Capítulo 8

MÉTODOS DE ORDENACIÓN

8.1 INTRODUCCIÓN

Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica. Los procesos de ordenación y búsqueda —este último se estudiará en el siguiente capítulo— son frecuentes en nuestra vida. Vivimos en un mundo desarrollado, automatizado, acelerado, donde la información representa un elemento de vital importancia. La sociedad debe estar informada y, por lo tanto, constantemente se necesita buscar y recuperar información.

La operación de búsqueda —recuperación— de información normalmente se efectúa sobre elementos ordenados, lo que demuestra que, en general, donde haya objetos que se deban buscar y recuperar estará presente el proceso de ordenación.

Los objetos ordenados aparecen por doquier. Directorios telefónicos, registros de pacientes de un hospital, registros de huéspedes de un hotel, índices de libros de una biblioteca, son tan sólo algunos ejemplos de objetos ordenados con los que el ser humano se encuentra frecuentemente. Incluso y de manera informal se puede señalar que desde niño se nos enseña a ser organizado, a poner las cosas en orden.

La ordenación es una actividad fundamental y relevante en la vida. Imagine el lector qué ocurriría si se deseara encontrar un libro en una biblioteca con más de 100 000 volúmenes y éstos estuvieran desordenados o registrados en los índices en el orden en el cual fueron recibidos; o, por ejemplo, si se quisiera hablar por teléfono con una persona y se encontrara que en el directorio los abonados están ordenados según su número telefónico, en forma ascendente o descendente. La tarea sería mayúscula, pero sin ningún sentido.

Formalmente se define **ordenación** de la siguiente manera:

Sea A una lista de N elementos:

$$A_1, A_2, A_3, ..., A_N$$

Ordenar significa permutar estos elementos de tal forma que queden de acuerdo con una distribución preestablecida.

- Ascendente: $A_1 \le A_2 \le A_3 \le ... \le A_N$
- Descendente: $A_1 \ge A_2 \ge A_3 \ge ... \ge A_N$

En el procesamiento de datos a los métodos de ordenación se les clasifica en dos grandes categorías, según donde hayan sido almacenados:

- Ordenación de arreglos.
- Ordenación de archivos.

La primera categoría se denomina también **ordenación interna**, ya que los elementos o componentes del arreglo se encuentran en la memoria principal de la computadora. La segunda categoría se llama **ordenación externa**, ya que los elementos se encuentran en archivos almacenados en dispositivos de almacenamiento secundario, como discos, cintas, tambores, etcétera.

Si se buscara una analogía entre los métodos de ordenación y la vida real, se podría mencionar que para la máquina, la ordenación interna representa lo que para un humano significa ordenar un conjunto de tarjetas que se encuentran visibles y extendidas todas sobre una mesa. La ordenación externa, en cambio, representa para la máquina lo que para un humano significa ordenar un conjunto de tarjetas que están dispuestas una debajo de otra y en donde sólo se visualiza la primera.

En la primera parte de este capítulo se estudiarán los métodos más importantes de ordenación interna y posteriormente los más interesantes de ordenación externa.

FIGURA 8.1
Ordenación interna.

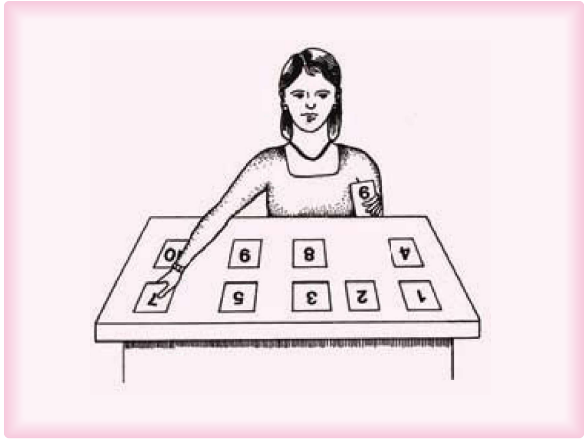
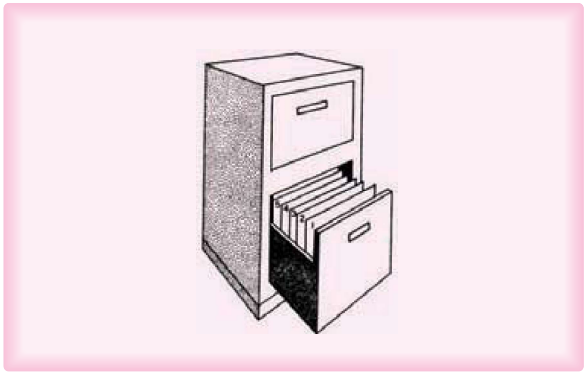


FIGURA 8.2
Ordenación externa.



8.2 ORDENACIÓN INTERNA

Los métodos de **ordenación interna** se explicarán con arreglos unidimensionales, pero su uso puede extenderse a otros tipos de arreglos y estructuras de datos. Es importante señalar, además, que se trabajará con métodos de ordenación *in situ*, es decir, métodos que no requieren de arreglos auxiliares para su ordenación. Los métodos que requieren de arreglos auxiliares son generalmente ineficientes e intrínsecamente de menor interés.

Los métodos de ordenación interna a su vez se pueden clasificar en dos tipos:

- Métodos directos (n^2).
- Métodos logarítmicos ($n * \log n$).

Los **métodos directos** tienen la característica de que su implementación es relativamente sencilla y son fáciles de comprender, aunque son ineficientes cuando N —el número de elementos del arreglo— es de tamaño mediano o grande. Los **métodos logarítmicos**, por su parte, son más complejos que los directos. Su elaboración es más sofisticada y, al ser menos intuitivos, resultan más difíciles de entender. Sin embargo, son más eficientes ya que requieren de menos comparaciones y movimientos para ordenar sus elementos.

Es importante destacar que una buena medida de eficiencia entre los distintos métodos la constituye el tiempo de ejecución del algoritmo y éste depende fundamentalmente del número de comparaciones y movimientos que se realicen entre sus elementos.

Como conclusión se puede señalar que cuando N es pequeño se deben utilizar métodos directos y cuando N es mediano o grande se usarán métodos logarítmicos.

Los métodos directos más conocidos son:

- Ordenación por intercambio.
- Ordenación por inserción.
- Ordenación por selección.

8.2.1 Ordenación por intercambio directo (burbuja)

El método de **intercambio directo**, conocido coloquialmente como **burbuja**, es el más utilizado entre los estudiantes principiantes de computación por su fácil comprensión y programación. Pero es preciso señalar que es quizás el método más ineficiente.

El método de intercambio directo puede trabajar de dos maneras diferentes: llevando los elementos más pequeños hacia la parte izquierda del arreglo o trasladando los elementos más grandes hacia su parte derecha. La idea básica de este algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentren ordenados. Se realizan $(n - 1)$ pasadas transportando en cada una de ellas el menor o mayor de elementos —según sea el caso— a su posición ideal. Al final de las $(n - 1)$ pasadas los elementos del arreglo estarán ordenados.

Ejemplo 8.1

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional A, transportando en cada pasada el menor elemento hacia la parte izquierda del arreglo.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son:

PRIMERA PASADA

$A[7] > A[8]$	(12 > 35)	no hay intercambio
$A[6] > A[7]$	(27 > 12)	sí hay intercambio
$A[5] > A[6]$	(44 > 12)	sí hay intercambio
$A[4] > A[5]$	(16 > 12)	sí hay intercambio
$A[3] > A[4]$	(08 > 12)	no hay intercambio
$A[2] > A[3]$	(67 > 08)	sí hay intercambio
$A[1] > A[2]$	(15 > 08)	sí hay intercambio

Luego de la primera pasada el arreglo queda así:

A: 08 15 67 12 16 44 27 35

Observe que el elemento más pequeño, en este caso 08, fue situado en la parte izquierda del arreglo.

SEGUNDA PASADA

$A[7] > A[8]$	(27 > 35)	no hay intercambio
$A[6] > A[7]$	(44 > 27)	sí hay intercambio
$A[5] > A[6]$	(16 > 27)	no hay intercambio

$A[4] > A[5]$ (12 > 16) no hay intercambio
$A[3] > A[4]$ (67 > 12) sí hay intercambio
$A[2] > A[3]$ (15 > 12) sí hay intercambio

Luego de la segunda pasada el arreglo queda así:

A: 08 12 15 67 16 27 44 35

y el segundo elemento más pequeño del arreglo, en este caso 12, fue situado en la segunda posición.

En la tabla 8.1 se presenta el resultado de las pasadas restantes.

TABLA 8.1

3a. pasada:	08	12	15	16	67	27	35	44
4a. pasada:	08	12	15	16	27	67	35	44
5a. pasada:	08	12	15	16	27	35	67	44
6a. pasada:	08	12	15	16	27	35	44	67
7a. pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de intercambio directo que transporta en cada pasada el *menor elemento* hacia la parte izquierda del arreglo es el siguiente:

Algoritmo 8.1 Burbuja_menor

Burbuja_menor (*A*, *N*)

{Este algoritmo ordena los elementos del arreglo unidimensional utilizando el método de la burbuja. Transporta en cada pasada el elemento más pequeño hacia la parte izquierda del arreglo. *A* es un arreglo unidimensional de *N* elementos}
{*I*, *J* y AUX son variables de tipo entero}

1. *Repetir* con *I* desde 2 hasta *N*
  1.1 *Repetir* con *J* desde *N* hasta *I*
    1.1.1 *Si* $A(J-1) > A[J]$ *entonces*
      Hacer AUX ← $A[J-1]$, $A[J-1] \leftarrow A[I]$ y $A[I] \leftarrow$ AUX
    1.1.2 {Fin del condicional del paso 1.1.1}
  1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}

Ejemplo 8.2

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional *A* transportando en cada pasada el mayor elemento hacia la parte derecha del arreglo:

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son:

PRIMERA PASADA

$A[1] > A[2]$ (15 > 67) no hay intercambio
$A[2] > A[3]$ (67 > 08) sí hay intercambio
$A[3] > A[4]$ (67 > 16) sí hay intercambio
$A[4] > A[5]$ (67 > 44) sí hay intercambio
$A[5] > A[6]$ (67 > 27) sí hay intercambio
$A[6] > A[7]$ (67 > 12) sí hay intercambio
$A[7] > A[8]$ (67 > 35) sí hay intercambio

A: 15 08 16 44 27 12 35 67

Observe que el elemento más grande, en este caso 67, fue situado en la última posición del arreglo.

SEGUNDA PASADA

$A[1] > A[2]$ (15 > 08) sí hay intercambio
$A[2] > A[3]$ (15 > 16) no hay intercambio
$A[3] > A[4]$ (16 > 44) no hay intercambio
$A[4] > A[5]$ (44 > 27) sí hay intercambio
$A[5] > A[6]$ (44 > 12) sí hay intercambio
$A[6] > A[7]$ (44 > 35) sí hay intercambio

A: 08 15 16 27 12 35 44 67

y el segundo elemento más grande del arreglo, en este caso 44, fue situado en la penúltima posición. En la tabla 8.2 se ve el resultado de las pasadas restantes.

TABLA 8.2

3a. pasada:	08	15	16	12	27	35	44	67
4a. pasada:	08	15	12	16	27	35	44	67
5a. pasada:	08	12	15	16	27	35	44	67
6a. pasada:	08	12	15	16	27	35	44	67
7a. pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de intercambio directo que transporta en cada pasada el elemento mayor hacia la parte derecha del arreglo es:

Algoritmo 8.2 Burbuja_mayor

Burbuja_mayor (*A*, *N*)

{El algoritmo ordena los elementos del arreglo unidimensional *A*. Transporta en cada pasada el elemento más grande hacia la parte derecha del arreglo. *A* es un arreglo de *N* elementos}
{*I*, *J* y AUX son variables de tipo entero}

```
1. Repetir con I desde N – 1 hasta 1
   1.1 Repetir con J desde 1 hasta I
       1.1.1 Si A[J] > A [J + 1] entonces
                 Hacer AUX ← A[J], A[J] ← A[J + 1] y A[J + 1] ← AUX
       1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
2. {Fin del ciclo del paso 1}
```

Análisis de eficiencia del método de intercambio directo

El número de comparaciones que se realizan en el método de la burbuja se puede contabilizar fácilmente. En la primera pasada se realizan $(n-1)$ comparaciones, en la segunda pasada $(n-2)$ comparaciones, en la tercera pasada $(n-3)$ comparaciones y así sucesivamente hasta llegar a 2 y 1 comparaciones entre claves, siendo n el número de elementos del arreglo. Por lo tanto, tenemos que el número de comparaciones es:

$$C = (n-1)+(n-2)+\ldots+2+1 = \frac{n*(n-1)}{2}$$

que es igual a:

$$C = \frac{n^2-n}{2} \qquad \textbf{Fórmula 8.1}$$

Como ya se mencionó en el capítulo 2, se hace uso del principio de inducción matemática para desarrollar ciertas fórmulas.

Respecto del número de movimientos, éstos dependen fundamentalmente de si el arreglo se encuentra ordenado, desordenado o en orden inverso. Los movimientos para cada uno de estos casos son:

$$M_{mín} = 0 \qquad M_{med} = 0.75 * (n^2-n) \qquad M_{máx} = 1.5 * (n^2-n) \qquad \textbf{Fórmula 8.2}$$

Así, por ejemplo, si se tiene que ordenar un arreglo que contiene 500 elementos, el número de movimientos que se tendrá que realizar es:

a) Si el arreglo se encuentra ordenado:
- 124 750 comparaciones
- 0 movimientos

b) Si los elementos del arreglo se encuentran dispuestos en forma aleatoria:
- 124 750 comparaciones
- 187 125 movimientos

c) Si los elementos del arreglo se encuentran en orden inverso:
- 124 750 comparaciones
- 374 250 movimientos

Ahora bien, el tiempo necesario para ejecutar el algoritmo de la burbuja es proporcional a n^2, $O(n^2)$, donde n es el número de elementos del arreglo.

A continuación se presentarán dos variantes del método de intercambio directo: método de intercambio directo con señal y método de *shaker sort*.

