

**Instituto Tecnológico de Tijuana**

**Subdirección Académica**

**Departamento de Sistemas y Computación**

**Semestre agosto-diciembre 2018**

**Carrera:** Ing. En Sistemas Computacionales Serie SC1A

**Materia:** Estructura de Datos

**Unidad 4 – Métodos de Ordenamiento**

**Alumno:**15211883 - Angeles Valadez Jonathan

**Profesor:** Ray Brunett Parra Galaviz

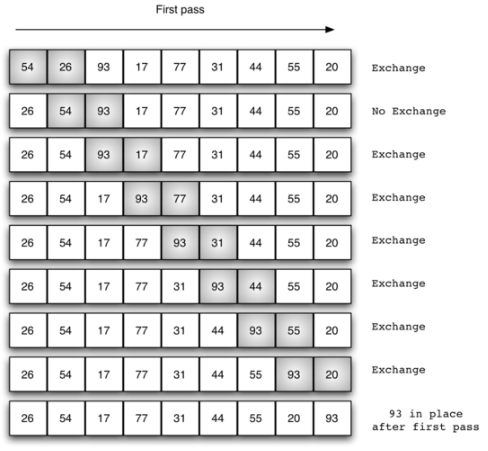
**Ordenamiento**

Ordenar es el proceso de ubicar elementos de una colección en algún orden. Por ejemplo, una lista de palabras podría ordenarse alfabéticamente o por longitud. Una lista de ciudades podría ordenarse por población, por área o por código postal. Se han desarrollado y analizado muchísimos algoritmos de ordenamiento. Esto sugiere que el ordenamiento es una importante área de estudio en las ciencias de la computación. Ordenar un gran número de ítems puede requerir una cantidad considerable de recursos informáticos. Al igual que la búsqueda, la eficiencia de un algoritmo de ordenamiento está relacionada con el número de ítems que se están procesando. Para las pequeñas colecciones, un método de ordenamiento complejo puede resultar más problemático que beneficioso. La sobrecarga puede ser demasiado alta. Por otra parte, para colecciones más grandes, queremos aprovechar tantas mejoras como sean posibles. En esta sección discutiremos varias técnicas de ordenamiento y las compararemos respecto a sus tiempos de ejecución.

**El ordenamiento burbuja**

El **ordenamiento burbuja** hace múltiples pasadas a lo largo de una lista. Compara los ítems adyacentes e intercambia los que no están en orden. Cada pasada a lo largo de la lista ubica el siguiente valor más grande en su lugar apropiado. En esencia, cada ítem “burbujea” hasta el lugar al que pertenece.

La siguiente figura muestra la primera pasada de un ordenamiento burbuja. Los ítems sombreados se comparan para ver si no están en orden. Si hay n ítems en la lista, entonces hay n−1 parejas de ítems que deben compararse en la primera pasada. Es importante tener en cuenta que, una vez que el valor más grande de la lista es parte de una pareja, éste avanzará continuamente hasta que la pasada se complete.



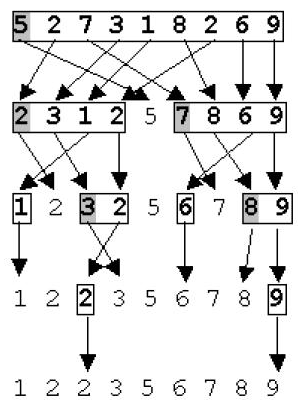
Al comienzo de la segunda pasada, el valor más grande ya está en su lugar. Quedan n−1 ítems por ordenar, lo que significa que habrá n−2 parejas. Puesto que cada pasada ubica al siguiente valor mayor en su lugar, el número total de pasadas necesarias será n−1. Después de completar la pasada n−1, el ítem más pequeño debe estar en la posición correcta sin requerir procesamiento adicional.

**El Ordenamiento QuickSort**

El método Quick Sort es actualmente el más eficiente y veloz de los método de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de Quick Sort, por la velocidad con la que ordena los elementos del arreglo.

Quicksort es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n. Quicksort es actualmente el más eficiente y veloz de los métodos de ordenación interna.

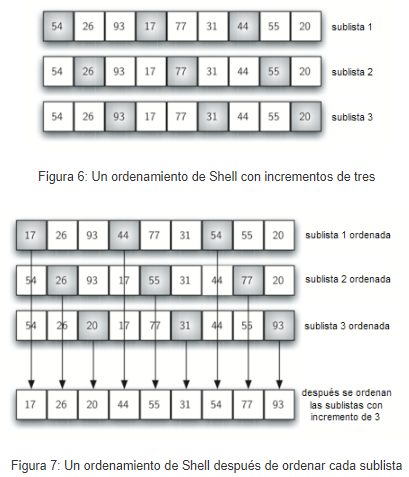
Este método fue creado por el científico británico Charles Antony Richard Hoare, también conocido como Tony Hoare en 1960, su algoritmo Quicksort es el algoritmo de ordenamiento más ampliamente utilizado en el mundo.



**El Ordenamiento de Shell**

El ordenamiento de Shell, a veces llamado “ordenamiento de incremento decreciente”, mejora el ordenamiento por inserción al romper la lista original en varias sublistas más pequeñas, cada una de las cuales se ordena mediante un ordenamiento por inserción. La manera única en que se eligen estas sublistas es la clave del ordenamiento de Shell. En lugar de dividir la lista en sublistas de ítems contiguos, el ordenamiento de Shell usa un incremento i, a veces denominado brecha, para crear una sublista eligiendo todos los ítems que están separados por i ítems.

Este proceso se puede ver en la Figura 6. La lista tiene nueve ítems. Si usamos un incremento de tres, hay tres sublistas, cada una de las cuales puede ordenarse mediante un ordenamiento por inserción. Después de completar estos ordenamientos, obtenemos la lista que se muestra en la Figura 7. Aunque esta lista no está completamente ordenada, ha ocurrido algo muy interesante. Al ordenar las sublistas, hemos acercado los ítems a donde realmente pertenecen.

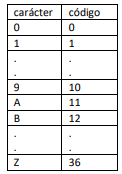


El Ordenamiento Radix Sort

El método de ordenamiento Radix Sort también llamado ordenamiento por residuos puede utilizarse cuando los valores a ordenar están compuestos por secuencias de letras o dígitos que admiten un orden lexicográfico.

El algoritmos ordena utilizando un algoritmo de ordenación estable, las letras o dígitos de forma individual, partiendo desde el que esta mas a la derecha (menos significativo) y hasta el que se encuentra más a la izquierda (el más significativo). Nota: A cada letra o digito se le asigna una llave o código representado por un número entero, el cual se utiliza para el ordenamiento de cada elemento que conforma el valor original.

Por ejemplo, en uno se planea que una línea aérea proporciona números de confirmación diseñados con cadenas formadas con 2 caracteres donde cada carácter es un digito o una letra que puede tomar 36 valores (26 letras y 10 dígitos) y así hay posibles códigos.



**Bibliografía**

[En Línea]: <http://interactivepython.org/runestone/static/pythoned/SortSearch/Ordenamiento.html>

[En Línea]: <http://interactivepython.org/runestone/static/pythoned/SortSearch/ElOrdenamientoBurbuja.html>

[En Línea]: <http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/mtodo_quick_sort.html>

[En Línea]: <http://odin.fi-b.unam.mx/salac/practicasEDAII/eda2_p3.pdf>

[En Línea]: <http://interactivepython.org/runestone/static/pythoned/SortSearch/ElOrdenamientoDeShell.html>