IMPLEMENTACIÓN DE MÉTODOS DE ORDENACIÓN

Daniel Fumero Cruz

Algoritmos Utilizados

Estable			
Algoritmo	Complejidad	Memoria	Método
Bubblesort	$O(n^2)$	O(1)	Intercambio
Insertion sort	$O(n^2)$("(en el peor de los casos)")	O(1)	Inserción

Inestables			
Nombre original	Complejidad	Memoria	Método
Shell sort	$O(n^{1.25})$	O(1)	Inserción
Heapsort	$O(n \log n)$	O(1)	Selección
Quicksort	Promedio: $O(n \log n)$, peor caso: $O(n^2)$	O(log *n*)	Partición

Método de la Burbuja (Bubblesort)

Características

- Es un algoritmo sencillo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo uno de los más sencillo de implementar.

Funcionamiento:

Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Ordenación por inserción

Características

- Es un algoritmo sencillo de entender y de codificar.
- Si el tamaño de la entrada es N, entonces el orden del tiempo de ejecución, para el peor caso es O(N2);
- Si la entrada esta "casi ordenada", el algoritmo se ejecuta mucho más rápidamente.
- Esta velocidad tiende a un tiempo O(N), peor caso que se cumple cuando la entrada está totalmente ordenada.
- Es por la propiedad anterior que este algoritmo, a pesar de no ser el más rápido para entradas grandes, suele usarse de la siguiente manera: Se semi ordena la entrada con algún otro algoritmo más rápido y más adecuado para entradas grandes. Luego, cuando tenemos la entrada "casi ordenada" usamos este algoritmo. La velocidad de ejecución será muy buena por dos razones: su tiempo de ejecución tiende a O(N) con entradas "casi ordenadas" (lo cual es un tiempo excelente), y la simpleza de su implementación hará que se ejecute más rápido que otros algoritmos más complejos.

Funcionamiento:

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay ***k*** elementos ordenados de menor a mayor, se toma el elemento ***k+1*** y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha). En este punto se *inserta* el elemento *k+1* debiendo desplazarse los demás elementos.

Ordenación de Shell (ShellSort)

Características

- A diferencia del algoritmo de ordenación por inserción, este algoritmo intercambia elementos distantes. Es por esto que puede deshacer más de una inversión en cada intercambio, hecho del cual nos aprovechamos para ganar velocidad.

- La velocidad del algoritmo dependerá de una secuencia de valores (llamados incrementos, de los cuales hablaremos más abajo) con los cuales trabaja utilizándolos como distancias entre elementos a intercambiar.

- Se considera la ordenación de Shell como el algoritmo más adecuado para ordenar entradas de datos moderadamente grandes (decenas de millares de elementos) ya que su velocidad, si bien no es la mejor de todos los algoritmos, es aceptable en la práctica y su implementación (código) es relativamente sencillo.

Funcionamiento:

Se determinan valores de espaciado utilizando una de las formulas disponibles, el primer valor es 1, y el limite se basa en el tamaño del vector.
Se divide el vector en subvectores cuya cantidad de elementos está definida por uno de los valores de espaciado, empezando por el mayor hasta llegar a 1. Cada subvector se ordena utilizando insertsort. una vez estén ordenado los subvectores se divide por el valor siguiente. Cada uno de los subvectores esta ordenado.
Una vez se alcance el espaciado 1, se ordena todo el vector completo.

Ordenación por montículos (Heapsort)

Características

- Es un algoritmo que se construye utilizando las propiedades de los montículos binarios.

- El orden de ejecución para el peor caso es $O(N \cdot \log(N))$, siendo N el tamaño de la entrada.

Funcionamiento:

Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo (heap), y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado. Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él.

Ordenación rápida (Quicksort)

Características

- Es el algoritmo de ordenación más rápido (en la práctica) conocido. Su tiempo de ejecución promedio es O(N log(N)).

- Para el peor caso tiene un tiempo O(N2), pero si se codifica correctamente las situaciones en las que sucede el peor caso pueden hacerse altamente improbables.

- Al igual que el algoritmo de ordenación por intercalación, el algoritmo de ordenación rápida es fruto de la técnica de resolución de algoritmos "divide y vencerás". La técnica de divide y vencerás se basa en, en cada recursión, dividir el problema en subproblemas más pequeños, resolverlos cada uno por separado (aplicando la misma técnica) y unir las soluciones.

Funcionamiento:

El algoritmo **quicksort** ordena un vector **V** eligiendo entre sus elementos un valor clave **P** que actúa como pivote, organiza tres secciones, izquierda-P-derecha, todos los elementos a la izquierda deberán ser menores a P, y los de la derecha mayores, los ordena sin tener que hacer ningún tipo de mezcla para combinarlos, ¿cómo elegimos el pivote?

Método 1: Lo ideal sería que el pivote fuera la mediana del vector para que las partes izquierda y derecha tuvieran el mismo tamaño.

Método 2: recorremos el vector con un índice **i** desde 0 a n-1, y otro índice **j** inversamente y cuando se crucen, es decir, tenga el mismo valor, ese se seleccionará como pivote.