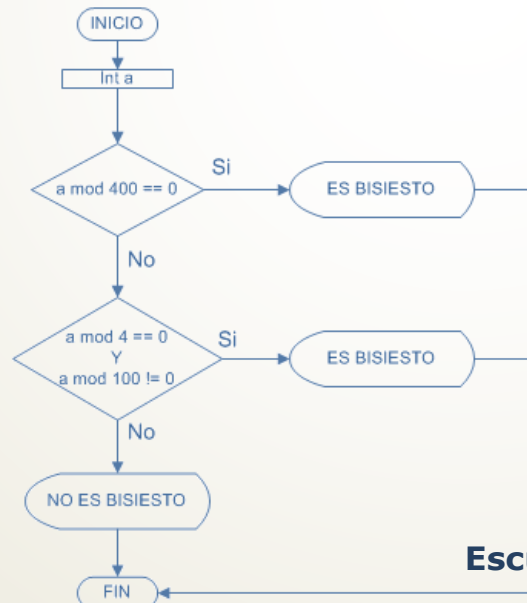


# Programación Estructurada

## UNIDAD V: ARREGLOS. INTERCALACIÓN Y ORDENACIÓN



# Arreglos

## ➤ Definición

- colección homogénea y finita de elementos cuyo acceso se realiza a través de índices.

## ➤ Índices

- valores, expresiones o constantes de tipo ordinal que permiten identificar de forma individual a cada elemento de un arreglo.

## ➤ Tipos (según su organización)

- Unidimensional (vector), Bidimensional (matriz) y Multidimensional.

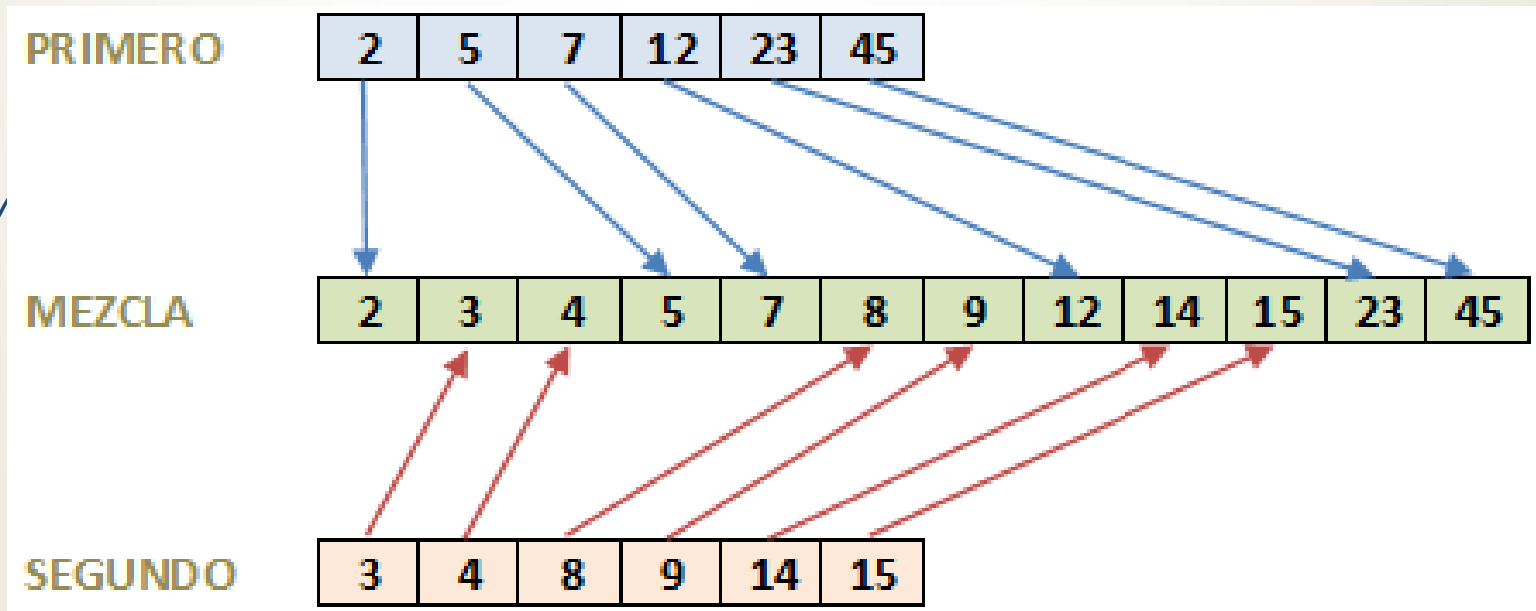
# Operaciones

---

- Asignación
- Lectura/escritura
- Recorrido
- Actualización
  - agregar, insertar y borrar
- Búsqueda
  - secuencial y binaria
- Intercalación
- Ordenación
  - burbuja, selección, inserción, shaker sort, rápido y shell

# Intercalación (1)

- La intercalación consiste en mezclar (intercalar) el contenido de dos vectores ordenados para producir un nuevo vector también ordenado.



