

# Métodos de ordenación interna



Eva Lucrecia Gibaja Galindo

Dpto. Informática y Análisis Numérico

# Introducción

- **Ordenar:** Proceso de reorganizar un conjunto dado, de objetos similares, en una secuencia especificada
- **Objetivo fundamental:** Facilitar la **búsqueda** de cualquier elemento de dicho conjunto (es mucho más fácil en un conjunto ordenado)
- Las técnicas de ordenación son un ejemplo muy ilustrativo sobre las ideas implicadas en el análisis de rendimiento de algoritmos que permitan la elección adecuada de los mismos

# Introducción

## Definición formal del problema:

- Se toman  $n$  elementos  $R_1, R_2, R_3, \dots, R_n$ , para ordenarlos
- Cada  $R_i$  se denomina **registro** y al conjunto de los  $n$  registros, **fichero**
- Cada registro  $R_i$ , tiene una **clave**  $K_i$ , que determina el orden
  - El registro puede contener información adicional que no tiene ningún efecto en el proceso de ordenación y permanece en el registro
- Se especifica entre las claves una **relación de orden**  $<$ , de forma que, para tres valores  $a, b, c$  se cumplen las siguientes condiciones:
  1. una de las posibilidades  $a < b, a = b, b < a$  es cierta
  2. si  $a < b$  y  $b < c$  entonces  $a < c$  (propiedad transitiva)
- **Ordenación:** Determinar una permutación  $p_1, p_2, \dots, p_n$  de los registros que coloque las claves en orden no decreciente.

$$K(p_1) \leq K(p_2) \leq K(p_3) \leq \dots \leq K(p_n)$$

# Introducción

- **Ordenación estable:** Los registros con igual clave, mantienen su orden relativo inicial
- **Tipos de ordenación en función de la memoria:**
  - **Ordenación interna (vectores).** Se almacenan en memoria. Más rápidos
  - **Ordenación externa (ficheros).** Se ubican en dispositivos de almacenamiento externo. Más lentos
- **Ordenación “*in situ*”.** Las permutaciones de los elementos a ordenar se realizan utilizando el espacio ocupado por estos
- **Tiempo de ordenación:**
  - **Comparaciones** (función del número de elementos a ordenar,  $n$ )
  - **Desplazamientos** (función del número de elementos a ordenar,  $n$ )
- **Métodos de orden  $n^2$  (métodos directos)**
  1. Son una introducción a la problemática y su estudio es útil
  2. Los programas para estos métodos son más cortos y fáciles de entender
  3. Son más rápidos cuando el número de elementos es suficientemente pequeño  
No deben usarse para números grandes de elementos



# Ordenación por selección

- La idea de este tipo de métodos se basa en las siguientes operaciones:
  1. Seleccionar el elemento más pequeño
  2. Intercambiarlo con el primero,  $v_1$
  3. A continuación se repiten estas operaciones con los  $n-1$ ,  $n-2$ , ..., elementos restantes, hasta que quede un único elemento, el mayor
- Los métodos basados en esta idea requieren la disponibilidad de todos los elementos antes de empezar el proceso de ordenación
- Esencialmente la idea es opuesta a la de inserción

# Ordenación por selección

```

algoritmo selección-directa(n; v; )
inicio
  para i de 1 a n-1
    menor  $\leftarrow$  i //Posición del menor
    para k de i+1 a n
      si v[k] < v[menor] entonces
        menor  $\leftarrow$  k
      fin-si
    fin-para

auxiliar  $\leftarrow$  v[menor]
      v[menor]  $\leftarrow$  v[i]
      v[i]  $\leftarrow$  auxiliar

fin-para
fin
  
