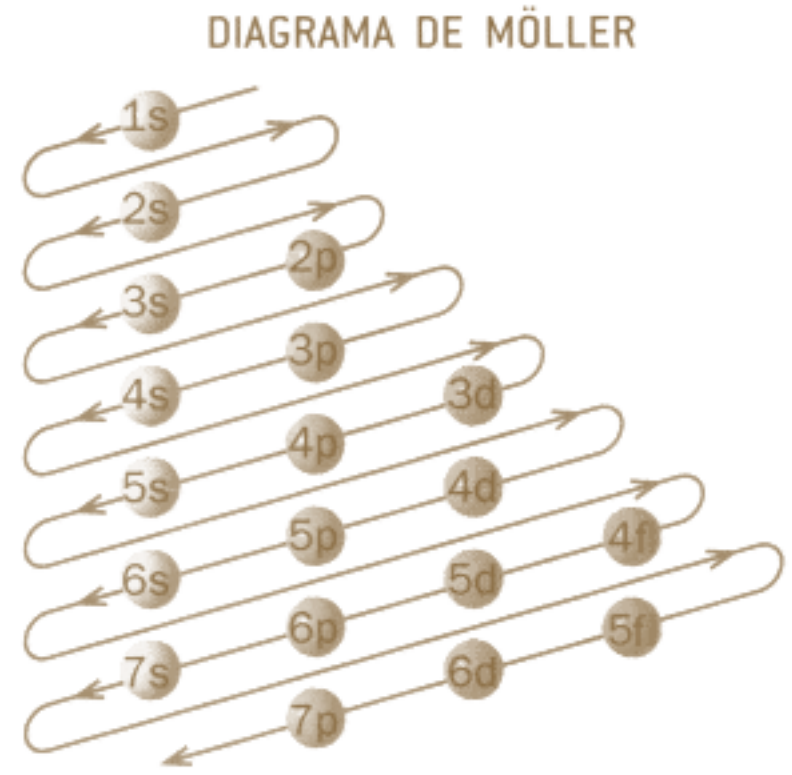
Universidad Nacional Autónoma de México
Facultad de Ingeniería
Algoritmos y estructuras de datos
Tema 6:
MÉTODOS DE ORDENAMIENTO

6 Métodos de ordenamiento

Objetivo: Aplicar los métodos internos y externos más importantes para efectuar ordenamientos en la computadora. Diseñar y aplicar algoritmos.

6 Métodos de ordenamiento

6.1 Generalidades.
6.2 Ordenamientos internos.
6.3 Ordenamientos externos.
6.4 Archivos auxiliares almacenados en disco.



6.1 Generalidades.

6.1 Generalidades.

Dentro de las estructuras de datos, el término ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica.

Por tanto, la clasificación (ordenación) de la información permite realizar búsquedas y/o actualización de información de una manera más eficiente.

De manera formal, un ordenamiento se define a partir de un conjunto de elementos. Sea A un conjunto de N elementos:

$$A_1, A_2, A_3, \ldots A_n,$$

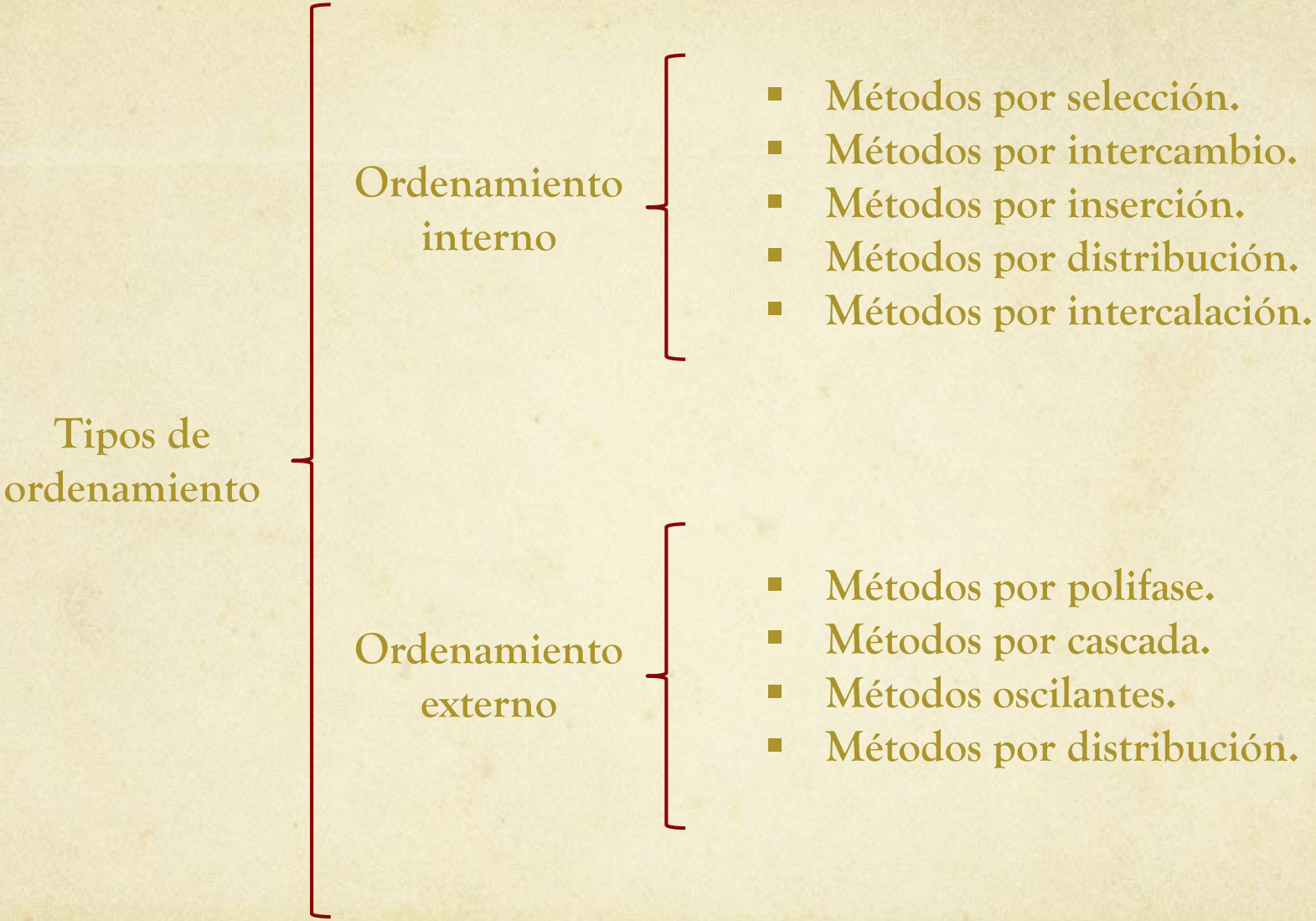
Ordenar significa permutar los elementos de tal forma que se clasifiquen de acuerdo a una distribución preestablecida.

- Ascendente $A_1 \leq A_2, \leq A_3 \leq \ldots \leq A_n,$
- Descendente $A_1 \geq A_2 \geq A_3 \geq \ldots \geq A_n$

Los métodos de ordenamiento se dividen en dos grupos: ordenamientos internos y ordenamientos externos.

Los ordenamientos internos se caracterizan porque solo ocupan memoria principal. Se utilizan cuando el volumen de información o datos que se va a manejar es pequeño, ya que la memoria principal es un dispositivo de baja capacidad de almacenamiento.

Los ordenamientos externos ocupan tanto memoria principal como memoria secundaria. Se utilizan cuando la cantidad de información que se maneja es muy grande y la capacidad de almacenamiento que posee la memoria principal no es suficiente y, por ende, se debe utilizar la memoria secundaria.


Tipos de ordenamiento
Ordenamiento interno
Métodos por selección.
Métodos por intercambio.
Métodos por inserción.
Métodos por distribución.
Métodos por intercalación.
Ordenamiento externo
Métodos por polifase.
Métodos por cascada.
Métodos oscilantes.
Métodos por distribución.

6.2 Ordenamientos internos.

6.2 Ordenamientos internos.

Los métodos de ordenación interna son aquellos que se encargan de clasificar la información contenida en la memoria principal.

Los métodos de ordenamiento interno se clasifican en:

- Metodo de selección
- Metodo de intercambio
- Metodo de inserción
- Metodo de distribución
- Metodo de intercalación

Los métodos de ordenación interna también se pueden clasificar, con base en su eficiencia, en:

- Métodos directos (n^2): tienen como característica que su implementación es relativamente sencilla, pero son ineficientes cuando la instancia de entrada es grande (n > 10,000).
- Métodos logarítmicos (n * log n): están compuestos por algoritmos más complejos que los directos, pero resultan más eficientes ya que realizan menos comparaciones y movimientos para ordenar los elementos.

La eficiencia de un algoritmo está definida por el tiempo de ejecución del mismo. Dentro de los métodos de ordenamiento, el tiempo de ejecución depende, fundamentalmente, del número de comparaciones y del número de movimientos que se realicen para ordenar los elementos.

Por tanto, cuando la instancia de entrada es pequeña, se recomienda utilizar métodos directos, cuando la instancia de entrada es grande es más recomendable utilizar métodos logarítmicos.

6.2.1 Métodos por selección

El método de ordenación por selección consiste en buscar el menor elemento del conjunto y colocarlo en la primera posición. Luego se busca el siguiente elemento más pequeño y se 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:

- Seleccionar el menor elemento del conjunto.
- Intercambiar dicho elemento con el primero.
- Repetir los pasos anteriores para los n–1 elementos.

Ordenar el siguiente conjunto, transportando el elemento de menor tamaño hacia la parte izquierda del conjunto por el método de selección.

Primer recorrido. Se realiza la asignación menor ← A[1]

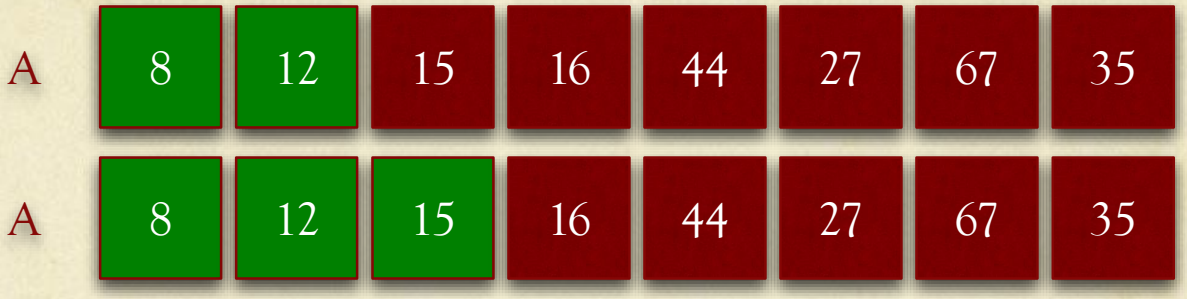
menor < A[2]	15 < 67	Sí se cumple, no se realiza mov.
menor < A[3]	15 < 8	No se cumple => menor ← A[3]
menor < A[4]	8 < 16	Sí se cumple, no se realiza mov.
menor < A[5]	8 < 44	Sí se cumple, no se realiza mov.
menor < A[6]	8 < 27	Sí se cumple, no se realiza mov.
menor < A[7]	8 < 12	Sí se cumple, no se realiza mov.
menor < A[8]	8 < 35	Sí se cumple, no se realiza mov.

