

# Estructuras y Tipos de Datos



Ordenamiento y Mezcla

# TEMA IV - Ordenamiento y Mezcla

## CONTENIDO

- Introducción
- Ordenamiento Interno
  - Método de Ordenamiento por Inserción
  - Método de Ordenamiento por Selección
  - Método de Ordenamiento por Intercambio (Burbuja)
  - Método de ordenamiento Shell
  - Método de Ordenamiento Rápido - Quicksort
- Merge Sort
- Standard Template Library (STL)

# Ordenamiento y Mezcla

## Introducción

*El proceso de ordenamiento consiste en reorganizar una colección de elementos en orden ascendente o descendente, de acuerdo a una clave.*

### **Objetivos:**

- *El mantener los elementos ordenados supone el poder emplear estrategias de búsqueda más fáciles y eficientes.*
- *Nos permite hacer estudios de diferentes algoritmos que resuelven el mismo problema.*

### **Tipos:**

- **Ordenamiento Interno:** *los datos residen en memoria principal, el proceso se lleva a cabo en memoria principal*
- **Ordenamiento Externo:** *cantidad de elementos es demasiado grande, hace uso de dispositivos de memoria secundaria,*

# Ordenamiento y Mezcla

## Introducción

### *Proceso de Ordenamiento:*

*Dados los elementos y una función, ordenar los elementos a través de la función.*

$$\begin{array}{l} e_1, e_2, e_3, \dots, e_n \\ f(e_i) \end{array}$$
$$\Rightarrow e_{k1} < e_{k2} < e_{k3} < \dots < e_{kn} \quad f(e_{ki}) \leq f(e_{kj}) \quad i \leq j$$

### *Análisis de la Eficiencia:*

- *Complejidad*
- *Número de comparaciones entre claves*
- *número de movimientos de registros.*

*Para efecto de los ejemplos y algoritmos que mostraremos mas adelante, definiremos los elementos a ordenar de la siguiente forma:*

```
TYPE elemento = RECORD  
    Clave : integer;  
    Otros componentes  
END
```

# Ordenamiento Interno

---

## Ordenamiento Interno

*Los datos residen en memoria principal,*

*El proceso se lleva a cabo en memoria principal*

***Métodos de ordenamiento en arreglos a revisar:***

- 1. Inserción*
- 2. Selección*
- 3. Intercambio o Burbuja*
- 4. ShellSort*
- 5. Quicksort*

# Ord. Int. – Método por Inserción

## Estrategia

*Mantener el arreglo dividido en dos conjuntos, uno ordenado y otro desordenado. Inicialmente todos los elementos están en el conjunto desordenado*

*Realizan los siguientes pasos:*

*Tomar el primer elemento del conjunto desordenado e insertarlo en el lugar apropiado del conjunto ordenado. Repetir hasta que el conjunto desordenado esté vacío.*

## Ejemplo

*Sea A un arreglo de N elementos, con  $N = 6$ . Ordenar A por el Método de Inserción*

