Algoritmos de ordenamiento

Burbuja

El método de la burbuja es uno de los más simples, es tan fácil como comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición.
Por ejemplo, imaginemos que tenemos los siguientes valores: 5 6 1 0 3

Lo que haría una burbuja simple, sería comenzar recorriendo los valores de izq. a derecha, comenzando por el 5. Lo compara con el 6, con el 1, con el 0 y con el 3, si es mayor o menor (dependiendo si el orden es ascendiente o descendiente) se intercambian de posición. Luego continúa con el siguiente, con el 6, y lo compara con todos los elementos de la lista, esperando ver si se cumple o no la misma condición que con el primer elemento. Así, sucesivamente, hasta el último elemento de la lista.

ShellSort

Este método es una mejora del algoritmo de ordenamiento por Inserción (Insertsort).
Si tenemos en cuenta que el ordenamiento por inserción es mucho más eficiente si nuestra lista de números esta semi-ordenada y que desplaza un valor una única posición a la vez.
Durante la ejecución de este algoritmo, los números de la lista se van casi-ordenando y finalmente, el último paso o función de este algoritmo es un simple método por inserción que, al estar casi-ordenados los números, es más eficiente.

- 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. Veremos que con algunas secuencias podremos obtener órdenes de tiempo de ejecución en el peor caso de O(n2), O(n^(3/2)) y O(n^(4/3)).

- 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.

Secuencia k-ordenada. Decimos que una secuencia A de n elementos está k-ordenada (siendo k un natural) si, para todo i de [0,n] se cumple que los elementos A[i + hk] están ordenados, siendo h un entero y siempre que el índice i+hk esté en [0,n].
El algoritmo de ordenación de Shell lo que hace en realidad es tomar una secuencia de incrementos h1, h2, ..., hp y en la etapa k-esima realizar una hk-ordenación de los datos. La única condición que debe respetar la secuencia de incrementos es hp=1 de modo tal que en la última etapa se hace una 1-ordenación (o sea una ordenación normal).

QuickSort

Sin duda, este algoritmo es uno de los más eficientes. Este método es el más rápido gracias a sus llamadas recursivas, basándose en la teoría de divide y vencerás.

Lo que hace este algoritmo es dividir recursivamente el vector en partes iguales, indicando un elemento de inicio, fin y un pivote (o comodín) que nos permitirá segmentar nuestra lista. Una vez dividida, lo que hace, es dejar todos los mayores que el pivote a su derecha y todos los menores a su izq. Al finalizar el algoritmo, nuestros elementos están ordenados.

- 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.

- En la práctica, el hecho de que sea más rápido que los demás algoritmos de ordenación con el mismo tiempo promedio O(N log(N)) está dado por un ciclo interno muy ajustado (pocas operaciones).

- 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". Como ya se explicó, 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.

Radix

Este ordenamiento se basa en los valores de los dígitos reales en las representaciones de posiciones de los números que se ordenan.
Por ejemplo el número 235 se escribe 2 en la posición de centenas, un 3 en la posición de decenas y un 5 en la posición de unidades.

Reglas para ordenar.

- Empezar en el dígito más significativo y avanzar por los dígitos menos significativos mientras coinciden los dígitos correspondientes en los dos números.
- El número con el dígito más grande en la primera posición en la cual los dígitos de los dos números no coinciden es el mayor de los dos (por supuesto sí coinciden todos los dígitos de ambos números, son iguales).

Este mismo principio se toma para Radix Sort, para visualizar esto mejor tenemos el siguiente ejemplo. En el ejemplo anterior se ordenó de izquierda a derecha. Ahora vamos a ordenar de derecha a izquierda.

Archivo original.

25 57 48 37 12 92 86 33

Asignamos colas basadas en el dígito menos significativo.

	Parte delantera	Parte posterior
0		
1		
2	12	92
3	33	
4		
5	25	
6	86	
7	57	37
8	48	
9		
10		

Después de la primera pasada:

12 92 33 25 86 57 37 48

Referencias:

Estructuras de Datos y Algoritmos, Mark Allen Weiss, Addison-Wesley Iberoamericana, 1995, ISBN 0-201-62571-7.