

# **Sesión 02**

## **Métodos de Ordenamiento Avanzado**

### **Unidad I**



Universidad  
Tecnológica  
del Perú

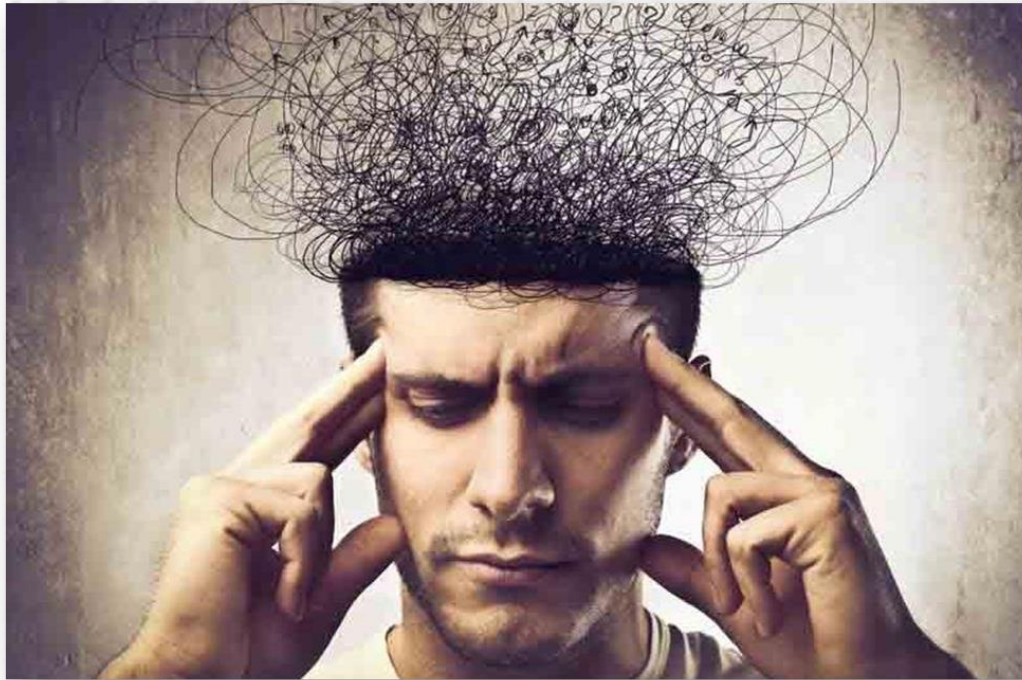
## Temario:

- Conceptos avanzados de ordenamiento en Java
- Algoritmos básicos:
  - Bubble Sort, Selection Sort, Insertion Sort
- Algoritmos avanzados
  - Quick Sort, Shell Sort, Merge Sort.
- Práctica

## Pautas de trabajo

- Los días que tengamos clases debemos conectarnos a través de Zoom.
- La participación de los estudiantes se dará través del **chat de Zoom**.