8.2.2 Ordenación por el método de intercambio directo con señal

Este método es una modificación del método de intercambio directo analizado en la sección anterior. La idea central de este algoritmo consiste en utilizar una marca o señal para indicar que no se ha producido ningún intercambio en una pasada. Es decir, se comprueba si el arreglo está totalmente ordenado después de cada pasada, terminando su ejecución en caso afirmativo. El algoritmo de ordenación por el método de la burbuja con señal es:

Algoritmo 8.3 Burbuja_señal

Burbuja_señal (*A*, *N*)

{El algoritmo ordena los elementos del arreglo utilizando el método de la burbuja con señal. *A* es un arreglo unidimensional de *N* elementos}
{*I*, *J* y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

```
1. Hacer I ← 1 y BAND ← FALSO
2. Mientras ((I ≤ N – 1) y (BAND = FALSO)) Repetir
       Hacer BAND ← VERDADERO
   2.1 Repetir con J desde 1 hasta N – 1
       2.1.1 Si (A[J] > A[J + 1]) entonces
                 Hacer AUX ← A[J], A[J] ← A[J + 1], A[J + 1] ← AUX
                 y BAND ← FALSO
       2.1.2 {Fin del condicional del paso 2.1.1}
   2.2 {Fin del ciclo del paso 2.1}
       Hacer I ← I + 1
3. {Fin del ciclo del paso 2}
```

A continuación presentamos la segunda variante del método de intercambio directo.

8.2.3 Ordenación por el método de la sacudida (*shaker sort*)

El método de *shaker sort,* más conocido como el método de la sacudida, es una optimización del método de intercambio directo. La idea básica de este algoritmo consiste en mezclar las dos formas en que se puede realizar el método de la burbuja.

En este algoritmo cada pasada tiene dos etapas. En la primera etapa, de derecha a izquierda, se trasladan los elementos más pequeños hacia la parte izquierda del arreglo, almacenando en una variable la posición del último elemento intercambiado. En la segunda etapa, de izquierda a derecha, se trasladan los elementos más grandes hacia la parte derecha del arreglo, almacenando en otra variable la posición del último elemento intercambiado. Las sucesivas pasadas trabajan con los componentes del arreglo comprendidos entre las posiciones almacenadas en las variables auxiliares. El algoritmo termina cuando en una etapa no se producen intercambios, o bien cuando el contenido de la variable que guarda el extremo izquierdo del arreglo es mayor que el contenido de la variable que almacena el extremo derecho.

Ejemplo 8.3

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional *A* utilizando el método de la sacudida.

A: 15 67 08 16 44 27 12 35

PRIMERA PASADA

Primera etapa (de derecha a izquierda)

$A[7] > A[8]$ (12 > 35) no hay intercambio
$A[6] > A[7]$ (27 > 12) sí hay intercambio
$A[5] > A[6]$ (44 > 12) sí hay intercambio
$A[4] > A[5]$ (16 > 12) sí hay intercambio
$A[3] > A[4]$ (08 > 12) no hay intercambio
$A[2] > A[3]$ (67 > 08) sí hay intercambio
$A[1] > A[2]$ (15 > 08) sí hay intercambio

Última posición de intercambio de derecha a izquierda: 2.

Luego de la primera etapa de la primera pasada, el arreglo queda así:

A: [08] 15 67 12 16 44 27 35

Segunda etapa (de izquierda a derecha)

$A[2] > A[3]$ (15 > 67) no hay intercambio
$A[3] > A[4]$ (67 > 12) sí hay intercambio
$A[4] > A[5]$ (67 > 16) sí hay intercambio
$A[5] > A[6]$ (67 > 44) sí hay intercambio
$A[6] > A[7]$ (67 > 27) sí hay intercambio
$A[7] > A[8]$ (67 > 35) sí hay intercambio

Última posición de intercambio de izquierda a derecha: 8.

Luego de la segunda etapa de la primera pasada, el arreglo queda así:

A: [08] 15 12 16 44 27 35 [67]

SEGUNDA PASADA

Primera etapa (de derecha a izquierda)

$A[6] > A[7]$ (27 > 35) no hay intercambio
$A[5] > A[6]$ (44 > 27) sí hay intercambio
$A[4] > A[5]$ (16 > 27) no hay intercambio
$A[3] > A[4]$ (12 > 16) no hay intercambio
$A[2] > A[3]$ (15 > 12) sí hay intercambio

Última posición de intercambio de derecha a izquierda: 3.

A: [08] [12] 15 16 27 44 35 [67]

Segunda etapa (de izquierda a derecha)

$A[3] > A[4]$ (15 > 16) no hay intercambio
$A[4] > A[5]$ (16 > 27) no hay intercambio
$A[5] > A[6]$ (27 > 44) no hay intercambio
$A[6] > A[7]$ (44 > 35) sí hay intercambio

Última posición de intercambio de izquierda a derecha: 7.

A: [08] [12] 15 16 27 34 [44] [67]

Al realizar la primera etapa de la tercera pasada se observa que no se realizaron intercambios; por lo tanto, la ejecución del algoritmo se termina. El algoritmo de ordenación por el método de la sacudida es el siguiente:

Algoritmo 8.4 Sacudida

Sacudida (*A*, *N*)

{El algoritmo ordena los elementos de un arreglo unidimensional utilizando el método de la sacudida. *A* es un arreglo de *N* elementos}
{*I*, IZQ, DER, *K* y AUX son variables de tipo entero}

1. Hacer IZQ ← 2, DER ← *N* y *K* ← *N*
2. *Mientras* (DER ≥ IZQ) *Repetir*
   2.1 *Repetir* con *I* desde DER hasta IZQ {Ciclo descendente}

```
      2.1.1 Si (A[I – 1] > A[I]) entonces
              Hacer AUX ← A[I – 1], A[I – 1] ← A[I], A[I] ← AUX y K ← I
      2.1.2 {Fin del condicional del paso 2.1.1}
   2.2 {Fin del ciclo del paso 2.1}
      Hacer IZQ ← K + 1
   2.3 Repetir con I desde IZQ hasta DER {Ciclo ascendente}
      2.3.1 Si (A[I – 1] > A[I]) entonces
              Hacer AUX ← A[I – 1], A[I – 1] ← A[I], A[I] ← AUX y K ← I
      2.3.2 {Fin del condicional del paso 2.2.1}
   2.4 {Fin del ciclo del paso 2.3}
      Hacer DER ← K – 1
3. {Fin del ciclo 2}
```

Análisis de eficiencia del método de la sacudida

El análisis del método de la sacudida, y en general el de los métodos mejorados y logarítmicos, es muy complejo. Para el análisis de este método es necesario tener en cuenta tres factores que afectan directamente al tiempo de ejecución del algoritmo: las comparaciones entre las claves, los intercambios entre ellas y las pasadas que se realizan. Encontrar fórmulas que permitan calcular cada uno de estos factores es una tarea muy difícil de realizar.

Los estudios que se han efectuado sobre el método de la sacudida demuestran que en él sólo se pueden reducir las dobles comparaciones entre claves, pero se debe recordar que la operación de intercambio es una tarea más complicada y costosa que la de comparación. Por lo tanto, es posible afirmar que las hábiles mejoras realizadas sobre el método de intercambio directo sólo producen resultados apreciables si el arreglo está parcialmente ordenado, lo cual resulta difícil saber de antemano; pero si el arreglo está desordenado el método se comporta, incluso, peor que otros métodos directos, como los de inserción y selección.

8.2.4 Ordenación por inserción directa

El método de ordenación por **inserción directa** es el que utilizan generalmente los jugadores de cartas cuando las ordenan, de ahí que también se conozca con el nombre de método de la baraja.

La idea central de este algoritmo consiste en insertar un elemento del arreglo en su parte izquierda, que ya se encuentra ordenada. Este proceso se repite desde el segundo hasta el n-ésimo elemento. Observemos un ejemplo.

Ejemplo 8.4

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional A utilizando el método de inserción directa:

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son:

PRIMERA PASADA

$A[2] < A[1]$ (67 < 15) no hay intercambio

A: [15] [67] 08 16 44 27 12 35

SEGUNDA PASADA

$A[3] < A[2]$ (08 < 67) sí hay intercambio
$A[2] < A[1]$ (08 < 15) sí hay intercambio

A: [08] [15] [67] 16 44 27 12 35

TERCERA PASADA

$A[4] < A[3]$ (16 < 67) sí hay intercambio
$A[3] < A[2]$ (16 < 15) no hay intercambio

A: [08] [15] [16] [67] 44 27 12 35

Observe que una vez que se determina la posición correcta del elemento se interrumpen las comparaciones. Por ejemplo, para el caso anterior no se realizó la comparación $A[2] < A[1]$. En la tabla 8.3 se presenta el resultado de las pasadas restantes.

El algoritmo de ordenación por el método de inserción directa es:

Algoritmo 8.5 Inserción

Inserción (A, N)

{Este algoritmo ordena los elementos del arreglo utilizando el método de inserción directa. A es un arreglo unidimensional de N elementos}
{I, AUX y K son variables de tipo entero}

```
1. Repetir con I desde 2 hasta N
      Hacer AUX ← A[I] y K ← I – 1
   1.1 Mientras ((K ≥ 1) y (AUX < A[K])) Repetir
         Hacer A[K + 1] ← A[K] y K ← K – 1
   1.2 {Fin del ciclo del paso 1.1}
      Hacer A[K + 1] ← AUX
2. {Fin del ciclo del paso 1}
```

TABLA 8.3

4a. pasada:	08	15	16	44	67	27	12	35
5a. pasada:	08	15	16	27	44	67	12	35
6a. pasada:	08	12	15	16	27	44	67	35
7a. pasada:	08	12	15	16	27	35	44	67

Análisis de eficiencia del método de inserción directa

El número mínimo de comparaciones y movimientos entre claves se produce cuando los elementos del arreglo ya están ordenados. Analice el siguiente caso:

Ejemplo 8.5

Sea *A* un arreglo formado por los elementos:

A: 15 20 45 52 86

Las comparaciones que se realizan son:

PRIMERA PASADA

$A[2] < A[1]$ $(20 < 15)$ no hay intercambio

SEGUNDA PASADA

$A[3] < A[2]$ $(45 < 20)$ no hay intercambio

TERCERA PASADA

$A[4] < A[3]$ $(52 < 45)$ no hay intercambio

CUARTA PASADA

$A[5] < A[4]$ $(86 < 52)$ no hay intercambio

Luego de las comparaciones realizadas, el arreglo correspondiente queda así:

A: | 15 | | 20 | | 45 | | 52 | | 86 |

Observe que para este ejemplo se efectuaron cuatro comparaciones. En general, podemos afirmar que si el arreglo se encuentra ordenado se efectúan como máximo $n - 1$ comparaciones y 0 movimientos entre elementos.

$$C_{min} = n - 1 \qquad \textbf{Fórmula 8.3}$$

El número máximo de comparaciones y movimientos entre elementos se produce cuando los elementos del arreglo están en orden inverso.

Ejemplo 8.6

Sea *A* un arreglo formado por los siguientes elementos:

A: 86 52 45 20 15

Las comparaciones que se realizan son:

PRIMERA PASADA

$A[2] < A[1]$ $(52 < 86)$ sí hay intercambio

SEGUNDA PASADA

$A[3] < A[2]$ $(45 < 86)$ sí hay intercambio
$A[2] < A[1]$ $(45 < 52)$ sí hay intercambio

TERCERA PASADA

$A[4] < A[3]$ $(20 < 86)$ sí hay intercambio
$A[3] < A[2]$ $(20 < 52)$ sí hay intercambio
$A[2] < A[1]$ $(20 < 45)$ sí hay intercambio

CUARTA PASADA

$A[5] < A[4]$ $(15 < 86)$ sí hay intercambio
$A[4] < A[3]$ $(15 < 52)$ sí hay intercambio
$A[3] < A[2]$ $(15 < 45)$ sí hay intercambio
$A[2] < A[1]$ $(15 < 20)$ sí hay intercambio

Observe que en la primera pasada se realizó una comparación; en la segunda, dos comparaciones; en la tercera, tres comparaciones, y así sucesivamente hasta $n - 1$ comparaciones entre elementos. Por lo tanto:

$$M_{máx} = 1 + 2 + 3 + \ldots + (n-1) = \frac{n*(n-1)}{2}$$

que es igual a

$$C_{máx} = \frac{(n^2 - n)}{2} \qquad \textbf{Fórmula 8.4}$$

Ahora bien, el número de comparaciones promedio, que es cuando los elementos aparecen en el arreglo en forma aleatoria, se puede calcular mediante la suma de las comparaciones mínimas y máximas dividida entre 2.

$$C_{med} = \frac{\left[(n-1) + \frac{(n^2 - n)}{2}\right]}{2}$$

Al hacer la operación queda:

$$C_{med} = \frac{(n^2 + n - 2)}{4}$$ **Fórmula 8.5**

Respecto del número de movimientos, si el arreglo se encuentra ordenado no se realiza ninguno. Por lo tanto:

$$M_{mín} = 0$$ **Fórmula 8.6**

El número máximo de movimientos se presenta cuando el arreglo está en orden inverso. Observe el ejemplo anterior. En la primera pasada se realizó un movimiento, en la segunda dos y así sucesivamente hasta $n - 1$ movimientos entre los elementos en la última pasada. Por lo tanto:

$$M_{máx} = 1 + 2 + 3 + \ldots + (n-1) = \frac{n*(n-1)}{2}$$

que es igual a

$$M_{máx} = \frac{(n^2 - n)}{2}$$ **Fórmula 8.7**

El número de movimientos promedio, que se da cuando los elementos se encuentran en el arreglo en forma aleatoria, se calcula como la suma de los movimientos mínimos y máximos dividida entre 2. Por lo tanto:

$$M_{med} = \frac{0 + \frac{(n^2 - n)}{2}}{2}$$

Al hacer la operación queda:

$$M_{med} = \frac{n^2 - n}{4}$$ **Fórmula 8.8**

Así, por ejemplo, si se tiene que ordenar un arreglo que contiene 500 elementos:

a) Si el arreglo se encuentra ordenado serán necesarias:
- 499 comparaciones
- 0 movimientos

b) Si los elementos del arreglo se encuentran dispuestos en forma aleatoria se realizarán:

- 62 624 comparaciones, en promedio.
- 62 375 movimientos, en promedio.

c) Si los elementos del arreglo se encuentran en orden inverso serán necesarias:
- 124 750 comparaciones.
- 124 750 movimientos.

Es importante señalar que el tiempo requerido para ejecutar el algoritmo de inserción directa es proporcional a n^2, $O(n^2)$, donde n es el número de elementos del arreglo.

A pesar de ser un método ineficiente y recomendable sólo cuando n es pequeño, el método de inserción directa se comporta mejor que los métodos de intercambio directo analizados anteriormente.

8.2.5 Ordenación por el método de inserción binaria

El método de ordenación por **inserción binaria** es una mejora del método de inserción directa presentado anteriormente. La mejora consiste en realizar una búsqueda binaria en lugar de una búsqueda secuencial, para insertar un elemento en la parte izquierda del arreglo, que ya se encuentra ordenado. El proceso, al igual que en el método de inserción directa, se repite desde el segundo hasta el n-ésimo elemento. Analicemos un ejemplo.

