

**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 – Algoritmos de Ordenamiento Interno**

**Alumno:**15211883 - Angeles Valadez Jonathan

**Profesor:** Ray Brunett Parra Galaviz

**Algoritmos de Ordenamiento Interno**

Los algoritmos de ordenamiento interno son aquellos que son manejados usando la memoria primaria, es decir la memoria de trabajo o memoria RAM.

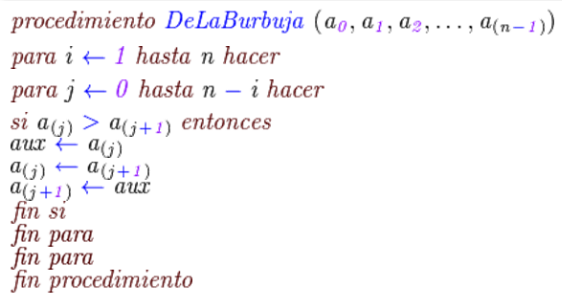
A estos algoritmos se les conoce porque su uso es con listas simples, los datos son de un solo tipo y se ordenan mientras se esté trabajando con la lista de forma preliminar, es decir; usando la lista, ya sea que los datos se inserten, o que se inicialicen.

Entre los algoritmos de ordenamiento interno tenemos:

1. Ordenamiento de Burbuja
2. Ordenamiento Shell
3. Ordenamiento Quick Sort
4. Ordenamiento Radix

**Burbuja Simple**

El método de la burbuja es una comparación lineal con cada uno de los elementos, el elemento que sea menor contra el que se está comparado intercambiaran posiciones. Este método no es recomendado para grandes comparaciones, ya que es un proceso muy lento y requiere de una gran cantidad de Memoria RAM.



**Quick Sort**

El ordenamiento rápido (quicksort en inglés) es un algoritmo creado por el científico británico en computación C. A. R. Hoare basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n.

El algoritmo trabaja de la siguiente forma:

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
2. Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
3. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
4. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

1. En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es O(n•log n).
2. En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de O(n²). El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente.

**Shell Sort**

Esta forma de ordenación es muy parecida a la ordenación con burbuja.

La diferencia es que no es una comparación lineal, sino que trabaja con una segmentación entre los datos.

Por lo tanto es un buen método, pero no el mejor para implementarlos en grandes arreglos.

Método de Shell Se tiene el siguiente arreglo con 16 elementos:

A={15,67,08,16,44,27,12,35,56,21,13,28,60,36 07,10}

Se parte a la mitad, quedando dos sublistas, cada una con 8 elementos, que son los siguientes: A1={15,67,08,16,44,27,12,35} A2={56,21,13,28,60,36,07,10}.

Se comparan los elementos de cada sublista, donde el 1ro de la sublista A1 con el 1ro de la sublista A2 y así sucesivamente.

Por ejemplo: 1

5>56 no, se queda como estaba

67>21 si, se cambian de posición

08>13 no, se queda como estaba

16>28 no, se queda como estaba

44>60 no, se queda como estaba

27>36 no, se queda como estaba

12>07 si, se cambian de posición

35>10 si, se cambian de posición

Quedando el arreglo de la siguiente forma

15 21 08 16 44 27 07 10 56 67 13 28 60 36 12 35

Ahora se divide el arreglo que quedo en grupos de 4 en 4, quedando así 4 sublistas:

A1={15,21,08,16} A2={44,27,07,10} A3={56,67,13,28} A4={60,36,12,35}

Ahora se va a comparar cada elemento de cada subsista según su posición y el que sea el menor se cambia.

15>44 no

44>56 no,

56>60 no, queda igual(misma posición)

21>27 no

27>67 no,

67>36 si se cambian( el 36 por 67)

08>07 si, cambian,

08>13 no, 13>12 se cambian(12 por 13)

16>10 si, cambian,

16>28 no

28>35 no

Quedando el arreglo de la siguiente forma:

15 21 07 10 44 27 08 16 56 36 12 28 60 67 13 35

Ahora se divide el arreglo que queda en grupos de 2 en 2 quedando así 8 sublistas de la siguiente forma:

A1={15,21} A2={07,10} A3={44,27} A4={08,16} A5={56,36} A6={12,28} A7={60,67} A8={13,35}

Ahora se va a comparar cada elemento de cada sablista según su posición y el que sea el menor se cambia.

15>07 si, cambia;

15>44 no,

44>08 si, cambia;

44>56 no,

56>12 si, cambia;

56>60 no,

60>13 si, cambia.

21>10 si, cambia;

21>27 no,

27>16 si, cambia;

27>36 no,

36>28 si, cambia;

36>67 no,

67>35 si, cambia.

Quedando el arreglo de la siguiente forma:

07 10 08 16 12 21 13 27 15 28 44 35 56 36 60 67

Ahora se comparan de uno en uno para quedar el arreglo ya propiamente ordenado, si es mayor se cambia.

Quedando el arreglo ordenado de la siguiente forma:

07,08,10,12,13,15,16,21,27,28,35,36,44,56,60,67

**Radix**

En informática, el ordenamiento Radix (radix sort en inglés) es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (por ejemplo, nombres o fechas) y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros.

**Ejemplo**

Vector original:

25 57 48 37 12 92 86 33

Asignamos los elementos en colas basadas en el dígito menos significativo de cada uno de ellos.

0:

1:

2:12 92

3:33

4:

5:25

6:86

7:57 37

8:48

9:

Después de la primera pasada, la ordenación queda:

12 92 33 25 86 57 37 48

Colas basadas en el dígito más significativo.

0:

1:12

2:25

3:33 37

4:48

5:57

6:

7:

8:86

9:92

Lista ordenada:

12 25 33 37 48 57 86 92

**Bibliografía**

[En Línea]: <http://itslr.edu.mx/archivos2013/TPM/temas/s3u5.html>