# Inicio:



**Notación Big O**

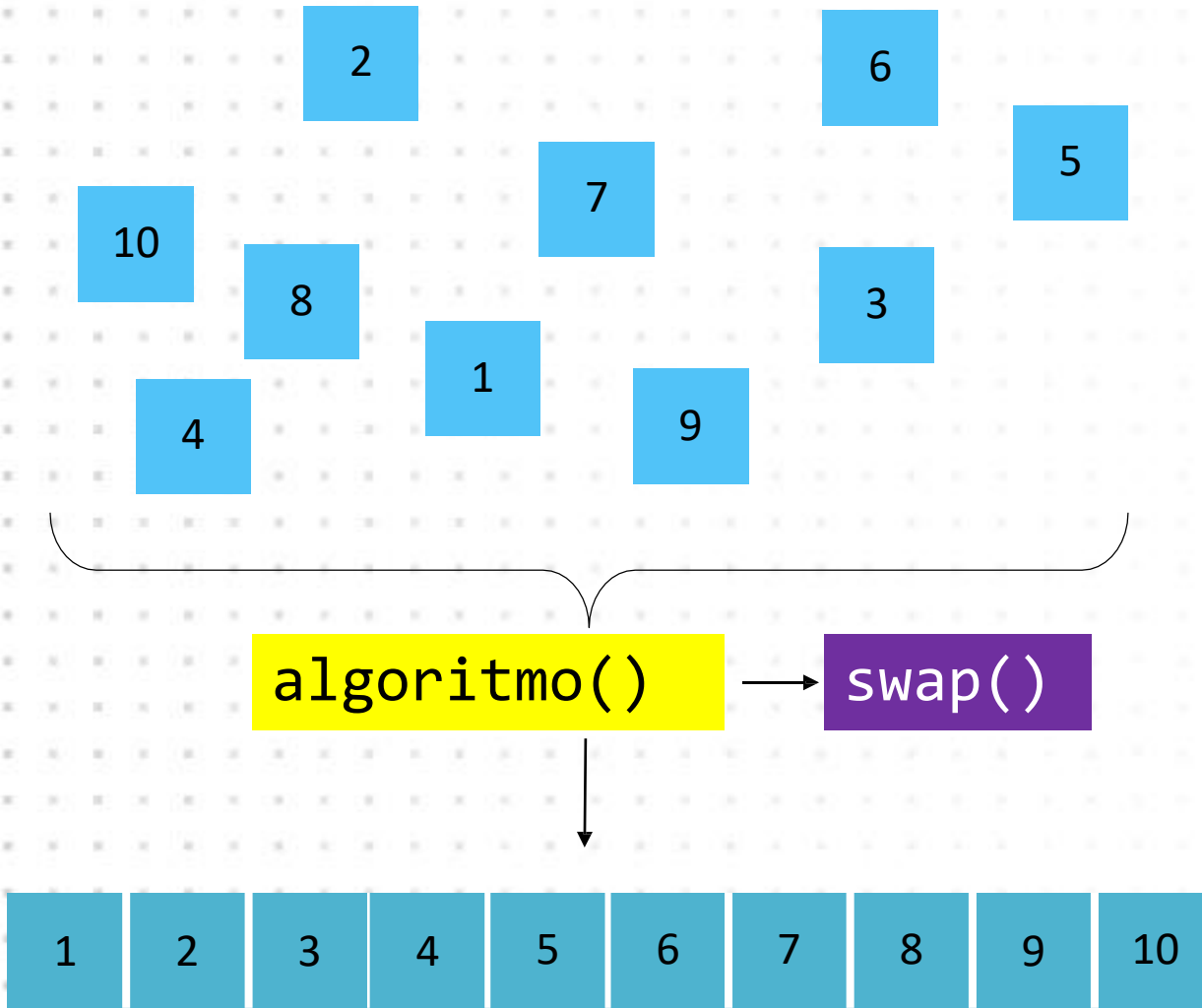
**Método de ordenamiento**

**Bubble Sort**

**Selection Sort**

**InsertionSort**

# Inicio:



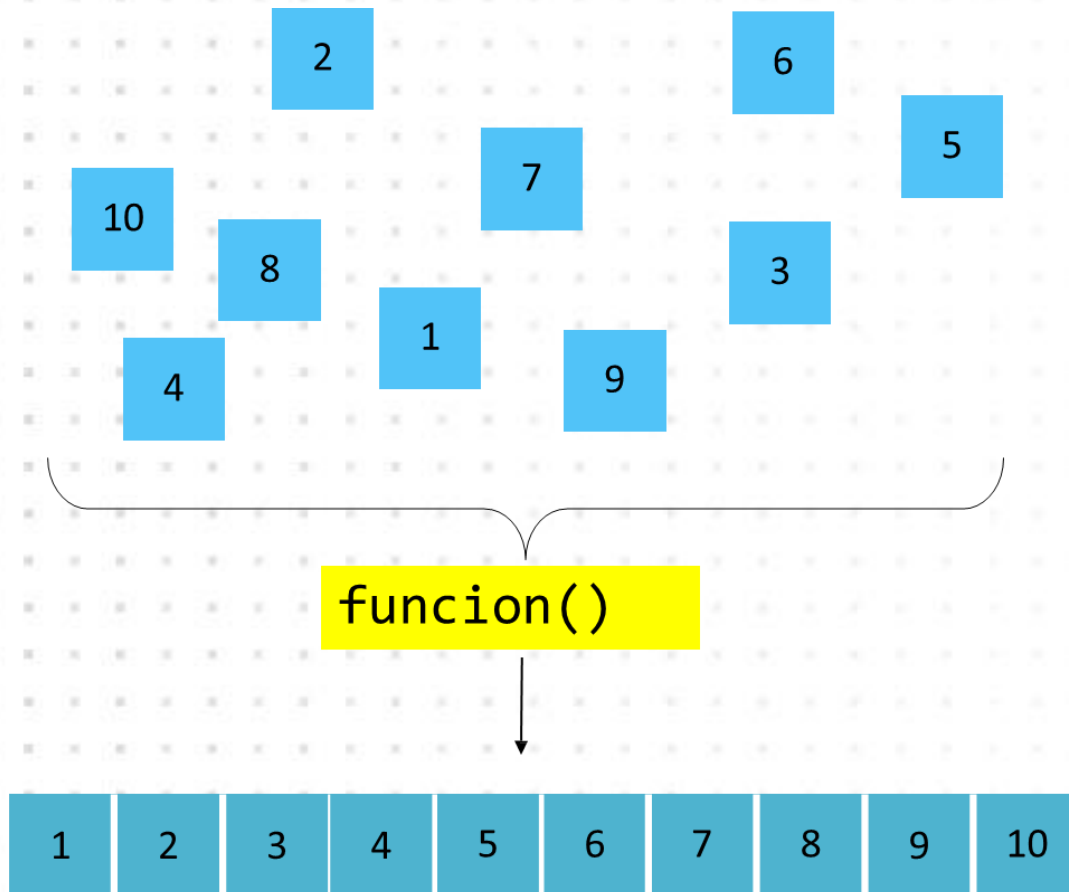
## Mira la imagen y responde

¿Qué hace la función `swap()`?

¿En que situaciones el `algoritmo()` utiliza la función `swap()`?

¿Qué algoritmos de ordenamiento conoces?

# Utilidad:



**Mira la imagen y responde**

¿Cómo se puede llamar al procedimiento que realiza el método `funcion()`?

¿En que situaciones has podido observar este proceso?

¿Qué estructura se está utilizando?

# Utilidad:

## Desarrollar algoritmos para ordenar información de manera rápida y flexible en Java

Mira la imagen y responde

Producto	Tipo	Precio	PorcentajeDscto	Pedidos	Total
Arduino NANO	IOT	75	0.05	35	2493.75
Kaspersky	SOFTWARE	97	0.02	74	7034.44
Arduino UNO	IOT	150	0.1	20	2700
Zoom	SOFTWARE	180	0.01	15	2673
Office 365	SOFTWARE	250	0.07	22	5115
HP DL380	SERVER	5800	0.15	4	19720
IBM SystemX	SERVER	7500	0.12	6	39600

¿Qué información tiene el reporte?

¿Qué orden tiene el reporte?

¿Se puede ordenar por más de un criterio?

¿En que te ayudaría en tu futuro profesional ordenar información de manera rápida y flexible?

## Logro de la sesión:

“Al finalizar la sesión, el estudiante resolverá problemas propuestos utilizando algoritmos de ordenamiento avanzados utilizando arreglos como estructura de datos”.







# Conceptos avanzados de ordenamiento en Java

# Ordenamiento en Java

¿Qué **conceptos** debemos tener en cuenta en Java?

Igualdad

`.equals()`

Comparación

`.compareTo()`

Intercambio

SWAP

# Ordenamiento en Java

## ¿Cómo funciona la **igualdad** en Java?

Igualdad

**.equals()**



Devuelve un **boolean** (true/false)  
Indica si un objeto es igual a otro

nombre1

Juan

nombre1.**equals**(nombre2)

nombre2.**equals**(nombre1)

nombre2

Rosa

=> **false**

=> **false**

nombre1

Juan

nombre1.**equals**(nombre2)

nombre2.**equals**(nombre1)

nombre2

Juan

=> **true**

=> **true**

# Ordenamiento en Java

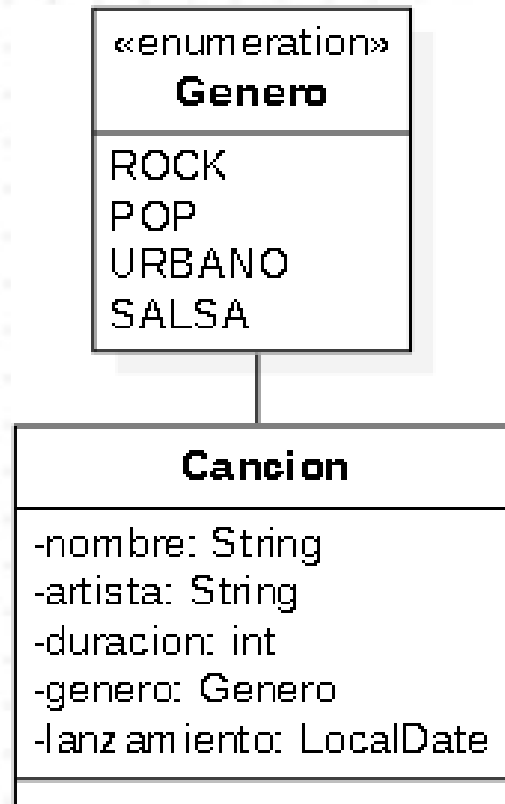
## ¿Cómo funciona la **igualdad** en Java?

Igualdad

**.equals()**



Devuelve un **boolean** (true/false)  
Indica si un objeto es igual a otro



cancion1

Pedro Navaja
Ruben Blades
441
SALSA
07/09/1978

cancion2

Pedro Navaja
Ruben Blades
441
SALSA
07/09/1978

cancion1.**equals**(cancion2) => **false**

# Ordenamiento en Java

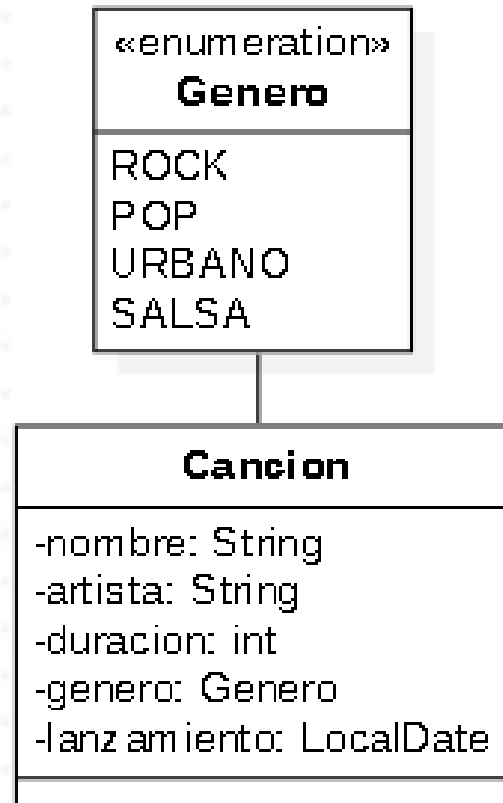
## ¿Cómo funciona la **igualdad** en Java?