# Intercalación (2)

```
PROCEDIMIENTO intercalar (E uno:tvector1,E dos:tvector1,
                        E/S tres:tvector2,E ocup1:entero,
                        E ocup2:entero,E/S ocup3:entero)
```

```
VARIABLES
```

```
    i,j,k:entero
```

```
INICIO
```

```
    i←1;j←1;k←0;
```

```
    MIENTRAS (i≤ocup1) Y (j≤ocup2) HACER
```

```
        SI uno[i]< dos[j] ENTONCES
```

*Copia un  
elemento del  
primer vector*

```
            k←k+1
            tres[k]←uno[i]
            i←i+1
```

```
        SINO
```

*Copia un  
elemento del  
segundo vector*

```
            k←k+1
            tres[k]←dos[j]
            j←j+1
```

```
        fin_si
```

```
    FIN_MIENTRAS
```

*Recorrido de los  
arreglos*

...

# Intercalación (3)

```
...  
MIENTRAS i<=ocup1 HACER  
    k←k+1  
    tres[k]←uno[i]  
    i←i+1  
FIN_MIENTRAS  
MIENTRAS j<=ocup2 HACER  
    k←k+1  
    tres[k]←dos[j]  
    j←j+1  
FIN_MIENTRAS  
ocup3← k  
FIN
```

*Copia los elementos restantes del primer vector*

*Copia los elementos restantes del segundo vector*

# Ordenación

---

- La ordenación de arreglos consiste en reorganizar el contenido (datos) de éstos según un criterio ascendente o descendente.
- Métodos de ordenación
  - Burbuja
  - Selección
  - Inserción
  - Shaker sort
  - Rápido
  - Shell



# Ordenación. Burbuja (1)

- El algoritmo de *intercambio* o *burbuja* compara pares de elementos adyacentes intercambiándolos (cuando sea necesario) hasta que queden todos ordenados.
  1. Se comparan el primer y segundo elementos del arreglo, y se intercambian si no están ordenados.
  2. Luego, se comparan el segundo y tercer elementos, y se intercambian si no están ordenados.
  3. El proceso continúa comparando cada elemento con el siguiente, intercambiándolos cuando sea necesario.
  4. El proceso finaliza cuando el vector queda ordenado.



# Ordenación. Burbuja (2)

<i>Primer recorrido</i>	1	2	3	4	5	
	11	88	53	40	28	<i>compara 1º y 2º elemento</i>
	1	2	3	4	5	
	11	88	53	40	28	<i>compara 2º y 3º elemento</i>
	1	2	3	4	5	
<i>Segundo recorrido</i>	11	53	88	40	28	<i>compara 3º y 4º elemento</i>
	1	2	3	4	5	
	11	53	40	88	28	<i>compara 4º y 5º elemento</i>
	1	2	3	4	5	
	11	53	40	28	88	<i>compara 1º y 2º elemento</i>
	1	2	3	4	5	
	11	53	40	28	87	<i>compara 2º y 3º elemento</i>
	1	2	3	4	5	
	11	40	53	28	87	<i>compara 3º y 4º elemento</i>
	1	2	3	4	5	
	11	40	28	53	87	<i>compara 4º y 5º elemento</i>

# Ordenación. Burbuja (3)

	1	2	3	4	5	
	11	40	53	28	87	<i>compara 3° y 4° elemento</i>
	1	2	3	4	5	
	11	40	28	53	87	<i>compara 4° y 5° elemento</i>
<i>Tercer recorrido</i>	1	2	3	4	5	
	11	40	28	53	87	<i>compara 1° y 2° elemento</i>
	1	2	3	4	5	
	11	28	40	53	87	<i>compara 2° y 3° elemento</i>
	1	2	3	4	5	
	11	28	40	53	87	<i>compara 3° y 4° elemento</i>
	1	2	3	4	5	
	11	28	40	53	87	<i>compara 4° y 5° elemento</i>
<i>Ordenado</i>	1	2	3	4	5	
	11	28	40	53	88	

# Ordenación. Burbuja (4)

PROCEDIMIENTO burbuja(E/S a:tvector,E ocupado:entero)