```



44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo Selección

i=1

44	55	12	42	94	18	<b>6</b>	67
----	----	----	----	----	----	----------	----

i=2

6	55	<b>12</b>	42	94	18	44	67
---	----	-----------	----	----	----	----	----

i=3

6	12	55	42	94	<b>18</b>	44	67
---	----	----	----	----	-----------	----	----

i=4

6	12	18	<b>42</b>	94	55	44	67
---	----	----	-----------	----	----	----	----

i=5

6	12	18	42	94	55	<b>44</b>	67
---	----	----	----	----	----	-----------	----

i=6

6	12	18	42	44	<b>55</b>	94	67
---	----	----	----	----	-----------	----	----

i=7

6	12	18	42	44	55	94	<b>67</b>
---	----	----	----	----	----	----	-----------

6	12	18	42	44	55	67	94
---	----	----	----	----	----	----	----

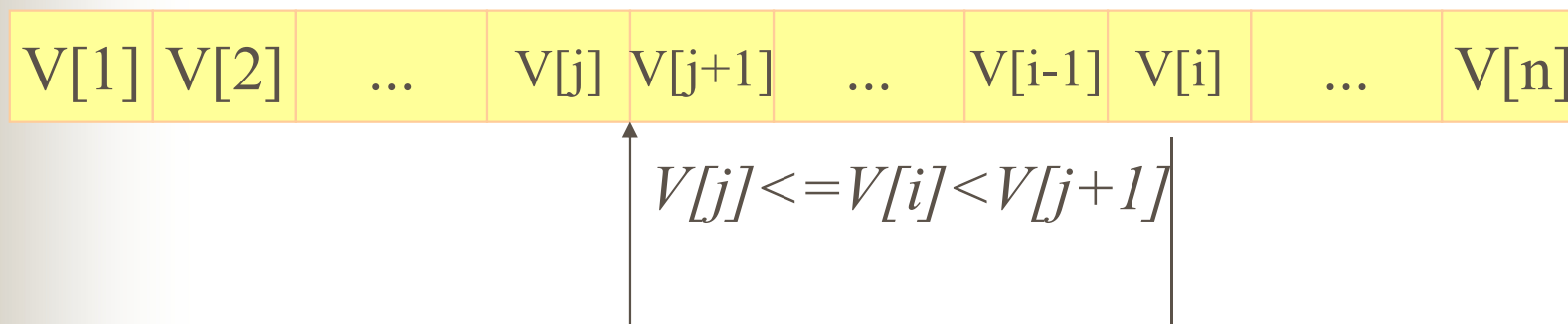
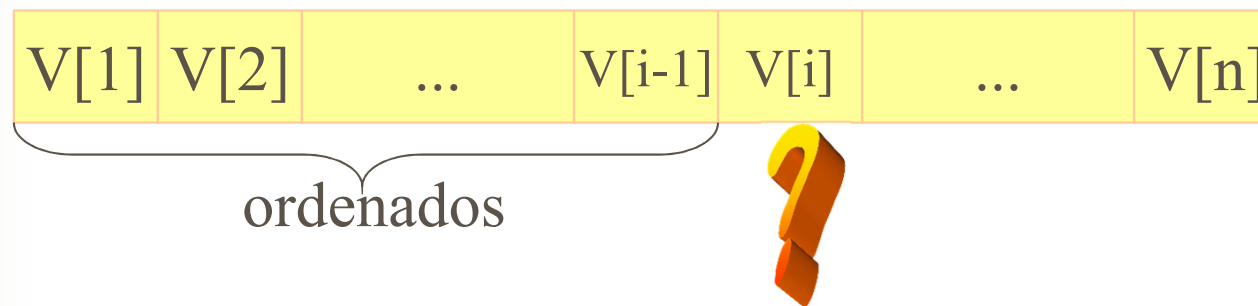
# Ordenación por inserción

## Inserción directa

- Supongamos que  $j < i \leq n$  y que se han ordenado ya los  $i-1$  elementos ya tratados
- Se compara el nuevo elemento,  $v_i$  con  $v_{i-1}$ ,  $v_{i-2}$ , ..., uno a uno, hasta encontrar dos elementos consecutivos  $v_j$  y  $v_{j+1}$  entre los cuales insertar el elemento  $v_i$
- Se desplazan un espacio los elementos  $v_{j+1}$ ,  $v_{j+2}$ , ...,  $v_{i-1}$ , y se coloca el nuevo elemento en la posición  $j+1$



# Ordenación por inserción



# Ordenación por inserción

**algoritmo inserción\_directa (n; v; )**

**inicio**

**para i de 2 a n hacer**

**auxiliar**  $\leftarrow v[i]$  //Elemento que hay que insertar

**j**  $\leftarrow i - 1$  //Tope de los ordenados

**mientras j > 0 Y  $v[j] > \text{auxiliar}$**

**$v[j+1] \leftarrow v[j]$**  //Desplaza

**j  $\leftarrow j - 1$**

**fin-mientras**

**$v[j+1] \leftarrow \text{auxiliar}$**

**fin-para**

**fin**

**Ojo!! En este tema los vectores  
comienzan en la posición 1**

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo inserción directa

i=2

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

i=3

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

auxiliar=12

44	55	55	42	94	18	6	67
----	----	----	----	----	----	---	----

j=2

44	44	55	42	94	18	6	67
----	----	----	----	----	----	---	----

j=1

12	44	55	42	94	18	6	67
----	----	----	----	----	----	---	----

j=0

i=4

12	44	55	42	94	18	6	67
----	----	----	----	----	----	---	----

auxiliar=42

12	44	55	55	94	18	6	67
----	----	----	----	----	----	---	----

j=3

12	44	44	55	94	18	6	67
----	----	----	----	----	----	---	----

j=2

12	42	44	55	94	18	6	67
----	----	----	----	----	----	---	----

j=1  $V[j] < 42$

# Ejemplo inserción directa

i=5

12	42	44	55	94	18	6	67
----	----	----	----	----	----	---	----

auxiliar=94

$V[j] < 94$

i=6

12	42	44	55	94	18	6	67
----	----	----	----	----	----	---	----

auxiliar=18

12	42	44	55	94	94	6	67
----	----	----	----	----	----	---	----

j=5

12	42	44	55	55	94	6	67
----	----	----	----	----	----	---	----

j=4

12	42	44	44	55	94	6	67
----	----	----	----	----	----	---	----

j=3

12	42	42	44	55	94	6	67
----	----	----	----	----	----	---	----

j=2

12	18	42	44	55	94	6	67
----	----	----	----	----	----	---	----

j=1



# Ejemplo inserción directa

i=7

12	18	42	44	55	94	6	67
----	----	----	----	----	----	---	----

 auxiliar=6

12	18	42	44	55	94	94	67
----	----	----	----	----	----	----	----

 j=6

12	18	42	44	55	55	94	67
----	----	----	----	----	----	----	----

 j=5

12	18	42	44	44	55	94	67
----	----	----	----	----	----	----	----

 j=4

12	18	42	42	44	55	94	67
----	----	----	----	----	----	----	----

 j=3

12	18	18	42	44	55	94	67
----	----	----	----	----	----	----	----

 j=2

12	12	18	42	44	55	94	67
----	----	----	----	----	----	----	----

 j=1

6	12	18	42	44	55	94	67
---	----	----	----	----	----	----	----

i=8

6	12	18	42	44	55	94	67
---	----	----	----	----	----	----	----

 auxiliar=67

6	12	18	42	44	55	94	94
---	----	----	----	----	----	----	----

 j=7

6	12	18	42	44	55	67	94
---	----	----	----	----	----	----	----

 j=6  $V[j] < 67$

# Ordenación por inserción

algoritmo inserción\_binaria (n; v; )