Paso 0

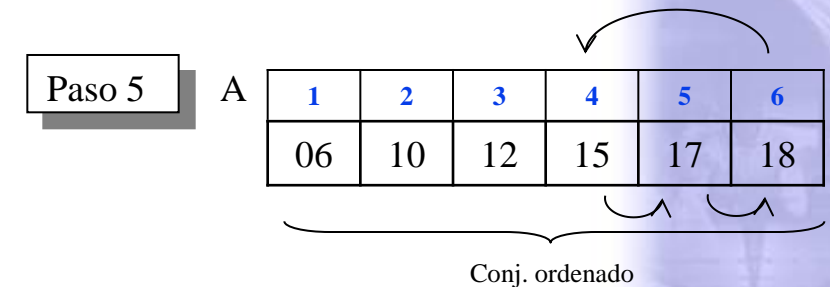
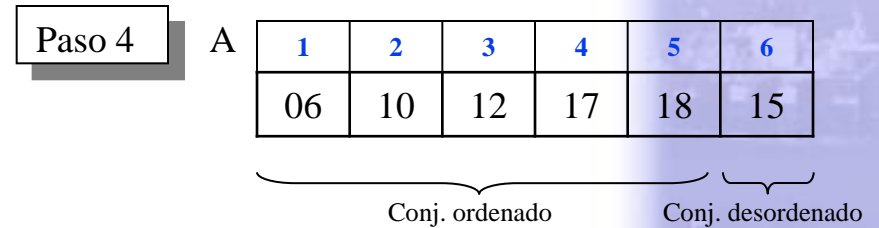
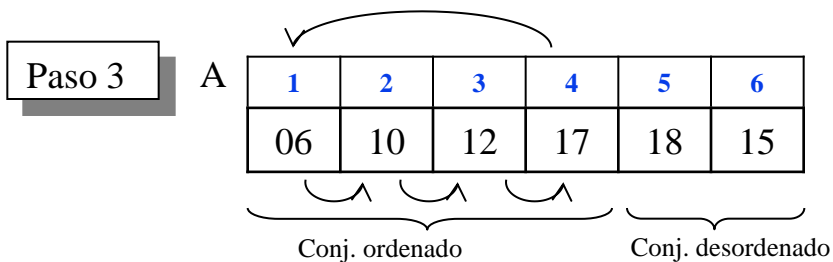
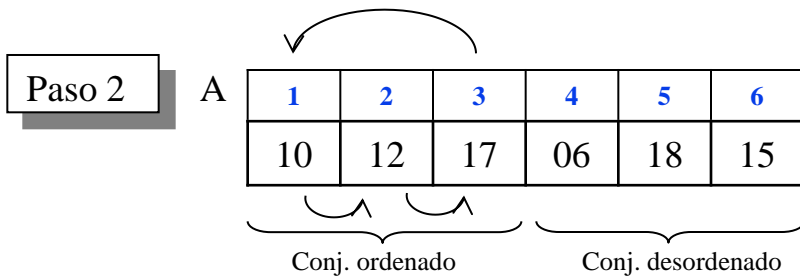
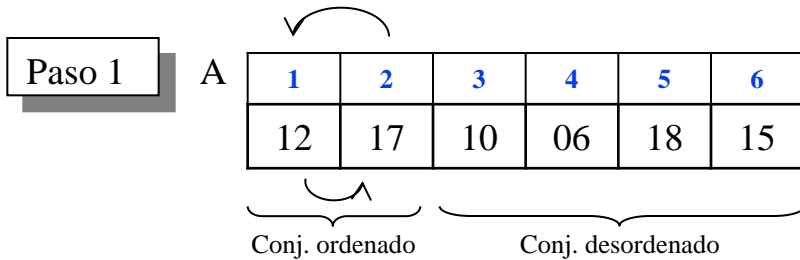
A

1	2	3	4	5	6
17	12	10	06	18	15

Conj. ordenado      Conj. desordenado

# Ord. Int. – Método por Inserción

## Ejemplo, Cont ...



# Ord. Int. – Método por Inserción

## Algoritmo

```
For  $i := 2$  to  $n$  do
Begin
   $x := a[i]$ ;
  insertar  $x$  en el lugar apropiado en  $a_1, \dots, a_{i-1}$ 
End
```

## Código

```
void insercionDirecta(int array[], int size) {
    int i, j;

    for (i=2; i<size; i++) {
        for (j=i; j>0 && array[j]<array[j-1]; j--) {
            int t; // Intercambiar
            t=array[j];
            array[j]=array[j-1];
            array[j-1]=t;
        }
    }
}
```



# Ord. Int. – Método por Inserción

## Análisis

### Complejidad

*En el for*                      *Se hacen*  $i$  *comparaciones*  
                                          $i-1$  *movimientos*  
                                         *orden*  $O(i)$

*el while interno está afectado por el for externo, entonces orden del algoritmo depende de cuantas veces se ejecute el for interno .*

$$\rightarrow \sum_{i=2}^n c_1 * i = c_1 \sum_{i=2}^n i = c_1 * \left( \frac{n * (n+1)}{2} - 1 \right) \approx O(n^2)$$