Ejemplo 8.7

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional A utilizando el método de inserción binaria.

A: 15 67 08 16 44 27 12 35

A continuación se presentan las comparaciones que se llevan a cabo:

PRIMERA PASADA

$A[2] < A[1]$ (67 < 15) no hay intercambio

A: [15] [67] 08 16 44 27 12 35

SEGUNDA PASADA

$A[3] < A[1]$ (08 < 15) sí hay intercambio

A: [08] [15] [67] 16 44 27 12 35

TERCERA PASADA

$A[4] < A[2]$ (16 < 15) no hay intercambio
$A[4] < A[3]$ (16 < 67) sí hay intercambio

A: [08] [15] [16] [67] 44 27 12 35

CUARTA PASADA

$A[5] < A[2]$ (44 < 15) no hay intercambio
$A[5] < A[3]$ (44 < 16) no hay intercambio
$A[5] < A[4]$ (44 < 67) sí hay intercambio

A: 08 15 16 44 67 27 12 35

QUINTA PASADA

$A[6] < A[3]$ (27 < 16) no hay intercambio
$A[6] < A[4]$ (27 < 44) sí hay intercambio

A: 08 15 16 27 44 67 12 35

SEXTA PASADA

$A[7] < A[3]$ (12 < 16) sí hay intercambio
$A[7] < A[1]$ (12 < 08) no hay intercambio
$A[7] < A[2]$ (12 < 15) sí hay intercambio

A: 08 12 15 16 27 44 67 35

SÉPTIMA PASADA

$A[8] < A[4]$ (35 < 16) no hay intercambio
$A[8] < A[6]$ (35 < 44) sí hay intercambio
$A[8] < A[5]$ (35 < 27) no hay intercambio

A: 08 12 15 16 27 35 44 67

El algoritmo de ordenación por el método de inserción binaria es:

Algoritmo 8.6 Inserción_binaria

Inserción_binaria (*A*, *N*)

{Este algoritmo ordena los elementos de un arreglo unidimensional utilizando el método de inserción binaria. *A* es un arreglo unidimensional de *N* elementos}
{*I*, AUX, IZQ, DER, *M* y *J* son variables de tipo entero}

```
1. Repetir con I desde 2 hasta N
        Hacer AUX ← A[I], IZQ ← 1 y DER ← I – 1
   1.1 Mientras (IZQ ≤ DER) Repetir
            Hacer M ← parte entera ((IZQ + DER) entre 2)
```



```
            1.1.1 Si (AUX ≤ A[M])
                  entonces
                     Hacer DER ← M – 1
                  si no
                     Hacer IZQ ← M + 1
            1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del ciclo del paso 1.1}
       Hacer J ← I – 1
   1.3 Mientras (J ≥ IZQ) Repetir
            Hacer A[J + 1] ← A[J] y J ← J – 1
   1.4 {Fin del ciclo del paso 1.3}
       Hacer A[IZQ] ← AUX
2. {Fin del ciclo del paso 1}
```

Análisis de eficiencia del método de inserción binaria

Al analizar el método de ordenación por inserción binaria se advierte la presencia de un caso antinatural. El método efectúa el menor número de comparaciones cuando el arreglo está totalmente desordenado y el máximo cuando se encuentra ordenado.

Es posible suponer que mientras en una búsqueda secuencial se necesitan K comparaciones para insertar un elemento, en una binaria se necesitará la mitad de las K comparaciones. Por lo tanto, el número de comparaciones promedio en el método de ordenación por inserción binaria se puede calcular como:

$$C = \frac{1}{2} + \frac{2}{2} + \frac{3}{2} + \ldots + \frac{(n-1)}{2} = \frac{n*(n-1)}{4}$$

que es igual a:

$$C = \frac{(n^2 - n)}{4} \qquad \textbf{Fórmula 8.9}$$

Éste es un algoritmo de comportamiento antinatural y, por lo tanto, es necesario ser muy cuidadoso cuando se hace un análisis de él. Las hábiles mejoras introducidas producen un efecto negativo cuando el arreglo está ordenado y resultados apenas satisfactorios cuando las claves están desordenadas. De todas maneras, se debe recordar que no se reduce el número de movimientos que es una operación más complicada y costosa que la operación de comparación. Por lo tanto, el tiempo de ejecución del algoritmo sigue siendo proporcional a n^2, $O(n^2)$.

8.2.6 Ordenación por selección directa

El método de ordenación por **selección directa** es más eficiente que los métodos analizados anteriormente. Pero, aunque su comportamiento es mejor que el de aquéllos y su

programación es fácil y comprensible, no se recomienda utilizarlo cuando el número de elementos del arreglo es mediano o grande. La idea básica de este algoritmo consiste en buscar el menor elemento del arreglo y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño del arreglo y se lo coloca en la segunda posición. El proceso continúa hasta que todos los elementos del arreglo hayan sido ordenados. El método se basa en los siguientes principios:

1. Seleccionar el menor elemento del arreglo.
2. Intercambiar dicho elemento con el primero.
3. Repetir los pasos anteriores con los $(n-1)$, $(n-2)$ elementos, y así sucesivamente hasta que sólo quede el elemento mayor.

Ejemplo 8.8

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional A utilizando el método de selección directa.

A: 15 67 08 16 44 27 12 35

Las comparaciones que se realizan son:

PRIMERA PASADA

Se realiza la asignación: MENOR ← $A[1]$ (15)

(MENOR < $A[2]$) (15 < 67) sí se cumple la condición
(MENOR < $A[3]$) (15 < 08) no se cumple la condición
MENOR ← $A[3]$ (8)
(MENOR < $A[4]$) (08 < 16) sí se cumple la condición
(MENOR < $A[5]$) (08 < 44) sí se cumple la condición
(MENOR < $A[6]$) (08 < 27) sí se cumple la condición
(MENOR < $A[7]$) (08 < 12) sí se cumple la condición
(MENOR < $A[8]$) (08 < 35) sí se cumple la condición

Luego de la primera pasada, el arreglo queda de la siguiente forma:

A: 08 67 15 16 44 27 12 35

Observe que el menor elemento del arreglo $A[3](08)$ se intercambió con el primer elemento $A[1](15)$, realizando solamente un movimiento.

SEGUNDA PASADA

Se realiza la siguiente asignación: MENOR ← $A[2](67)$

(MENOR < $A[3]$) (67 < 15) no se cumple la condición
MENOR ← $A[3]$ (15)
(MENOR < $A[4]$) (15 < 16) sí se cumple la condición
(MENOR < $A[5]$) (15 < 44) sí se cumple la condición

(MENOR < $A[6]$) (15 < 27) sí se cumple la condición
(MENOR < $A[7]$) (15 < 12) no se cumple la condición
MENOR ← $A[7](12)$
(MENOR < $A[8]$) (12 < 35) sí se cumple la condición

A: 08 12 15 16 44 27 67 35

Observe que el segundo elemento más pequeño del arreglo $A[7](12)$ se intercambió con el segundo elemento, $A[2](67)$.

En la tabla 8.4 se observa el resultado de las pasadas restantes.

TABLA 8.4

3a. pasada:	08	12	15	16	44	27	67	35
4a. pasada:	08	12	15	16	44	27	67	35
5a. pasada:	08	12	15	16	27	44	67	35
6a. pasada:	08	12	15	16	27	35	67	44
7a. pasada:	08	12	15	16	27	35	44	67

El algoritmo de ordenación por el método de selección directa es:

Algoritmo 8.7 Selección

Selección (A, N)

{Este algoritmo ordena los elementos de un arreglo unidimensional utilizando el método de selección directa. A es un arreglo unidimensional de N elementos}
{I, MENOR, K y J son variables de tipo entero}

1. *Repetir* con I desde 1 hasta $N-1$
Hacer MENOR ← $A[I]$ y K ← I
1.1 *Repetir* con J desde $I+1$ hasta N
1.1.1 *Si* ($A[J]$ < MENOR) *entonces*
Hacer MENOR ← $A[J]$ y K ← J
1.1.2 {Fin del condicional del paso 1.1.1}
1.2 {Fin del ciclo del paso 1.1}
Hacer $A[K]$ ← $A[I]$ y $A[I]$ ← MENOR
2. {Fin del ciclo del paso 1}

Análisis de eficiencia del método de selección directa

El análisis del método de selección directa es relativamente simple. Se debe considerar que el número de comparaciones entre elementos es independiente de la disposición inicial de éstos en el arreglo. En la primera pasada se realizan $(n-1)$ comparaciones, en la segunda pasada $(n-2)$ comparaciones y así sucesivamente hasta 2 y 1 comparaciones, en la penúltima y última pasadas, respectivamente. Por lo tanto:

$$C = (n-1)+(n-2)+\ldots+2+1 = \frac{n*(n-1)}{2}$$

que es igual a:

$$C = \frac{n^2 - n}{2} \qquad \textbf{Fórmula 8.10}$$

Respecto del número de intercambios, siempre será $n - 1$, a excepción de que se tenga incorporada en el algoritmo alguna técnica para prevenir el intercambio de un elemento consigo mismo. Por lo tanto:

$$M = n - 1 \qquad \textbf{Fórmula 8.11}$$

Así, por ejemplo, si se tiene que ordenar un arreglo que contiene 500 elementos, se efectuarán 124 750 comparaciones y 499 movimientos.

El tiempo de ejecución del algoritmo es proporcional a n^2, $O(n^2)$, aun cuando es más rápido que los métodos presentados con anterioridad.

8.2.7 Análisis de eficiencia de los métodos directos

La tabla 8.5 contiene las fórmulas necesarias para obtener el número de comparaciones y movimientos para ordenar un arreglo con los tres métodos directos analizados. Las columnas indican si los elementos del arreglo se encuentran en forma ordenada, desordenada o en orden inverso.

En la tabla 8.6, por otra parte, se observan los números de comparaciones y movimientos necesarios para ordenar un arreglo con los tres métodos directos analizados.

TABLA 8.5

		Ordenada	Desordenada	Orden inverso
Intercambio directo	C	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$
	M	0	$0.75*(n^2-n)$	$1.5*(n^2-n)$
Inserción directa	C	$(n-1)$	$\frac{(n^2+n-2)}{4}$	$\frac{n^2-n}{2}$
	M	0	$\frac{n^2-n}{4}$	$\frac{n^2-n}{2}$
Selección directa	C	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$	$\frac{n^2-n}{2}$
	M	$n-1$	$n-1$	$n-1$

TABLA 8.6

		Ordenada		Desordenada		Orden inverso	
Intercambio directo	C	124 750	499 500	124 750	499 500	124 750	499 500
	M	0	0	187 125	749 250	374 250	1 498 500
Inserción directa	C	499	999	62 624	250 249	124 750	499 500
	M	0	0	62 375	249 750	124 750	499 500
Selección directa	C	124 750	499 500	124 750	499 500	124 750	499 500
	M	499	999	999	999	499	999

Las columnas indican si los elementos del arreglo se encuentran en forma ordenada, desordenada o en orden inverso. Observe que estas columnas se encuentran divididas en dos. La subcolumna izquierda representa un arreglo de 500 elementos y la subcolumna derecha representa un arreglo de 1 000 elementos.

Es fácil observar que el método de selección directa es el mejor y sólo es superado por el método de inserción directa cuando los elementos del arreglo ya se encuentran ordenados. El peor método, sin duda, es el de intercambio directo.

En los siguientes incisos se analizarán los métodos logarítmicos más importantes.

8.2.8 Ordenación por el método de Shell

El **método de Shell** es una versión mejorada del método de inserción directa. Recibe ese nombre en honor de su autor, Donald L. Shell, quien lo propuso en 1959. Este método también se conoce como **inserción con incrementos decrecientes**.

En el método de ordenación por inserción directa cada elemento se compara para su ubicación correcta en el arreglo con los elementos que se encuentran en su parte izquierda. Si el elemento a insertar es más pequeño que el grupo de elementos que se encuentran a su izquierda, será necesario efectuar varias comparaciones antes de su ubicación.

Shell propone que las comparaciones entre elementos se efectúen con saltos de mayor tamaño, pero con incrementos decrecientes; así, los elementos quedarán ordenados en el arreglo más rápidamente. Para comprender mejor este algoritmo analice el siguiente caso.

Consideremos un arreglo que contenga 16 elementos. En primer lugar, se dividirán los elementos del arreglo en ocho grupos, teniendo en cuenta los elementos que se encuentran a ocho posiciones de distancia entre sí y se ordenarán por separado. Quedarán en el primer grupo los elementos ($A[1]$, $A[9]$); en el segundo, ($A[2]$, $A[10]$); en el tercero, ($A[3]$, $A[11]$), y así sucesivamente. Después de este primer paso se dividirán los elementos del arreglo en cuatro grupos, teniendo en cuenta ahora los elementos que se encuentren a cuatro posiciones de distancia entre sí y se les ordenará por separado. Que-

darán en el primer grupo los elementos ($A[1]$, $A[5]$, $A[9]$, $A[13]$); en el segundo ($A[2]$, $A[6]$, $A[10]$, $A[14]$), y así sucesivamente. En el tercer paso se dividirán los elementos del arreglo en grupos, tomando en cuenta los elementos que se encuentran ahora a dos posiciones de distancia entre sí y nuevamente se les ordenará por separado. En el primer grupo quedarán ($A[1]$, $A[3]$, $A[5]$, $A[7]$, $A[9]$, $A[11]$, $A[13]$, $A[15]$) y en el segundo ($A[2]$, $A[4]$, $A[6]$, $A[8]$, $A[10]$, $A[12]$, $A[14]$, $A[16]$).

Finalmente se agruparán y ordenarán los elementos de manera normal; es decir, de uno en uno. Se presenta a continuación un ejemplo.

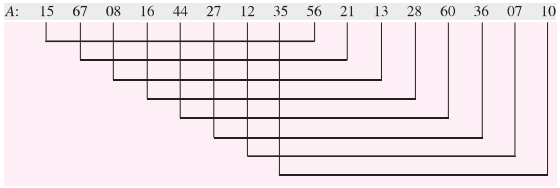
Ejemplo 8.9

Supongamos que se desea ordenar los elementos que se encuentran en el arreglo unidimensional A utilizando el método de Shell.