inicio

para i de 2 a n hacer

auxiliar  $\leftarrow v[i]$  //Elemento a insertar

izquierda  $\leftarrow 1$  // Posicion de insercion

derecha  $\leftarrow i - 1$  //Parte del vector parcialmente ordenada

mientras izquierda  $\leq$  derecha

mitad  $\leftarrow (izquierda + derecha) \div 2$

si auxiliar  $< v[mitad]$  entonces

derecha  $\leftarrow mitad - 1$

sino

izquierda  $\leftarrow mitad + 1$

fin-si

fin-mientras

para j de i-1 a izquierda paso -1 // desplazamientos

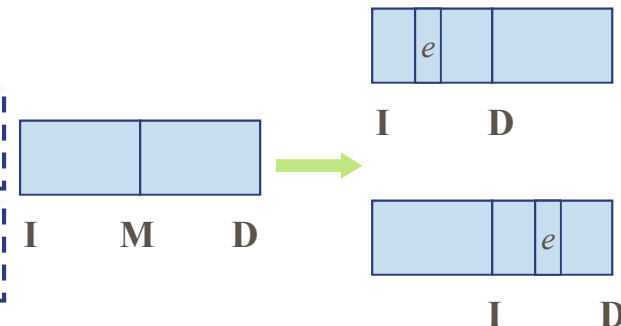
$v[j+1] \leftarrow v[j]$

fin-para

$v[izquierda] \leftarrow auxiliar$  //izquierda indica la posición donde debe ir el elemento

fin-para

fin



44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo inserción binaria

i=2

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

D

44 <sub>m</sub>	55	12	42	94	18	6	67
-----------------	----	----	----	----	----	---	----

D I

i=3

44 <sub>m</sub>	55	12	42	94	18	6	67
-----------------	----	----	----	----	----	---	----

I D

44 <sub>m</sub>	55	12	42	94	18	6	67
-----------------	----	----	----	----	----	---	----

D I

44	55	55	42	94	18	6	67
----	----	----	----	----	----	---	----

j=2

44	44	55	42	94	18	6	67
----	----	----	----	----	----	---	----

j=1

12	44	55	42	94	18	6	67
----	----	----	----	----	----	---	----

v[izd]=12

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo inserción binaria

i=4

12	44 <sub>m</sub>	55	42	94	18	6	67
----	-----------------	----	----	----	----	---	----

I

D

12	44 <sub>m</sub>	55	42	94	18	6	67
----	-----------------	----	----	----	----	---	----

DI

12 <sub>m</sub>	44	55	42	94	18	6	67
-----------------	----	----	----	----	----	---	----

D

I

12	44	55	55	94	18	6	67
----	----	----	----	----	----	---	----

j=3

12	44	44	55	94	18	6	67
----	----	----	----	----	----	---	----

j=2

12	42	44	55	94	18	6	67
----	----	----	----	----	----	---	----

v[izd]=42



44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo inserción binaria

i=7

12	18	42 <sub>m</sub>	44	55	94	6	67
----	----	-----------------	----	----	----	---	----

I

D

12 <sub>m</sub>	18	42	44	55	94	6	67
-----------------	----	----	----	----	----	---	----

I

D

12 <sub>m</sub>	18	42	44	55	94	6	67
-----------------	----	----	----	----	----	---	----

D

I

12	18	42	44	55	94	94	67	j=6
12	18	42	44	55	55	94	67	j=5
12	18	42	44	44	94	94	67	j=4
12	18	42	42	55	94	94	67	j=3
12	18	18	42	55	94	94	67	j=2
12	12	18	42	55	94	94	67	j=1
6	12	18	42	55	94	94	67	v[izd]=6 17

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo inserción binaria

i=8

6	12	18	42 <sub>m</sub>	44	55	94	67
---	----	----	-----------------	----	----	----	----

I

D

6	12	18	42 <sub>m</sub>	44	55	94	67
---	----	----	-----------------	----	----	----	----

I

D

6	12	18	42	44	55 <sub>m</sub>	94	67
---	----	----	----	----	-----------------	----	----

ID

6	12	18	42	44	55	94 <sub>m</sub>	67
---	----	----	----	----	----	-----------------	----

D

I

6	12	18	42	44	55	94	94	j=7
---	----	----	----	----	----	----	----	-----

6	12	18	42	44	55	67	94	v[izd]=67
---	----	----	----	----	----	----	----	-----------

# Ordenación por inserción

```

algoritmo inserción_shell(n; v; )
inicio
  distancia  $\leftarrow$  n
  repetir
    distancia  $\leftarrow$  distancia div 2
  repetir
    ordenado  $\leftarrow$  verdad
    i  $\leftarrow$  1
    repetir
      si  $v[i] > v[i+distancia]$  entonces
        auxiliar  $\leftarrow v[i]$ 
         $v[i] \leftarrow v[i+distancia]$ 
         $v[i+distancia] \leftarrow$  auxiliar
        ordenado  $\leftarrow$  falso
      fin-si
      i  $\leftarrow$  i + 1
    hasta que i+d > n
  hasta que ordenado
  hasta distancia=1
fin
  
```

*intercambio*

# Ordenación por inserción

## *Inserción shell*

- La secuencia de incrementos tomada (las sucesivas mitades), no tiene por qué ser única:
  - Se puede usar cualquier secuencia con tal de que el incremento final sea 1
  - En particular no se sabe qué secuencia de incrementos produce el mejor resultado
  - Knuth recomienda:
    - 1, 4, 13, 40, 121
    - 1, 3, 5, 7, 15, 31