# Ord. Int. – Método por Inserción

## Características

- *Tiene un comportamiento natural*
- *Es estable*

Un **algoritmo** de ordenamiento **es estable** si el orden original entre las claves iguales se mantienen después de aplicarse el algoritmo, en caso de no conservarse el orden entre estas claves se dice que el algoritmo no es estable

### Ejemplo

<i>Entrada</i>	<i>Algoritmo de ordenamiento</i>	<i>Salida</i>	
$3_1 3_2 2 5$	<i>Alg.. Orden 1</i>	$2 3_1 3_2 5.$	<i>Este algoritmo es estable.</i>
$3_1 3_2 2 5$	<i>Alg.. Orden 2</i>	$2 3_2 3_1 5.$	<i>Este algoritmo no es estable.</i>

# Ord. Int. – Método por Selección

## Estrategia

*Mantener una parte del arreglo ordenado y otra no ordenado. Inicialmente todos están no ordenados.*

*Realizan los siguientes pasos:*

*Buscar el elemento más pequeño del conjunto no ordenado,*

*Intercambiar elto mas pequeño con el primer elemento del conjunto no ordenado*

*Incorporarlo al conjunto ordenado.*

*Repetir hasta que todos los elementos están en el conjunto ordenado*

## Ejemplo

*Sea A un arreglo de N elementos, con  $N = 6$ . Ordenar A por el Método de Selección*

Paso 0

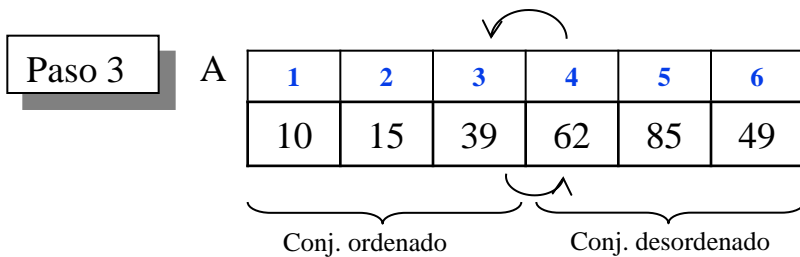
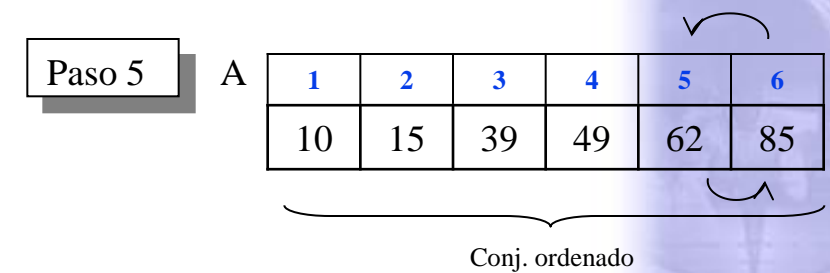
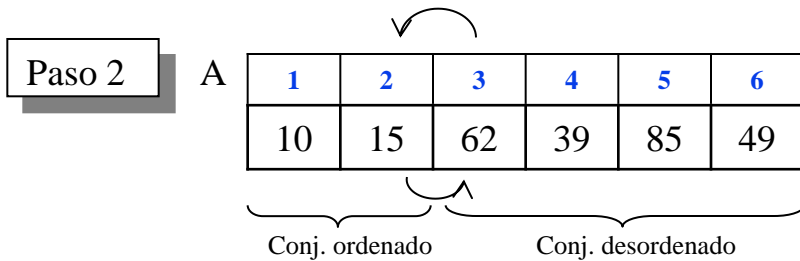
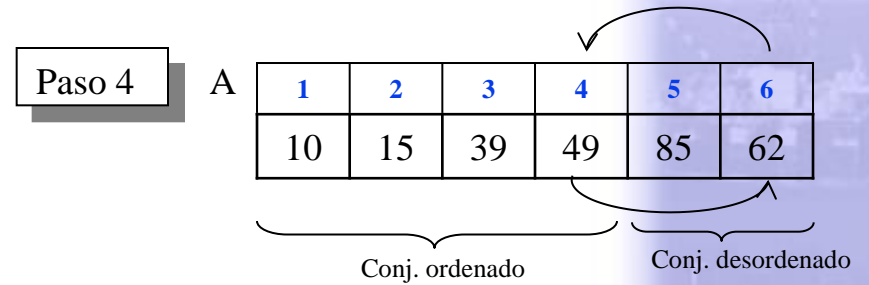
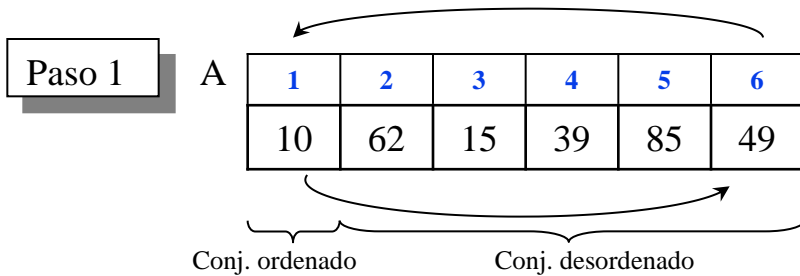
A