Igualdad

**.equals()**



Devuelve un **boolean** (true/false)  
Indica si un objeto es igual a otro  
Cuando queremos comparar  
objetos personalizados, debemos  
implementar el método equals()



```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Cancion)) return false;
    Cancion cancion = (Cancion) o;
    return getNombre()
        .equals(cancion.getNombre())
        && getArtista()
        .equals(cancion.getArtista());
}
```

cancion1.**equals**(cancion2) => **true**

# Ordenamiento en Java

## ¿Cómo funciona la **comparación** en Java?

### Comparación

**o1.compareTo(o2)**



Devuelve 0 si los objetos son iguales  
Devuelve <0 si los o1 es menor a o2  
Devuelve >0 si los o1 es mayor a o2

nombre1

Juan

nombre2

Rosa

nombre1.compareTo(nombre2) => -8 (Es menor)

nombre2.compareTo(nombre1) => 8 (Es mayor)

nombre1

Juan

nombre2

Juan

nombre1.compareTo(nombre2) => 0 (Es igual)

nombre2.compareTo(nombre1) => 0 (Es igual)

# Ordenamiento en Java

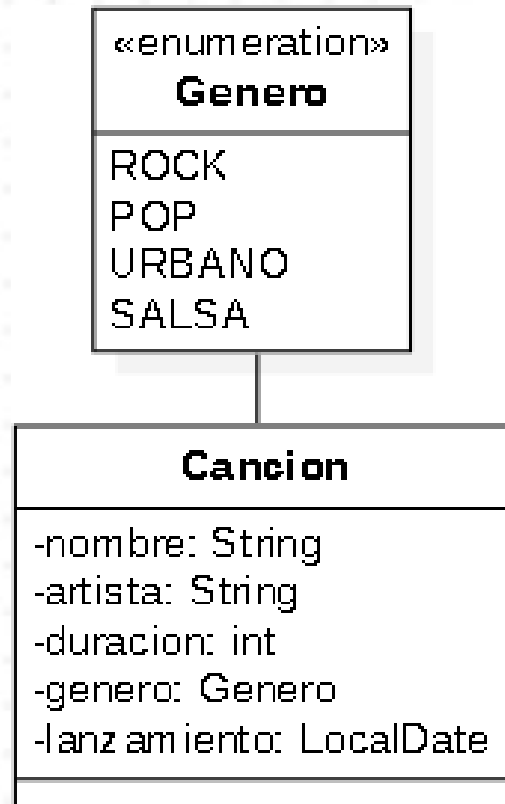
## ¿Cómo funciona la **comparación** en Java?

### Comparación

**o1.compareTo(o2)**



Devuelve 0 si los objetos son iguales  
Devuelve <0 si los o1 es menor a o2  
Devuelve >0 si los o1 es mayor a o2



cancion1

Pedro Navaja
Ruben Blades
441
SALSA
07/09/1978

cancion2

Thriller
Michael Jackson
358
POP
23/01/1984

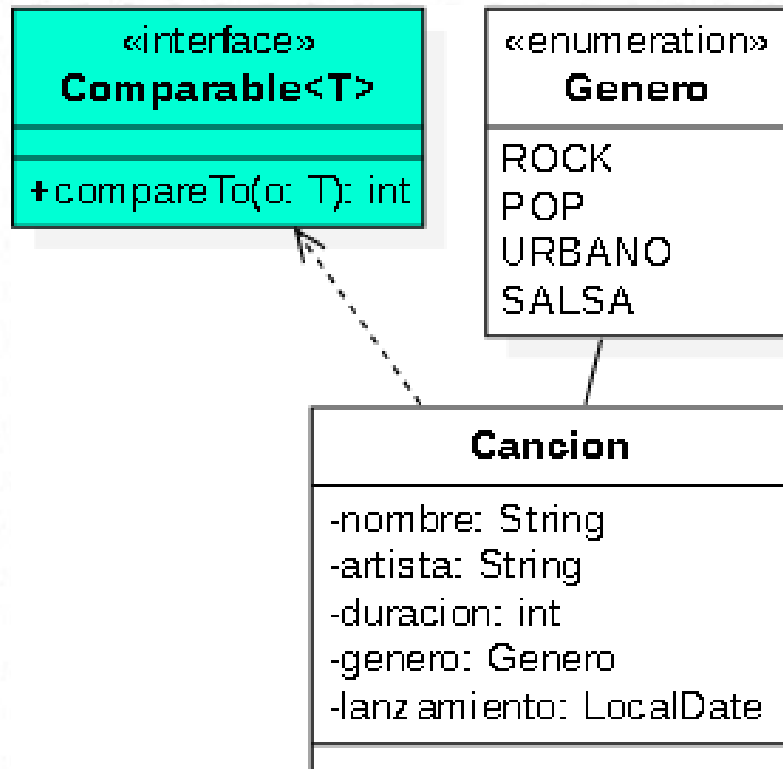
cancion1.**compareTo**(cancion2) => **ERROR**

**ERROR:** Se desconoce el “**Orden Natural**” del objeto Canción



# Ordenamiento en Java

## ¿Cómo funciona la **comparación** en Java?



```

public class Cancion implements Comparable<Cancion> {

    public static final Comparator<Cancion>
        // orden de compación natural: nombre, artista
        CANCION_COMPARATOR_NATURAL_ORDER =
        Comparator.comparing(Cancion::getNombre)
                    .thenComparing(Cancion::getArtista);

    @Override
    public int compareTo(Cancion o) {
        // Orden Natural: nombre, artista
        return CANCION_COMPARATOR_NATURAL_ORDER.compare(this, o);
    }
}
    
```



# Ordenamiento en Java

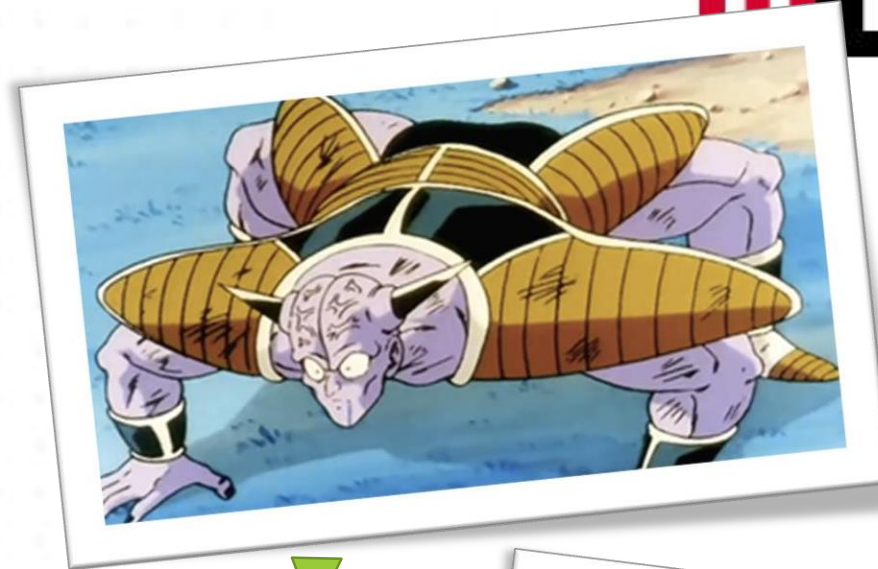
## ¿Qué es Swap?