5	4	6	8	9	5	7	3	3
---	---	---	---	---	---	---	---	---

# Ejemplo inserción Shell

d=4

5	4	6	8	9	5	7	3	3
---	---	---	---	---	---	---	---	---

5	4	6	3	3	5	7	8	9
---	---	---	---	---	---	---	---	---

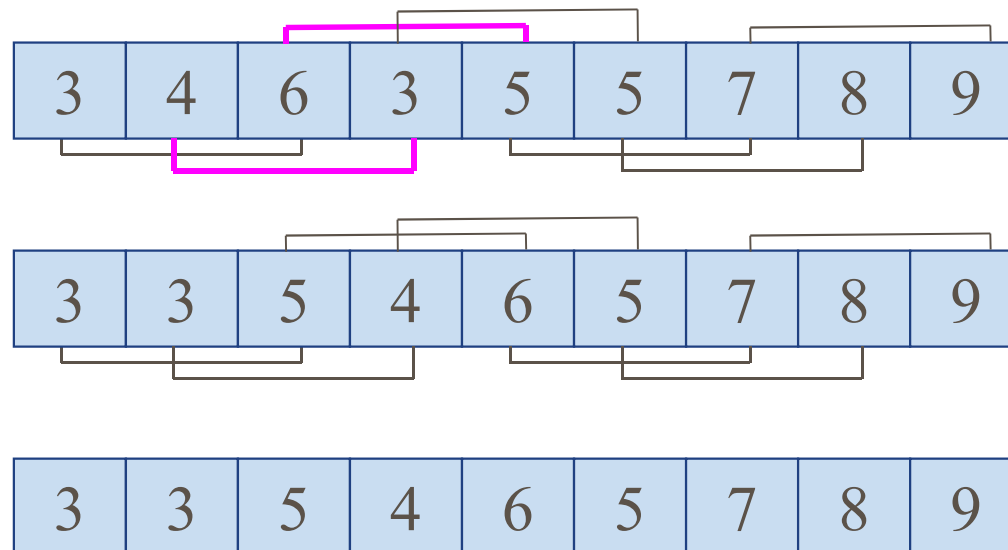
3	4	6	3	5	5	7	8	9
---	---	---	---	---	---	---	---	---

*Ordenados los pares de distancia 4*

5	4	6	8	9	5	7	3	3
---	---	---	---	---	---	---	---	---

# Ejemplo inserción Shell

d=2



*Ordenados los pares de distancia 2*

5	4	6	8	9	5	7	3	3
---	---	---	---	---	---	---	---	---

# Ejemplo inserción Shell

d=1

3	3	5	4	6	5	7	8	9
---	---	---	---	---	---	---	---	---

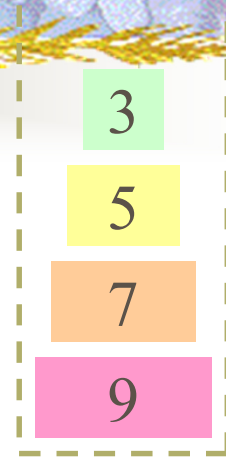
3	3	4	5	5	6	7	8	9
---	---	---	---	---	---	---	---	---

*Ordenados los pares de distancia 1*

# Ordenación por intercambio

## Intercambio directo o método de la burbuja

- La base de este método es la comparación e intercambio de pares de elementos adyacentes, hasta que todos los elementos del vector están ordenados
- Para ello, se realizan varias pasadas sobre el vector, moviendo en cada una de ellas el elemento mínimo, hasta el extremo izquierdo del vector
- Si se observase el vector como si estuviera en posición vertical en vez de horizontal, los elementos se asemejan a burbujas en un depósito de agua, con pesos acordes al valor de los mismos; de cada pasada sobre el vector resulta la ascensión a una burbuja hasta el nivel del peso que le corresponde





# Ordenación por intercambio

algoritmo burbuja(n; v; )

inicio

para i de 2 a n

para j de n a i inc -1

si  $v[j-1] > v[j]$  entonces

auxiliar  $\leftarrow v[j-1]$

$v[j-1] \leftarrow v[j]$

$v[j] \leftarrow$  auxiliar

fin-si

fin-para

fin-para

fin

*intercambio*

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo burbuja

i=2

44	55	12	42	94	18	6	67	j=8
44	55	12	42	94	18	6	67	j=7
44	55	12	42	94	6	18	67	j=6
44	55	12	42	6	94	18	67	j=5
44	55	12	6	42	94	18	67	j=4
44	55	6	12	42	94	18	67	j=3
44	6	55	12	42	94	18	67	j=2
6	44	55	12	42	94	18	67	

# Ejemplo burbuja

$i=3$

6	44	55	12	42	94	18	<b>67</b>
---	----	----	----	----	----	----	-----------

$j=8$

6	44	55	12	42	94	<b>18</b>	67
---	----	----	----	----	----	-----------	----

$j=7$

6	44	55	12	42	<b>18</b>	94	67
---	----	----	----	----	-----------	----	----

$j=6$

6	44	55	12	<b>18</b>	42	94	67
---	----	----	----	-----------	----	----	----

$j=5$

6	44	55	<b>12</b>	18	42	94	67
---	----	----	-----------	----	----	----	----

$j=4$

6	44	<b>12</b>	55	18	42	94	67
---	----	-----------	----	----	----	----	----

$j=3$

6	<b>12</b>	44	55	18	42	94	67
---	-----------	----	----	----	----	----	----

# Ejemplo burbuja

i=4

6	12	44	55	18	42	94	<b>67</b>
6	12	44	55	18	42	<b>67</b>	94
6	12	44	55	18	<b>42</b>	67	94
6	12	44	55	<b>18</b>	42	67	94
6	12	44	<b>18</b>	55	42	67	94
6	12	<b>18</b>	44	55	42	67	94

j=8

j=7

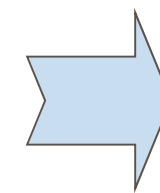
j=6

j=5

j=4

# Ejemplo burbuja

i=4	6	12	18	44	55	42	67	94
i=5	6	12	18	42	44	55	67	94
i=6	6	12	18	42	44	55	67	94
i=7	6	12	18	42	44	55	67	94
i=8	6	12	18	42	44	55	67	94