1	2	3	4	5	6
49	62	15	39	85	10

Conj. desordenado

# Ord. Int. – Método por Selección

## Ejemplo, Cont ...



# Ord. Int. – Método por Selección

## Algoritmo

```
For  $i := 1$  to  $n-1$  do Begin
    Asignar el índice del menor elemento de
         $a[i], \dots, a[n]$  a  $k$ ;
    Intercambiar  $a[i]$  y  $a[k]$ 
End
```

```
void seleccion(int array[], int size) {
    int e, i, j;

    for(i=0; i<size-1; i++){
        e=i;
        for(j=i+1; j<size; j++){
            if(array[e]>array[j])
                e=j;
        }
        if(i!=e) {
            int t; // intercambiar
            t=array[i];
            array[i]=array[e];
            array[e]=t;
        }
    }
}
```

# Ord. Int. – Método por Selección

## Análisis

### Complejidad

*El for interno se ejecuta  $c_i * (n-i)$  veces,  
pero está afectado por el for externo*

$$\rightarrow \sum_{i=1}^{n-1} (c_2 + c_1 * (n-i)) = \sum_{i=1}^{n-1} c_2 + c_1 * \sum_{i=1}^{n-1} (n-i) = c_2 * (n-1) + c_1 * \left( \frac{n * (n-1)}{2} \right) \approx O(n^2)$$

## Características

- *No es estable*

$$2_1 \ 2_2 \ 1 \rightarrow 1 \ 2_2 \ 2_1$$

# Ord. Int. – Método por Intercambio (Burbuja)

## Estrategia

*Tiene inicialmente un arreglo desordenado, en cada pasada lleva (sube) al principio del arreglo desordenado las claves más livianas, repite este proceso hasta que se agotan todos los elementos del arreglo y éste queda ordenado.*

## Ejemplo

*Sea A un arreglo de N elementos, con  $N = 8$ . Ordenar A por el Método de Burbuja*