Segundo recorrido. Se realiza la asignación menor ← A[2]

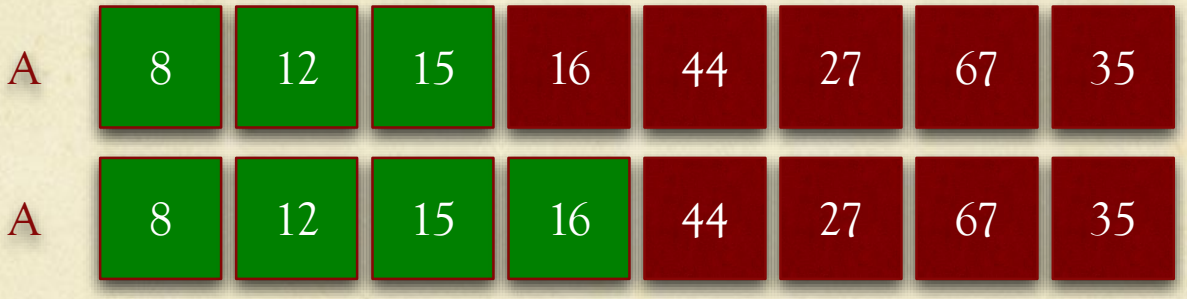
menor < A[3]	67 < 15	No se cumple => menor ← A[3]
menor < A[4]	15 < 16	Sí se cumple, no se realiza mov.
menor < A[5]	15 < 44	Sí se cumple, no se realiza mov.
menor < A[6]	15 < 27	Sí se cumple, no se realiza mov.
menor < A[7]	15 < 12	No se cumple, => menor ← A[7]
menor < A[8]	12 < 35	Sí se cumple, no se realiza mov.

El recorrido continúa n–1 veces.

3

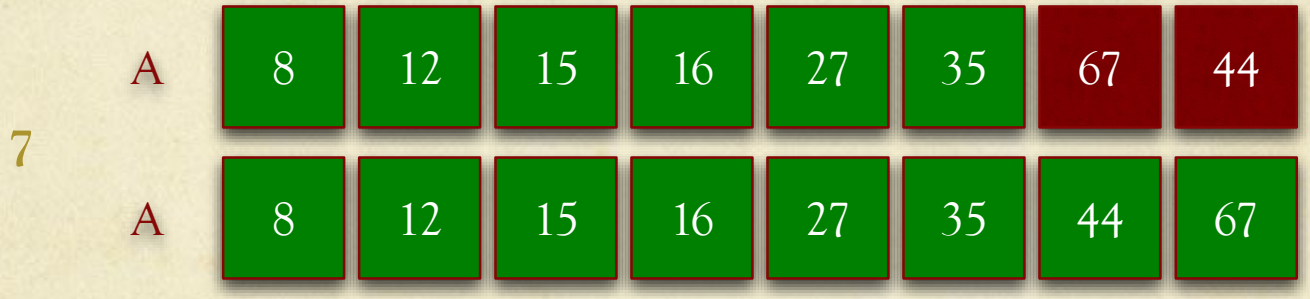


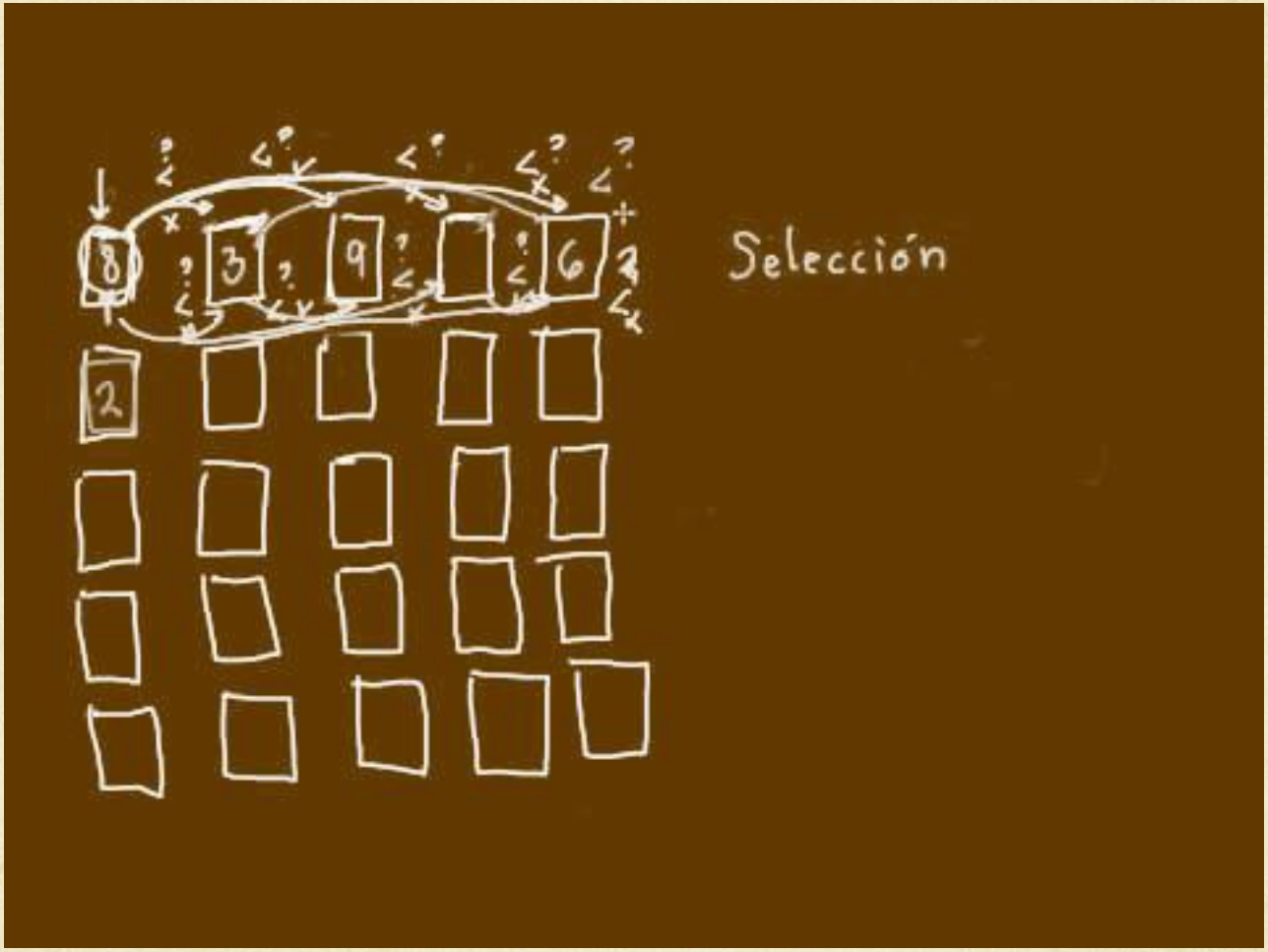
4



El recorrido continúa n–1 veces.

El recorrido continúa n–1 veces.




8
3
9
6
2
Selección

El algoritmo de ordenación utilizando el método de selección directo es el siguiente:

```
FUNC seleccionDirecta (n: Entero, A[]: Entero): DEV vacío
        cont ← 1, menor, aux: ENTERO
        MIENTRAS cont < n HACER
                menor ← A[cont] y aux ← cont
                cont2 ← cont : Entero
                MIENTRAS cont2+1 < N HACER
                        SI menor > A[j] HACER
                                menor ← A[cont2]
                                aux ← cont2
                        FIN_SI
                FIN_MIENTRAS
                A[cont2]← A[cont]
                A[cont] ← menor
        FIN_MIENTRAS
FIN_FUNC
```

El análisis de complejidad del algoritmo de selección directo está dado por el número de comparaciones que se realizan.

En el primer recorrido se realizan (n–1) comparaciones, en el segundo recorrido se realizan (n–2) comparaciones, en el tercer recorrido se realizan (n–3) comparaciones y así sucesivamente hasta el penúltimo recorrido donde se realizan 2 comparaciones y en el último recorrido se realiza 1 comparación.

Con base en el análisis anterior, el número de comparaciones está dado por:

$$C = (n-1) + (n-2) + \ldots + 2 + 1$$

Por inducción matemática, se tiene:

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

El número máximo de intercambios siempre será n–1, entonces el número de movimientos es:

$$M = n-1$$

Si se quiere ordenar un conjunto de 500 elementos, se efectuarán 124,750 comparaciones y 499 movimientos.

El tiempo promedio para ejecutar el algoritmo de intercambio es $0(n^2)$.

6.2.2 Métodos por intercambio

El método de intercambio directo (conocido como burbuja) es uno de los métodos de ordenamiento más simples tanto en su comprensión como en su implementación, aunque es un método muy ineficiente.

El método de intercambio directo puede trabajar de dos maneras distintas: moviendo los elementos más pequeños a la izquierda del conjunto o transladando los elementos más grandes a la derecha del conjunto.

El algoritmo consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que todos se encuentren ordenados, partiendo del extremo izquierdo o derecho, según se desee.

Por tanto, se realizan n–1 comparaciones para mover el elemento menor o mayor (según sea el caso) a su posición final.

Ordenar el siguiente conjunto, transportando el elemento de menor tamaño hacia la parte izquierda del conjunto, por el método de intercambio,

Primer recorrido.

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]	8 > 12	No hay intercambio
A[2]>A[3]	67 > 8	Sí hay intercambio
A[1]>A[2]	15 > 8	Sí hay intercambio

Segundo recorrido.

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

El recorrido y las comparaciones internas se siguen realizando hasta que se llegue al elemento n–1.

Los recorridos y las comparaciones se siguen realizando hasta que se llegue al elemento n–1.


a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]
0 1 3 2 5 4 6 7 8 9

El algoritmo de ordenación utilizando el método de intercambio directo es el siguiente:

```
FUNC intercambioDirecto (n: Entero, A[]: Entero): DEV vacío
        cont ← 2, comp, aux: ENTERO
        MIENTRAS cont < n HACER
                comp ← n
                MIENTRAS comp > cont HACER
                        SI A[comp-1] > A[j] HACER
                                aux ← A[comp-1]
                                A[comp-1] ← A[comp]
                                A[comp] ← aux
                        FIN_SI
                FIN_MIENTRAS
        FIN_MIENTRAS
FIN_FUNC
```

El análisis de complejidad del algoritmo de intercambio directo está dado por el número de comparaciones que se realizan.

En el primer recorrido se realizan (n–1) comparaciones, en el segundo recorrido se realizan (n–2) comparaciones, en el tercer recorrido se realizan (n–3) comparaciones y así sucesivamente hasta el penúltimo recorrido donde se realizan 2 comparaciones y en el último recorrido se realiza 1 comparación.

Con base en el análisis anterior, el número de comparaciones está dado por:

$$C = (n-1) + (n-2) + \ldots + 2 + 1$$

Por inducción matemática, se tiene:

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

Considerando el análisis anterior, el número de movimientos depende de las veces que la comparación se cumpla y depende directamente de la manera en la que esté acomodado el conjunto de elementos (ordenado, desordenado o en orden inverso). Los movimientos promedio para el mejor, el peor y el caso base son:

$$M_{min} = 0$$
$$M_{med} = 0.75 * (n^2 - n)$$
$$M_{max} = 1.5 * (n^2 - n)$$

Por tanto, el tiempo promedio para ejecutar el algoritmo de intercambio es $0 (n^2)$.