*No se han  
producido  
cambios*



# Ordenación por intercambio

## ■ Mejoras al algoritmo de la burbuja:

1. Las tres últimas pasadas no tienen efecto, por estar ya estos ordenados
  - Controlar que termine la ordenación al realizar una última pasada sin intercambios
2. Una burbuja mal situada en el extremo pesado del vector se sitúa en la posición correcta en una sola pasada. Un elemento mal colocado en el extremo ligero del vector, se hunde hacia su posición correcta a un ritmo de una posición por pasada
  - Esto sugiere la alternancia en la dirección de dos pasadas consecutivas

# Ordenación por intercambio

3. Todos los pares de elementos adyacentes por debajo de la posición del último intercambio realizado están ya ordenados
  - Las pasadas siguientes pueden terminarse en esta posición, sin tener que llegar al límite inferior,  $i$

	2	5	7	18	105	94	42	12	
i=2	2	5	7	18	105	94	12	42	j=8
	2	5	7	18	105	12	94	42	j=7
	2	5	7	18	12	105	94	42	j=6
	2	5	7	12	18	105	94	42	j=5

↑

# Ordenación por intercambio

algoritmo sacudida(n; v; )

inicio

izquierda  $\leftarrow 2$

derecha  $\leftarrow n$

k  $\leftarrow n$  //Posición del último intercambio

repetir

para j de derecha a izquierda paso -1



si  $v[j-1] > v[j]$  entonces

auxiliar  $\leftarrow v[j-1]$  //Intercambio

$v[j-1] \leftarrow v[j]$

$v[j] \leftarrow auxiliar$

k  $\leftarrow j$  //Actualiza posición del último cambio

fin-si

fin-para

izquierda  $\leftarrow k+1$

para j de izquierda a derecha



si  $v[j-1] > v[j]$  entonces

auxiliar  $\leftarrow v[j-1]$  //Intercambio

$v[j-1] \leftarrow v[j]$

$v[j] \leftarrow auxiliar$

k  $\leftarrow j$  //Actualiza posición del último cambio

fin-si

fin-para

derecha  $\leftarrow k-1$

hasta que izquierda > derecha

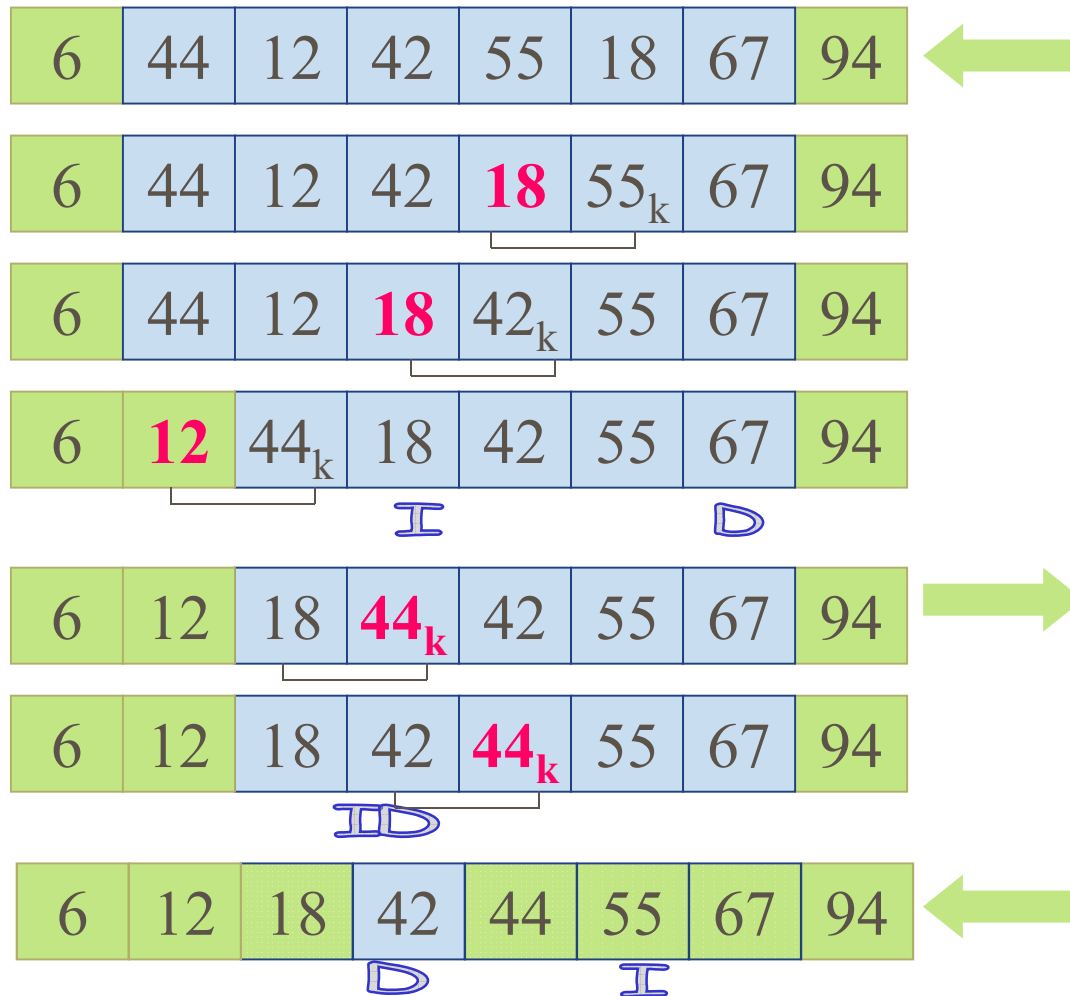
fin

44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

## Ejemplo sacudida



# Ejemplo sacudida





# Métodos sofisticados

- **Ordenación por montículo (*heapsort*).** No lo vamos a ver
- **Ordenación por mezcla (*mergesort*).** No lo vamos a ver
- **Ordenación rápida (*quicksort*)**

# Métodos sofisticados

## Ordenación rápida (*quicksort*)

- Se toma arbitrariamente un elemento ( $x$ ) del vector:
