**Algoritmo de ordenamiento**

**Sensible:** es cuando a diferentes entradas varía significativamente el tiempo de ejecución

**Estable:** es cuando se ordena un vector con claves repetidas y estas se mantienen en el orden inicial.

**Inserción:** estable y sensible. n(n-1)/2 desplazamiento en el peor caso, 0 en el caso ideal. n(n-1)/2 comparaciones en el peor caso, n-1 en el caso ideal.

**Selección:** no es estable ni sensible. n-1 intercambios, n(n-1)/2 comparaciones.

**Burbujeo:** estable y sensible. n(n-1)/2 intercambios y comparaciones.

Si está casi ordenado elijo burbujeo

Si tengo registros grandes con claves pequeñas elijo seleccion porque burbujeo hace mas intercambios en el peor caso.

**SHELL**

O(n^(3/2)) ( es sensible no es estable)

Ordena los k distantes con inserción.(ojo en el video lo hace con burbuja).

Luego ordeno los k/2 distantes con inserción.

**Quick sort**

Eligo un pivot del medio al azar O(n.log n ) ( no es sensible ni estable)

**Merge sort** O(n.log n ) ( no es sensible y es estable)

Divide recursivamente un vector hasta que tenga 2 elementos en la funcion recursiva y ordena esos dos elementos y los devuelve.

**Cuenta de distribución particular.**

Numeros consecutivos no admite repetidos. Tiene que ser numeros contables, no reales.

La posición es el elemento menos un k. Complejidad N (no es sensible, no podemos hablar de estable porque no tiene elementos repetidos)

For

Aux[ver[i]-k] = vec[i];

**Cuenta de distribución general:**

Complejidad lineal N (no es sensible, es estable si lo recorrer de atras para adelante)

Pocas claves consecutivas repetidas muchas veces en el vector que tengo que ordenar.

B a bb c a c a

Abc

3 3 2

Con un for (n) anota cuantas veces aparecen las claves.

Acumulo lsa claves.

Abc

368

Tomo un aux

Y pongo la cantidad de letras de atras para adelantes.

**Monticulo** Tipo de datos abstracto.

Arbol binario (cada nodo tiene dos hijos)

Completo o casi completo porque se llenan de izq a der.

El padre de cualquier subarbol tiene que ser mayor a los hijos si es monticulo maximo, alreves si es minimo.

Guardo todo el arbol en un vector en donde la raiz esta en el indice 1 ( el indice 0 se descarta)

Despues guardo los hijos:

Hijo Izq = i\*2;

Hijo der = i\*2+1;

Para buscar un padre de un i;

Padre = i/2 (parte entera);

Para agregar un elemento lo pongo en la hoja que puedo y pregunto si el padre cumple con la relacion de monticulo. Si no cumple lo intercambio. Una vez intercambiado pregunto si cumple la realcion de monticulo. Sino cumple lo intercambio.

Los monticulos se usan pora cola de prioridad. Los grafos necesitan ordenar las aristas de menor peso.(poner O(log n))

Para sacar, saca el elemento y pongo el ultimo elemento en el lugar donde lo saque. Despues pregunto por el hijo si cumple la funcion si no cumple lo intercambio por el hijo mayor y vuelvo a preguntar lo mismo. (sacar O(log n))

**Programación dinámica**

Usa màs memoria para guardar el resultado del algoritmo pero intenta usar menos procesamiento.

Algoritmo greeding en cada iteración busca la mejor opción.

Si quiero conseguir la submatriz en donde la suma sea la maxima.

Problema de la mochila, expendedora de cafe.

|  |  |
| --- | --- |
| **Complejidad Computacional**  C=10^8 operaciones por segundo | N aproximada para **1 seg** |
| Log n | 2^108 |
| Raiz de n | 10^16 |
| n | 10^8 |
| N log n | 4.500.000 |
| n^2 | 10.000 |
| n^2 log n | 3.000 |
| n^3 | 500 |
| n^4 | 100 |
| 2^n | 27 |
| 3^n | 17 |
| n! | 11 |

t(n) = C.f(n)