Paso 0

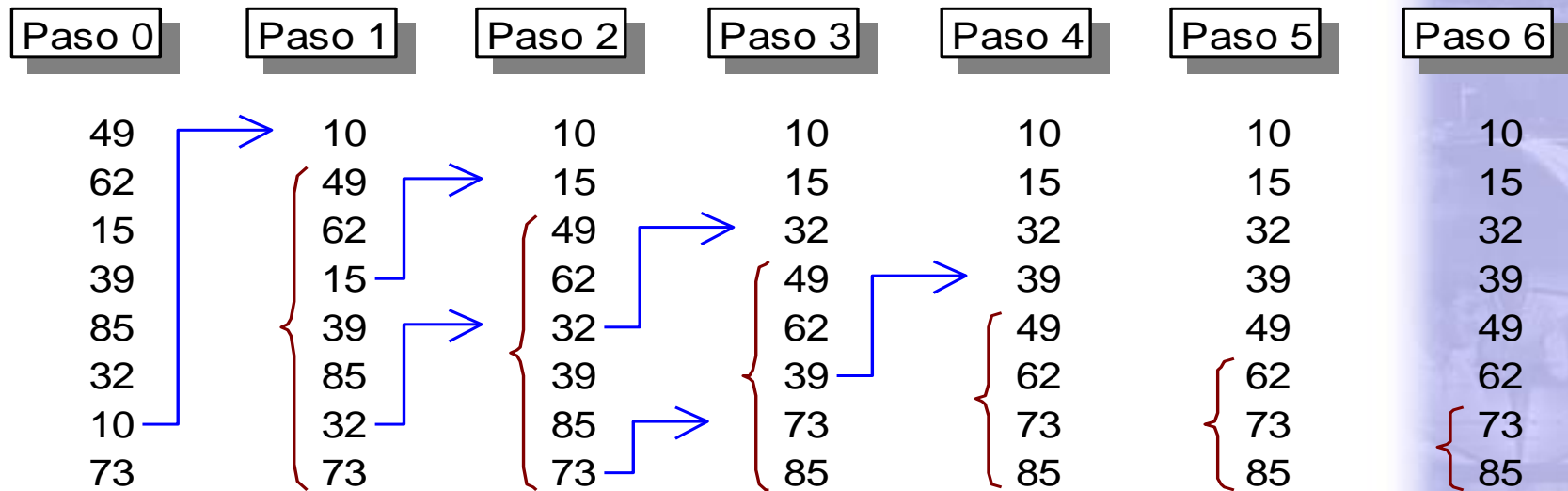
A

1	2	3	4	5	6	7	8
49	62	15	39	85	32	10	73

Conj. desordenado

# Ord. Int. – Método por Intercambio (Burbuja)

## Ejemplo, Cont ...



*Como se observa, en los pasos 5 y 6 aunque el arreglo está ordenado, el método de burbuja continúa repitiendo el proceso hasta agotar todos los elementos del arreglo.*



# Ord. Int. – Método por Intercambio (Burbuja)

## Algoritmo

```
void burbuja(int array[],int size){
    int i,j;

    for(i=1;i<size;i++)
        for(j=size-1;j>=i;--j)
            if(array[j-1]>array[j]){
                int t; // intercambio
                t=array[j-1];
                array[j-1]=array[j];
                array[j]=t;
            }
}
```

# Ord. Int. – Método por Intercambio (Burbuja)

## Análisis

### Complejidad

*El for interno se ejecuta (n-i) veces por cada iteración del for externo.*

*Pero el for externo se ejecuta (n-2) veces, entonces la complejidad sería*

$$\rightarrow \sum_{i=2}^n c_1 * (n-i) = c_1 * \sum_{i=1}^{n-1} (n-i) = c_1 * \left( \frac{n * (n-1)}{2} \right) \approx O(n^2)$$

## Características

*Es estable, el algoritmo conserva el orden original de las claves iguales.*

*Aunque es interesante su estudio, es ineficiente. Adicionalmente, burbuja no se da cuenta si el arreglo ya está ordenado*

# Ord. Int. – Método de Shell (Shellsort)

## Introducción

*Debe su nombre a su inventor D. L. Shell, suele llamarse también **ordenación por disminución de incremento** o **Inserción por incremento decreciente**.*

*Es un refinamiento del método de Inserción Directa.*

## Estrategia

- 1) Se divide la lista general de elementos en  $n/2$  grupos de 2, considerándose un incremento, salto o intervalo de  $n/2$ .*
- 2) Se clasifica cada grupo (elementos apartados  $n/2$  posiciones) por separado, se comparan las parejas de elementos y si no están ordenados se intercambian entre sí. Esto se conoce como clasificación  $n/2$ .*
- 3) Se divide la lista ahora en  $n/4$  grupos de 4 (se agrupan los elementos que están apartados  $n/4$  posiciones) y se ordenan por separado. A este proceso se le llama clasificación  $n/4$ .*
- 4) Se sigue dividiendo la lista de elementos en cada iteración por la mitad, formando en la iteración  $i$   $n/2^i$  grupos de  $2^i$  elementos, se clasifican por separado hasta que queda un solo grupo formado por los  $n$  elementos.*