VARIABLES

j:entero

bandera:lógico

INICIO

ordenado←FALSO

MIENTRAS (ordenado=FALSO) HACER

ordenado←VERDADERO

PARA j DESDE 1 HASTA ocupado-1 HASTA

SI a[j]>a[j+1] ENTONCES

cambio(a[j],a[j+1])

ordenado←FALSO

FIN\_SI

FIN\_PARA

FIN\_MIENTRAS

*Comparación e  
intercambio de  
elementos*

*Recorrido  
del vector*

*Bucle que  
repite el  
recorrido  
del vector*

FIN

# Ordenación. Selección (1)

- El algoritmo de *selección* consiste en recorrer el arreglo (comparando cada posición con las siguientes) e intercambiar elementos cuando sea necesario.
  1. Se compara la primera posición del arreglo con las restantes y se intercambia su contenido cuando corresponda.
  2. Se compara la segunda posición del arreglo con las restantes y se intercambia su contenido cuando corresponda.
  3. Este proceso se repite para cada posición del arreglo.
  4. El algoritmo finaliza cuando se evalúa la penúltima posición.

# Ordenación. Selección (2)

Primer recorrido	1	2	3	4	5	
	11	88	53	40	28	compara 1° y 2° elemento
	1	2	3	4	5	
	11	88	53	40	28	compara 1° y 3° elemento
	1	2	3	4	5	
	11	53	88	40	28	compara 1° y 4° elemento
	1	2	3	4	5	
	11	53	40	88	28	compara 1° y 5° elemento
Segundo recorrido	1	2	3	4	5	
	11	88	53	40	28	compara 2° y 3° elemento
	1	2	3	4	5	
	11	53	88	40	28	compara 2° y 4° elemento
	1	2	3	4	5	
	11	40	88	53	28	compara 2° y 5° elemento

# Ordenación. Selección (3)

<i>Tercer recorrido</i>	1	2	3	4	5	<i>compara 3° y 4° elemento</i>
	11	28	88	53	40	
	1	2	3	4	5	<i>compara 3° y 5° elemento</i>
	11	28	53	88	40	
<i>Cuarto recorrido</i>	1	2	3	4	5	<i>compara 4° y 5° elemento</i>
	11	28	40	88	53	
<i>Ordenado</i>	1	2	3	4	5	
	11	28	40	53	88	

# Ordenación. Selección (4)

PROCEDIMIENTO seleccion(E/S a:tvector,E ocupado:entero)

VARIABLES

i,j:entero

INICIO

PARA i DESDE 1 HASTA ocupado-1 HACER

PARA j DESDE i+1 hasta ocupado HACER

SI a[i]>a[j] ENTONCES

cambio(a[i],a[j])