Intercambio

SWAP



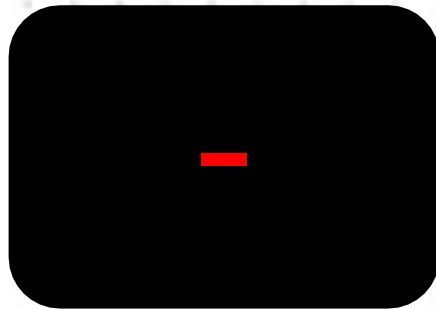
Es un procedimiento que permite intercambiar valores u objetos



# Ordenamiento en Java

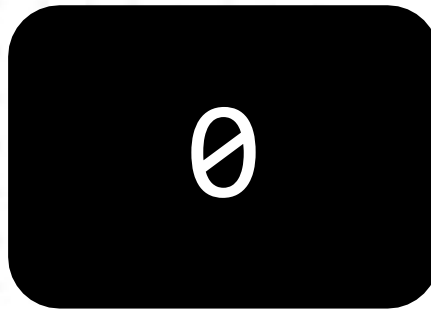
¿Cómo funciona la **comparación** en Java?

$< 0$



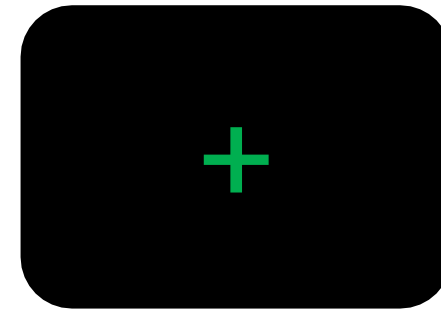
Es Menor

$= 0$



Es igual

$> 0$



Es Mayor

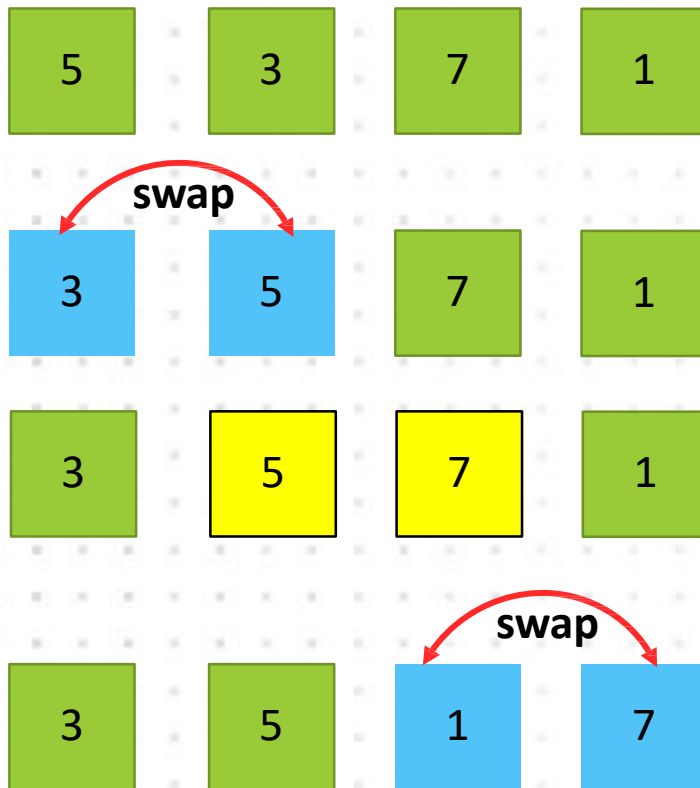


# Algoritmos de ordenamiento básicos

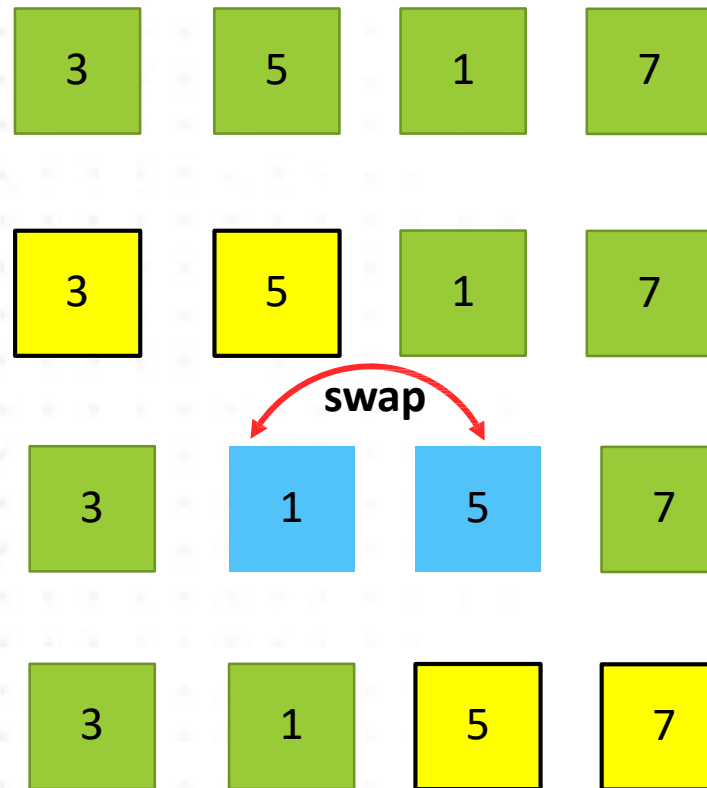
# Métodos de Ordenamiento

## Bubble Sort

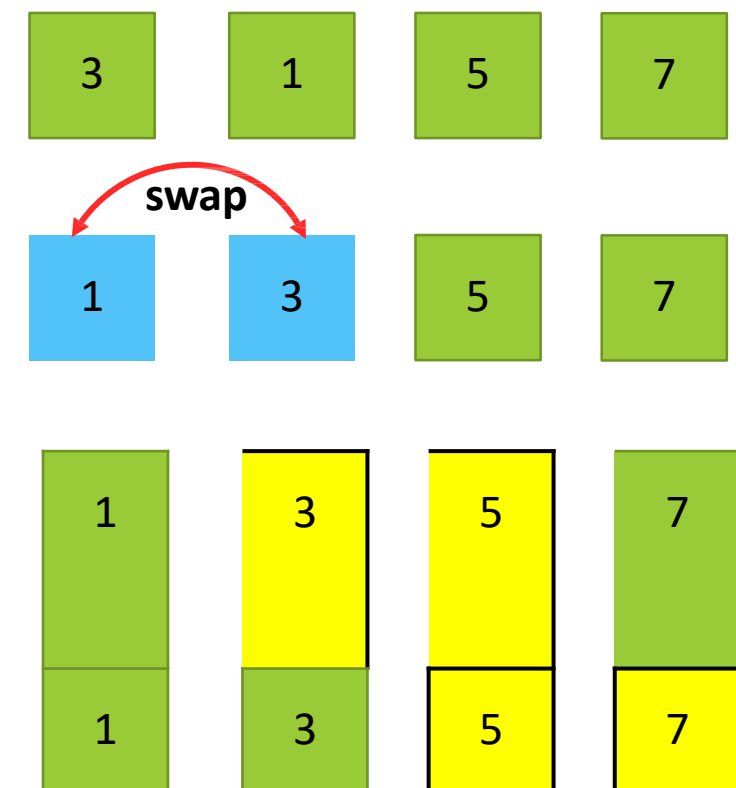
Iteración 1



Iteración 2



Iteración 3



# Métodos de Ordenamiento

## Bubble Sort en Java usando compareTo

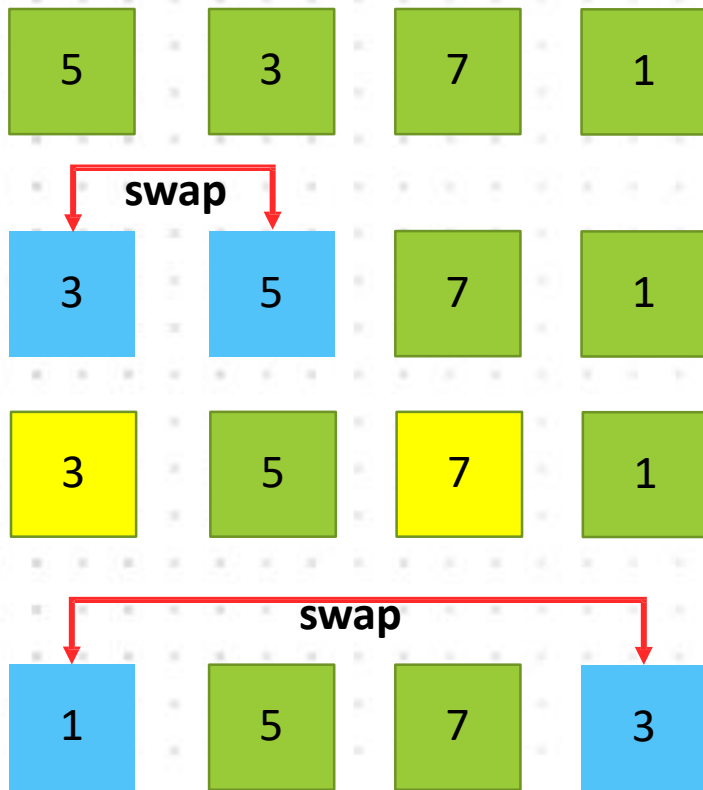
### Pseudocódigo

```
Para i = 0 hasta i < longitud(arreglo)
    Para j = 0 hasta j < longitud(arreglo)-1
        Si (arreglo[j] > arreglo[j+1])
            swap(arreglo, j, j+1)
        FinSi
    FinPara
FinPara
```

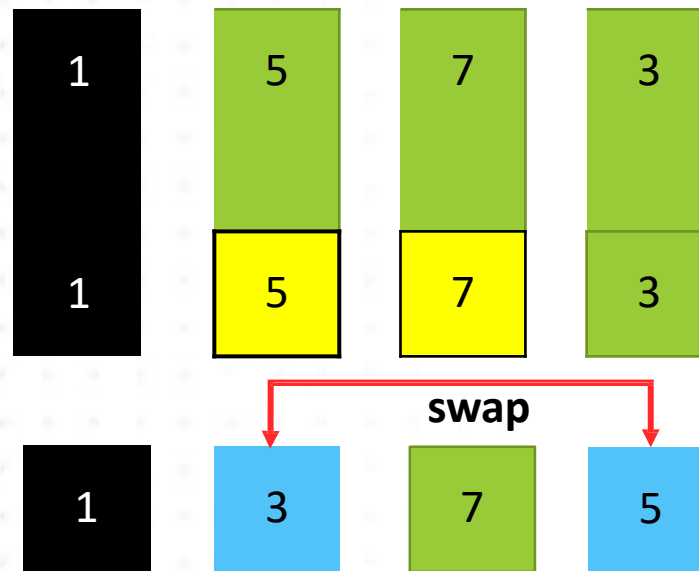
```
public static void bubbleSort(Object[] data){
    for (int i = 0; i < data.length; i++) {