# Ord. Int. – Método de Shell (Shellsort)

## Ejemplo

Sea  $A$  un arreglo de  $N$  elementos, con  $N = 8$ . Ordenar  $A$  por el Método de Shellsort

Paso 0

A

1	2	3	4	5	6	7	8
49	62	15	39	85	32	10	73

Conj. desordenado

Paso 1

A

1	2	3	4	5	6	7	8
49	62	15	39	85	32	10	73

Paso 3

A

1	2	3	4	5	6	7	8
10	32	15	39	49	62	85	73

Paso 2

A

1	2	3	4	5	6	7	8
49	32	10	39	85	62	15	73

Paso 4

A

1	2	3	4	5	6	7	8
10	15	32	39	49	62	73	85

# Ord. Int. – Método de Shell (Shellsort)

## Algoritmo

```
void shellsort(int array[], int size) {
    int gap, i, j;

    for (gap=size/2; gap>0; gap/=2)
        for (i=gap; i<size; i++)
            for (j=i-gap; j>=0 && array[j+gap]<array[j]; j-=gap) {
                int t; // Intercambiar

                t=array[j];
                array[j]=array[j+gap];
                array[j+gap]=t;
            }
}
```

## Complejidad

*Peor caso:  $O(n^2)$*

*Promedio:  $O(n^{3/2})$*

# Ord. Int. – Método Rápido (Quicksort)

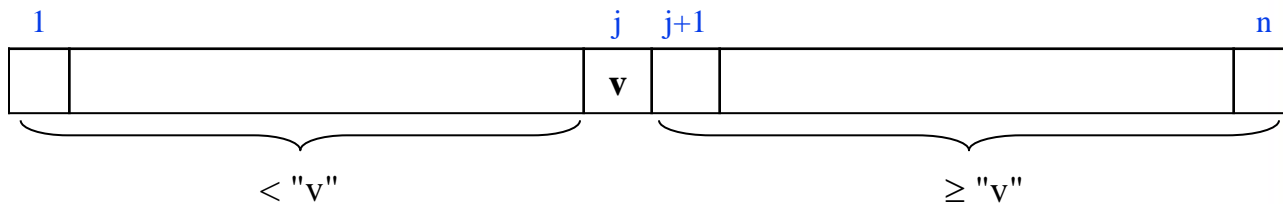
## Estrategia

*Seleccionar un elemento entre los datos que se encuentran en el arreglo, a este elemento lo llamamos pivote, y alrededor de él reorganizamos los datos de tal manera que a su izquierda quedarán los elementos menores al pivote y a la derecha los mayores o iguales*

*Una vez particionado el arreglo de esta forma, se aplica el mismo proceso a cada una de las particiones hasta que no podamos subdividir mas el arreglo.*

## Algoritmo

- 1) *Tomar una clave "v" al azar como pivote.*
- 2) *Colocar a la izquierda del arreglo las claves menores que "v" y a la derecha los elementos mayores o iguales a "v".*



- 3) *Aplicar el mismo proceso a cada una de las particiones*

# Ord. Int. – Método Rápido (Quicksort)

## Refinamiento (Versión Recursiva)

```
void quicksort(int array[], int inicio, int fin){  
    int p;  
  
    if(inicio<fin){  
        p=particion(array, inicio, fin);  
        quicksort(array, inicio, p-1);  
        quicksort(array, p+1, fin);  
    }  
}
```

## Complejidad

*Peor caso:  $O(n^2)$*

*Promedio:  $n \log(n)$*

# Ord. Int. – Método Rápido (Quicksort)

```
int particion(int array[],int inicio, int fin){
    int pivote, inferior=inicio+1, superior=fin;
    pivote=array[inicio];
    do{
        while(array[inferior]<=pivote && inferior<=superior)
            inferior++;
        while(array[superior]>pivote && inferior<=superior)
            superior--;
        if(inferior <= superior){
            swap(array,inferior,superior);
            inferior++;
            superior--;
        }
    }while(inferior<=superior);
    swap(array,inicio,superior);
    return superior;
}

void swap(int array[], int a,int b){
    int t;

    t=array[b];
    array[b]=array[a];
    array[a]=t;
}
```