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

PRIMERA PASADA

Se dividen los elementos en 8 grupos:

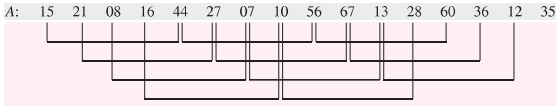


La ordenación produce:

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

SEGUNDA PASADA

Se dividen los elementos en 4 grupos:

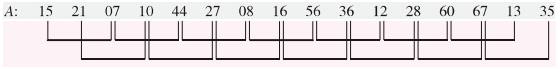


La ordenación produce:

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

TERCERA PASADA

Se dividen los elementos en 2 grupos:

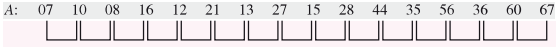


La ordenación produce

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

CUARTA PASADA

Se dividen los elementos en un solo grupo:



La ordenación produce:

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

A continuación se presenta el algoritmo de ordenación por el método de Shell.

Algoritmo 8.8 Shell

Shell (A, N)

{Este algoritmo permite ordenar los elementos de un arreglo unidimensional utilizando el método de Shell. A es un arreglo unidimensional de N elementos}
{INT, I y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

```
1. Hacer INT ← N + 1
2. Mientras (INT > 1) Repetir
       Hacer INT ← parte entera (INT / 2) y BAND ← VERDADERO
   2.1 Mientras (BAND = VERDADERO) Repetir
           Hacer BAND ← FALSO e I ← 1
       2.1.1 Mientras ((I + INT) ≤ N) Repetir
             2.1.1.1 Si A[I] > A[I + INT] entonces
                         Hacer AUX ← A[I], A[I] ← A[I + INT], A[I + INT] ← AUX
                           y BAND ← VERDADERO
             2.1.1.2 {Fin del condicional del paso 2.1.1.1}
                         Hacer I ← I + 1
       2.1.2 {Fin del ciclo del paso 2.1.1}
   2.2 {Fin del ciclo del paso 2.1}
3. {Fin del ciclo del paso 2}
```

Análisis de eficiencia del método de Shell

El análisis de eficiencia del método de Shell es un problema muy complicado y aún no resuelto. Hasta el momento no se ha podido establecer la mejor secuencia de incrementos cuando n es grande. Cabe recordar que cada vez que se propone una secuencia de intervalos, es necesario *correr* el algoritmo para analizar su tiempo de ejecución.

En 1969, Pratt descubrió que el tiempo de ejecución del algoritmo es del orden de $n * (\log n)^2$. Unas pruebas exhaustivas realizadas para obtener la mejor secuencia de intervalos cuando el número de elementos del arreglo es igual a 8 arrojaron como resultado que la mejor secuencia corresponde a un intervalo de 1, que no es más que el método de inserción directa estudiado previamente. Estas pruebas también determinaron que el menor número de movimientos se registraba con la secuencia 3, 2, 1. Cabe aclarar que las pruebas exhaustivas corresponden al análisis de (8!) posibilidades; es decir, 40 320 casos diferentes.

En la tabla 8.7 se muestran las diez mejores secuencias obtenidas al evaluar las 40 320 posibilidades de secuencias que se presentan cuando se tiene un arreglo de 8 elementos.

Para concluir con el análisis de eficiencia de método de Shell, se menciona que estudios de Peterson y Russell, en la Universidad de Stanford, en 1971, muestran que las mejores secuencias para valores de N comprendidos entre 100 y 60 000 son las que se presentan en la tabla 8.8, donde $k = 0, 1, 2, 3,...$

Por último, y para aclarar aún más los conceptos vertidos sobre el método de Shell, se incluye un segundo ejemplo.

Ejemplo 8.10

Supongamos que se desea ordenar las siguientes claves del arreglo unidimensional A utilizando el método de Shell. La secuencia de intervalos que se utilizará corresponde a la fórmula $(2k - 1)$ presentada por Peterson y Russell.

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

TABLA 8.7

Posición	Secuencia	
1	1	
2	6	1
3	5	1
4	7	1
5	4	1
6	3	1
7	2	1
8	5	3	1
9	4	2	1
10	3	2	1

TABLA 8.8

Secuencias	1, 3, 5, 9, ..., $2^k + 1$
	1, 3, 7, 15, ..., $2^k - 1$
	1, 3, 5, 11, ..., $(2^k \pm 1)/3$
	1, 4, 13, 40, ..., $(3^k + 1)/2$

Los resultados parciales de cada pasada, así como el resultado final, se observan en la tabla 8.9, donde PAS representa el número de pasada e INT representa el intervalo en el cual se está trabajando.

8.2.9 Ordenación por el método *quicksort*

El método de ordenación ***quicksort*** es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido como **método rápido** y de **ordenación por partición**. Este método es una mejora sustancial del método de intercambio directo y se denomina *quicksort* —rápido— por la velocidad con que ordena los elementos del arreglo. Su autor, C. A. Hoare, lo llamó así. La idea central de este algoritmo consiste en lo siguiente:

1. Se toma un elemento X de una posición cualquiera del arreglo.
2. Se trata de ubicar a X en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentren a su izquierda sean menores o iguales a X y todos los que se encuentren a su derecha sean mayores o iguales a X.
3. Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición de X en el arreglo.
4. El proceso termina cuando todos los elementos se encuentran en su posición correcta en el arreglo.

Se debe seleccionar, entonces, un elemento X cualquiera. En este caso se seleccionará $A[1]$. Se empieza a recorrer el arreglo de derecha a izquierda comparando si los elementos son mayores o iguales a X. Si un elemento no cumple con esta condición, se intercambian aquéllos y se almacena en una variable la posición del elemento intercambiado —se acota el arreglo por la derecha—. Se inicia nuevamente el recorrido, pero ahora de izquierda a derecha, comparando si los elementos son menores o iguales a X.

TABLA 8.9

PAS	Arreglo A																INT
1	15	67	08	16	44	27	12	35	56	21	13	28	60	36	07	10	15
2	10	67	08	16	44	27	12	35	56	21	13	28	60	36	07	15	07
3	07	15	08	13	28	27	12	10	56	21	16	44	60	36	35	67	03
4	07	10	08	12	15	27	13	16	35	21	28	44	60	36	56	67	01
	07	08	10	12	13	15	16	21	27	28	35	36	44	56	60	67	

Si un elemento no cumple con esta condición, entonces se intercambian aquéllos y se almacena en otra variable la posición del elemento intercambiado —se acota el arreglo por la izquierda—. Se repiten los pasos anteriores hasta que el elemento X encuentra su posición correcta en el arreglo. Analicemos a continuación el siguiente ejemplo.

Ejemplo 8.11

Supongamos que se desea ordenar los elementos que se encuentran en el arreglo A utilizando el método.

A: 15 67 08 16 44 27 12 35

Se selecciona $A[1]$, por lo tanto, $X \leftarrow 15$.
Se llevan a cabo las comparaciones que se muestran a continuación:

PRIMERA PASADA

Recorrido de derecha a izquierda

$A[8] \geq X$ $(35 \geq 15)$ no hay intercambio
$A[7] \geq X$ $(12 \geq 15)$ sí hay intercambio

A: 12 67 08 16 44 27 15 35
↑ ↑

Recorrido de izquierda a derecha

$A[2] \leq X$ $(67 \leq 15)$ sí hay intercambio

A: 12 15 08 16 44 27 67 35
↑ ↑

SEGUNDA PASADA

Recorrido de derecha a izquierda

$A[6] \geq X$ $(27 \geq 15)$ no hay intercambio
$A[5] \geq X$ $(44 \geq 15)$ no hay intercambio
$A[4] \geq X$ $(16 \geq 15)$ no hay intercambio
$A[3] \geq X$ $(08 \geq 15)$ sí hay intercambio

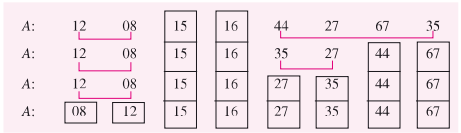
A: 12 08 15 16 44 27 67 35
↑ ↑

Como el recorrido de izquierda a derecha se debería iniciar en la misma posición donde se encuentra el elemento X, el proceso termina ya que se detecta que el elemento X se encuentra en la posición correcta. Observe que los elementos que forman parte del primer conjunto son menores o iguales a X, y los del segundo conjunto son mayores o iguales a X.

A: 12 08 15 16 44 27 67 35
1er conjunto 2o. conjunto

Este proceso de particionamiento aplicado para localizar la posición correcta de un elemento X en el arreglo se repite cada vez que queden conjuntos formados por dos o más elementos. El método se puede aplicar de manera iterativa o recursiva. En la tabla 8.10 se presenta la ubicación del resto de los elementos en el arreglo.

TABLA 8.10



El algoritmo de ordenación por el método *quicksort* en su versión recursiva es:

Algoritmo 8.9 Rápido_recursivo

Rápido_recursivo (A, N)

{Este algoritmo ordena los elementos del arreglo unidimensional utilizando el método rápido, de manera recursiva. A es un arreglo unidimensional de N elementos}

1. Llamar al algoritmo Reduce_recursivo con 1 y N

Observe que el algoritmo *Rápido_recursivo* requiere para su funcionamiento de otro algoritmo, el cual se presenta en la tabla 8.10.

Algoritmo 8.10 Reduce_recursivo

Reduce_recursivo (INI, FIN)

{INI y FIN representan las posiciones del extremo izquierdo y derecho, respectivamente, del conjunto de elementos a ordenar}
{IZQ, DER, POS y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

1. Hacer IZQ ← INI, DER ← FIN, POS ← INI y BAND ← VERDADERO
2. Mientras (BAND = VERDADERO) *Repetir*
   Hacer BAND ← FALSO
   2.1 Mientras ((A[POS] ≤ A[DER]) y (POS ≠ DER)) *Repetir*
   Hacer DER ← DER – 1
   2.2 {Fin del ciclo del paso 2.1}
   2.3 *Si* (POS ≠ DER) *entonces*
   Hacer AUX ← A[POS], A[POS] ← A[DER], A[DER] ← AUX y POS ← DER

```
      2.3.1 Mientras ((A[POS] ≥ A[IZQ]) y (POS ≠ IZQ)) Repetir
                Hacer IZQ ← IZQ + 1
      2.3.2 {Fin del ciclo del paso 2.3.1}
      2.3.3 Si (POS ≠ IZQ) entonces
                Hacer BAND ← VERDADERO, AUX ← A[POS], A[POS] ← A[IZQ],
                  A[IZQ] ← AUX y POS ← IZQ
      2.3.4 {Fin del condicional del paso 2.3.3}
   2.4 {Fin del condicional del paso 2.3}
3. {Fin del ciclo del paso 2}
4. Si ((POS – 1) > INI) entonces
            Regresar a Reduce_recursivo con INI y (POS – 1) {Llamada recursiva}
5. {Fin del condicional del paso 4}
6. Si (FIN > (POS + 1)) entonces
            Regresar a Reduce_recursivo con (POS + 1) y FIN {Llamada recursiva}
7. {Fin del condicional del paso 6}
```

Aun cuando el algoritmo del *quicksort* presentado resulte claro, es posible aumentar su velocidad de ejecución eliminando las llamadas recursivas. La recursividad es un instrumento muy poderoso, pero la eficiencia de ejecución es un factor muy importante en un proceso de ordenación que es necesario cuidar y administrar muy bien. Estas llamadas recursivas se pueden sustituir utilizando pilas, dando lugar entonces a la iteratividad.

Consideremos el arreglo unidimensional *A* del ejemplo 8.11. Luego de la primera partición, *A* queda de la siguiente manera:

A: 12 08 **15** 16 44 27 67 35

1er. conjunto — 2o. conjunto

Al utilizar la iteratividad, se deben almacenar en las pilas los índices de los dos conjuntos de datos que falta tratar. Se utilizarán dos pilas, PILAMENOR y PILAMAYOR. En la primera se almacenará el extremo izquierdo y en la otra se almacenará el extremo derecho de los conjuntos de datos que falta tratar. En la figura 8.3 se observa el estado de las pilas, luego de cargar los extremos de los conjuntos que falta tratar.

Los índices del primer conjunto quedaron almacenados en la primera posición de PILAMENOR y PILAMAYOR, respectivamente. La posición del extremo izquierdo del primer conjunto (1) en PILAMENOR y la del extremo derecho del mismo conjunto (2) en PILAMAYOR. Las posiciones de los extremos izquierdo y derecho del segundo conjunto (4 y 8) fueron almacenados en la cima de PILAMENOR y PILAMAYOR, respectivamente.

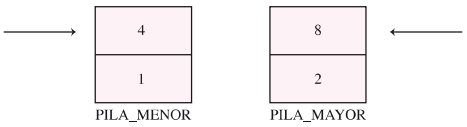


FIGURA 8.3
Pilas para sustituir la recursividad.

A continuación se presenta el algoritmo de ordenación por el método del *quicksort* utilizando iteratividad en lugar de recursividad.

Algoritmo 8.11 Rápido_iterativo

```
Rápido_iterativo (A, N)

{Este algoritmo ordena los elementos de un arreglo unidimensional utilizando el método
rápido, de manera iterativa. A es un arreglo unidimensional de N elementos}
{TOPE, INI, FIN y POS son variables de tipo entero. PILAMENOR y PILAMAYOR son
arreglos unidimensionales, que funcionan como pilas}

1. Hacer TOPE ← 1, PILAMENOR[TOPE] ← 1 y PILAMAYOR[TOPE] ← N
2. Mientras (TOPE > 0) Repetir
        Hacer INI ← PILAMENOR[TOPE], FIN ← PILAMAYOR[TOPE] y
          TOPE ← TOPE – 1
        Llamar al algoritmo Reduce_iterativo con INI, FIN y POS.
   2.1 Si (INI < (POS – 1)) entonces
            Hacer TOPE ← TOPE + 1, PILAMENOR[TOPE] ← INI y
              PILAMAYOR[TOPE] ← POS – 1
   2.2 {Fin del condicional del paso 2.1}
   2.3 Si (FIN > (POS + 1)) entonces
            Hacer TOPE ← TOPE + 1, PILAMENOR[TOPE] ← POS + 1 y
              PILAMAYOR[TOPE] ← FIN
   2.4 {Fin del condicional del paso 2.3}
3. {Fin del ciclo del paso 2}
```

Note que el algoritmo **Rápido_iterativo** necesita para su funcionamiento de otro algoritmo, el cual se presenta a continuación.