Se desea ordenar un conjunto de 500 elementos, ¿cuál es el número de comparaciones y movimientos para el mejor, el peor y el caso base?

- Si el conjunto se encuentra ordenado:
  - C = 124,750
  - $M_{min} = 0$
- Si el conjunto se encuentra desordenado de forma aleatoria:
  - C = 124,750
  - $M_{med} = 187,125$
- Si el conjunto se encuentra desordenado:
  - C = 124,750
  - $M_{max} = 374,250$

Ordenación por el método de intercambio directo con señal

Este método es una modificación del método de intercambio directo. La modificación consiste en utilizar una bandera (señal) que permita validar si no se ha o no efectuado algún intercambio, esto con el fin de comprobar si el conjunto ya se encuentra ordenado.

El algoritmo de ordenación utilizando el método de intercambio con señal es el siguiente:

```
FUNC intercambioSeñal (n: Entero, A[]: Entero): DEV vacío
        cont ← 2, comp, aux: ENTERO y band ← Falso
        MIENTRAS (cont < n) Y (band = Falso) HACER
                comp ← n y band ← Verdadero
                MIENTRAS comp > cont HACER
                        SI A[comp-1] > A[j] HACER
                                aux ← A[comp-1]
                                A[comp-1] ← A[comp]
                                A[comp] ← aux
                                band ← Falso
                        FIN_SI
                FIN_MIENTRAS
        FIN_MIENTRAS
FIN_FUNC
```

Ordenación por el método de intercambio de la sacudida

El método shaker sort (método de la sacudida), es una optimización del método de intercambio directo. La idea de este método es combinar las dos maneras en se puede realizar el método de la burbuja (intercambio directo).

Dentro del método de la sacudida se pueden identificar dos etapas distintas en cada iteración.

La primera etapa consiste en recorrer los elementos de derecha a izquierda, moviendo los elementos más pequeños hacia la parte izquierda del conjunto. Se debe almacenar la posición del último elemento intercambiado (izq).

La segunda etapa consiste en recorrer los elementos de izquierda a derecha, moviendo los elementos más grandes hacia la parte derecha del conjunto. Se debe almacenar la posición del último elemento intercambiado (der).

Las iteraciones siguientes se realizan sobre el intervalo [izq, der]. El algoritmo termina cuando en alguna iteración no se producen intercambios o bien cuando el elemento extremo izquierdo del conjunto (izq) es mayor al contenido del elemento extremo derecho del conjunto (der).

Ordenar el siguiente conjunto de forma ascendente utilizando el método de la sacudida (shaker sort).

Primer recorrido (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]	8 > 12	No hay intercambio
A[2]>A[3]	67 > 8	Sí hay intercambio
A[1]>A[2]	15 > 8	Sí hay intercambio

izq = 2

Primer recorrido (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

der = 7

Segundo recorrido (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

izq = 3

Segundo recorrido (De izquierda a derecha).

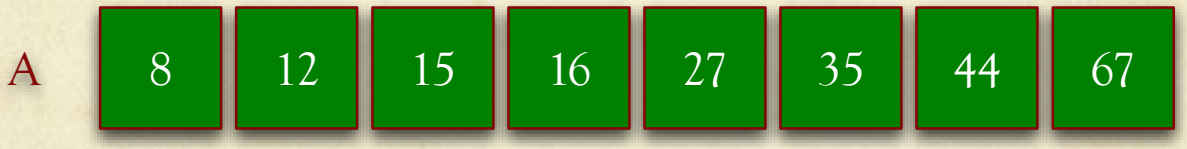
A[3]>A[4]	15 > 16	No hay intercambio
A[4]>A[5]	16 > 27	No hay intercambio
A[5]>A[6]	27 > 44	No hay intercambio
A[6]>A[7]	44 > 35	Sí hay intercambio

der = 6

Tercer recorrido (De derecha a izquierda).

A[6]>A[7]	27 > 35	No hay intercambio
A[5]>A[6]	16 > 27	No hay intercambio
A[4]>A[5]	15 > 16	No hay intercambio

Debido a que en el tercer recorrido no se realizó intercambio alguno, la ejecución del algoritmo termina.



El algoritmo de ordenación utilizando el método de la sacudida es el siguiente:

```
FUNC shakerSort (n: Entero, A[]: Entero): DEV vacío
        izq ← 2, der ← n, pos ← n, aux, cont: ENTERO
        MIENTRAS (izq ≤ der) HACER      ** Inicia ciclo general
                cont ← der
                ** Recorrido de derecha a izquierda
                MIENTRAS cont ≥ izq HACER
                        SI A[cont-1] > A[cont] ENTONCES
                                aux ← A[cont-1]
                                A[cont-1 ] ← A[cont]
                                A[cont] ← aux
                                pos ← cont
                        FIN_SI
                FIN_MIENTRAS
                izq ← pos-1
```

El algoritmo de ordenación utilizando el método de la sacudida es el siguiente:

```
                    ** Recorrido de izquierda a derecha
                    cont ← izq
                    MIENTRAS cont ≤ der HACER
                              SI A[cont-1] > A[cont] ENTONCES
                                        aux ← A[cont-1]
                                        A[cont-1 ] ← A[cont]
                                        A[cont] ← aux
                                        pos ← cont
                              FIN_SI
                    FIN_MIENTRAS
                    der ← pos-1
          FIN_MIENTRAS              ** Termina ciclo
FIN_FUNC
```

Ordenación por el método de intercambio quicksort

El método quick sort es uno de los más eficientes y veloces de los métodos de ordenación interna. También es conocido como método rápido o de ordenación por partición.

El algoritmo sigue los siguientes pasos:

- Se toma un elemento X de una posición cualquiera del conjunto.
- Se intenta ubicar al elemento X en la posición correcta del conjunto, de tal manera que todos los elementos que se encuentren a su izquierda sean menores o iguales a X y todos los elementos que se encuentren a su derecha sean mayores o iguales a X.
- Se repiten los pasos anteriores para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición de X en el conjunto.
- El proceso termina cuando todos los elementos se encuentran ordenados.

Ordenar el siguiente conjunto de forma ascendente utilizando el método de intercambio quicksort.

Primer recorrido.

$x \leftarrow A[1]$

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

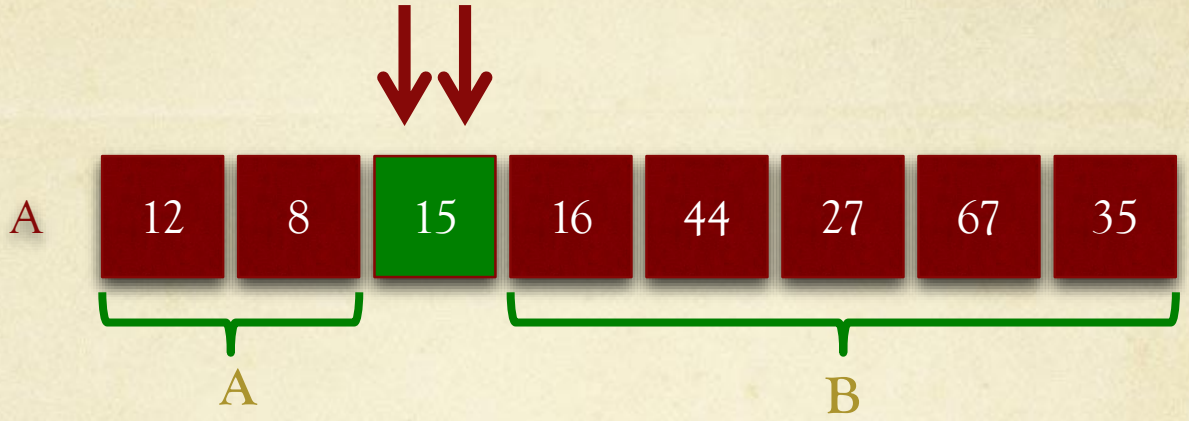
Primer recorrido.

Recorrido de izquierda a derecha

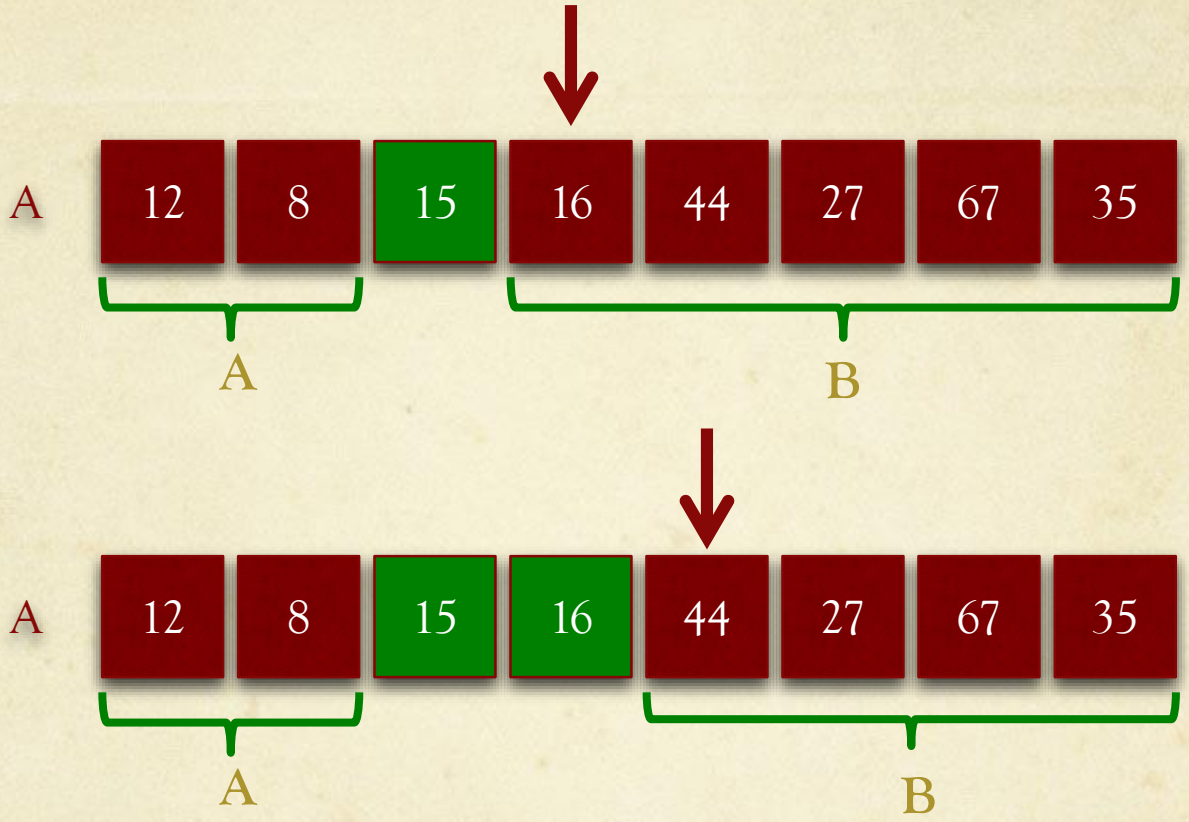
$A[2] \leq x$ $67 \leq 15$ Sí hay intercambio