        for (int j = 0; j < data.length-1; j++) {
            if ( ((Comparable) data[j])
                .compareTo(data[j+1]) > 0){
                swap(data, j, j+1);
            }
        }
    }
}
```

# Métodos de Ordenamiento

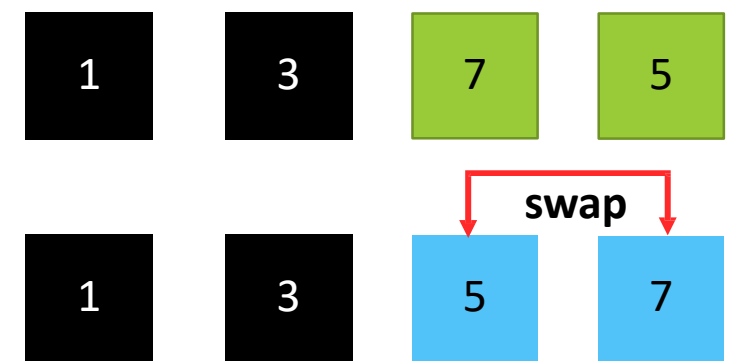
## Selection Sort



Iteración 1



Iteración 2



Iteración 3



# Métodos de Ordenamiento

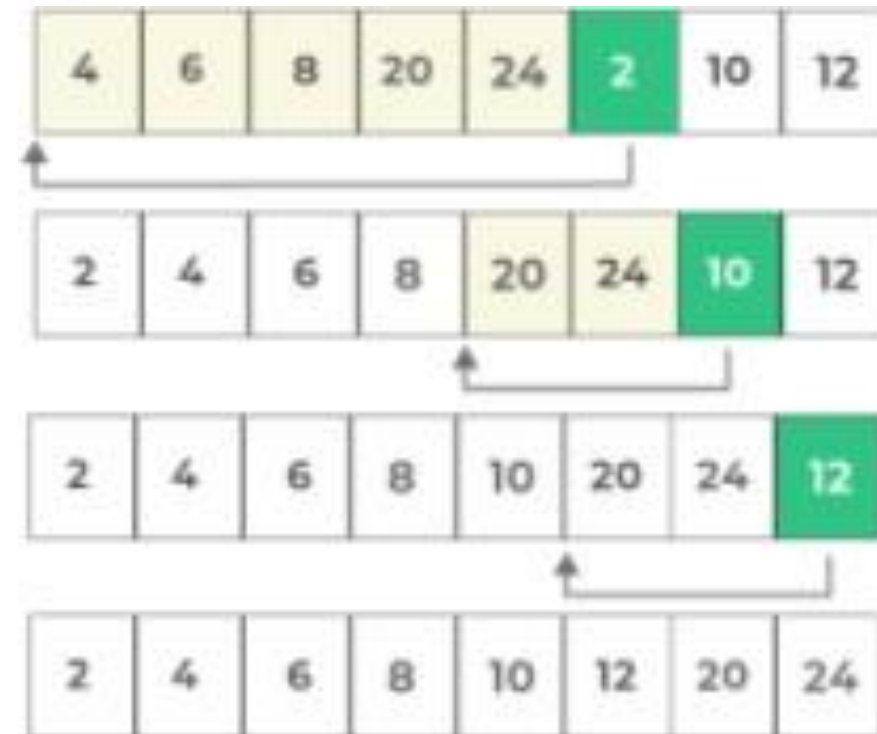
## Selection Sort en Java usando compareTo

```
private static int getIdxMinimo(Object[] data,
                                int inicio){
    int idxMin = inicio;
    for (int i = inicio; i < data.length; i++) {
        if ( ((Comparable) data[i])
            .compareTo(data[idxMin]) < 0){
            idxMin = i;
        }
    }
    return idxMin;
}
```

```
public static void selectionSort(Object[] data,
                                int inicio){
    if (inicio >= data.length) return;
    int min = getIdxMinimo(data, inicio);
    if (min != inicio) swap(data, inicio, min);
    selectionSort(data, inicio+1);
}
```