FIN\_SI

FIN\_PARA

FIN\_PARA

FIN

Comparación e  
intercambio de  
elementos

Recorrido del  
arreglo a partir  
de la posición  
i+1

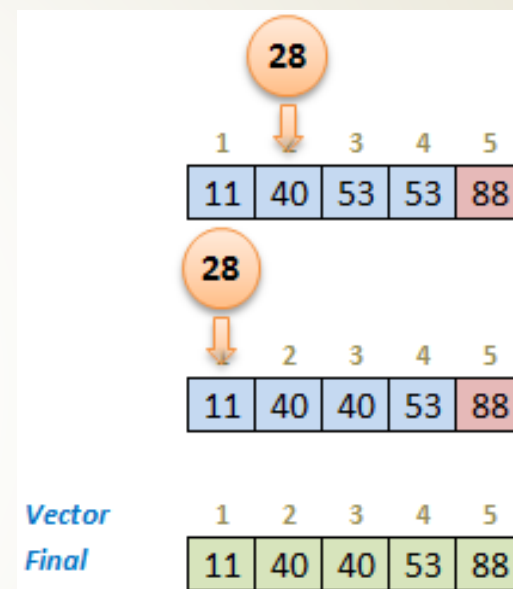
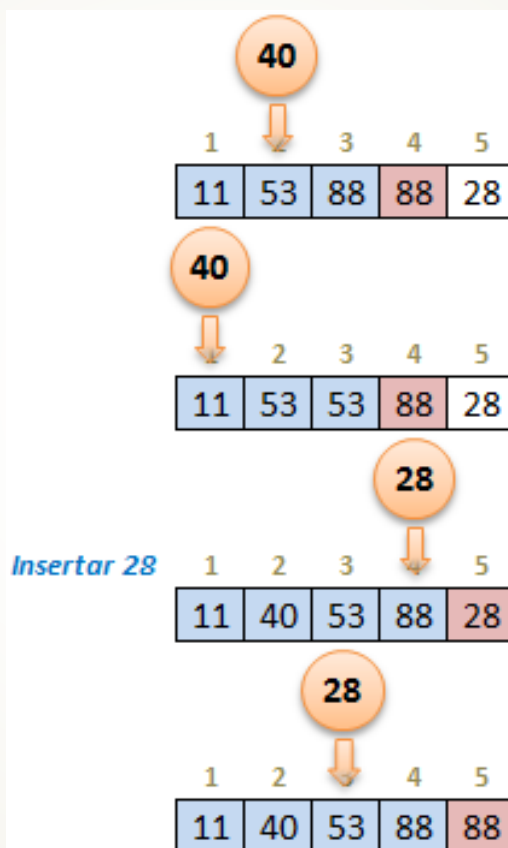
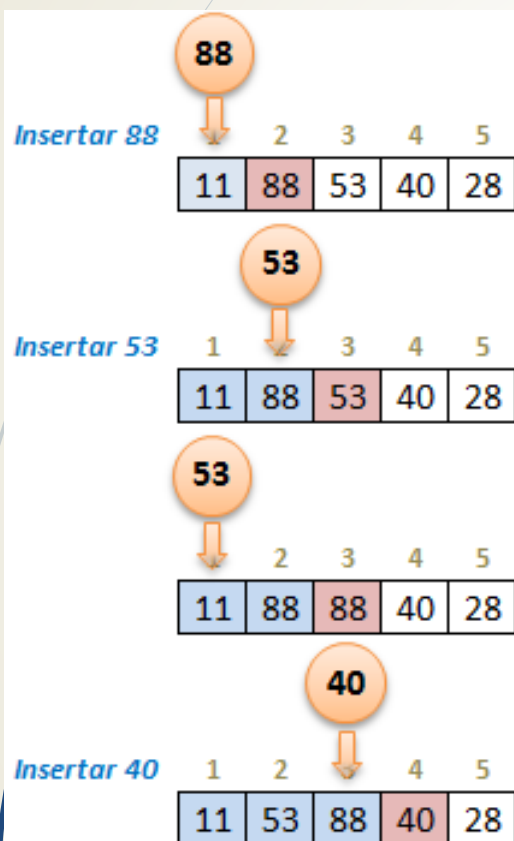
Recorrido  
principal  
del arreglo



# Ordenación. Inserción (1)

- El algoritmo de *inserción* ordena un vector insertando el elemento  $i$  entre los  $i-1$  elementos anteriores (ya ordenados).
  1. Se parte de un subarreglo de 1 elemento, en el que se inserta (en orden) el segundo elemento del arreglo original.
  2. Se parte de un subarreglo de 2 elementos, en el que se inserta (en orden) el tercer elemento del arreglo original.
  3. Se parte de un subarreglo de  $i-1$  elementos, en el que se inserta (en orden) el elemento  $i$  del arreglo original.
  4. El algoritmo finaliza cuando se evalúa la última posición.

# Ordenación. Inserción (2)



# Ordenación. Inserción (3)

PROCEDIMIENTO insercion(E/S a:tvector, E ocupado: entero)

VARIABLES

i, j, aux: entero

INICIO

PARA i DESDE 2 HASTA ocupado HACER

aux ← a[i]

j ← i - 1

MIENTRAS (j ≥ 1) Y (a[j] > aux) HACER

a[j + 1] ← a[j]

j ← j - 1

FIN\_MIENTRAS

a[j + 1] ← aux

FIN\_PARA

FIN

*Recorrido del arreglo  
(a partir de la 2da posición)*

*Desplazamiento  
de elementos y  
búsqueda de la  
posición de  
inserción*

*Inserción del  
elemento en el  
subarreglo ordenado*

# Ordenación. Shaker sort (1)

- El algoritmo *shaker sort* o sacudida es una variante del método Burbuja que funciona de la siguiente forma:
  1. primero, se recorre el arreglo de **derecha** a **izquierda** intercambiando pares de valores según el criterio de ordenación elegido.
  2. segundo, se recorre el arreglo de **izquierda** a **derecha** intercambiando pares de valores según el criterio de ordenación elegido.
  3. cada recorrido ignora las posiciones ya ordenadas lo que reduce las posiciones a comparar y ordenar
  4. El algoritmo finaliza cuando todas las posiciones fueron ordenadas o cuando en un recorrido no se realizaron intercambios .

