mis registros originalmente no ordenadospara ordenarlos. Es decir, no crear un array nuevo para poner mi array ordenado

Clasificación de algoritmos según métodos internos y externos:

* Método interno -> diseñado para ordenar en memoria ram porque la performance del algoritmo es menor.
* Método externo -> diseñado para ordenar datos en memoria externa o disco porque le alcanza la performance.

Complejidad computacional de un algoritmo -> cuan complejo es para la computadora resolver el algoritmo que le doy. Acá hay una clasificación de problemas entre P (posibles de resolver por computadora) y NP (no posibles de resolver por computadora). El algoritmo para NP existe pero no puede ejecutarse para un numero determinado de datos.

El orden de complejidad se define como O(f(n)), siendo f una función matemática asociada a un algoritmo que representa la cantidad de comparaciones que se realizan para una cantidad n de datos. Me dice cuanto tarda en ejecutarse un algoritmo en función de la cantidad de datos que recibe.

Como se evalua el orden de complejidad

* La computadora puede hacer operaciones matemáticas y comparaciones booleanas
* Para evaluar la complejidad de un algoritmo se evalúan la cantidad de comparaciones realizadas.
* El motivo es que un if puede llegar a ser hasta 100 veces mas lenta que una operación matemática

Métodos de ordenamiento

Bubble sort -> compara por pares, si el de la izquierda es menor que la derecha lo intercambia mientras recorre de izquierda a derecha. El mas grande de todos burbujea hasta la ultima posición.

Tiende a n^2 como complejidad porque son dos for anidados de un if. Pero si viene ordenado, el orden de complejidad es n porque hago n comparaciones porque son n elementos.

Selection sort -> compara por pares pero de otra forma. Compara el elemento 0 con los n-1 restantes. De tal forma que cada iteración encuentra al mismo. Complejidad O(n^2)

Insertion sort -> compara de izquierda a derecha buscando un marcador ->un elemento mas grande que todos los que estén a la izquierda. Cuando el marcador es mas grande que el elemento que le sigue, agarra ese elemento y lo inserta ordenado en la lista de la izquierda. Si el marcador es mas chico, agarra ese elemento nuevo y lo usa como marcador sabiendo que a su izquierda, están todos ordenados. Es de complejidad O(n^2) porque tiene 2 for anidados pero aun así hace menos comparaciones porque solo usa el segundo for si se da el caso de que el siguiente elemento es mas pequeño que el marcador

Shell sort -> es una modificación del insertion sort. Compara por secuencia de incrementos, es decir, agarra un elemento y compara con k posiciones adelante y si es menor los intercambia. Así se arman grupos de k elementos que sabemos que están desordenados dentro de si pero si comparamos entre grupos, estos están ordenados entre si. Su complejidad es O(n^3/2)

Ahora vienen algoritmos que bajan la complejidad lineal. Estos son algoritmos externos

Merge sort -> Divide el universo a la mitad hasta que no pueda dividir mas y ahí la comparación es lineal. Luego fusiona todo en orden. Se arma un árbol de comparación para unificar niveles. Es decir se compara por niveles. El nivel de complejidad es O(nlogn) porque para un árbol completo y balanceado es la cantidad de niveles. Para este algoritmo, mientras mas elementos, menos tarda relativamente.

Quick sort -> Dado un conjunto de elementos, elige un pivote o referencia, compara todos con el pivote y crea 2 conjuntos, uno con los mas chicos que el pivote y otros con los mas grandes que el pivote. Los deja a la izquierda y derecha de el respectivamente. En la próxima iteración, hace lo mismo con otro pivote por cada uno de esos conjuntos.

Esto termina creando un árbol binario de búsqueda. La complejidad es O(nlogn) excepto en un caso donde los elementos me vienen totalmente ordenados

Mean sort -> toma como pivote el mas parecido a la media. Este tiene sentido si los datos están acotados.

B sort -> elige el elemento que esta en el medio del array como pivote.

Heap sort -> arma un heap con los datos y luego lo desarma

Heap -> árbol binario que se arma en forma completa, donde el padre es mayor que los hijos