# Métodos de Ordenamiento

## Insertion Sort





# Métodos de Ordenamiento

## Insertion Sort en Java usando compareTo

```
public static void insertionSort(Object[] data){  
    int i=1;  
    while(i < data.length){  
        int j=i;  
        while (j >0 && ( ((Comparable) data[j-1])  
                        .compareTo(data[j]) > 0) ){  
            swap(data, j, j-1);  
            j--;  
        }  
        i++;  
    }  
}
```



# Algoritmos de ordenamiento avanzados

# Métodos de Ordenamiento

## Merge Sort

- Sigue el enfoque “divide y vencerás”
- Subdivide el array en 2 subarrays de  $n/2$  elementos cada uno de manera recursiva
- Una vez que la división de arrays ya no puede continuar, se realiza el proceso de “fusión” (merge)
- Su complejidad es  $O(n \log n)$



Universidad  
Tecnológica  
del Perú

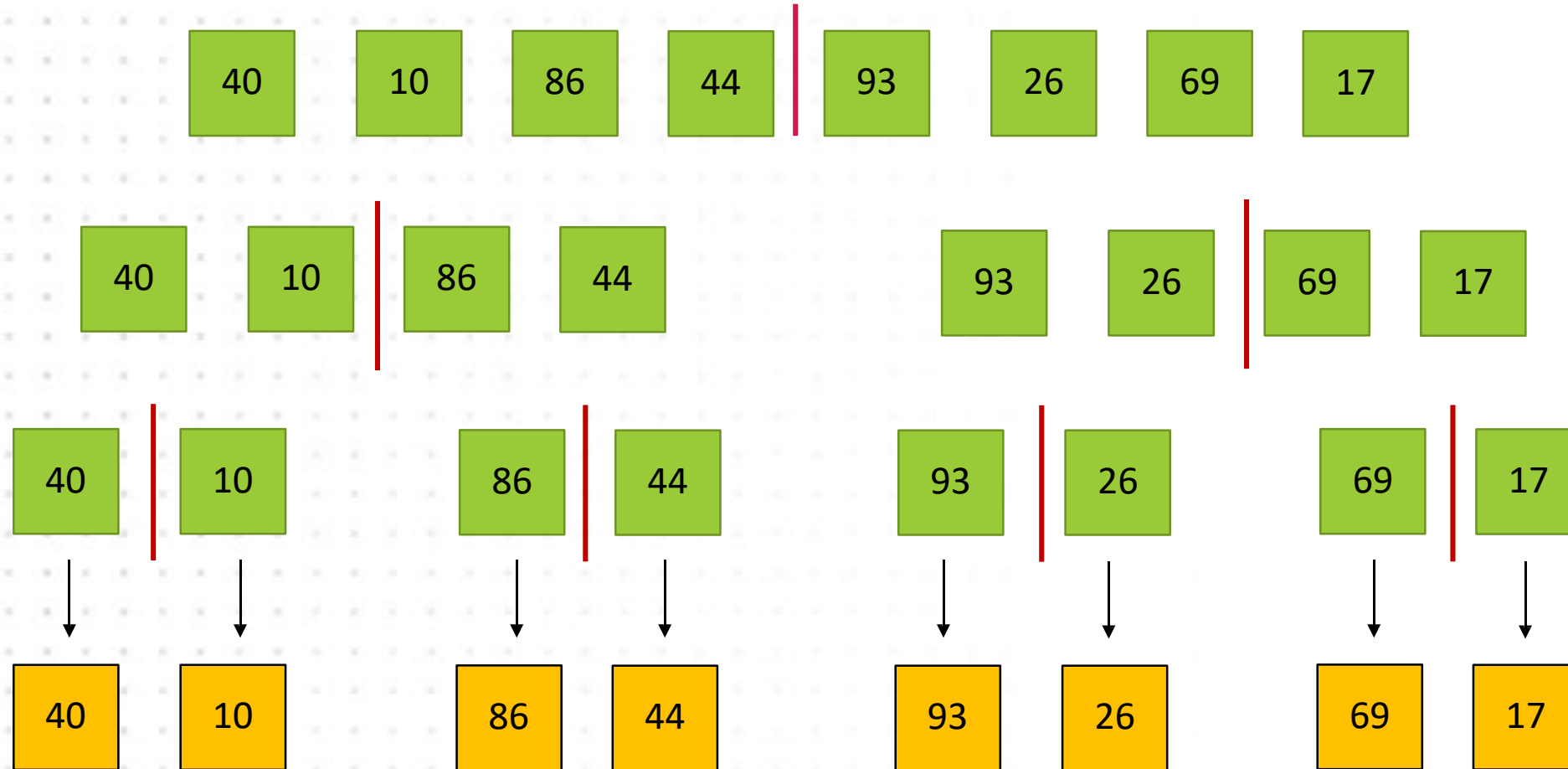


Imágenes extraídas de:  
<https://www.brokenjoysticks.net/wp-content/uploads/2016/01/fusion-e-fredda-dragon-ball.jpg> y  
<https://pics.awwwmemes.com/futubandera-cl-fail-meme-templates-imgflip-50106429.png>  
(Obra de: Akira Toriyama)