  1. Se recorre el vector de izquierda a derecha hasta encontrar un elemento  $v_i$ , que sea mayor que  $x$
  2. Se recorre el vector de derecha a izquierda hasta encontrar un elemento  $v_j$  que sea menor que  $x$
  3. Se intercambian  $v_i$  y  $v_j$ , y se continúa este proceso hasta que los recorridos se encuentran
- Como resultado:
  - Tenemos el vector partido en dos, una parte izquierda con elementos menores que  $x$  y la derecha con los mayores que  $x$
  - Tras aplicar el algoritmo de partición, el elemento  $x$  ocupará su posición correcta

# Métodos sofisticados

## Ordenación rápida (*quicksort*)

- Caso más favorable: Se selecciona la mediana como elemento de partición en todos los casos:
  - Cada proceso de partición divide al vector en dos partes iguales, y el número de pasadas para ordenarlo es  $\log n$
  - El total de comparaciones resultantes es del orden  $n \log n$
  - Evidentemente la posibilidad de que esto ocurra es prácticamente nula. No obstante el rendimiento medio del algoritmo sigue siendo proporcional a  $n \log n$
- Caso más desfavorable: Se selecciona como elemento de partición el mayor del subvector analizado
- El principal inconveniente es su bajo rendimiento para valores pequeños de  $n$

**algoritmo quicksort (izquierda, derecha; v;)**

**inicio**

**$i \leftarrow \text{izquierda}$**

**$j \leftarrow \text{derecha}$**

**$\text{mitad} \leftarrow v[(\text{izquierda} + \text{derecha}) \text{ div } 2]$**

**repetir**

**mientras  $v[i] < \text{mitad}$**

**$i \leftarrow i + 1$**

**fin-mientras**

**mientras  $v[j] > \text{mitad}$**

**$j \leftarrow j - 1$**

**fin-mientras**

**si  $i \leq j$  entonces**

**$\text{auxiliar} \leftarrow v[i]$**

**$v[i] \leftarrow v[j]$**

**$v[j] \leftarrow \text{auxiliar}$**

**$i \leftarrow i + 1$**

**$j \leftarrow j - 1$**

**fin-si**

**hasta que  $i > j$**

**si  $\text{izquierda} < j$  entonces**

**quicksort ( $\text{izquierda}, j, v$ )**

**fin-si**

**si  $i < \text{derecha}$  entonces**

**quicksort ( $i, \text{derecha}, v$ )**

**fin-si**

**fin**

*Se intercambia el pivote y por tanto  $i++$ ,  $j++$  por lo que el pivote no se incluye en las llamadas recursivas*