# Ordenación. Shaker sort (2)

1	2	3	4	5	6
78	21	24	33	11	13
1	2	3	4	5	6
78	21	24	33	11	13
1	2	3	4	5	6
78	21	24	11	33	13
1	2	3	4	5	6
78	21	11	24	33	13
1	2	3	4	5	6
78	11	21	24	33	13
1	2	3	4	5	6
11	78	21	24	33	13

Elemento Ordenado

Recorrido Derecha-Izquierda

1	2	3	4	5	6
11	78	21	24	33	13
1	2	3	4	5	6
11	21	78	24	33	13
1	2	3	4	5	6
11	21	24	78	33	13
1	2	3	4	5	6
11	21	24	33	78	13
1	2	3	4	5	6
11	21	24	33	13	78

Elemento Ordenado      Elemento Ordenado

Recorrido Izquierda-Derecha

PRIMERA PASADA

1	2	3	4	5	6
11	21	24	33	13	78
1	2	3	4	5	6
11	21	24	13	33	78
1	2	3	4	5	6
11	21	13	24	33	78
1	2	3	4	5	6
11	13	21	24	33	78

Elementos Ordenados      Elemento Ordenado

Recorrido Derecha-Izquierda

1	2	3	4	5	6
11	13	21	24	33	78
1	2	3	4	5	6
11	13	21	24	33	78

NO SE REALIZARON INTERCAMBIOS DURANTE EL RECORRIDO

Recorrido Izquierda-Derecha

SEGUNDA PASADA

# Ordenación. Shaker sort (3)

PROCEDIMIENTO sacudida(E/S a:tvector,E ocup:ENTERO)

VARIABLES

ordenado:LÓGICO

pri,ult,i:ENTERO

INICIO

ordenado←FALSO

pri←1

ult←ocup

MIENTRAS (ordenado=FALSO Y pri<ult) HACER

ordenado←VERDADERO

PARA i DESDE ult HASTA pri+1 CON PASO -1 HACER

SI (a[i] < a[i-1]) ENTONCES

aux(a[i],a[i-1])

ordenado←FALSO

FIN SI

FIN PARA

pri←-pri+1

SI (ordenado=FALSO) ENTONCES

ordenado←VERDADERO

PARA i DESDE pri HASTA ult-1 CON PASO 1 HACER

SI (a[i]>a[i+1]) ENTONCES

aux(a[i],a[i+1])

ordenado←FALSO

FIN SI

FIN PARA

FIN SI

ult←-ult-1

FIN MIENTRAS

FIN

Comparación  
e intercambio  
de elementos

Recorrido  
Derecha-Izquierda

Comparación  
e intercambio  
de elementos

Recorrido  
Izquierda-Derecha

Bucle de  
Ordenación



# Ordenación. Shell (1)

- ▶ Shell (o inserción con incrementos decrecientes) es una mejora del método de inserción.
- ▶ Compara elementos que se encuentran a cierta distancia (salto  $> 1$ ) consiguiendo una ordenación más rápida.
- ▶ El valor inicial de salto puede definirse como  $N/2$  ( $N$  elementos del arreglo)
- ▶ El salto se reduce a la mitad en cada repetición hasta alcanzar 1 (algoritmo de inserción básico).
- ▶ El algoritmo finaliza cuando el salto es CERO.



# Ordenación. Shell (2)

**Inserción**

**con Salto=5**

**(11 div 2)**

La ordenación se realiza por grupos de elementos. Los elementos de un grupo están separados por una distancia igual a salto.