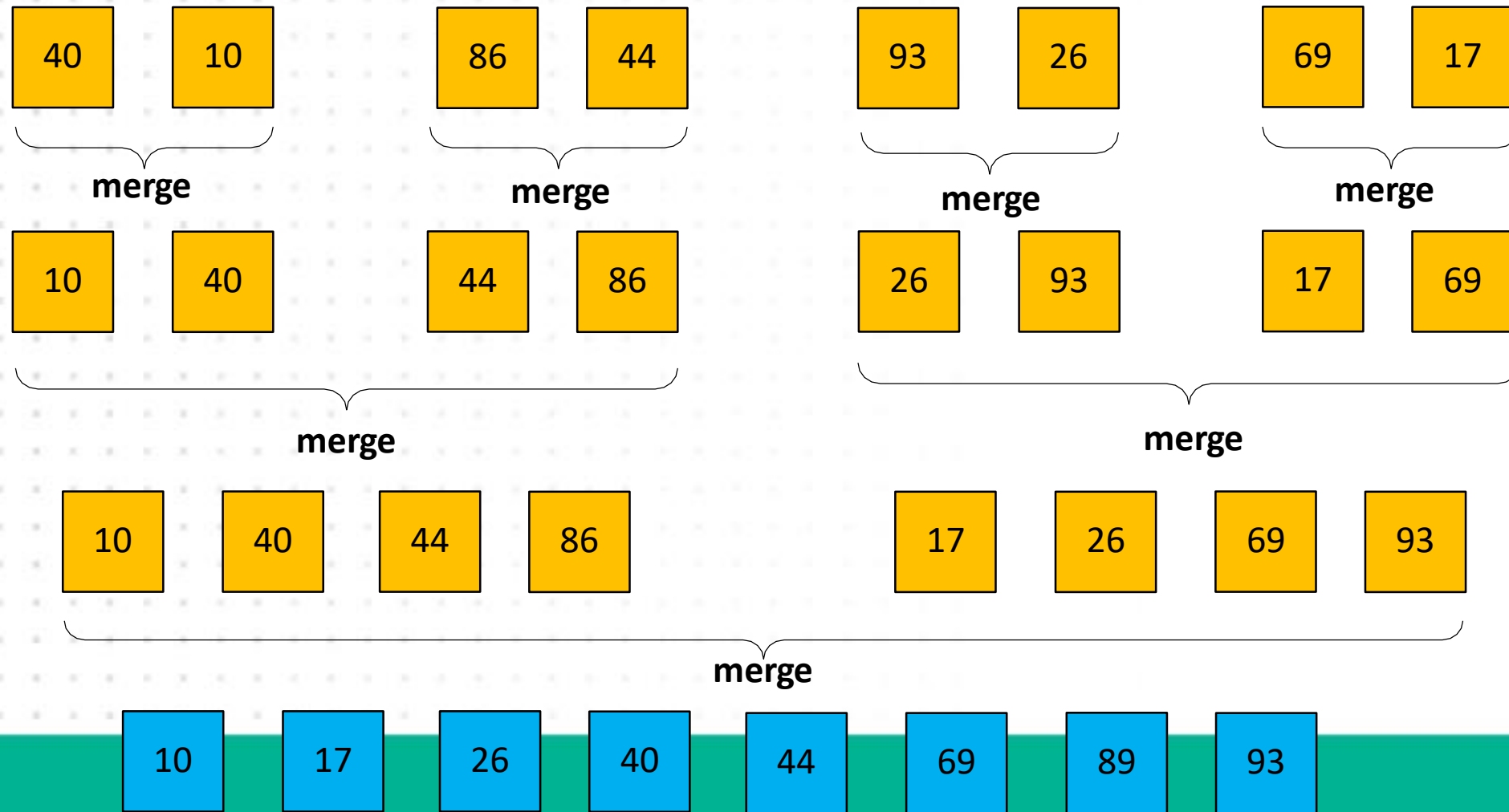
# Métodos de Ordenamiento

## Proceso de división



# Métodos de Ordenamiento

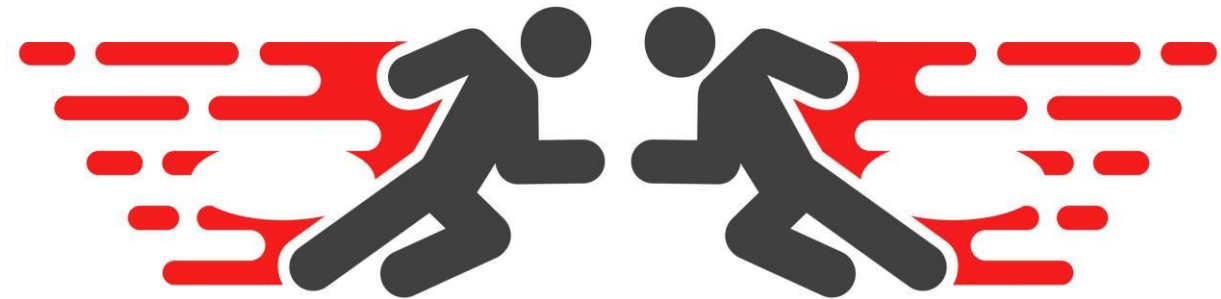
## Proceso de mezclado (merge)



# Métodos de Ordenamiento

## Quick Sort

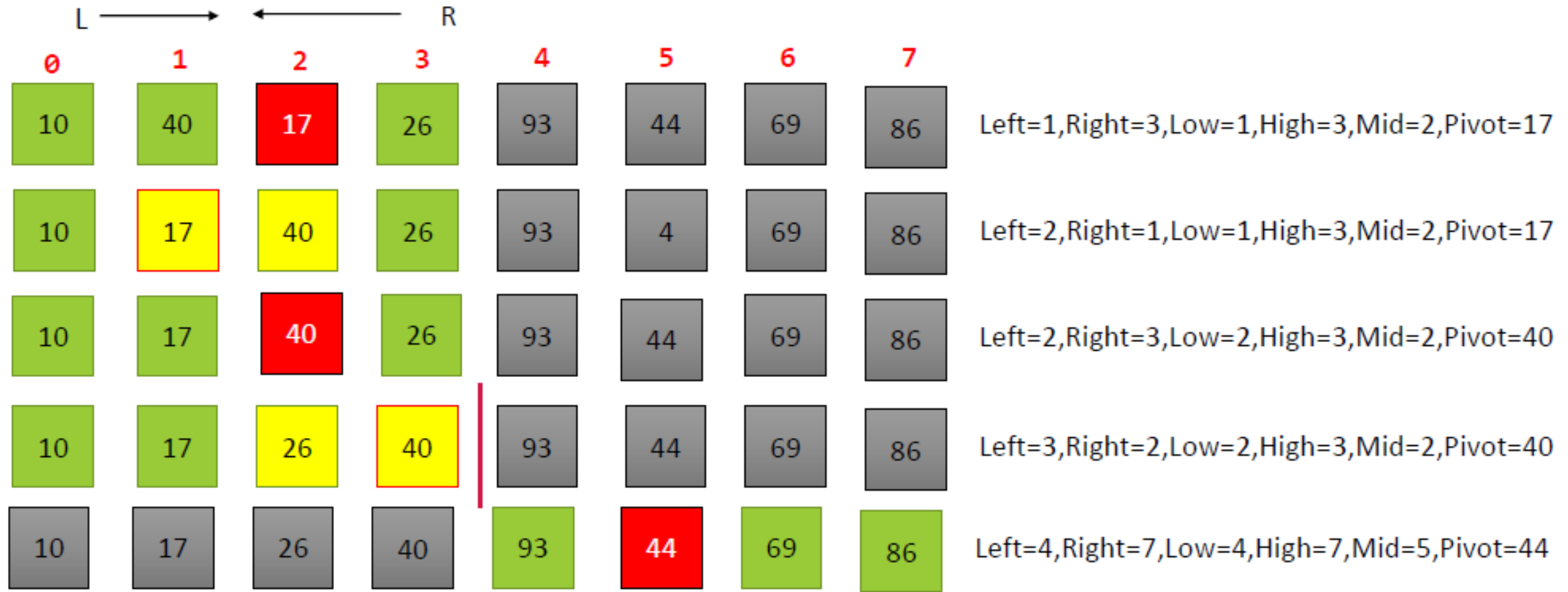
- También sigue el enfoque “divide y vencerás”.  
Subdivide el array en 2 subarrays de  $n/2$  elementos cada uno de manera recursiva
- Se elige un elemento Pivot cercano a la mitad de cada subarray y se utilizan las posiciones Left y Right del array acercándolos al centro y evaluando donde corresponda el swap.
- El método va ordenando las mitades izquierdas de los subarrays, y el algoritmo se ordena de izquierda a derecha
- Su complejidad es  $O(n \log n)$