# Ord. Int. – Método Rápido (Quicksort)

## Ordenamiento

### Ordenación rápida (quicksort)

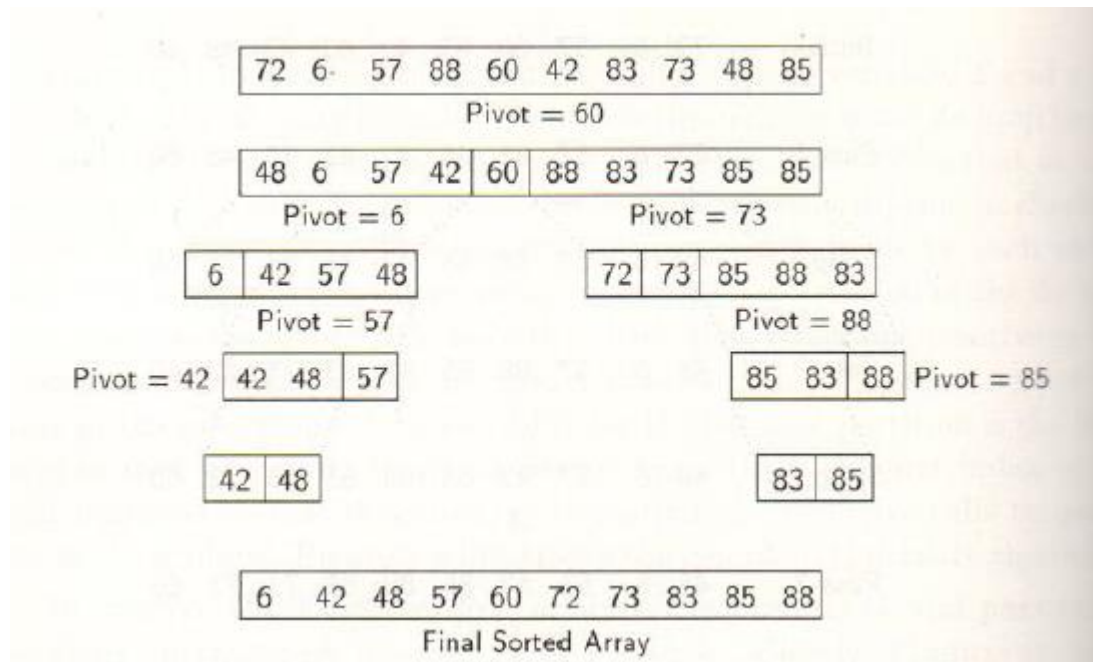
Initial	72	6	57	88	85	42	83	73	48	60	
										r	
Pass 1	72	6	57	88	85	42	83	73	48	60	
										r	
Swap 1	48	6	57	88	85	42	83	73	72	60	
										r	
Pass 2	48	6	57	88	85	42	83	73	72	60	
						r					
Swap 2	48	6	57	42	85	88	83	73	72	60	
						r					
Pass 3	48	6	57	42	85	88	83	73	72	60	
				r							
Swap 3	48	6	57	85	42	88	83	73	72	60	
				r							
Reverse Swap	48	6	57	42		85	88	83	73	72	60
				r							