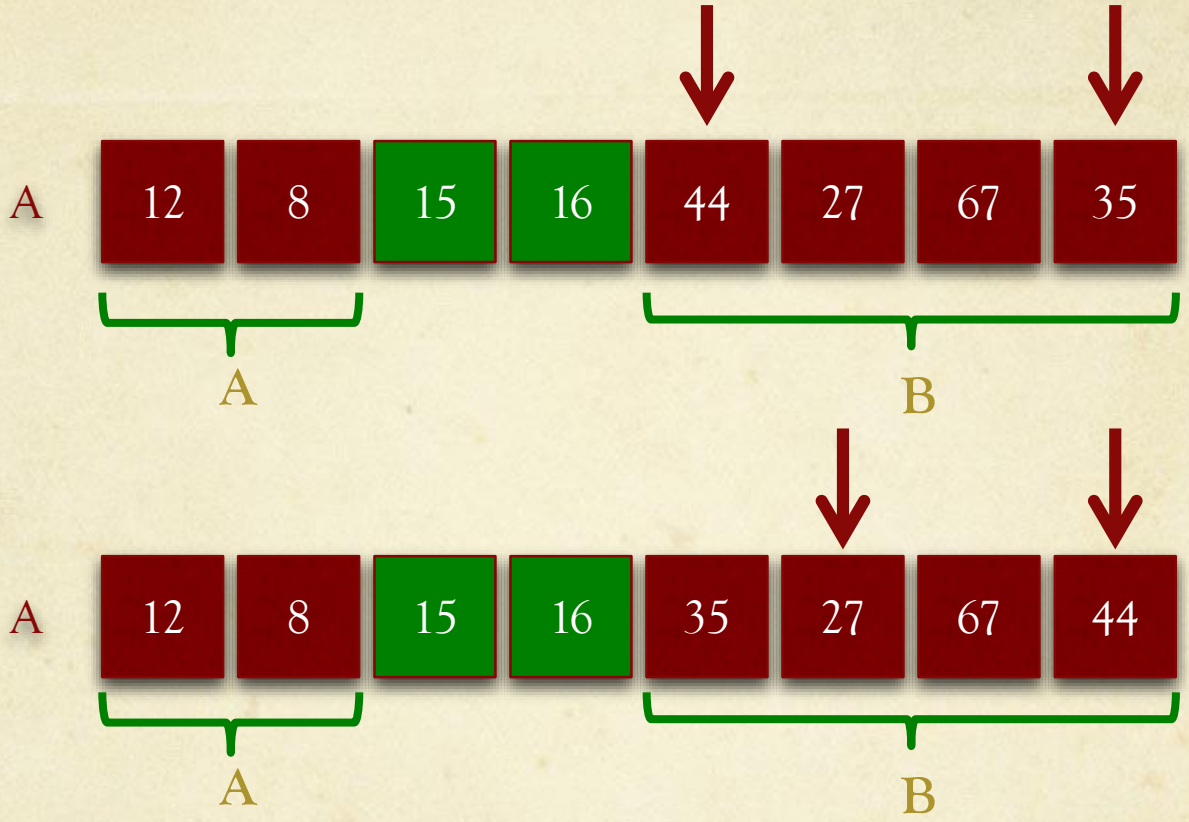
Recorrido de derecha a izquierda

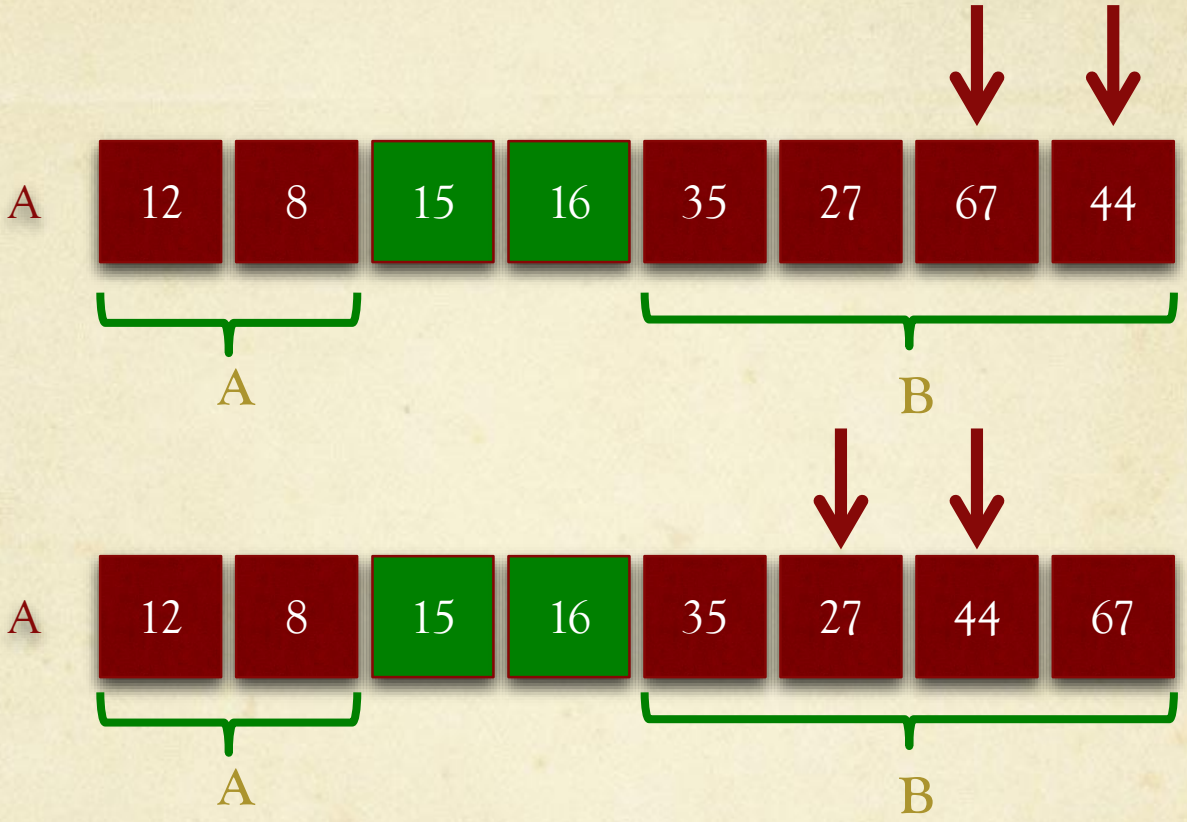
A[6] ≥ x	27 ≥ 15	No hay intercambio
A[5] ≥ x	44 ≥ 15	No hay intercambio
A[4] ≥ x	16 ≥ 15	No hay intercambio
A[3] ≥ x	8 ≥ 15	Sí hay intercambio

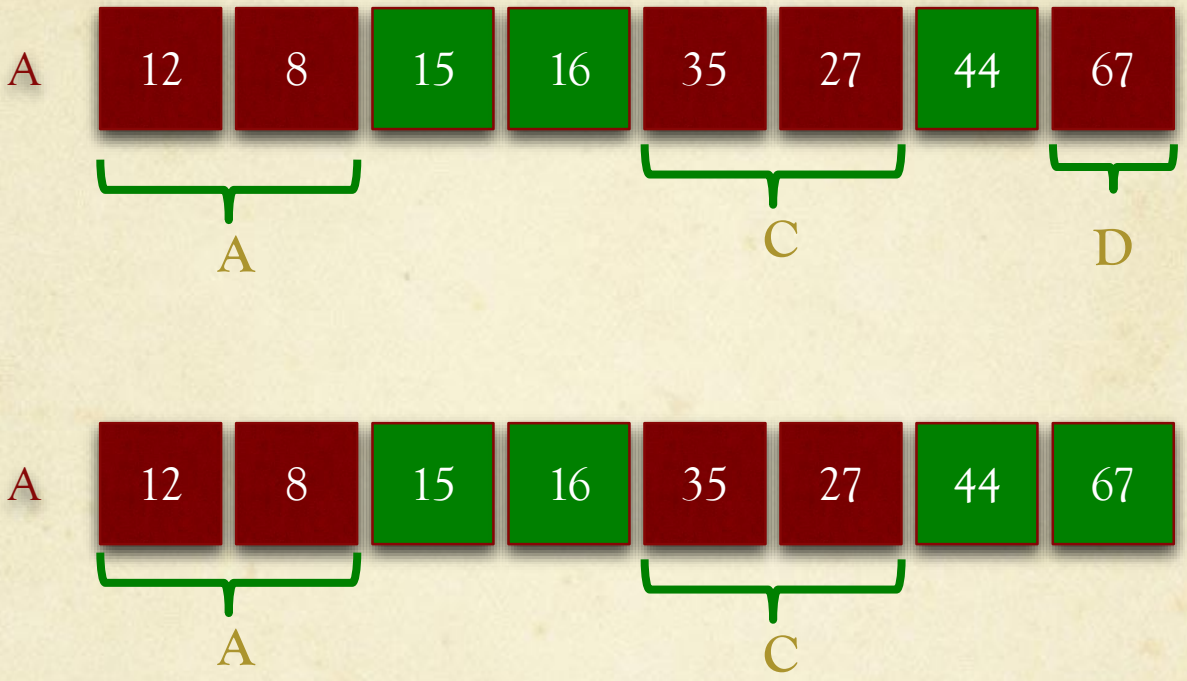


En este momento el proceso para el número x elegido termina debido a que el recorrido de izquierda a derecha debería iniciar en la misma posición donde se encuentra el elemento x y, por tanto, x se encuentra en la posición correcta (los elementos del primer conjunto son menores o iguales a x y los elementos del segundo conjunto son mayores o iguales a x).

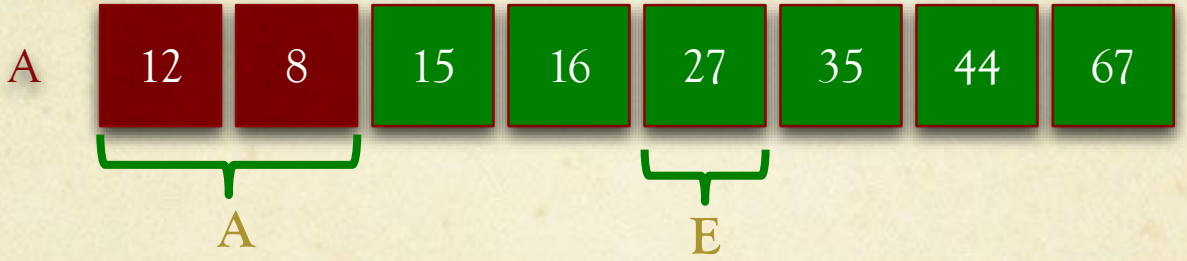

A
12
8
15
16
44
27
67
35
A
B
A
12
8
15
16
44
27
67
35
A
B


A
12
8
15
16
44
27
67
35
A
B
A
12
8
15
16
35
27
67
44
A
B


A
12
8
15
16
35
27
67
44
A
B
A
12
8
15
16
35
27
44
67
A
B


A
12
8
15
16
35
27
44
67
A
C
D
A
12
8
15
16
35
27
44
67
A
C


A
12
8
15
16
27
35
44
67
A
E


A
12
8
15
16
27
35
44
67
A
E


A
8
12
15
16
27
35
44
67
F


A
8
12
15
16
27
35
44
67

Como se puede observar en el ejemplo, el algoritmo de ordenación utilizando el método de intercambio quicksort es iterativ.

```
FUNC quicksort (ini: Entero, fin: Entero): DEV vacío
        izq ← ini, der ← fin, pos ← ini, band ← verdad: ENTERO
        MIENTRAS band = verdad HACER
                band ← falso
                MIENTRAS A[pos] ≤ A[der] Y pos <> der HACER
                        der ← der–1
                FIN_MIENTRAS
                SI pos <> der HACER
                        aux ← A[pos]
                        A[pos] ← A[der]
                        A[der] ← aux
                        pos ← der
                FIN_SI
```

```
                MIENTRAS A[pos] ≥ A[izq] Y pos <> izq HACER
                        izq ← izq-1
                FIN_MIENTRAS
                SI pos <> izq HACER
                        band ← verdad
                        aux ← A[pos]
                        A[pos] ← A[izq]
                        A[izq] ← aux
                        pos ← izq
                FIN_SI
        FIN_MIENTRAS
        SI (pos-1) > ini HACER
                quicksortRecursivo (ini, pos-1)
        SI fin > (pos+1) HACER
                quicksortRecursivo (pos+1, fin)
FIN_FUNC
```

El análisis de complejidad del algoritmo de intercambio quicksort, demuestra que éste es el método de ordenación interna más rápido en la actualidad.