1	2	3	4	5	6	7	8	9	10	11
11	88	53	40	28	87	7	37	5	9	63



1	2	3	4	5	6	7	8	9	10	11
11	88	53	40	28	87	7	37	5	9	63



1	2	3	4	5	6	7	8	9	10	11
11	88	53	40	28	87	7	37	5	9	63



1	2	3	4	5	6	7	8	9	10	11
11	88	53	40	28	63	88	53	40	28	87

**Grupos**

**ordenados**

1	2	3	4	5	6	7	8	9	10	11
11	7	37	5	9	63	88	53	40	28	87

# Ordenación. Shell (3)

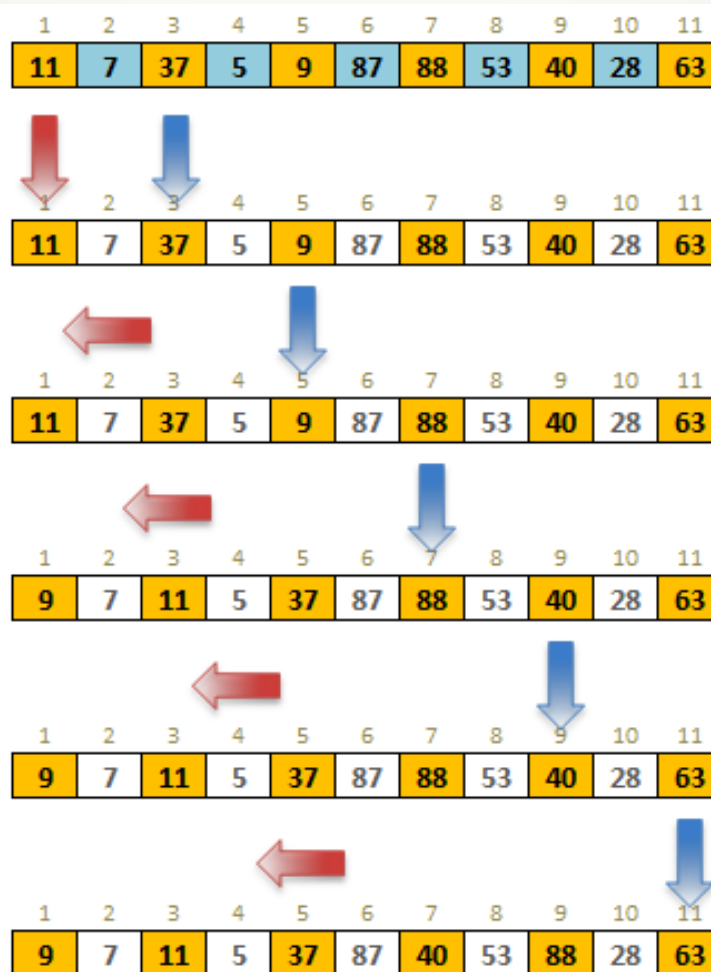
**Inserción**

**con Salto=2**

**(5 div 2)**

**Ordenando**

**grupo  
naranja**



# Ordenación. Shell (4)

Grupos  
ordenados

1	2	3	4	5	6	7	8	9	10	11
9	5	11	7	37	28	40	53	63	87	88

Inserción  
con Salto=1  
(2 div 2)

1	2	3	4	5	6	7	8	9	10	11
9	5	11	7	37	28	40	53	63	87	88

1	2	3	4	5	6	7	8	9	10	11
5	9	11	7	37	28	40	53	63	87	88

1	2	3	4	5	6	7	8	9	10	11
5	9	11	7	37	28	40	53	63	87	88

Se continua  
hasta ...

1	2	3	4	5	6	7	8	9	10	11
5	7	9	11	37	28	40	53	63	87	88

Vector  
Ordenado  
(Salto=0)

1	2	3	4	5	6	7	8	9	10	11
5	7	9	11	28	37	40	53	63	87	88

# Ordenación. Shell (5)

PROCEDIMIENTO shell(E/S a:tvector, E ocupado: entero)

VARIABLES

i, j, aux, salto: entero

INICIO

salto ← ocupado div 2 *Salto Inicial*

*Ordenación de  
elementos por grupos*

MIENTRAS salto > 0 HACER

PARA i DESDE salto+1 HASTA ocupado HACER

aux ← a[i]

j ← i - salto

MIENTRAS (j >= 1) Y (a[j] > aux) HACER

a[j+salto] ← a[j]