Algoritmo 8.12 Reduce_iterativo

```
Reduce_iterativo (INI, FIN, POS)

{INI y FIN representan las posiciones de los extremos izquierdo y derecho, respectivamente,
del conjunto de elementos a evaluar. POS es una variable donde se almacenará el resultado de
este algoritmo}
{IZQ, DER y AUX son variables de tipo entero. BAND es una variable de tipo boolcano}

1. Hacer IZQ ← INI, DER ← FIN, POS ← INI y BAND ← VERDADERO
2. Mientras (BAND = VERDADERO) Repetir
   2.1 Mientras ((A[POS] ≤ A[DER]) y (POS ≠ DER)) Repetir
            Hacer DER ← DER – 1
   2.2 {Fin del ciclo del paso 2.1}
   2.3 Si (POS = DER)
        entonces
```

```
            Hacer BAND ← FALSO
        si no
            Hacer AUX ← A[POS], A[POS] ← A[DER], A[DER] ← AUX y
              POS ← DER
    2.3.1 Mientras ((A[POS] ≥ A[IZQ]) y (POS ≠ IZQ)) Repetir
                  Hacer IZQ ← IZQ + 1
    2.3.2 {Fin del ciclo del paso 2.3.1}
    2.3.3 Si (POS = IZQ)
              entonces
                  Hacer BAND ← FALSO
              si no
                  Hacer AUX ← A[POS], A[POS] ← A[IZQ],
                  A[IZQ] ← AUX y POS ← IZQ
    2.3.4 {Fin del condicional del paso 2.3.3}
  2.4 {Fin del condicional del paso 2.3}
3. {Fin del ciclo del paso 2}
```

Ejemplo 8.12

En la tabla 8.11 se exponen los pasos necesarios para ordenar las claves del arreglo unidimensional *A*.

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

La columna EXTREMO contiene los extremos *izquierdo* y *derecho* del conjunto de elementos a evaluar. En PILAMENOR se almacena el extremo izquierdo y en PILAMAYOR el extremo derecho de los conjuntos pendientes de tratar.

Una leve mejora en el funcionamiento del método rápido se puede producir si el primer elemento posicionado en el arreglo se encuentra en la mitad o muy próximo a su mitad. Para lograr esto último se necesita permutar los componentes del arreglo de tal forma que para el valor *X* todos los elementos que se encuentren a su izquierda desde *A*[1] hasta *A*[*i*], donde *i* es igual a ((*n*/2) – 1), sean menores o iguales a él y todos los que se encuentren a su derecha desde *A*[*i* + 1] hasta *A*[*N*] sean mayores o iguales a él. Por ejemplo, en el siguiente arreglo unidimensional *A*:

A: 45 21 76 08 17 96 55 36 43

43 es el elemento que ocupa la posición central del arreglo ordenado y divide a éste en dos mitades iguales. Queda la tarea de construir el algoritmo que encuentra al elemento *X* que ocupa la posición central del arreglo.

Análisis de eficiencia del método *quicksort*

El método *quicksort* es el más rápido de ordenación interna que existe en la actualidad. Esto es sorprendente, porque el método tiene su origen en el método de intercambio directo, el peor de todos los métodos directos. Diversos estudios realizados sobre su comportamiento demuestran que si se escoge en cada pasada el elemento que ocupa la

TABLA 8.11

POSICIONES																EXTREMO	PILA MENOR	PILA MAYOR
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16			
15	67	08	16	44	27	12	35	56	21	13	28	60	36	07	10	(01, 16)	NIL	NIL
10	07	08	13	12	15	27	35	56	21	44	28	60	36	16	67	(07, 16)	01	05
																	07	08
10	07	08	13	12	15	16	21	27	56	44	28	60	36	35	67	(10, 16)	01	05
																	10	13
10	07	08	13	12	15	16	21	27	35	44	28	36	56	60	67	(15, 16)	07	08
																	01	05
10	07	08	13	12	15	16	21	27	35	44	28	36	56	60	67	(10, 13)	07	08
																	01	05
10	07	08	13	12	15	16	21	27	28	35	44	36	56	60	67	(12, 13)	07	08
																	01	05
10	07	08	13	12	15	16	21	27	28	35	36	44	56	60	67	(07, 08)	01	05
10	07	08	13	12	15	16	21	27	28	35	36	44	56	60	67	(01, 05)	NIL	NIL
08	07	10	13	12	15	16	21	27	28	35	36	44	56	60	67	(04, 05)	01	02
08	07	10	12	13	15	16	21	27	28	35	36	44	56	60	67	(01, 02)	NIL	NIL
07	08	10	12	13	15	16	21	27	28	35	36	44	56	60	67	NIL	NIL	NIL

posición central del conjunto de datos a analizar, el número de pasadas necesarias para ordenarlo es del orden de log *n*. Respecto del número de comparaciones, si el tamaño del arreglo es una potencia de 2, en la primera pasada realizará $(n-1)$ comparaciones, en la segunda $(n-1)/2$ comparaciones, pero en dos conjuntos diferentes, en la tercera realizará $(n-1)/4$ comparaciones, pero en cuatro conjuntos diferentes y así sucesivamente. Por lo tanto:

$$C = (n-1) + 2*\frac{(n-1)}{2} + 4*\frac{(n-1)}{4} + \ldots + (n-1)*\frac{(n-1)}{(n-1)}$$

lo cual es lo mismo que:

$$C = (n-1) + (n-1) + (n-1) + \ldots + (n-1)$$

Si se considera a cada uno de los componentes de la sumatoria como un término y el número de términos de la sumatoria es igual a m, entonces se tiene que:

$$C = (n-1) * m$$

Considerando que el número de términos de la sumatoria (m) es igual al número de pasadas, y que éste es igual a log n, la expresión anterior queda:

$$C = (n-1) * \log n \qquad \textbf{Fórmula 8.12}$$

Sin embargo, encontrar el elemento que ocupe la posición central del conjunto de datos que se van a analizar es una tarea difícil, ya que existen $1/n$ posibilidades de lograrlo. Además, el rendimiento medio del método es aproximadamente (2 * ln 2) inferior al caso óptimo, por lo que Hoare, el autor del método, propone como solución que el elemento X se seleccione arbitrariamente o bien entre una muestra relativamente pequeña de elementos del arreglo.

El peor caso ocurre cuando los elementos del arreglo ya se encuentran ordenados, o bien cuando se encuentran en orden inverso. Supongamos, por ejemplo, que se debe ordenar el siguiente arreglo unidimensional que ya se encuentra ordenado:

A: 08 12 15 16 27 35 44 67

Si se escoge arbitrariamente el primer elemento (08), entonces se particionará el arreglo en dos mitades, una de 0 y otra de $(n-1)$ elementos.

08 | 12 15 16 27 35 44 67

Si se continúa con el proceso de ordenación y se escoge nuevamente el primer elemento (12) del conjunto de datos que se analizarán, entonces se dividirá el arreglo en dos nuevos conjuntos, nuevamente uno de 0 y otro de $(n-2)$ elementos. Por lo tanto, el número de comparaciones que se realizarán será:

$$C_{máx} = n + (n-1) + (n-2) + \ldots + 2 = \frac{n*(n+1)}{2} - 1$$

que es igual a:

$$C_{máx} = \frac{n^2+n}{2} - 1 \qquad \textbf{Fórmula 8.13}$$

Como conclusión, se puede afirmar que el tiempo promedio de ejecución del algoritmo es proporcional a (n * log n), $O(n * \log n)$. En el peor caso, el tiempo de ejecución es proporcional a n^2, $O(n^2)$.

8.2.10 Ordenación por el método *heapsort* (montículo)

El método de ordenación ***heapsort*** se conoce también como **montículo**. Su nombre se debe a su autor, J. W. Williams, quien lo llamó así. Es el más eficiente de los métodos de ordenación que trabajan con árboles. La idea central de este algoritmo se basa en dos operaciones:

1. Construir un montículo.
2. Eliminar la raíz del montículo en forma repetida.

Es importante señalar que un montículo se define como: *Para todo nodo del árbol se debe cumplir que su valor sea mayor o igual que el valor de cualquiera de sus hijos.*

Ejemplo 8.13

En la figura 8.4 se muestra un montículo. Allí se puede observar que para cada nodo K del árbol, su valor es mayor o igual que el valor de cualquiera de sus hijos.

Ahora bien, para representar un montículo en un arreglo lineal se debe tener en cuenta para todo nodo K lo siguiente:

1. El nodo K se almacena en la posición K correspondiente del arreglo.
2. El hijo izquierdo del nodo K se almacena en la posición 2 * K.
3. El hijo derecho del nodo K se almacena en la posición 2 * K + 1.

La figura 8.5 contiene la representación del montículo de la figura 8.4 en un arreglo unidimensional A.

Observe que si se desea obtener el hijo izquierdo del nodo $A[4]$, cuyo valor es 28, se hace $A[4 * 2] = A[8]$ y su contenido es 27. Si deseamos obtener en cambio el hijo derecho de $A[4]$, hacemos $A[4 * 2 + 1] = A[9]$ y su contenido es 16.

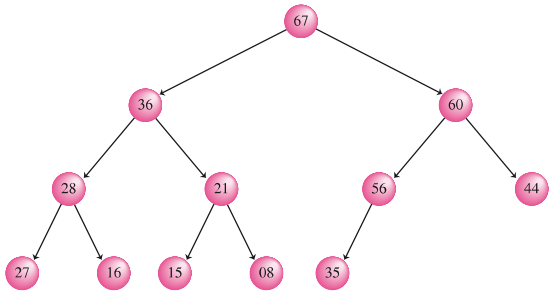


FIGURA 8.4
Montículo con 12 elementos.

FIGURA 8.5
Representación de un montículo en un arreglo lineal.

	1	2	3	4	5	6	7	8	9	10	11	12
A	67	36	60	28	21	56	44	27	16	15	08	35

A su vez, es posible calcular el padre de un nodo no raíz K cualquiera, tomando la parte entera de (K entre 2). Así, por ejemplo, si se desea obtener el padre del nodo $A[11]$, cuyo valor es 8, se hace A[parte entera(11 entre 2)] = $A[5]$ y su contenido es 21.

Inserción de un elemento en un montículo

La inserción de un elemento en un montículo se lleva a cabo por medio de los siguientes pasos:

1. Se inserta el elemento en la primera posición disponible.
2. Se verifica si su valor es mayor que el de su padre. Si se cumple esta condición, entonces se efectúa el intercambio. Si no se cumple esta condición, entonces el algoritmo se detiene y el elemento queda ubicado en su posición correcta en el montículo.

Cabe aclarar que el paso 2 se aplica de manera recursiva y desde abajo hacia arriba.

Ejemplo 8.14

Supongamos que se quiere incorporar al montículo de la figura 8.6 el elemento 63. Las comparaciones que realizamos son:

63 > 56 sí hay intercambio
63 > 60 sí hay intercambio
63 > 67 no hay intercambio

Luego de haber insertado el elemento 63, el montículo queda como se muestra en la figura 8.7.

FIGURA 8.6
Montículo con 12 elementos.

Nota: Se utiliza flecha discontinua para indicar la posición inicial donde se inserta el elemento 63.

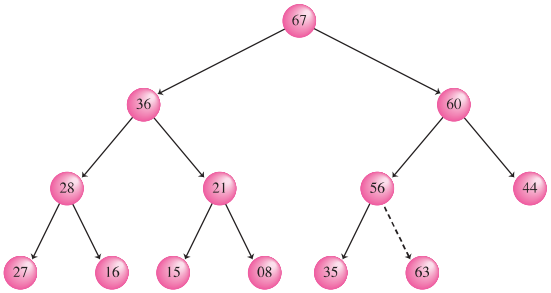
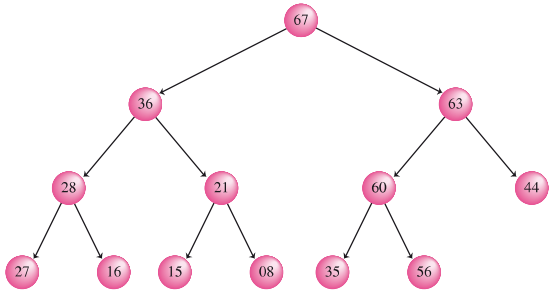


FIGURA 8.7
Montículo luego de haber insertado la clave 63.



Ejemplo 8.15

Supongamos que se desea insertar las siguientes claves en un montículo que se encuentra vacío.

15 60 08 16 44 27 12 35

Los resultados parciales que ilustran cómo funciona el procedimiento se observan en la figura 8.8. El montículo se representa como árbol y también como arreglo. Las flechas discontinuas indican la posición inicial donde se inserta el o los elementos.

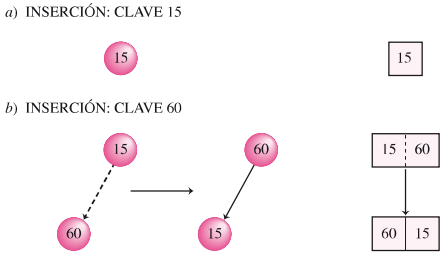
Ejemplo 8.16

Dado el montículo de la figura 8.8*f*, verifique si éste queda igual al montículo, representado como arreglo, de la figura 8.9, luego de insertar las siguientes claves:

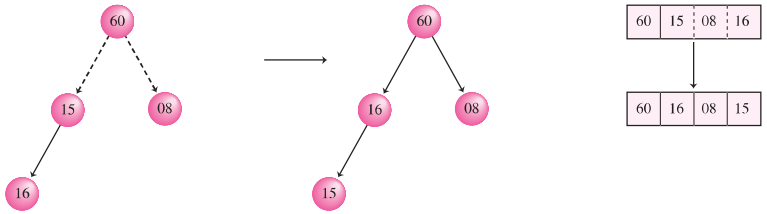
56 21 13 28 67 36 07 10

FIGURA 8.8
Inserciones en un montículo.

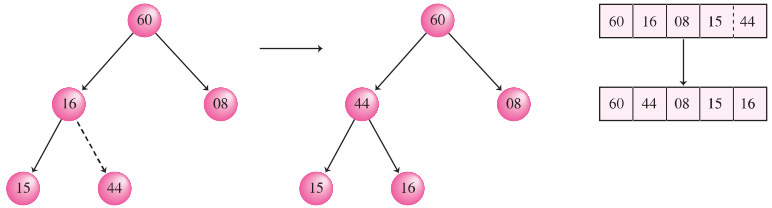
Nota: Se utiliza la flecha discontinua para indicar la posición inicial donde se inserta el o los elementos.



c) INSERCIÓN: CLAVES 08 y 16



d) INSERCIÓN: CLAVE 44



e) INSERCIÓN: CLAVE 27

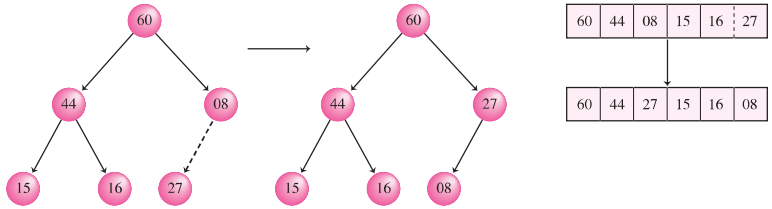


FIGURA 8.8
(*continuación*)
Inserciones en un montículo.