Si se escoge, en cada recorrido, el elemento que ocupa la posición central del conjunto de datos, el número de recorridos necesarios para ordenarlo es del orden de log n.

Por otro lado, el número de comparaciones que se realizan está dado por la fórmula:

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

Debido a que el número de términos de la sumatoria (m) es igual al número de recorridos realizados (log n) se puede afirmar que:

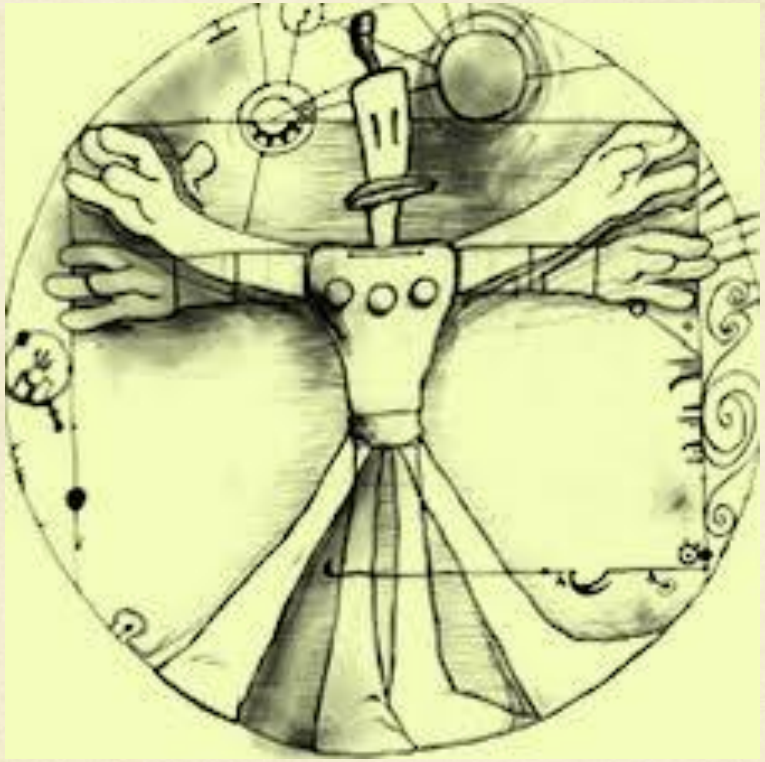
$$C = (n-1) * \log n$$

El peor caso de complejidad ocurre cuando el conjunto está ordenado o cuando se encuentra en orden inverso. En tales casos el número de comparaciones está dado por la fórmula:

$$C = n + (n-1) + (n-2) + \ldots + 2$$

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

Por lo tanto, en general, el tiempo de ejecución es proporcional a $O(n * \log n)$. En el peor caso, el tiempo de ejecución es proporcional a $O(n^2)$.



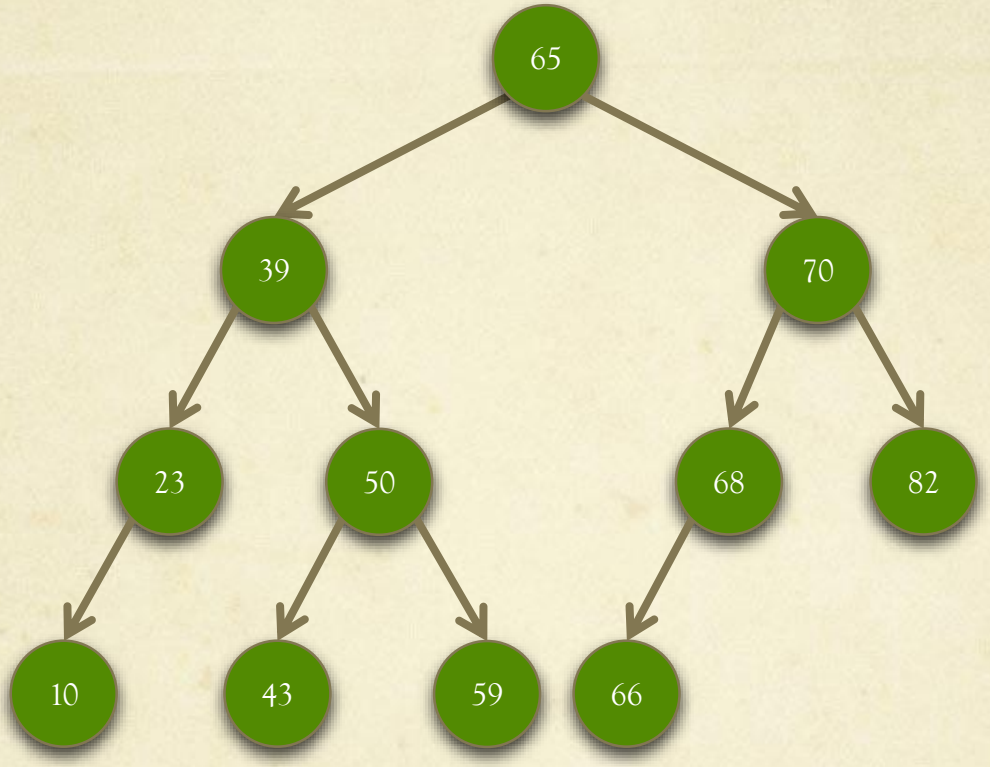
6. Métodos de ordenamiento.

Ordenación por el método de intercambio heapsort

El método heapsort (montículo), define que para todo nodo de la estructura se debe cumplir que su valor sea mayor o igual que el valor de su hijo izquierdo y menor o igual al valor de su hijo derecho.

El método heapsort se basa en dos operaciones:

- Construir un montículo.
- Eliminar la raíz del montículo de forma repetida.

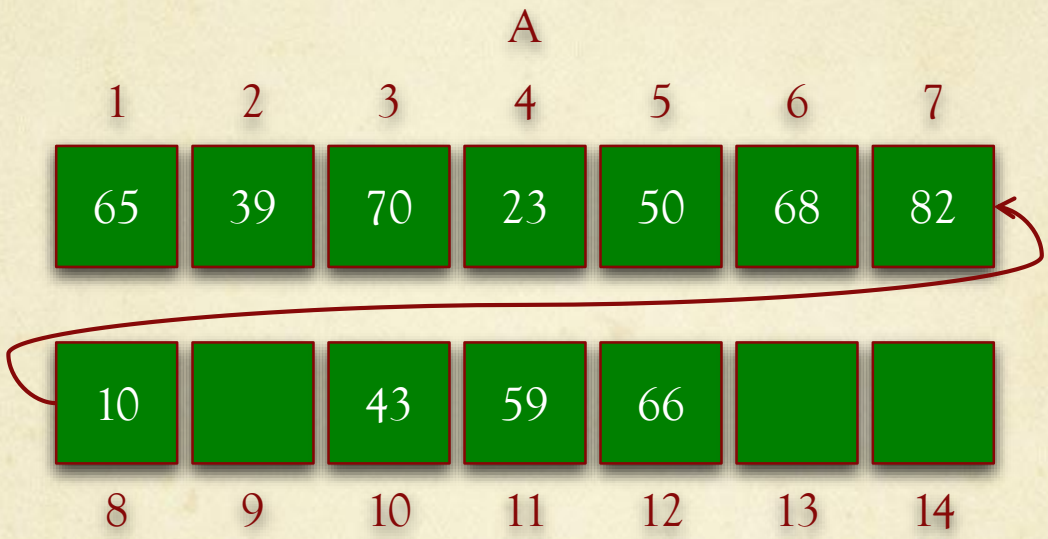


Montículo (heapsort).

Un montículo, al igual que un árbol binario, se puede representar en una estructura lineal teniendo en cuenta que para todo nodo k:

- El nodo k se almacena en la posición k de la estructura de datos lineal.
- El nodo (hijo) izquierdo del nodo k se almacena en la posición 2*k.
- El nodo (hijo) derecho del nodo k se almacena en la posición 2*k+1.

Por tanto, el montículo de la figura anterior se puede representar de la siguiente manera:



El análisis de eficiencia del método del montículo, al igual que el de los métodos logarítmicos, es complejo. Sin embargo, se puede afirmar que este método es muy rápido incluso para valores grandes de n, ya que su tiempo de ejecución, incluso en el peor caso, está dado por $\Theta(n*\log n)$

6.2.3 Métodos por inserción

El método consiste en insertar un elemento del conjunto en la parte izquierda del mismo, que ya se encuentra ordenado. El proceso se repite a partir del segundo elemento hasta el enésimo.

Este método es parecido al que utilizan los jugadores de cartas cuando acomodan (ordenan) sus jugadas, de ahí que también se le conoce como el método de la baraja.

Ordenar el siguiente conjunto de forma ascendente utilizando el método de inserción directa (método de la baraja).

Primer recorrido.

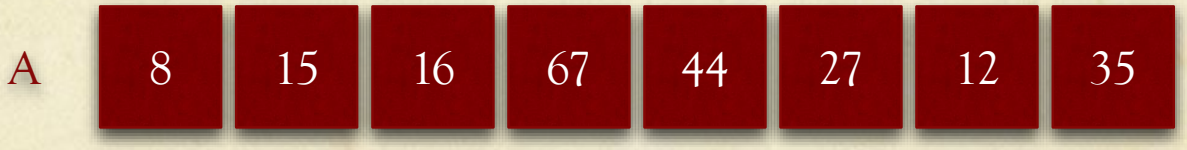
A[2]<A[1] 67 < 15 No hay intercambio

Segundo recorrido.

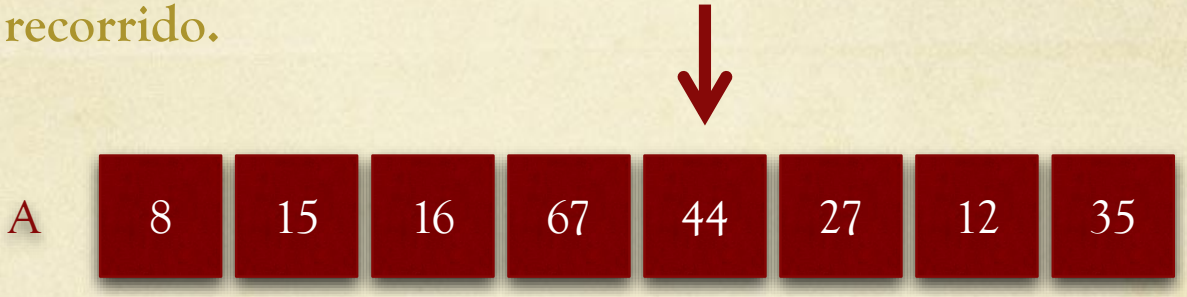
A[3]<A[2]	8 < 67	Sí hay intercambio
A[2]<A[1]	8 < 15	Sí hay intercambio

Tercer recorrido.