j ← j - salto

FIN MIENTRAS

a[j+salto] ← aux

*Inserción en el  
subgrupo ordenado*

FIN PARA

salto ← salto div 2 *Reducción del salto*

*Desplazamiento  
de elementos y  
búsqueda de la  
posición de  
inserción*

FIN\_MIENTRAS

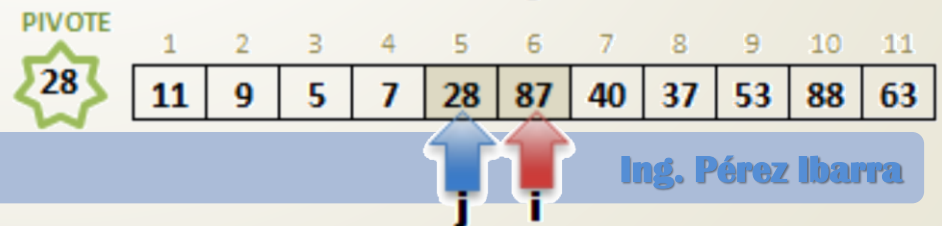
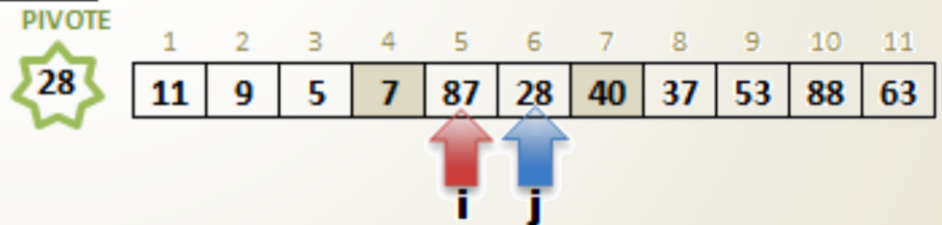
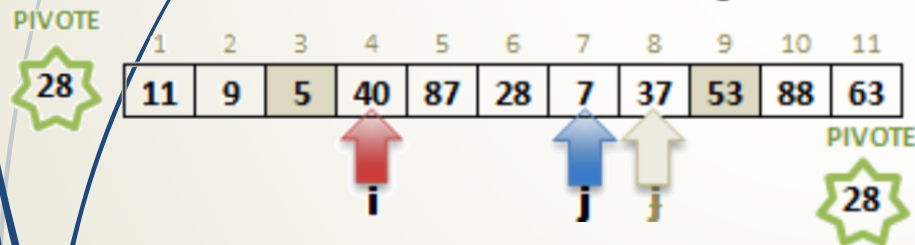
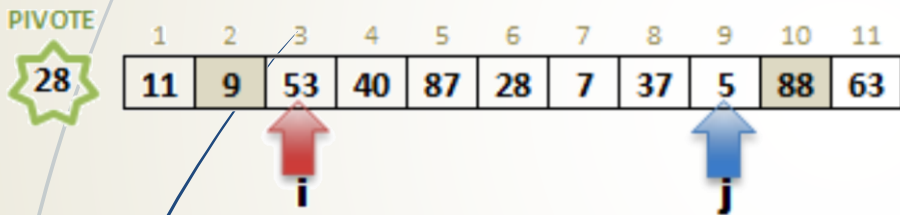
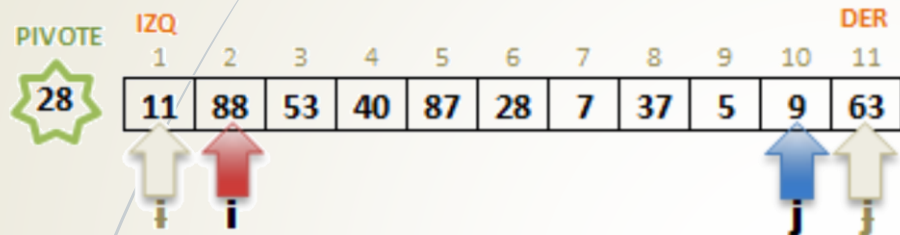
FIN

# Ordenación. Rápido (1)

- El algoritmo *rápido* (*quick sort*) se basa en la técnica divide y vencerás.
- El método particiona, sucesivamente, el arreglo en subarreglos hasta reducir el espacio de ordenación a 0 (ningún elemento).
- Antes de cada partición se elige un valor **pivote** y se reorganizan los elementos del arreglo agrupando por un lado los valores **menores** al pivote y por otro los **mayores**.
- El **pivote** puede ser cualquier valor arbitrario del vector.

# Ordenación. Rápido (2)

Primer llamado a la ordenación, pivote=28 ((IZQ+DER) div 2)

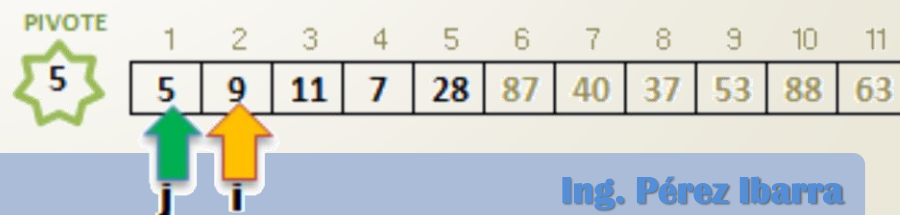
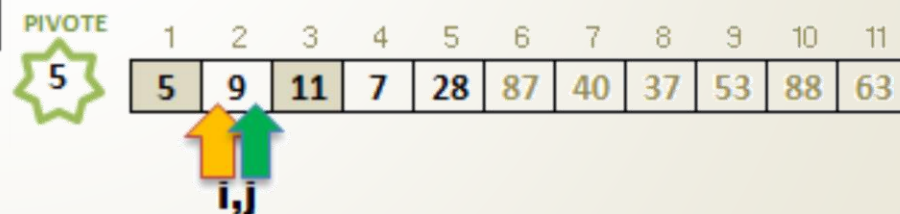
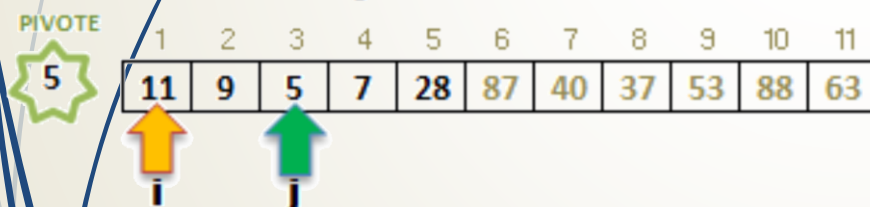
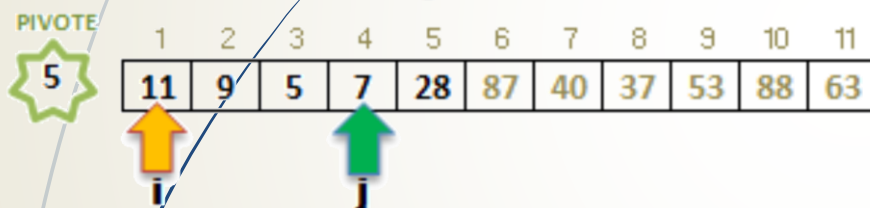
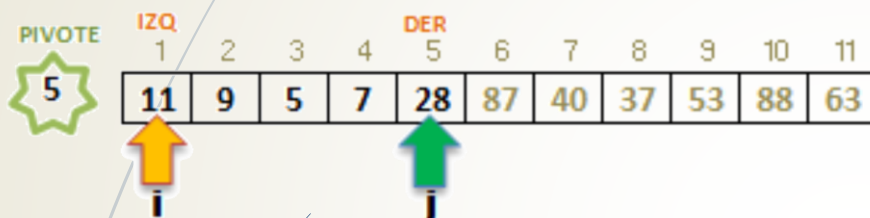