Nota: Se utiliza la flecha discontinua para indicar la posición inicial donde se inserta el o los elementos.

f) INSERCIÓN: CLAVES 12 y 35

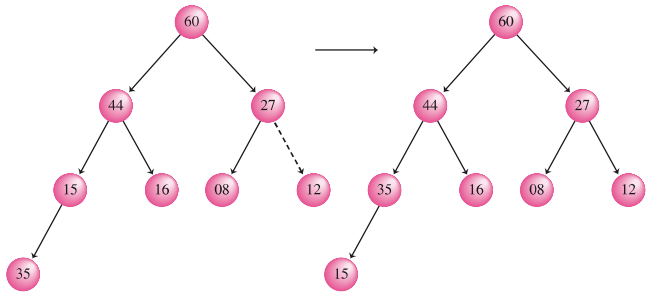


FIGURA 8.8
(*continuación*)
Inserciones en un montículo.

Nota: Se utiliza la flecha discontinua para indicar la posición inicial donde se inserta el o los elementos.

El algoritmo para insertar elementos en un montículo es:

Algoritmo 8.13 Inserta_montículo

Inserta_montículo (*A*, *N*)

{El algoritmo inserta los elementos en un montículo representado como arreglo. *A* es un arreglo unidimensional de *N* elementos}
{*I*, *K* y AUX son variables de tipo entero. BAND es una variable de tipo booleano}

```
1.  Repetir con I desde 2 hasta N
        Hacer K ← I y BAND ← VERDADERO
    1.1 Mientras ((K > 1) y (BAND = VERDADERO)) Repetir
            Hacer BAND ← FALSO
        1.1.1 Si (A[K] > A[parte entera(K entre 2)]) entonces
                  Hacer AUX ← A[parte entera(K entre 2)],
                  A[parte entera(K entre 2)] ← A[K], A[K] ← AUX,
                  K ← parte entera(K entre 2) y BAND ← VERDADERO
        1.1.2 {Fin del condicional del paso 1.1.1}
    1.2 {Fin del ciclo del paso 1.1}
2.  {Fin del ciclo del paso 1}
```

FIGURA 8.9
Montículo representado como arreglo unidimensional.

67	56	60	44	21	28	36	15	35	16	13	08	27	12	17	10
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Eliminación de un montículo

El proceso para obtener los elementos ordenados se efectúa eliminando la raíz del montículo en forma repetida. Ahora bien, los pasos necesarios para lograr la eliminación de la raíz del montículo son:

1. Se reemplaza la raíz con el elemento que ocupa la última posición del montículo.
2. Se verifica si el valor de la raíz es menor que el valor más grande de sus hijos. Si se cumple la condición, entonces se efectúa el intercambio. Si no se cumple la condición, entonces el algoritmo se detiene y el elemento queda ubicado en su posición correcta en el montículo.

Cabe aclarar que el paso 2 se aplica de manera recursiva y desde arriba hacia abajo.

Ejemplo 8.17

Supongamos que se desea eliminar la raíz del montículo (67) de la figura 8.10.

Reemplazamos la raíz, 67, por el elemento que ocupa la última posición del montículo, 35. A continuación efectuamos las siguientes comparaciones:

35 < 60 sí hay intercambio
60 es el mayor de los hijos de 35

35 < 56 sí hay intercambio
56 es el mayor de los hijos de 35

El montículo, luego de haberse eliminado la raíz, queda como el que se presenta en la figura 8.11.

FIGURA 8.10
Montículo representado como árbol.

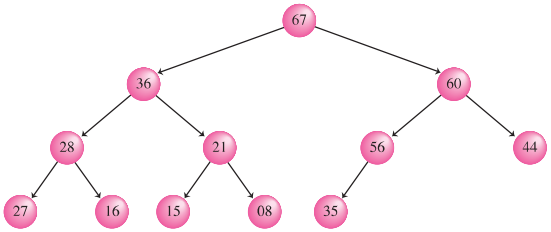
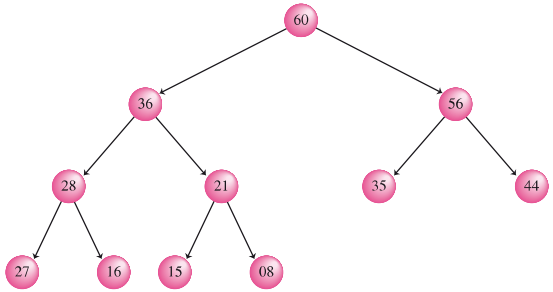


FIGURA 8.11
Montículo representado como árbol.



Ejemplo 8.18

Supongamos que se desea eliminar la raíz del montículo, presentado como arreglo, de la figura 8.12, en forma repetida.

Cabe aclarar que al reemplazar la raíz por el último elemento del montículo, ésta se coloca en la posición del último elemento. Es decir, la primera vez la raíz será colocada en la posición n, la segunda vez en la posición $(n-1)$, la tercera vez en la posición $(n-2)$ y así sucesivamente hasta que quede colocada en las posiciones 2 y 1, en forma respectiva. Los pasos a realizar son:

PRIMERA ELIMINACIÓN DE LA RAÍZ

Se intercambia la raíz, 67, con el elemento que ocupa la última posición del montículo, 10. Las comparaciones que se realizan son:

$A[1] < A[3]$ (10 < 60) sí hay intercambio
$A[3]$ es el mayor de los hijos de $A[1]$

$A[3] < A[7]$ (10 < 36) sí hay intercambio
$A[7]$ es el mayor de los hijos de $A[3]$

$A[7] < A[14]$ (10 < 12) sí hay intercambio
$A[14]$ es el mayor de los hijos de $A[7]$

Luego de eliminar la primera raíz, el montículo queda así:

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

FIGURA 8.12
Montículo representado como arreglo unidimensional.

67	56	60	44	21	28	36	15	35	16	13	08	27	12	07	10
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Observe que el elemento más grande se ubicó en la última posición del arreglo.

SEGUNDA ELIMINACIÓN DE LA RAÍZ

Se intercambia la raíz, 60, con el elemento que ocupa la última posición del montículo, 07. Las comparaciones que se realizan son:

$A[1] < A[2]$ (07 < 56) sí hay intercambio
$A[2]$ es el mayor de los hijos de $A[1]$

$A[2] < A[4]$ (07 < 44) sí hay intercambio
$A[4]$ es el mayor de los hijos de $A[2]$

$A[4] < A[9]$ (07 < 35) sí hay intercambio
$A[9]$ es el mayor de los hijos de $A[4]$

Luego de eliminar la segunda raíz, el montículo queda así:

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

TERCERA ELIMINACIÓN DE LA RAÍZ

Se intercambia la raíz, 56, con el elemento que ocupa la última posición del montículo, 10. Las comparaciones que se realizan son:

$A[1] < A[2]$ (10 < 44) sí hay intercambio
$A[2]$ es el mayor de los hijos de $A[1]$

$A[2] < A[4]$ (10 < 35) sí hay intercambio
$A[4]$ es el mayor de los hijos de $A[2]$

$A[4] < A[8]$ (10 < 15) sí hay intercambio
$A[8]$ es el mayor de los hijos de $A[4]$

Luego de eliminar la tercera raíz, el montículo queda así:

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

En la tabla 8.12 se presenta el resultado de las restantes eliminaciones. Observe que luego de eliminar la raíz del montículo, en forma repetida, el arreglo queda ordenado.

A continuación se presenta el algoritmo que elimina sucesivamente la raíz del montículo. Cabe aclarar que para efecto de aumentar la eficiencia del algoritmo se eliminan los intercambios parciales utilizando una variable auxiliar, en la que se almacena el último elemento del montículo.

TABLA 8.12

Eliminación	Montículo															
4	36	35	28	15	21	27	12	10	07	16	13	08	[44]	[56]	[60]	[67]
5	35	21	28	15	16	27	12	10	07	08	13	[36]	[44]	[56]	[60]	[67]
6	28	21	27	15	16	13	12	10	07	08	[35]	[36]	[44]	[56]	[60]	[67]
7	27	21	13	15	16	08	12	10	07	[28]	[35]	[36]	[44]	[56]	[60]	[67]
8	21	16	13	15	07	08	12	10	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
9	16	15	13	10	07	08	12	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
10	15	12	13	10	07	08	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
11	13	12	08	10	07	[15]	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
12	12	10	08	07	[13]	[15]	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
13	10	07	08	[12]	[13]	[15]	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
14	08	07	[10]	[12]	[13]	[15]	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]
15	[07]	[08]	[10]	[12]	[13]	[15]	[16]	[21]	[27]	[28]	[35]	[36]	[44]	[56]	[60]	[67]

Algoritmo 8.14 Elimina_montículo

Elimina_montículo (*A*, *N*)

{El algoritmo elimina la raíz del montículo en forma repetida. *A* es un arreglo unidimensional de *N* elementos}
{*I*, AUX, IZQ, DER, *K* y AP son variables de tipo entero. BOOL es una variable de tipo booleano}

1. *Repetir* con *I* desde *N* hasta 2 {Ciclo descendente}
Hacer AUX ← *A*[*I*], *A*[*I*] ← *A*[1], IZQ ← 2, DER ← 3, *K* ← 1 y
BOOL ← VERDADERO
1.1 Mientras ((IZQ < *I*) y (BOOL = VERDADERO)) *Repetir*
Hacer MAYOR ← *A*[IZQ] y AP ← IZQ
1.1.1 *Si* ((MAYOR < *A*[DER]) y (DER ≠ *I*)) *entonces*
Hacer MAYOR ← *A*[DER] y AP ← DER
1.1.2 {Fin del condicional del paso 1.1.1}
1.1.3 *Si* (AUX < MAYOR)
entonces
Hacer *A*[*K*] ← *A*[AP] y *K* ← AP
si no
Hacer BOOL ← FALSO
1.1.2 {Fin del condicional del paso 1.1.3}
Hacer IZQ ← *K* * 2 y DER ← IZQ + 1
1.2 {Fin del ciclo del paso 1.1}
Hacer *A*[*K*] ← AUX
2. {Fin del ciclo del paso 1}

El proceso de ordenación por el método del montículo consta de dos partes:

1. Construir el montículo. Esta operación se basa en la de inserción presentada en el algoritmo 8.13.
2. Eliminar repetidamente la raíz del montículo. Esta operación se basa en la de eliminación presentada en el algoritmo 8.14.

El algoritmo de ordenación, resulta entonces de la siguiente manera:

Algoritmo 8.15 Montículo

Montículo (*A*, *N*)

{El algoritmo ordena los elementos del arreglo utilizando el método del montículo. *A* es un arreglo unidimensional de *N* elementos}

1. Llamar al algoritmo Inserta_montículo con *A* y *N*.
2. Llamar al algoritmo Elimina_montículo con *A* y *N*.

Análisis de eficiencia del método del montículo

El análisis del método del montículo, como el de los métodos logarítmicos, es complejo. Es importante tener en cuenta tanto la fase de construcción del montículo como la fase donde se elimina repetidamente su raíz, para finalmente obtener el arreglo ordenado.

A diferencia de lo que se pudiera pensar, ya que en la fase de construcción del montículo los elementos mayores se cargan hacia la izquierda y en la fase de eliminación de la raíz los elementos mayores se cargan hacia la derecha, éste es un método muy rápido, sobre todo para valores grandes de *N*. Los estudios realizados al respecto demuestran que el tiempo de ejecución del algoritmo en ambas fases es de $O(n * \log n)$.

Aunque el método del montículo puede ser un poco más lento que el *quicksort* (se estima en 70%), es el único que garantiza que aun en el peor caso su tiempo de ejecución es proporcional a $(n * \log n)$, $O(n * \log n)$. Recuerde que el tiempo de ejecución del método *quicksort*, en el peor caso, es proporcional a n^2, $O(n^2)$.

8.3 ORDENACIÓN EXTERNA

En la actualidad es muy común procesar tales volúmenes de información que los datos no se pueden almacenar en la memoria principal de la computadora. Estos datos, organizados en archivos, se guardan en dispositivos de almacenamiento secundario, como cintas, discos, etcétera.

El proceso de ordenar los datos almacenados en varios archivos se conoce como **fusión** o **mezcla**; se entiende por este concepto a la combinación o intercalación de dos o más secuencias ordenadas en una única secuencia ordenada. Se debe hacer hincapié en

que sólo se colocan en la memoria principal de la computadora los datos que se pueden acceder en forma directa.

8.3.1 Intercalación de archivos

Por **intercalación de archivos** se entiende la unión o fusión de dos o más archivos, ordenados de acuerdo con un determinado campo clave, en un solo archivo. Analicemos el siguiente ejemplo.

Ejemplo 8.19

Suponga que se tienen dos archivos, *F*1 y *F*2, cuya información está ordenada de acuerdo con un campo clave:

***F*1:** 06 09 18 20 35
***F*2:** 10 16 25 28 66 82 87

Se debe producir, entonces, un archivo *F*3 ordenado, como resultado de la mezcla de *F*1 y *F*2. Sólo se pueden acceder en forma directa dos claves, la primera del archivo *F*1 y la segunda del archivo *F*2. Las comparaciones que se realizan para producir el archivo *F*3 son:

(06 < 10) sí se cumple la condición
Se escribe 06 en el archivo de salida *F*3 y se vuelve a leer otra clave de *F*1(09)

(09 < 10) sí se cumple la condición
Se escribe 09 en el archivo de salida *F*3 y se vuelve a leer otra clave de *F*1(18)

(18 < 10) no se cumple la condición
Se escribe 10 en el archivo de salida *F*3 y se vuelve a leer otra clave de *F*2(16)

El estado de los archivos *F*1, *F*2 y *F*3, hasta el momento, es como se muestra más abajo. La flecha señala el último elemento leído de los archivos que se están intercalando.

***F*1:** 06 09 ↓18 20 35
***F*2:** 10 ↓16 25 28 66 82 87

F3: 06 09 10

El proceso continúa hasta que en uno u otro archivo se detecte su final; en tal caso sólo se tendrá que copiar la información del archivo no vacío al archivo de salida *F*3. El resultado final de la intercalación entre los archivos *F*1 y *F*2 es:

F3: 06 09 10 16 18 20 25 28 35 66 82 87

A continuación se muestra el algoritmo de intercalación de archivos.