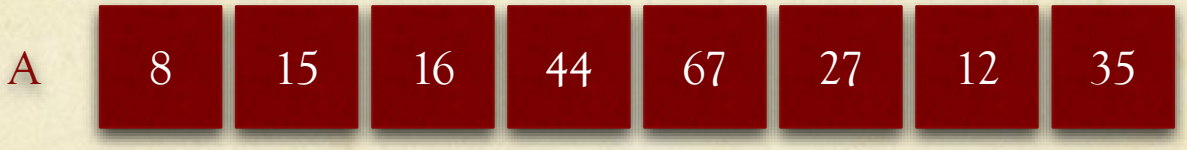
A[4]<A[3]	16 < 67	Sí hay intercambio
A[3]<A[2]	16 < 15	No hay intercambio



Cuarto recorrido.



A[5]<A[4]	44 < 67	Sí hay intercambio
A[4]<A[3]	44 < 16	No hay intercambio



Quinto recorrido.

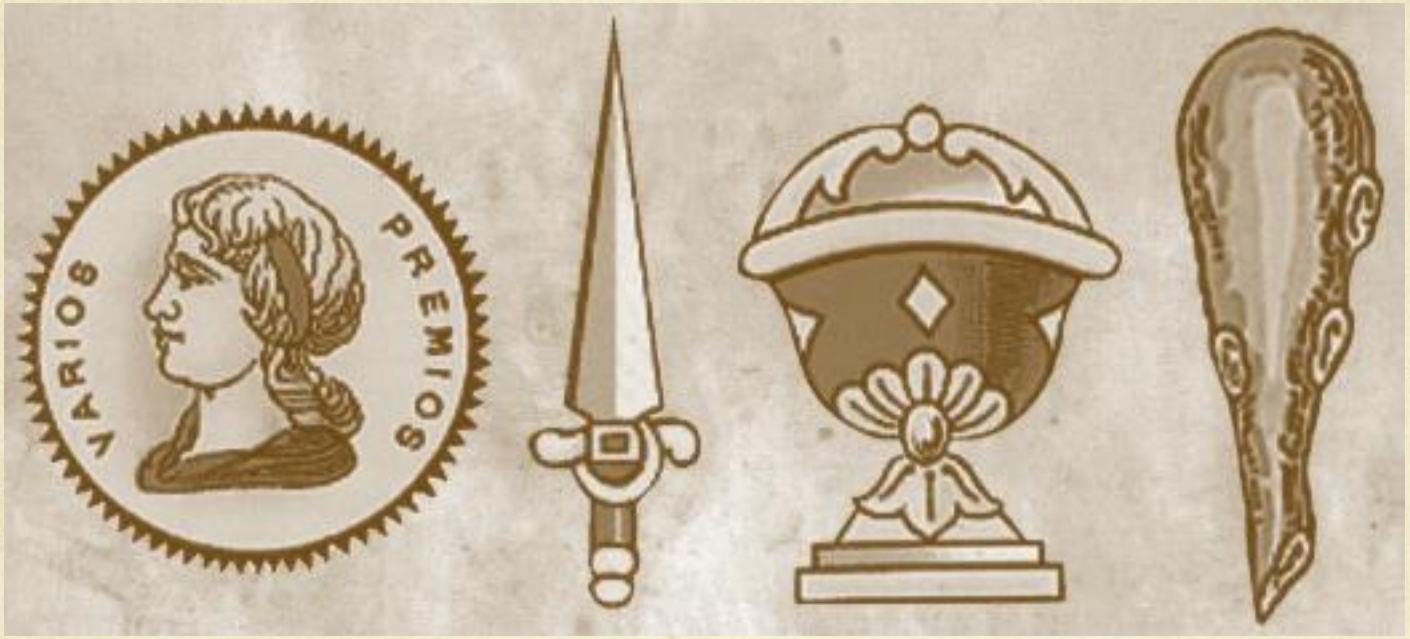
A[6]<A[5]	27 < 67	Sí hay intercambio
A[5]<A[4]	27 < 44	Sí hay intercambio
A[4]<A[3]	27 < 16	No hay intercambio

Sexto recorrido.

A[7]<A[6]	12 < 67	Sí hay intercambio
A[6]<A[5]	12 < 44	Sí hay intercambio
A[5]<A[4]	12 < 27	Sí hay intercambio
A[4]<A[3]	12 < 16	Sí hay intercambio
A[3]<A[2]	12 < 15	Sí hay intercambio
A[2]<A[1]	12 < 8	No hay intercambio

Séptimo recorrido.

A[8]<A[7]	35 < 67	Sí hay intercambio
A[7]<A[6]	35 < 44	Sí hay intercambio
A[6]<A[5]	35 < 27	No hay intercambio



6. Métodos de ordenamiento.

El algoritmo de ordenación utilizando el método de inserción directa es el siguiente:

```
FUNC insercion (n: Entero, A[]: Entero): DEV vacío
        cont, pos, aux,: ENTERO
        cont ← 2
        MIENTRAS cont < n HACER        ** Inicia ciclo general
                aux ← A[cont] y pos ← cont–1
                MIENTRAS pos > 1 AND aux < A[pos] HACER
                        A[pos+1] ← A[pos]
                        pos ← pos–1
                FIN_MIENTRAS
                A[pos+1] ← aux
        FIN_MIENTRAS
FIN_FUNC
```

El análisis de complejidad del algoritmo de inserción directa está dado, al igual que los anteriores, por el número de comparaciones que se realizan.

El mejor caso del algoritmo $\Omega(g(n))$ se presenta cuando el conjunto se encuentra ordenado, donde el número de comparaciones está dado por:

$$C_{min} = n-1$$

El peor caso del algoritmo se presenta cuando el conjunto está en orden inverso. En tal caso, el primer recorrido se realiza una comparación, el segundo dos comparaciones, el tercero tres comparaciones y así sucesivamente hasta n–1 comparaciones:

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

El número de comparaciones promedio, cuando los elementos se encuentrar de forma aleatoria, se puede calcular mediante la suma de las comparaciones mínimas y las máximas entre 2:

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

El número de movimientos para el mejor, el peor y el caso promedio está dado por:

$$M_{min} = 0$$

$$M_{max} = \frac{n * (n-1)}{2}$$

$$M_{med} = \frac{0 + \frac{n * (n-1)}{2}}{2}$$

Se desea ordenar un conjunto de 500 elementos, ¿cuál es el número de comparaciones y movimientos para el mejor, el peor y el caso base?

- Si el conjunto se encuentra ordenado:
  - C = 499
  - $M_{min} = 0$
- Si el conjunto se encuentra desordenado de forma aleatoria:
  - C = 62,624
  - $M_{med} = 62,375$
- Si el conjunto se encuentra desordenado:
  - C = 124,750
  - $M_{max} = 124,750$

6.2.4 Métodos por distribución

Los métodos de distribución son análogos al proceso de ordenamiento que emplea un clasificador de tarjetas, esto es, consiste en una colección de casilleros donde se depositan las tarjetas que coincidan con la marca del casillero.

El ordenamiento radix (radix sort) es el más representativo de los métodos de distribución. Es un algoritmo de ordenamiento estable que puede ser usado para ordenar elementos identificados por llaves (o claves) únicas. Cada llave debe ser una cadena o un número capaz de ser ordenada alfanuméricamente.

También es conocido como Clasificación por Urnas, debido que supone el almacenamiento de datos en cajas para construir conjuntos de elementos.

Los pasos que sigue este método son:

- Empezar en el dígito más significativo y avanzar por los dígitos menos significativos y hacer coincidir el dígito con los dígitos correspondientes en las urnas o casillas.
- Se extraen los elementos de las casillas y se repite el proceso hasta que el número de dígitos termina.

Ordenar el siguiente conjunto de forma ascendente utilizando el método de distribución radix.

Se realizan diversos montones de elementos. El número máximo de elementos es 10 [0–9]. Cada elemento se lee de derecha a izquierda iniciando por el dígito menos significativo (0).

A	31	41	81	54	23	33	64	98	93	88

0	1	2	3	4	5	6	7	8	9
	31		23	54				98	
	41		33	64				88	
	81		93						

Después se extraen los montones en orden, es decir, de la columna uno se estraen todos los elementos, de la columna dos se extraen todos los elementos y así hasta la columna n.

0	1	2	3	4	5	6	7	8	9
	31		23	54				98	
	41		33	64				88	
	81		93						

Se vuelven realizar montones de elementos con el dígito siguiente (1).

A	31	41	81	23	33	93	54	64	98	88

0	1	2	3	4	5	6	7	8	9
		23	31	41	54	64		81	93
			33					88	98

Se vuelven a extraer los montones en orden de la columna 1 a la columna n.

0	1	2	3	4	5	6	7	8	9
		23	31	41	54	64		81	93
			33					88	98

A

23	31	33	41	54	64	81	88	93	98

El tiempo total para el ordenamiento radix está dado por el número de elementos del conjunto multiplicado por el número de dígitos de las llaves. Por tanto, el algoritmo ordena una lista en tiempo lineal, es decir, O(n).

6.2.5 Métodos por intercalación

Este método consiste en unir dos o más grupos de elementos en un solo conjunto. Los conjuntos a unir deben estar previamente ordenados para poder realizar la intercalación. Por tanto, no se pueden intercalar conjuntos ordenados de manera inversa (uno ascendente y otro descendente) y, mucho menos, conjuntos desordenados.

Si se desea realizar la intercalación de un conjunto A con 10 elementos y un conjunto B con 15 elementos, el conjunto resultante C tendría 25 elementos.

Para llevar a cabo el proceso de forma ascendente, se elige el primer elemento del conjunto A y el primer elemento del conjunto B. El elemento con menor valor se almacena en el conjunto C y, de donde se tomó el elemento que se guardó, se obtiene el siguiente elemento y se siguen realizando las comparaciones y el almacenamiento de los elementos más pequeños en C.

Las comparaciones terminan cuando se agota uno de los conjuntos y, finalmente, los elementos del otro conjunto son pasados al conjunto C, con lo que termina el proceso.

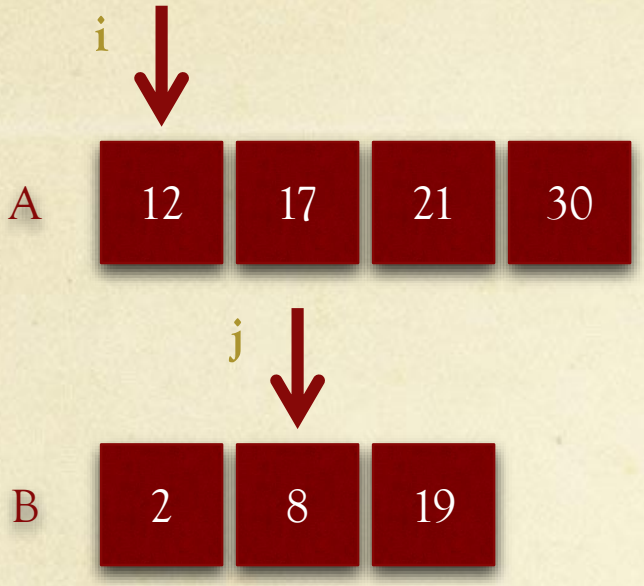
Si el número de conjuntos a unir es mayor a 2, se debe realizar el proceso repetidamente, ya sea de forma iterativa como recursiva.