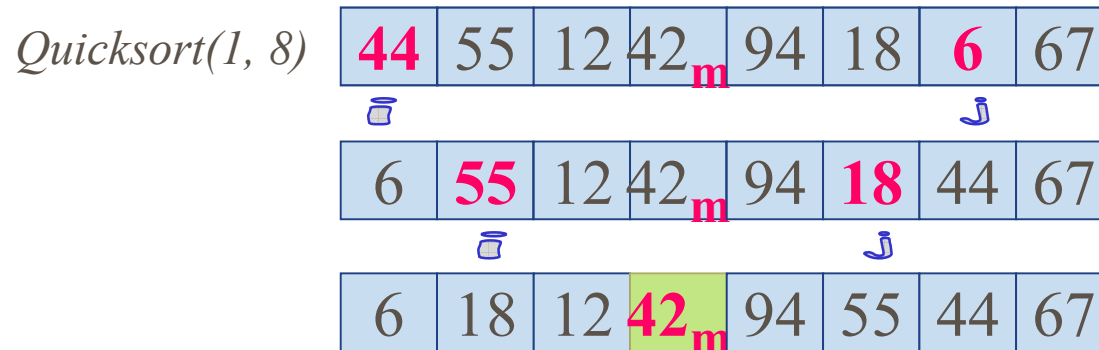
**intercambio**

*Si hay un solo elemento, para la recursividad*

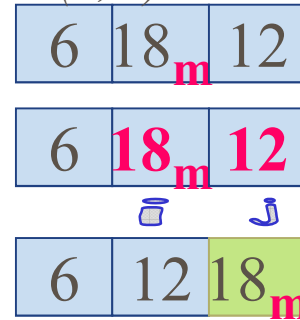


44	55	12	42	94	18	6	67
----	----	----	----	----	----	---	----

# Ejemplo quicksort



*Quicksort(1, 3)*

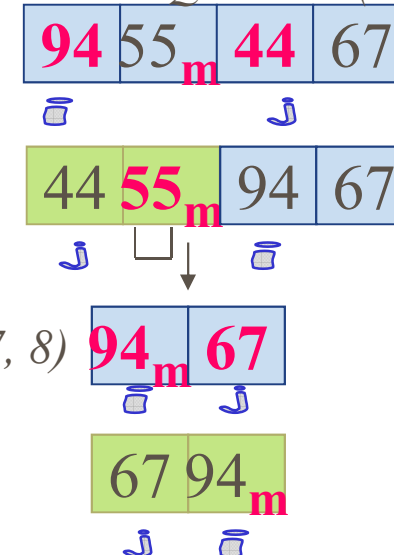


*el pivote cambia,  
se incluye en la  
llamada  
recursiva*

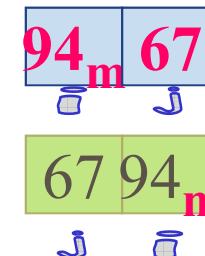
*Quicksort(1, 2)*



*Quicksort(5, 8)*



*Quicksort(7, 8)*





2	4	6	8	1	5	3
---	---	---	---	---	---	---

# Ejemplo quicksort

*Quicksort(1, 7)*

2	4	6	8 <sub>m</sub>	1	5	3
---	---	---	----------------	---	---	---

2	4	6	3	1	5	8 <sub>m</sub>
---	---	---	---	---	---	----------------

*Quicksort(1, 6)*

2	4	6 <sub>m</sub>	3	1	5
---	---	----------------	---	---	---

2	4	5	3	1	6 <sub>m</sub>
---	---	---	---	---	----------------

2	4	5 <sub>m</sub>	3	1
---	---	----------------	---	---

*Quicksort(1, 5)*

2	4	1	3	5 <sub>m</sub>
---	---	---	---	----------------

2	4 <sub>m</sub>	1	3
---	----------------	---	---

*Quicksort(1, 4)*

2	3	1	4 <sub>m</sub>
---	---	---	----------------

# Método de contabilización de frecuencias

- Sólo se puede aplicar bajo condiciones muy especiales
- **NO** es un método *in-situ* (requiere un vector auxiliar para la ordenación)
- Su importancia radica en que su tiempo de ejecución es  $O(n)$
- Si el rango de los valores del vector de partida es muy amplio, definir un vector que abarque dicho rango es inviable
- Este algoritmo se puede plantear en más ocasiones de lo que a priori parece

# Métodos sofisticados

## Método de contabilización de frecuencias

- Se basa en contabilizar frecuencias de los elementos del vector:
  - En un vector ordenado, si un elemento está en la posición  $k$ -ésima, se debe a que en el estado inicial había  $k-1$  elementos menores que el (si suponemos que todos los elementos son distintos)
  - Contar para cada elemento del vector  $v$ , el número de elementos inferiores a el, después colocar cada valor en un vector  $v_{ordenado}$  en función de la contabilización realizada
- Para realizar la contabilización necesaria, hay que comparar cada elemento del vector con los demás ( $n^2$ )
  - Cuando el número de posibles elementos del vector es finito, y está acotado entre unos límites razonables, se puede formar un vector que contabilice las frecuencias de cada elemento, y obtener un algoritmo de contabilización eficaz

# Métodos sofisticados

algoritmo frecuencias(n, v; ; vordenado)

inicio

para i de mínimo a máximo *//Inicializa vector de frecuencias*

frec [i]  $\leftarrow$  0

fin-para

para i de 1 a n *//Construye vector de frecuencias*

frec[v[i]]  $\leftarrow$  frec[v[i]] + 1

fin-para

para i de mínimo+1 a máximo *//Construye vector de frecuencias acumuladas*

frec[i]  $\leftarrow$  frec[i-1]+frec[i]

fin-para

para i de n a 1 paso -1 *//Recorrido inverso para que la ordenación sea estable*

vordenado[frec[v[i]]]  $\leftarrow$  v[i]