Algoritmo 8.16 Intercalación

Intercalación (*F1*, *F2*, *F3*)

{El algoritmo intercala los elementos de dos archivos ya ordenados *F1* y *F2* y almacena el resultado en el archivo *F3*}
{R1 y R2 son variables de tipo entero. BAN1 y BAN2 son variables de tipo booleano}

```
1. Abrir los archivos F1 y F2 para lectura.
2. Abrir el archivo F3 para escritura.
3. Hacer BAN1 ← VERDADERO y BAN2 ← VERDADERO
4. Mientras (((no sea el fin de archivo de F1) o (BAN1 = FALSO)) y ((no sea el fin de
   archivo de F2) o (BAN2 = FALSO))) Repetir
   4.1 Si (BAN1 = VERDADERO) entonces {Se debe leer R1 de F1}
           Leer R1 de F1
           Hacer BAN1 ← FALSO
   4.2 {Fin del condicional del paso 4.1}
   4.3 Si (BAN2 = VERDADERO) entonces {Se debe leer R2 de F2}
           Leer R2 de F2
           Hacer BAN2 ← FALSO
   4.4 {Fin del condicional del paso 4.3}
   4.5 Si (R1 < R2)
       entonces
           Escribir R1 en F3
           Hacer BAN1 ← VERDADERO
       si no
           Escribir R2 en F3
           Hacer BAN2 ← VERDADERO
   4.6 {Fin del condicional del paso 4.5}
5. {Fin del ciclo del paso 4}
     {Verifica si se leyó un elemento de F1 y no se copió en F3}
6. Si (BAN1 = FALSO) entonces
     Escribir R1 en F3
   6.1 Mientras (no sea el fin de archivo de F1) Repetir
         Leer R1 de F1
         Escribir R1 en F3
   6.2 {Fin del ciclo del paso 6.1}
7. {Fin del condicional del paso 6}
     {Verifica si se leyó un elemento de F2 y no se copió en F3}
8. Si (BAN2 = FALSO) entonces
     Escribir R2 en F3
   8.1 Mientras (no sea el fin de archivo de F2) Repetir
         Leer R2 de F2
         Escribir R2 en F3
   8.2 {Fin del ciclo del paso 8.1}
9. {Fin del condicional del paso 8}
10. {Cerrar los archivos F1, F2 y F3}
```


8.3.2 Ordenación de archivos

La ordenación de archivos se efectúa cuando el volumen de los datos es demasiado grande y éstos no caben en la memoria principal de la computadora. Al ocurrir esta situación no se pueden aplicar los métodos de ordenación interna estudiados en la primera parte de este capítulo, de modo que se debe pensar en otro tipo de algoritmos para ordenar datos almacenados en archivos.

Por ordenación de archivos se entiende, entonces, la ordenación o clasificación de éstos, ascendente o descendentemente, de acuerdo con un campo determinado al que se denominará campo clave. La principal desventaja de esta ordenación es el tiempo de ejecución, debido a las sucesivas operaciones de lectura y escritura a/de archivo.

Los dos métodos de ordenación externa más importantes son los basados en la mezcla directa y en la mezcla equilibrada.

8.3.3 Ordenación por mezcla directa

El método de ordenación por **mezcla directa** es probablemente el más utilizado por su fácil comprensión. La idea central de este algoritmo consiste en la realización sucesiva de una partición y una fusión que produce secuencias ordenadas de longitud cada vez mayor. En la primera pasada, la partición es de longitud 1 y la fusión o mezcla produce secuencias ordenadas de longitud 2. En la segunda pasada, la partición es de longitud 2 y la fusión o mezcla produce secuencias ordenadas de longitud 4. Este proceso se repite hasta que la longitud de la secuencia para la partición sea:

$$\text{Parte entera } ((n+1)/2)$$

Donde n representa el número de elementos del archivo original.

Ejemplo 8.20

Supongamos que se desea ordenar las claves del archivo *F*. Para realizar tal actividad se utilizan dos archivos auxiliares a los que se les denominará *F1* y *F2*.

F: 09 75 14 68 29 17 31 25 04 05 13 18 72 46 61

PRIMERA PASADA

Partición en secuencias de longitud 1.

F1: 09′ 14′ 29′ 31′ 04′ 13′ 72′ 61′
F2: 75′ 68′ 17′ 25′ 05′ 18′ 46′

Fusión en secuencias de longitud 2.

F: 09 75′ 14 68′ 17 29′ 25 31′ 04 05′ 13 18′ 46 72′ 61′

SEGUNDA PASADA

Partición en secuencias de longitud 2.

F1: 09 75′ 17 29′ 04 05′ 46 72′
F2: 14 68′ 25 31′ 13 18′ 61′

Fusión en secuencias de longitud 4.

F: 09 14 68 75′ 17 25 29 31′ 04 05 13 18′ 46 61 72′

TERCERA PASADA

Partición en secuencias de longitud 4.

F1: 09 14 68 75′ 04 05 13 18′
F2: 17 25 29 31′ 46 61 72′

Fusión en secuencias de longitud 8.

F: 09 14 17 25 29 31 68 75′ 04 05 13 18 46 61 72′

CUARTA PASADA

Partición en secuencias de longitud 8.

F1: 09 14 17 25 29 31 68 75′
F2: 04 05 13 18 46 61 72′

Fusión en secuencias de longitud 16.

F: 04 05 09 13 14 17 18 25 29 31 46 61 68 72 75

A continuación se presenta el algoritmo de ordenación de archivos por el método de mezcla directa.

Algoritmo 8.17 Mezcla_directa

Mezcla_directa (*F*, *F*1, *F*2, *N*)

{El algoritmo ordena los elementos del archivo *F* por el método de mezcla directa. Utiliza dos archivos auxiliares *F*1 y *F*2. *N* es el número de elementos del archivo *F*}
{PART es una variable de tipo entero}

1. Hacer PART ← 1
2. Mientras (PART < parte entera ((*N* + 1) / 2)) *Repetir*
   Llamar al algoritmo Particiona con *F*, *F*1, *F*2 y PART
   Llamar al algoritmo Fusiona con *F*, *F*1, *F*2 y PART
   Hacer PART ← PART * 2
3. {Fin del ciclo del paso 2}

Observe que el algoritmo requiere para su funcionamiento de dos algoritmos auxiliares, los cuales se presentan a continuación.

Algoritmo 8.18 Particiona

Particiona (*F*, *F*1, *F*2, PART)

{El algoritmo genera dos archivos auxiliares, *F*1 y *F*2, a partir del archivo *F*. PART es la longitud de la partición que se va a realizar}
{*K*, *L* y *R* son variables de tipo entero}

1. Abrir el archivo *F* para lectura.
2. Abrir los archivos *F*1 y *F*2 para escritura.
3. Mientras (no sea el fin de archivo de *F*) *Repetir*
   Hacer *K* ← 0
   3.1 Mientras ((*K* < PART) y (no sea el fin de archivo de *F*)) *Repetir*
   Leer *R* de *F*
   Escribir *R* en *F*1
   Hacer *K* ← *K* + 1
   3.2 {Fin del ciclo del paso 3.1}
   Hacer *L* ← 0
   3.3 Mientras ((*L* < PART) y (no sea el fin de archivo de *F*)) *Repetir*
   Leer *R* de *F*
   Escribir *R* en *F*2
   Hacer *L* ← *L* + 1
   3.4 {Fin del ciclo del paso 3.3}
4. {Fin del ciclo del paso 3}

Algoritmo 8.19 Fusiona

Fusiona (*F*, *F*1, *F*2, PART)

{El algoritmo fusiona los archivos *F*1 y *F*2 en el archivo *F*. PART es la longitud de la partición que se realizó previamente}
{*R*1, *R*2, *K* y *L* son variables de tipo entero. *B*1 y *B*2 son variables de tipo booleano}

```
1.  Abrir el archivo F para escritura.
2.  Abrir los archivos F1 y F2 para lectura.
3.  Hacer B1 ← VERDADERO y B2 ← VERDADERO
4.  Si (no es el fin de archivo de F1) entonces
        Leer R1 de F1
        Hacer B1 ← FALSO
5.  {Fin del condicional del paso 4}
6.  Si (no es el fin de archivo de F2) entonces
        Leer R2 de F2
        Hacer B2 ← FALSO
7.  {Fin del condicional del paso 6}
8.  Mientras ((no sea el fin de archivo de F1) o (B1 = FALSO)) y ((no sea el
    fin de archivo de F2) o (B2 = FALSO)) Repetir
        Hacer K ← 0 y L ← 0
    8.1 Mientras ((K < PART) y (B1 = FALSO)) y ((L < PART) y ((B2 = FALSO))
        Repetir
        8.1.1 Si (R1 ≤ R2)
              entonces
                    Escribir R1 en F
                    Hacer B1 ← VERDADERO y K ← K + 1
              8.1.1.1 Si (no es el fin de archivo de F1) entonces
                        Leer R1 de F1
                        Hacer B1 ← FALSO
              8.1.1.2 {Fin del condicional del paso 8.1.1.1}
              si no
                    Escribir R2 en F
                    Hacer B2 ← VERDADERO y L ← L + 1
              8.1.1.3 Si (no es el fin de archivo de F2) entonces
                        Leer R2 de F2
                        Hacer B2 ← FALSO
              8.1.1.4 {Fin del condicional del paso 8.1.1.3}
        8.1.2 {Fin del condicional del paso 8.1.1}
    8.2 {Fin del ciclo del paso 8.1}
    8.3 Mientras ((K < PART) y (B1 = FALSO)) Repetir
              Escribir R1 en F
              Hacer B1 ← VERDADERO y K ← K + 1
        8.3.1 Si (no es el fin de archivo de F1) entonces
                Leer R1 de F1
                Hacer B1 ← FALSO
        8.3.2 {Fin del condicional del paso 8.3.1}
    8.4 {Fin del condicional del paso 8.3}
```



```
    8.5 Mientras ((L < PART) y (B2 = FALSO)) Repetir
              Escribir R2 en F
              Hacer B2 ← VERDADERO y L ← L + 1
        8.5.1 Si (no es el fin de archivo de F2) entonces
                Leer R2 de F2
                Hacer B2 ← FALSO
        8.5.2 {Fin del condicional del paso 8.5.1}
    8.6 {Fin del ciclo del paso 8.5}
9.  {Fin del ciclo del paso 8}
10. Si (B1 = FALSO) entonces
        Escribir R1 en F
11. {Fin del condicional del paso 10}
12. Si (B2 = FALSO) entonces
        Escribir R2 en F
13. {Fin del condicional del paso 12}
14. Mientras (no sea el fin de archivo de F1) Repetir
        Leer R1 de F1
        Escribir R1 en F
15. {Fin del condicional del paso 14}
16. Mientras (no sea el fin de archivo de F2) Repetir
        Leer R2 de F2
        Escribir R2 en F
17. {Fin del ciclo del paso 16}
18. {Cerrar los archivos F, F1 y F2}
```

Ejemplo 8.21

Supongamos que se desea ordenar las claves del archivo *F* utilizando el método de mezcla directo.

***F*:** 25 33 15 18 21 07 12 36 84 90 19 38 40 22
64 77 29 36 11

Los pasos que se realizan son:

PRIMERA PASADA

Partición en secuencias de longitud 1.

***F*1:** 25′ 15′ 21′ 12′ 84′ 19′ 40′ 64′ 29′ 11′
***F*2:** 33′ 18′ 07′ 36′ 90′ 38′ 22′ 77′ 36′

Fusión en secuencias de longitud 2.

***F*:** 25 33′ 15 18′ 07 21′ 12 36′ 84 90′ 19 38′
22 40′ 64 77′ 29 36′ 11′

SEGUNDA PASADA

Partición en secuencias de longitud 2.

F1: 25 33′ 07 21′ 84 90′ 22 40′ 29 36′
F2: 15 18′ 12 36′ 19 38′ 64 77′ 11′

Fusión en secuencias de longitud 4.

F: 15 18 25 33′ 07 12 21 36′ 19 38 84 90′
22 40 64 77′ 11 29 36′

TERCERA PASADA

Partición en secuencias de longitud 4.

F1: 15 18 25 33′ 19 38 84 90′ 11 29 36′
F2: 07 12 21 36′ 22 40 64 77′

Fusión en secuencias de longitud 8.

F: 07 12 15 18 21 25 33 36′ 19 22 38 40 64′
77 84 90′ 11 29 36′

CUARTA PASADA

Partición en secuencias de longitud 8.

F1: 07 12 15 18 21 25 33 36′ 11 29 36′
F2: 19 22 38 40 64 77 84 90′

Fusión en secuencias de longitud 16.

F: 07 12 15 18 19 21 22 25 33 36 38 40 64
77 84 90′ 11 29 36′

QUINTA PASADA

Partición en secuencias de longitud 16.

F1: 07 12 15 18 19 21 22 25 33 36 38 40 64 77 84 90′
F2: 11 29 36′

Fusión en secuencias de longitud 32.

F: 07 11 12 15 18 19 21 22 25 29 33 36 36 38 40
64 77 84 90

8.3.4 Ordenación por el método de mezcla equilibrada

El método de ordenación por **mezcla equilibrada**, conocido también cómo **mezcla natural**, es una optimización del método de mezcla directa.

La idea central de este algoritmo consiste en realizar las particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias de tamaño fijo previamente determinadas. Luego se realiza la fusión de las secuencias ordenadas, en forma alternada, sobre dos archivos. Aplicando estas acciones en forma repetida se logrará que el archivo original quede ordenado. Para la realización de este proceso de ordenación se necesitarán cuatro archivos. El archivo original *F* y tres archivos auxiliares a los que se denominará *F*1, *F*2 y *F*3. De estos archivos, dos serán considerados de entrada y dos de salida; esto, de manera alternada, con el objeto de realizar la fusión-partición. El proceso termina cuando en la realización de una fusión-partición el segundo archivo quede vacío.

Ejemplo 8.22

Supongamos que se desea ordenar las claves del archivo *F* utilizando el método de mezcla equilibrada.

F: 09 75 14 68 29 17 31 25 04 05 13 18 72 46 61

Los pasos que se realizan son:

PARTICIÓN INICIAL

F2: 09 75′ 29′ 25′ 46 61′
F3: 14 68′ 17 31′ 04 05 13 18 72′

PRIMERA FUSIÓN-PARTICIÓN

F: 09 14 68 75′ 04 05 13 18 25 46 61 72′
F1: 17 29 31′

SEGUNDA FUSIÓN-PARTICIÓN

F2: 09 14 17 29 31 68 75′
F3: 04 05 13 18 25 46 61 72′

TERCERA FUSIÓN-PARTICIÓN

F: 04 05 09 13 14 17 18 25 29 31 46 61 68 72 75
F1:

Observe que al realizar la tercera fusión-partición el segundo archivo queda vacío; por lo tanto, se puede afirmar que el archivo ya se encuentra ordenado. A continuación se presenta la descripción formal del algoritmo de mezcla equilibrada.