# Métodos de Ordenamiento

## Quick Sort



# Métodos de Ordenamiento

## Quick Sort

L <span style="margin-left: 100px;">→</span> <span style="margin-left: 100px;">←</span> R								
0	1	2	3	4	5	6	7	
10	40	26	40	44	93	69	86	Left=5,Right=4,Low=4,High=7,Mid=5,Pivot=44
10	17	26	40	44	93	69	86	Left=5,Right=7,Low=5,High=7,Mid=6,Pivot=69
10	17	26	40	44	69	93	86	Left=6,Right=5,Low=5,High=7,Mid=6,Pivot=69
10	17	26	40	44	69	93	86	Left=6,Right=7,Low=6,High=7,Mid=6,Pivot=93
10	17	26	40	44	69	86	93	Left=7,Right=6,Low=6,High=7,Mid=6,Pivot=93
10	17	26	40	44	69	86	93	ORDENADO!



# Métodos de Ordenamiento

## Shell Sort

- Está basado en el algoritmo InsertionSort
- El algoritmo rompe el conjunto original en subconjuntos más pequeños y luego los ordena usando InsertionSort
- Utiliza un intervalo o gap para la creación de subconjuntos
- Su complejidad es  $O(n \log n)$



Imagen extraída de:  
<https://avatars.mds.yandex.net/i?id=0e6c03e457bcc999e7a5b802f59d5da3-5869483-images-thumbs&n=13&exp=1>

# Métodos de Ordenamiento

## Shell Sort

0	1	2	3	4	5	6	7	
40	10	86	44	93	26	69	17	Gap=4
40	10	69	44	93	26	86	17	Gap=4, i=6, j=2
40	10	69	17	93	26	86	44	Gap=4, i=7, j=3
40	10	69	17	93	26	86	44	Fin Gap=4
40	10	69	17	93	26	86	44	Gap=2
40	10	69	17	86	26	93	44	Gap=2, i=6, j=4

# Métodos de Ordenamiento

## Shell Sort

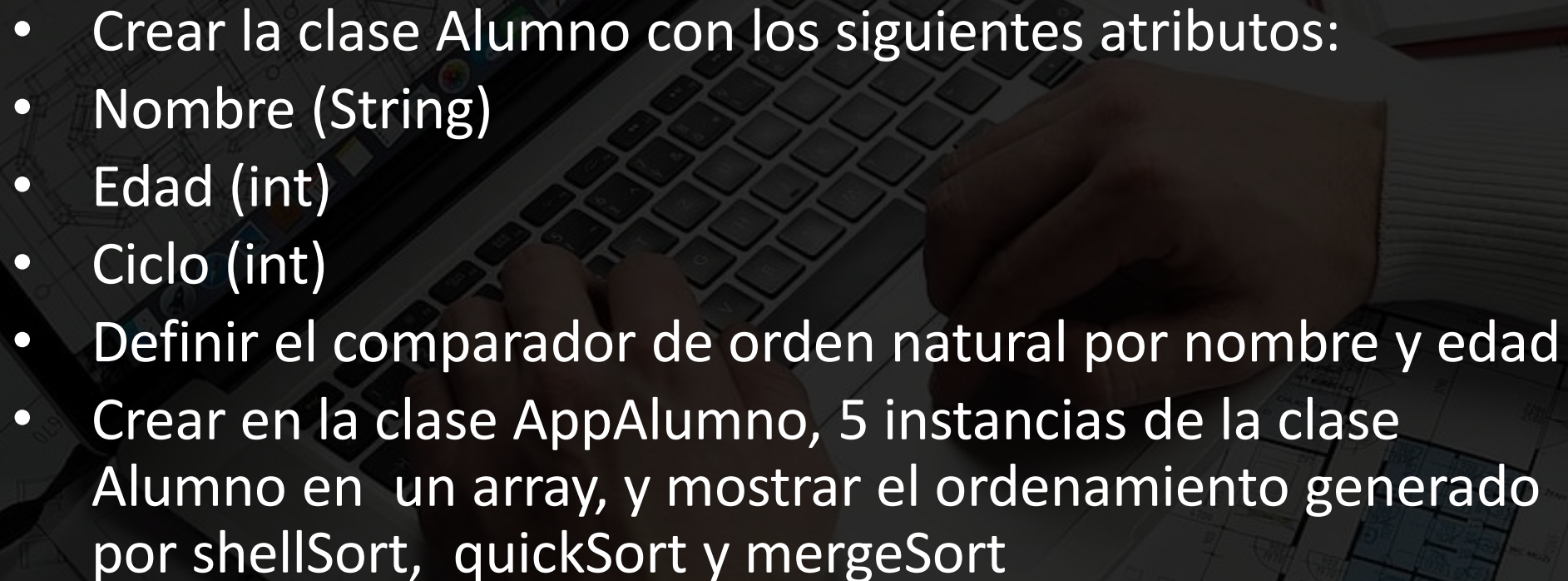
0	1	2	3	4	5	6	7	
40	10	69	17	86	26	93	44	Gap=1
10	40	69	17	86	26	93	44	Gap=1, i=1, j=0
10	17	40	69	86	26	93	44	Gap=1, i=3, j=1
10	17	26	40	69	86	93	44	Gap=1, i=5, j=2
10	17	26	40	44	69	86	93	Gap=1, i=7, j=4
10	17	26	40	44	69	86	93	<b>ORDENADO!</b>

# Consultas





# Practiquemos

- 
- Crear la clase Alumno con los siguientes atributos:
  - Nombre (String)
  - Edad (int)
  - Ciclo (int)
  - Definir el comparador de orden natural por nombre y edad
  - Crear en la clase AppAlumno, 5 instancias de la clase Alumno en un array, y mostrar el ordenamiento generado por shellSort, quickSort y mergeSort

# Tarea

- Dado un arreglo de 7 elementos numéricos, representar gráficamente el proceso de ordenamiento de todos los métodos de ordenamiento avanzado vistos en clase.
- Explicar la tarea en un video (max. 3 min) explicando el ejercicio planteado



# ¿Que hemos aprendido hoy?



- ¿Para que sirve compareTo en Java?
- ¿Para que sirven los algoritmos MergeSort, ShellSort y QuickSort?
- ¿Cuáles de los algoritmos de ordenamiento utilizan el enfoque de “divide y vencerás”?

# Bibliografía

- Tanenbaum & Van Steen (2008). Algoritmos y Estructuras de Datos - Principios y Paradigmas, 2da Edición. Pearson Education
- Khalid A. Mughal & Rolf W. Rasmussen (2017). A Programmer's guide to Java SE 8 Oracle Certified Associate





**Universidad  
Tecnológica  
del Perú**