frec[v[i]]  $\leftarrow$  frec[v[i]] - 1

fin-para

fin



# Ejemplo frecuencias

6	4	5	6	4	8	6
---	---	---	---	---	---	---

Paso 1. Construir vector de frecuencias

8	7	6	5	4
1	0	3	1	2

Paso 2. Construir vector de frecuencias acumuladas

8	7	6	5	4
1	0	3	1	2
8	7	6	5	4
7	6	6	3	2

Paso 3. Colocación de  $v$  en ordenado

	$l$						$n$
$v$	6	4	5	6	4	8	6
						6	

8	7	6	5	4
7	6	5	3	2
$max$				$min$
				$frec$



# Ejemplo frecuencias

6	4	5	6	4	8	6
---	---	---	---	---	---	---

					6	8
--	--	--	--	--	---	---



8	7	6	5	4
6	6	5	3	2

6	4	5	6	4	8	6
---	---	---	---	---	---	---

	4				6	8
--	---	--	--	--	---	---



8	7	6	5	4
6	6	5	3	1

6	4	5	6	4	8	6
---	---	---	---	---	---	---

	4			6	6	8
--	---	--	--	---	---	---



8	7	6	5	4
6	6	4	3	1

# Ejemplo frecuencias

6	4	5	6	4	8	6
	4	5		6	6	8



8	7	6	5	4
6	6	4	2	1

6	4	5	6	4	8	6
4	4	5		6	6	8



8	7	6	5	4
6	6	4	2	0

6	4	5	6	4	8	6
4	4	5	6	6	6	8

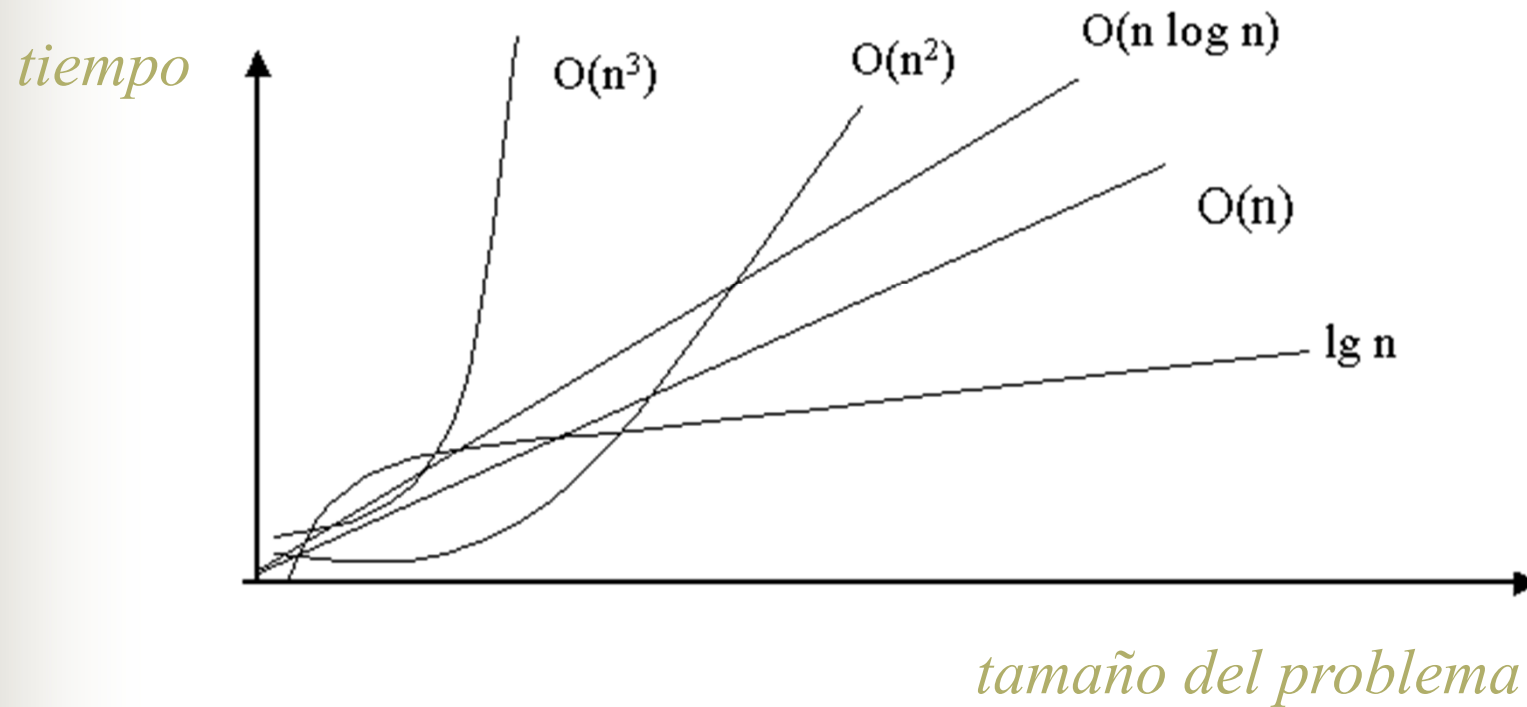


8	7	6	5	4
6	6	3	2	0

# Ordenación de ficheros (externa)

- **Ordenación de ficheros.** Ordenar los registros de un fichero según algún campo. Operación muy común en aplicaciones de gestión
- Si el fichero es pequeño, se puede copiar en memoria y utilizar los algoritmos de ordenación (burbuja, inserción directa, *quicksort*, ...)
- Si el fichero es grande los métodos de ordenación interna no resultan eficientes cuando los datos están almacenados en un fichero (los intercambios de elementos son muy costosos)
- Existen métodos de ordenación de ficheros, basados en otras técnicas, que utilizan estructuras de datos adicionales y que se escapan de nuestros objetivos:
  - **Ordenación mediante fusión**
  - **Ordenación n-vías**

# Ordenes de eficiencia





# Conclusiones

- El método de la burbuja es el peor de todos, y su versión mejorada, de la sacudida, es incluso peor que los de inserción y selección directa
- En general, el método de selección directa es preferible al de inserción directa e inserción binaria, aunque en los casos en que los elementos estén inicialmente ordenados o casi ordenados, estos últimos pueden ser algo más rápidos
- El método Shell es más rápido que todos los anteriores, pero menos eficaz que los de orden  $n \log n$
- Los métodos sofisticados (orden  $n \log n$ ) requieren menos operaciones que los demás, pero al ser normalmente más complejos en detalle, no se deben utilizar para valores pequeños de  $n$
- De los métodos sofisticados, el *quicksort* es el más eficaz



# Conclusiones

	→ mejor		
mejor ↑	$O(n^2)$	$O(n \log n)$	$O(n)$
	Inserción Shell Selección directa <sub>1</sub> Inserción binaria Inserción directa Burbuja (sacudida) Burbuja	Quicksort <sub>2</sub>	Contabilización de frecuencias
	1. Cuando los elementos están prácticamente ordenados inserción binaria y directa pueden ser algo más rápidos 2. No se deben utilizar con pequeños valores de $n$		