Este proceso de intercalación o mezcla se puede aplicar tanto a conjuntos de elementos (arreglos o listas) como a archivos (secuenciales).

Ordenar los siguientes conjuntos de forma ascendente utilizando el método de intercalación.

12 < 2

```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```

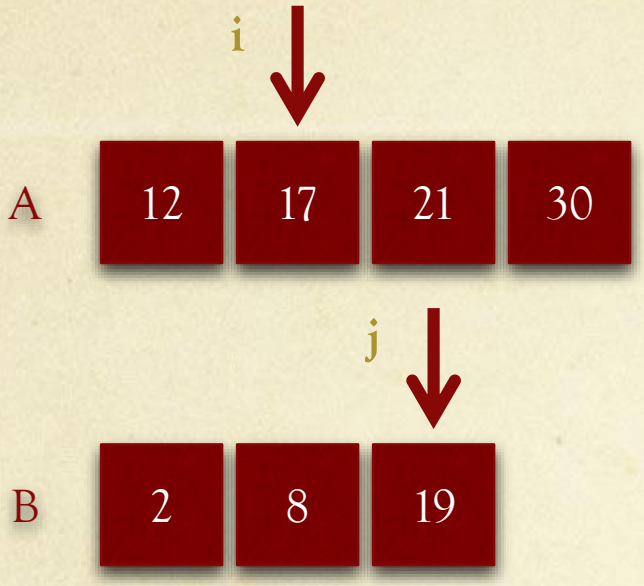


12 < 8

```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```

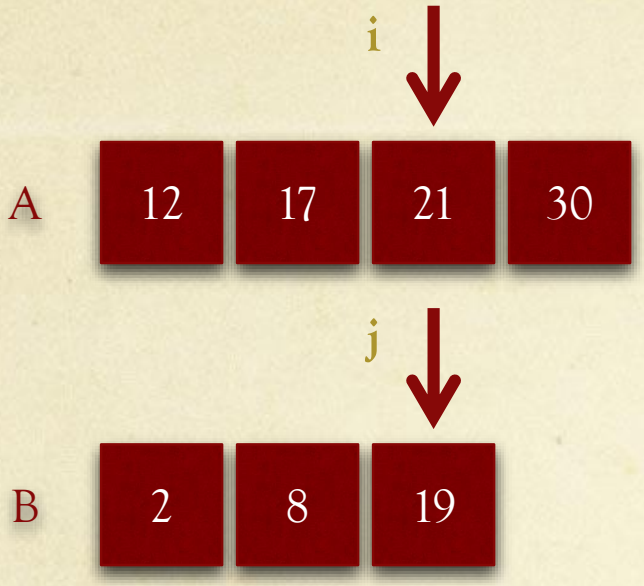
12 < 19

```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```



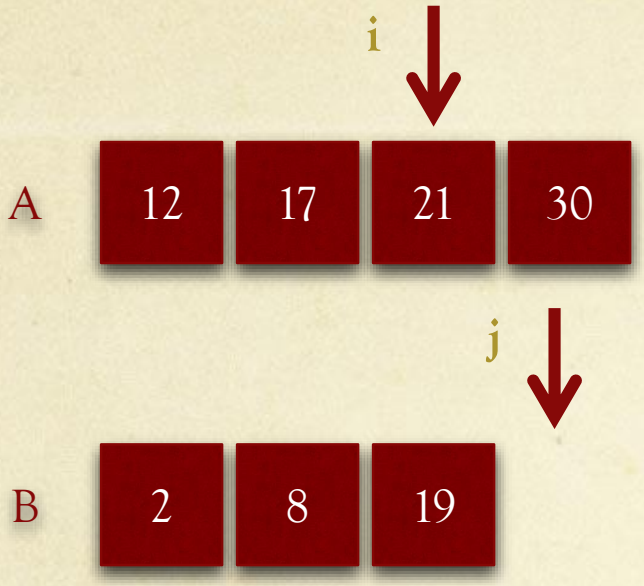
17 < 19

```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```

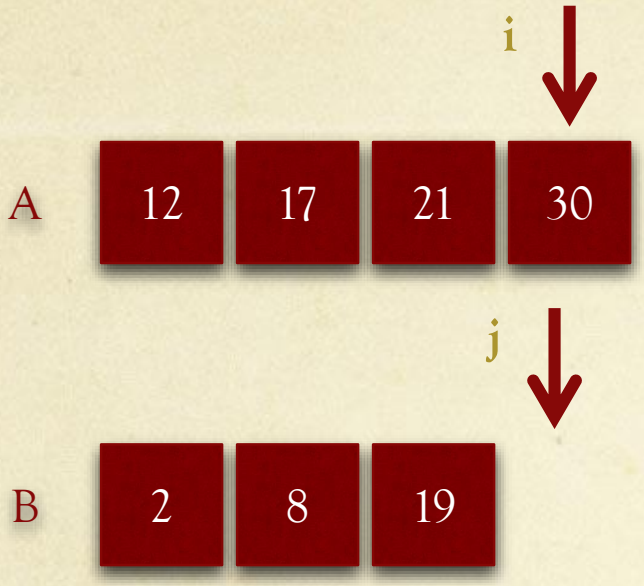


21 < 19

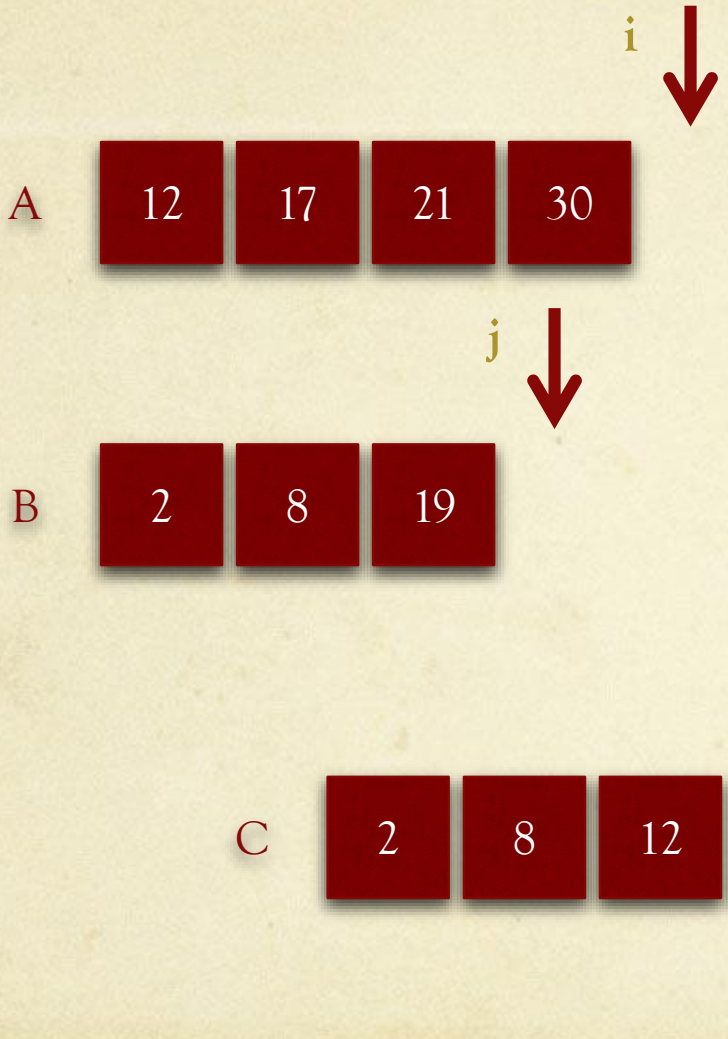
```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```




```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```




```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```