# Ordenación. Rápido (3)

Llamado recursivo por izquierda

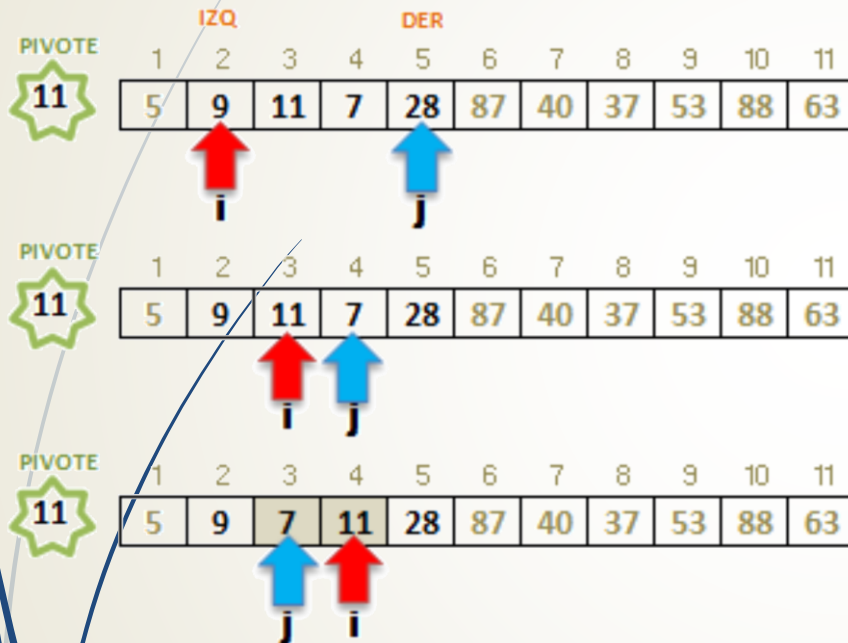
*rapido(vector, IZQ,j), pivote=5*





# Ordenación. Rápido (4)

Llamado recursivo por derecha  
*rapido(vector, i, DER), pivote=11*



Llamado recursivo por izquierda  
*rapido(vector, IZQ, j), pivote=11*



Ordenación de la primer mitad del vector (izquierda)

5	7	9	11	28	87	40	37	53	88	63
---	---	---	----	----	----	----	----	----	----	----

# Ordenación. Rápido(4)

PROCEDIMIENTO rapido(E/S a:tvector,E izq:entero,E der:entero)

VARIABLES

i,j,pivote:entero

INICIO

i ← izq } *Intervalo de  
ordenación*  
j ← der }

pivote ← a[(izq+der) div 2]

MIENTRAS i ≤ j HACER

MIENTRAS a[i] < pivote HACER

i ← i+1

FIN\_MIENTRAS

MIENTRAS a[j] > pivote HACER

j ← j-1

FIN\_MIENTRAS

SI i ≤ j ENTONCES

cambio(a[i],a[j])

i ← i+1

j ← j-1

FIN\_SI

FIN\_MIENTRAS

*Ordenación  
por izquierda* { SI izq < j ENTONCES  
rapido(a,izq,j)  
FIN\_SI

*Ordenación  
por derecha* { SI i < der ENTONCES  
rapido(a,i,der)  
FIN\_SI

FIN

*Intercambio de  
elementos a  
izquierda y  
derecha del  
pivote*

# Bibliografía

---

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.