Algoritmo 8.20 Mezcla_equilibrada

Mezcla_equilibrada (*F*, *F1*, *F2*, *F3*)

```
{El algoritmo ordena los elementos del archivo F por el método de mezcla equilibrada. Utiliza
tres archivos auxiliares F1, F2 y F3}
{BAND es una variable de tipo booleano}

1. Llamar al algoritmo Partición_inicial con F, F2 y F3
2. Llamar al algoritmo Partición_fusión con F2, F3, F y F1
3. Hacer BAND ← FALSO
4. Mientras ((F1 ≠ VACÍO) o (F3 ≠ VACÍO)) Repetir
   4.1 Si (BAND = VERDADERO)
                entonces
                    Llamar al algoritmo Partición_fusión con F2, F3, F y F1
                    Hacer BAND ← FALSO
               si no
                    Llamar al algoritmo Partición_fusión con F, F1, F2 y F3
                    Hacer BAND ← VERDADERO
   4.2 {Fin del condicional del paso 4.1}
5. {Fin del ciclo del paso 4}
```

El algoritmo 8.20 requiere para su funcionamiento de dos algoritmos auxiliares, **Partición_inicial** y **Partición_fusión**, los cuales se presentan a continuación.

Algoritmo 8.21 Partición_inicial

Partición_inicial (*F*, *F2*, *F3*)

```
{El algoritmo produce la partición inicial del archivo F en dos archivos auxiliares, F2 y F3}
{AUX y R son variables de tipo entero. BAND es una variable de tipo booleano}

1. Abrir el archivo F para lectura.
2. Abrir los archivos F2 y F3 para escritura.
3. Leer R de F.
4. Escribir R en F2.
5. Hacer BAND ← VERDADERO y AUX ← R
6. Mientras (no sea el fin de archivo de F) Repetir
        Leer R de F
   6.1 Si (R ≥ AUX)
            entonces
                Hacer AUX ← R
       6.1.1 Si (BAND = VERDADERO)
                    entonces
                        Escribir R en F2
                    si no
```



```
                        Escribir R en F3
       6.1.2 {Fin del condicional del paso 6.1.1}
             si no
                    Hacer AUX ← R
       6.1.3 Si (BAND = VERDADERO)
                    entonces
                        Escribir R en F3
                        Hacer BAND ← FALSO
                    si no
                        Escribir R en F2
                        Hacer BAND ← VERDADERO
       6.1.4 {Fin del condicional del paso 6.1.3}
   6.2 {Fin del condicional del paso 6.1}
7. {Fin del ciclo del paso 6}
8. {Cerrar los archivos F, F2 y F3}
```

Algoritmo 8.22 Partición_fusión

Partición_fusión (*FA, FB, FC, FD*)

```
{El algoritmo produce la partición y la fusión de los archivos FA y FB, en los archivos FC y
FD}
{R1, R2 y AUX son variables de tipo entero. BAN1, BAN2 y BAN3 son variables de tipo
booleano}

1. Abrir los archivos FA y FB para lectura.
2. Abrir los archivos FC y FD para escritura.
3. Hacer BAN1 ← VERDADERO, BAN2 ← VERDADERO, BAN3 ← VERDADERO y
   AUX ← –32 768 {AUX se inicializa con un valor negativo alto}
4. Mientras ((no sea el fin de archivo de FA) o (BAN1 = FALSO)) y ((no
   sea el fin de archivo de FB) o ((BAN2 = FALSO)) Repetir
   4.1 Si (BAN1 = VERDADERO) entonces
                Leer R1 de FA
                Hacer BAN1 ← FALSO
   4.2 {Fin del condicional del paso 4.1}
   4.3 Si (BAN2 = VERDADERO) entonces
                Leer R2 de FB
                Hacer BAN2 ← FALSO
   4.4 {Fin del condicional del paso 4.3}
   4.5 Si (R1 < R2)
            entonces
       4.5.1 Si (R1 ≥ AUX)
                    entonces
           4.5.1.1 Si (BAN3 = VERDADERO)
                        entonces
                            Escribir R1 en FC
                        si no
```

```
                    Escribir R1 en FD
          4.5.1.2  {Fin del condicional del paso 4.5.1.1}
                    Hacer BAN1 ← VERDADERO y AUX ← R1
                si no
          4.5.1.3  Si (BAN3 = VERDADERO)
                    entonces
                       Escribir R2 en FC
                       Hacer BAN3 ← FALSO
                    si no
                       Escribir R2 en FD
       Hacer BAN3 ← VERDADERO
          4.5.1.4  {Fin del condicional del paso 4.5.1.3}
       Hacer BAN2 ← VERDADERO y AUX ← –32 768
       4.5.2  {Fin del condicional del paso 4.5.1}
          si no
       4.5.3  Si (R2 ≥ AUX)
                 entonces
          4.5.3.1  Si (BAN3 = VERDADERO)
                    entonces
                       Escribir R2 en FC
                          si no
                       Escribir R2 en FD
          4.5.3.2  {Fin del condicional del paso 4.5.3.1}
                    Hacer BAN2 ← VERDADERO y AUX ← R2
                si no
          4.5.3.3  Si (BAN3 = VERDADERO)
                    entonces
                       Escribir R1 en FC
                       Hacer BAN3 ← FALSO
                    si no
                       Escribir R1 en FD
                       Hacer BAN3 ← VERDADERO
          4.5.3.4  {Fin del condicional del paso 4.5.3.3}
                    Hacer BAN1 ← VERDADERO y AUX ← –32 768
       4.5.4  {Fin del condicional del paso 4.5.3}
    4.6  {Fin del condicional del paso 4.5}
5.  {Fin del ciclo del paso 4}
6.  Si (BAN1 = FALSO) entonces
    6.1  Si (BAN3 = VERDADERO)
            entonces
               Escribir R1 en FC
         6.1.1  Mientras (no sea el fin de archivo de FA) Repetir
                    Leer R1 de FA
                    Escribir R1 en FC
         6.1.2  {Fin del ciclo del paso 6.1.1}
            si no
               Escribir R1 en FD
         6.1.3  Mientras (no sea el fin de archivo de FA) Repetir
                    Leer R1 de FA
                    Escribir R1 en FD
```



```
         6.1.4  {Fin del ciclo del paso 6.1.3}
    6.2  {Fin del condicional del paso 6.1}
7.  {Fin del condicional del paso 6}
8.  Si (BAN2 = FALSO) entonces
    8.1  Si (BAN3 = VERDADERO)
            entonces
               Escribir R2 en FC
         8.1.1  Mientras (no sea el fin de archivo de FB) Repetir
                    Leer R2 de FB
                    Escribir R2 en FC
         8.1.2  {Fin del ciclo del paso 8.1.1}
            si no
               Escribir R2 en FD
         8.1.3  Mientras (no sea el fin de archivo de FB) Repetir
                    Leer R2 de FB
                    Escribir R2 en FD
         8.1.4  {Fin del ciclo del paso 8.1.3}
    8.2  {Fin del condicional del paso 8.1}
9.  {Fin del condicional del paso 8}
10. {Cerrar los archivos FA, FB, FC y FD}
```

Ejemplo 8.23

Supongamos que se desea ordenar las claves del archivo *F* utilizando el método de mezcla equilibrada.

***F*:** 25 33 15 18 21 07 12 36 84 90 19 38 40 22 64
77 29 36 11

Los pasos que se realizan son:

PARTICIÓN INICIAL

***F*2:** 25 33′ 07 12 36 84 90′ 22 64 77′ 11′
***F*3:** 15 18 21′ 19 38 40′ 29 36′

PRIMERA FUSIÓN-PARTICIÓN

***F*:** 15 18 21 25 33′ 22 29 36 64 77′
***F*1:** 07 12 19 36 38 40 84 90′ 11′

SEGUNDA FUSIÓN-PARTICIÓN

***F*2:** 07 12 15 18 19 21 25 33 36 38 40 84 90′
***F*3:** 11 22 29 36 64 77′

TERCERA FUSIÓN-PARTICIÓN

F: 07 11 12 15 18 19 21 22 25 29 33 36 36 38 40 64
77 84 90
F1:

EJERCICIOS

Ordenación interna

1. En un arreglo se guardan los apellidos de N alumnos. Aplique el método de la burbuja —ordenación por el método de intercambio directo— para ordenar el arreglo en forma ascendente, de manera que:

$$Ap_1 \leq Ap_2 \leq \ldots \leq Ap_n$$

2. Resuelva el problema 1 aplicando el método de la burbuja con señal.

3. Resuelva el problema 1 aplicando el método *shakersort*.

4. Compare el tiempo de ejecución de las soluciones 1, 2 y 3 para distintos valores de N.

5. Dado un arreglo unidimensional de N números enteros, ordénelo en forma descendente aplicando:

a) Inserción directa
b) Inserción binaria
c) Selección directa

Compare el tiempo de ejecución de las soluciones *a*, *b* y *c* para distintos valores de N.

6. Se tienen tres arreglos paralelos. El primero de ellos almacena las matrículas de N alumnos; el segundo, las calificaciones de los N alumnos obtenidas en un examen final, y el tercero, el número total de materias aprobadas por cada alumno. Los elementos de los arreglos se corresponden.

a) Aplique el método de inserción directa para ordenar los arreglos, de manera que queden ordenados en forma ascendente por matrícula.
b) Aplique el método *quicksort* para ordenar los arreglos, de manera que queden ordenados en forma descendente por el número total de materias aprobadas.

7. En cierta empresa se maneja una lista de precios de los N artículos que se venden. De cada artículo se tiene la siguiente información:

- Clave del artículo
- Nombre del artículo
- Precio del artículo

a) Ordene el arreglo en forma ascendente según el campo "clave". Aplique el método *quicksort*.

b) Ordene el arreglo en forma descendente según el campo "precio". Aplique el método del *montículo*.

8. Resuelva el inciso *a)* del problema 7 aplicando el método de selección directa. Compare el tiempo de ejecución de esta solución con la obtenida en el ejercicio anterior, para distintos valores de *N*.

9. Una compañía propietaria de una cadena de hoteles quiere que se ordene su información. De cada hotel se tienen los siguientes datos:

- Nombre del hotel
- Ciudad en la que se encuentra el hotel
- Número de estrellas
- Número de cuartos

En una misma ciudad puede haber varios hoteles de la compañía.

Escriba un programa que ordene el arreglo según el campo "ciudad", en primer término, y luego, según el campo, "nombre". Es decir, el arreglo debe quedar:

$Nombre_1$	$Ciudad_1$	$Núm.estrellas_1$	$Núm.cuartos_1$
$Nombre_2$	$Ciudad_1$	$Núm.estrellas_2$	$Núm.cuartos_2$
...			
$Nombre_i$	$Ciudad_2$	$Núm.estrellas_i$	$Núm.cuartos_i$
...			
$Nombre_n$	$Ciudad_k$	$Núm.estrellas_k$	$Núm.cuartos_n$

Donde:

- $Ciudad_1 < Ciudad_2 < ... < Ciudad_k$
- $Nombre_1 < Nombre_2 < ... < Nombre_{j-1}$
- $Nombre_i < Nombre_{i+1} < ... < Nombre_{j-1}$
- ...
- $Nombre_i < Nombre_{i+1} < ... < Nombre_n$

10. Retome el problema 8 del capítulo 1. Ordene los arreglos SUR, CENTRO y NORTE, aplicando:

a) Inserción directa
b) Inserción binaria
c) Shell

11. Una escuela tiene almacenados los principales datos de cada alumno. Éstos son:

- Nombre del alumno
- Matrícula
- Número de materias aprobadas
- Promedio

a) Aplique el método de selección directa para ordenar el arreglo de *N* alumnos en forma ascendente, según el campo "nombre".
b) Aplique el método *quicksort* para ordenar el arreglo de *N* alumnos en forma ascendente, según el campo "número de materias aprobadas".

12. Se tiene un arreglo de *N* números enteros.

a) ¿Cuántas comparaciones y cuántos intercambios se deben realizar si se ordena el arreglo con el método de la burbuja?
b) ¿Cuántas comparaciones y cuántos intercambios se deben realizar si se ordena el arreglo con el método de selección directa?

13. Dado un arreglo unidimensional de *N* números enteros que debe ser ordenado en forma ascendente, conteste las siguientes preguntas:

a) Cuántas comparaciones e intercambios se deben realizar, aplicando el método de inserción directa, si:

- El arreglo ya está ordenado.
- El arreglo está ordenado en forma descendente.

b) Conteste la pregunta del inciso anterior, para el método *quicksort*.
c) Conteste la pregunta del inciso anterior, para el método del montículo.

14. Retome el problema 7, considerando que se definió la clase *Artículo* y se tiene un arreglo de objetos de este tipo. Escriba una función que ordene el arreglo de objetos, utilizando el algoritmo de *quicksort*.

15. Defina una clase *Arreglo*, según lo visto en el capítulo 1. En la clase debe incluir por lo menos 2 métodos de los estudiados en este capítulo para ordenar los elementos del arreglo.

16. Escriba una función que anime el proceso de ordenación usando el algoritmo de *quicksort*. Es decir, en la medida en que se va ordenando el arreglo, en la pantalla se debe ir reflejando gráficamente su cambio de estado.

Ordenación externa

17. Dado un archivo de números enteros, ordénelo e imprímalo.

18. En el archivo EMPLEADOS se tiene información sobre los empleados de una empresa. Los datos almacenados por cada empleado son:

- Nombre
- Estado civil
- Antigüedad

- Categoría
- Sueldo

Ordene el archivo según el campo "nombre".

a) Aplique mezcla directa.
b) Aplique mezcla equilibrada.

19. Se tiene un archivo con información sobre huéspedes de un hotel:

- Número de habitación
- Nombre del huésped
- Fecha de llegada

Ordene el archivo aplicando el método de mezcla directa.

a) Según el campo "número de habitación".
b) Según el campo "nombre".

20. Dado un archivo de cadenas de caracteres, ordénelo en forma descendente aplicando el método de mezcla equilibrada.

21. Se tienen dos archivos ordenados con los nombres de los estudiantes de una escuela. No se han actualizado de la misma manera los dos archivos, habiéndose dado de alta a algunos alumnos en un archivo pero no en el otro. Escriba un programa que obtenga un tercer archivo, también ordenado, intercalando la información de los dos archivos existentes. (No deben quedar elementos repetidos.)

22. Se tienen tres archivos A1, A2 y A3 con información sobre recitales efectuados en un teatro, a lo largo de los tres últimos años. Cada registro de los archivos tiene los siguientes campos:

- Nombre del cantante u orquesta que ofreció el recital
- Número de presentaciones
- Fechas de las presentaciones

Los tres archivos están ordenados en forma ascendente según el primer campo. Escriba un programa que intercale los tres archivos, formando el archivo RECITALES.