```
SI A[i] < B[j] ENTONCES
        C[k] ← A[i]
        i ← i + 1
FIN_SI
EN_CASO_CONTRARIO
        C[k] ← B[j]
        j ← j + 1
FIN_ECC
k ← k + 1
```

El análisis de complejidad del algoritmo de intercalación directa está dado por el producto del número de elementos de A y el número de elementos de B, $\Theta(n)$.

6.3 Ordenamientos externos.

6.3 Ordenamientos externos.

Cuando el conjunto de elementos que se desea clasificar es mayor a la capacidad de almacenamiento y, por ende, los datos se encuentran almacenados en la memoria secundaria, los ordenamientos se realizan directamente sobre los archivos.

Suponiendo que se desea ordenar un conjunto de 5000 registros (R_1, R_2, R_3, ..., R_{5000}), si las llaves de los registros solo pueden ser llevadas a la memoria principal en grupos de 1000, es posible dividir el conjunto en 5 subconjuntos, ordenarlos de forma independiente y, posteriormente, mezclarlos por intercalación.

Por tanto, los ordenamientos externos se basan en los ordenamientos internos vistos anteriormente.

El ordenamiento de archivos tiene tres fases:

1. Fase de ordenamiento interno, en la cual se ordenan los registros mediante varias ejecuciones distribuidas en dos o más dispositivos de almacenamiento.
2. Fase de intercalación, en la cual se combinan los subarchivos ordenados en una sola ejecución.
3. Fase de salida, en la cual se copia el archivo ordenado en su medio de almacenamiento final.

Prácticamente todas las técnicas de ordenamiento de archivos operan esencialmente de la misma manera.

El conjunto de registros por ordenar se divide en varias sublistas, cada una de las cuales se ordena mediante un método de ordenación interno. Cada sublista ordenada se escribe como un archivo secuencial. Estos archivos ordenados se intercalan para formar un solo archivo ordenado.

Por tanto, a estas técnicas de ordenamiento se les llama intercalación.

2.3.1 Métodos por polifase

Este método se apoya en los algoritmos de intercalación, por ello ha sido llamado ordenamiento de intercalación en polifase.

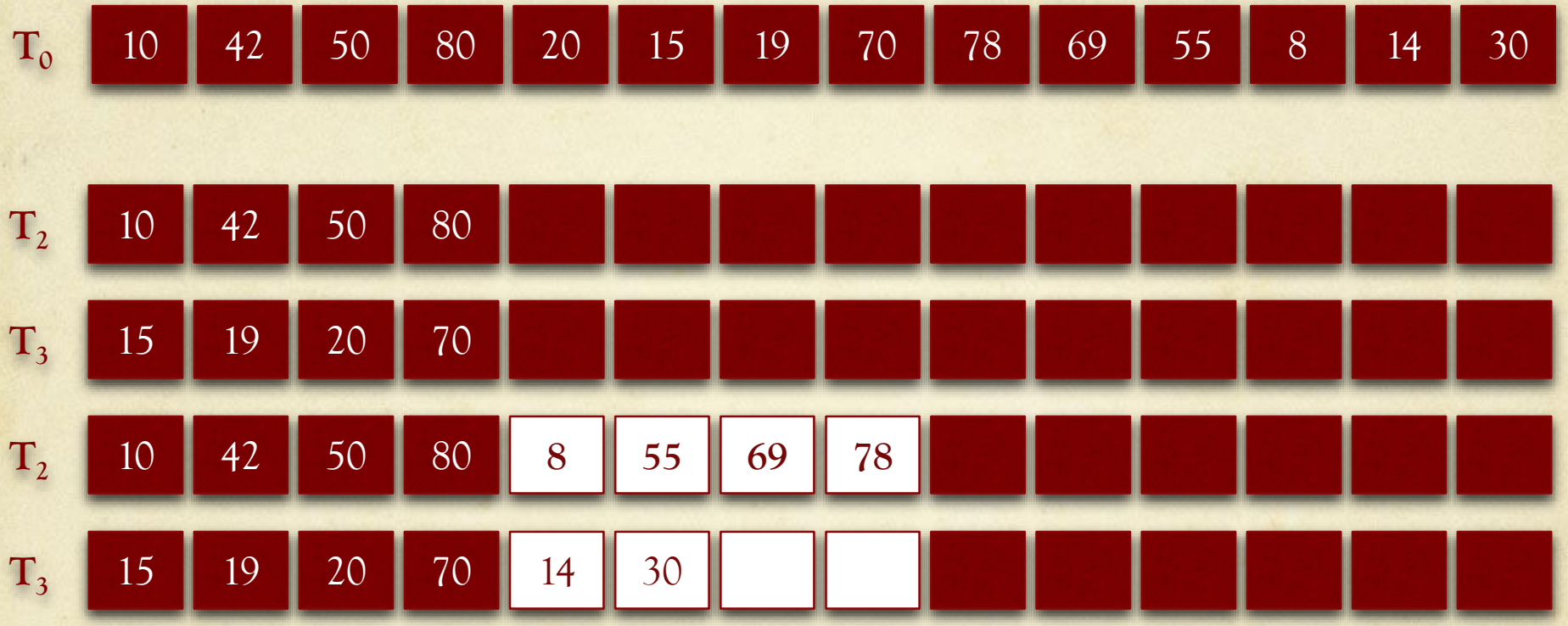
El método polifase es un tipo de intercalación desbalanceada, en la cual se utiliza un número constante de archivos auxiliares (T_1, T_2, T_3 y T_4), para almacenar la distribución de los elementos.

Se desea ordenar el siguiente conjunto de elementos que se encuentra en un archivo. El número máximo de llaves (m) que pueden ser llevadas a la memoria principal es 4.

La ordenación polifase sigue 2 fases. En la primera fase se construyen los arreglos ordenados siguiendo los siguientes pasos:

1. Se leen las m llaves posibles.
2. Se ordenan las m llaves por un método interno.
3. Las m llaves se colocan en los archivos auxiliares T_2 y T_3 de manera intercalada (una lectura en T_2 y la otra en T_3).

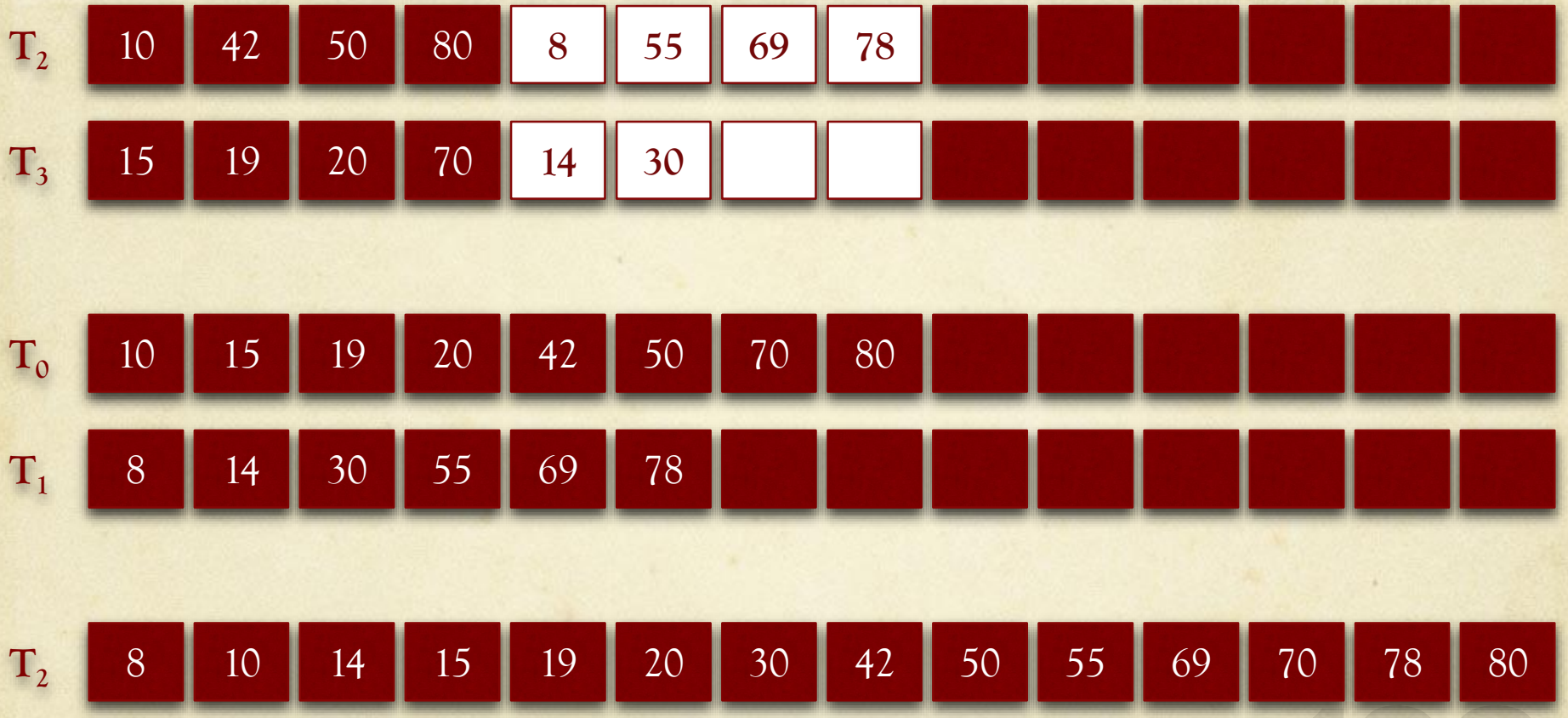
Primera fase



En la segunda fase se intercalas los elementos de los archivos auxiliares hasta formar un solo archivo ordenado, es decir:

1. Se intercalan los primeros m elementos del primer archivo con los primeros elementos del segundo archivo y se guardan en otro archivo auxiliar T_0.
2. Se intercalan el siguiente bloque de m elementos del primer archivo con el siguiente bloque de m elementos del segundo archivo y se guardan en otro archivo auxiliar T_1.
3. El proceso se repite hasta que los datos se agoten. El resultado de la intercalación se guarda en los primeros archivos auxiliares.

Segunda fase



El número total de recorridos (iteraciones) y comparaciones que se realizan en el método polifase, respectivamente, son:

$$R = \log r + i$$

$$C = n \log m + \sum_{i=1}^{\log r} n - \left(\frac{r}{2^i} \right)$$

Donde:
n = número de elementos
m = número de elementos por arreglo
r = n/m

2.3.2 Métodos por cascada

El método de cascada, al igual que el método polifase, se inicia con una distribución de los elementos de el o los archivos. La diferencia con el método anterior es que la distribución de los elementos en cada archivo sigue una secuencia que involucra a los números de Fibonacci. La distribución se realiza sobre seis archivos auxiliares (T_0, T_1, T_2, T_3, T_4, y T_5).

2.3.3 Métodos oscilantes

Los métodos de ordenamiento externo anteriores tienen dos fases muy marcadas, una fase de distribución y una fase de intercalación, con la característica de que para pasar a la fase de intercalación se debe terminar antes la fase de distribución.

El método de ordenamiento oscilante propone un algoritmo que combina la distribución y la mezcla de arreglos auxiliares. La distribución y mezcla se realiza sobre 5 archivos auxiliares.

2.3.4 Métodos por distribución

El método de distribución para ordenamientos externos funciona de manera similar al método de ordenamiento interno.

Este método también es llamado ordenamiento digital. Una característica particular es que se opone a los conceptos de intercalación utilizados en otros ordenamientos externos.

Ordenar un conjunto de elementos numéricos base 4, es decir, expresiones que solo ocupan dígitos del 0 al 3.

Archivo original

1023	0122	1131	3123	0012	0132	2013	1110

Se realizan 4 montones de elementos [0–4]. Cada elemento se lee de derecha a izquierda iniciando por el dígito más significativo (0).

Archivo original

1023	0122	1131	3123	0012	0132	2013	1110

0	1	2	3
1110	1131	0122	1023
		0012	3123
		0132	2013

Después se extraen los montones en orden, es decir, de la columna uno se estraen todos los elementos, de la columna dos se extraen todos los elementos y así hasta la columna n.

0	1	2	3
1110	1131	0122	1023
		0012	3123
		0132	2013

Archivo auxiliar

1110	1131	0122	0012	0132	1023	3123	2013

Se repite el proceso ahora con el archivo auxiliar y el siguiente dígito (1).

Archivo auxiliar

1110	1131	0122	0012	0132	1023	3123	2013

0	1	2	3
	1110	0122	1131
	0012	1023	0132
	2013	3123	

Se vuelven a extraer los elementos columna por columna y en orden ascendente.

0	1	2	3
	1110	0122	1131
	0012	1023	0132
	2013	3123	

Archivo auxiliar

1110	0012	2013	0122	1023	3123	1131	0132

Se repite el proceso ahora con el archivo auxiliar y el siguiente dígito (1).

Archivo auxiliar

1110	0012	2013	0122	1023	3123	1131	0132

0	1	2	3
0012	1110		
2013	0122		
1023	3123		
	1131		
	0132		

Se vuelven a extraer los elementos columna por columna y en orden ascendente.

0	1	2	3
0012	1110		
2013	0122		
1023	3123		
	1131		
	0132		

Archivo auxiliar

0012	2013	1023	1110	0122	3123	1131	0132

Se repite el proceso ahora con el archivo auxiliar y el siguiente dígito (1).

Archivo auxiliar

0012	2013	1023	1110	0122	3123	1131	0132

0	1	2	3
0012	1023	2013	3123
0122	1110		
0132	1131		

Se vuelven a extraer los elementos columna por columna y en orden ascendente.

0	1	2	3
0012	1023	2013	3123
0122	1110		
0132	1131		

Archivo final

0012	0122	0132	1023	1110	1131	2013	3123

En este caso, cada columna de la matriz representa un archivo auxiliar. Si se desean reducir el número de archivos se pueden transformar los elementos del archivo a base 2.

6.4 Archivos auxiliares almacenados en disco.

6.4 Archivos auxiliares almacenados en disco.

Los discos magnéticos permiten el manejo de archivos auxiliares con acceso directo a la información y buena velocidad de transmisión (para ser un almacenamiento secundario).

Cuando se utilizan discos magnéticos para el almacenamiento de arreglos auxiliares se debe optimizar el acceso a los datos haciendo más eficiente el código y intentando realizar la menor cantidad de accesos posibles.

6 Métodos de ordenamiento

Objetivo: Aplicar los métodos internos y externos más importantes para efectuar ordenamientos en la computadora. Diseñar y aplicar algoritmos.

6.1 Generalidades.
6.2 Ordenamientos internos.
6.3 Ordenamientos externos.
6.4 Archivos auxiliares almacenados en disco.