Pivote = 60

# Ord. Int. – Método Rápido (Quicksort)

## Ordenamiento

## Ordenación rápida (quicksort)

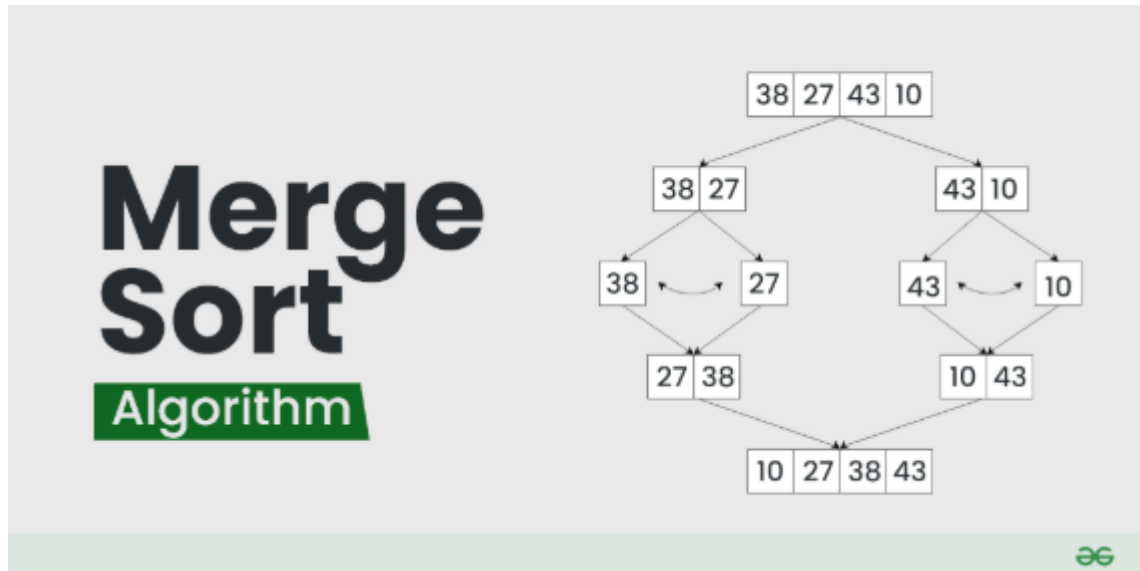


# Mezcla o Mergesort

## Estrategia

*Mezcla o merge en inglés, consiste en la ordenación por combinación es un algoritmo de ordenación que sigue el método «divide y vencerás». Funciona dividiendo recursivamente la matriz de entrada en submatrices más pequeñas, ordenándolas y volviéndolas a unir para obtener la matriz ordenada.*

## Algoritmo



# Mezcla o Mergesort

## Complejidad

*$O(n \log n)$*

*Complejidad de Espacio ( $n$ )*



# Ordenamiento en C++ STL

## Standard Template Library

*La Biblioteca de Plantillas Estándar (STL) de C++ es una potente biblioteca que proporciona un conjunto de estructuras de datos, algoritmos e iteradores de uso común para hacer la programación más fácil, eficiente y reutilizable. Forma parte de la Biblioteca Estándar de C++ y permite a los desarrolladores centrarse en la resolución de problemas en lugar de implementar estructuras de datos y algoritmos de bajo nivel.*



# Ordenamiento en C++ STL

## Standard Template Library

### 1. `std::sort`

- Provided by `<algorithm>`.
- Time Complexity:  $O(n \log n)$  (average-case).
- Uses a hybrid of Quick Sort, Heap Sort, and Insertion Sort.
- Example:

```
cpp Copy code  
  
#include <algorithm>  
#include <vector>  
#include <iostream>  
  
int main() {  
    std::vector<int> arr = {5, 2, 9, 1, 5, 6};  
    std::sort(arr.begin(), arr.end()); // Ascending order  
    for (int x : arr)  
        std::cout << x << " ";  
    return 0;  
}
```




# Ordenamiento en C++ STL

## Standard Template Library

### 2. `std::stable_sort`

- Provided by `<algorithm>`.
- Guarantees stability (preserves the relative order of equal elements).
- Time Complexity:  $O(n \log^2 n)$ .
- Example:

cpp

 Copy code

```
std::stable_sort(arr.begin(), arr.end());
```


# Ordenamiento en C++ STL

## Standard Template Library

### 3. `std::partial_sort`

- Sorts only a part of the array (e.g., finding top  $k$  elements).
- Example:

cpp


 Copy code

```
std::partial_sort(arr.begin(), arr.begin() + k, arr.end());
```

### 4. `std::nth_element`

- Rearranges elements so the  $n$ -th element is in its sorted position.
- Example:

cpp

 Copy code

```
std::nth_element(arr.begin(), arr.begin() + k, arr.end());
```