



**UNIVERSIDAD NACIONAL DE INGENIERÍA**

# **Facultad de Ciencias**

**Escuela Profesional de Ciencia de la Computación**

- **Curso: Fundamentos de Programación**
- **Docente: Américo Chulluncuy Reynoso**

**2025-3**

Sesión 2:

# Algoritmos de Ordenamiento y Búsqueda I

# Contenido

## 1. Ordenamiento

- Algoritmo de burbuja
- Algoritmo de selección
- Algoritmo de inserción

## 2. Búsqueda

- Algoritmo de búsqueda lineal

# 1. Ordenamiento

- Es una operación fundamental en la computación. Permite organizar los datos, facilitando su acceso y procesamiento:
  - ✓ Un conjunto de datos ordenado permite aplicar algoritmos de búsqueda más eficientes (búsqueda binaria). Ejemplo: un directorio telefónico está ordenado alfabéticamente.
  - ✓ Los datos ordenados facilitan la interpretación y toma de decisiones. Ejemplo: En estadísticas, ordenar datos ayuda a calcular la mediana o el percentil.
  - ✓ Algoritmos de aprendizaje automático y minería de datos suelen requerir datos ordenados para mejorar la eficiencia y precisión de los modelos.
- **Ejercicio:** Investiga cómo los motores de búsqueda como Google utilizan algoritmos de ordenamiento para mostrar resultados relevantes en cuestión de milisegundos.

# Convenciones

- En esta presentación asumiremos que los elementos de los arreglos son números enteros y se ordenará en forma ascendente (de menor a mayor).
- Para **intercambiar** dos elementos en un arreglo es necesario utilizar una variable temporal (temp) para guardar una copia del primer elemento, asignarle el segundo al primero y el temporal al segundo.

**Ejercicio:** implemente la función intercambio

- Si el arreglo de nombre lista tiene N elementos, el primer elemento es lista[0] y el último es lista[N-1].

# Algoritmo de Burbuja (Bubble sort)

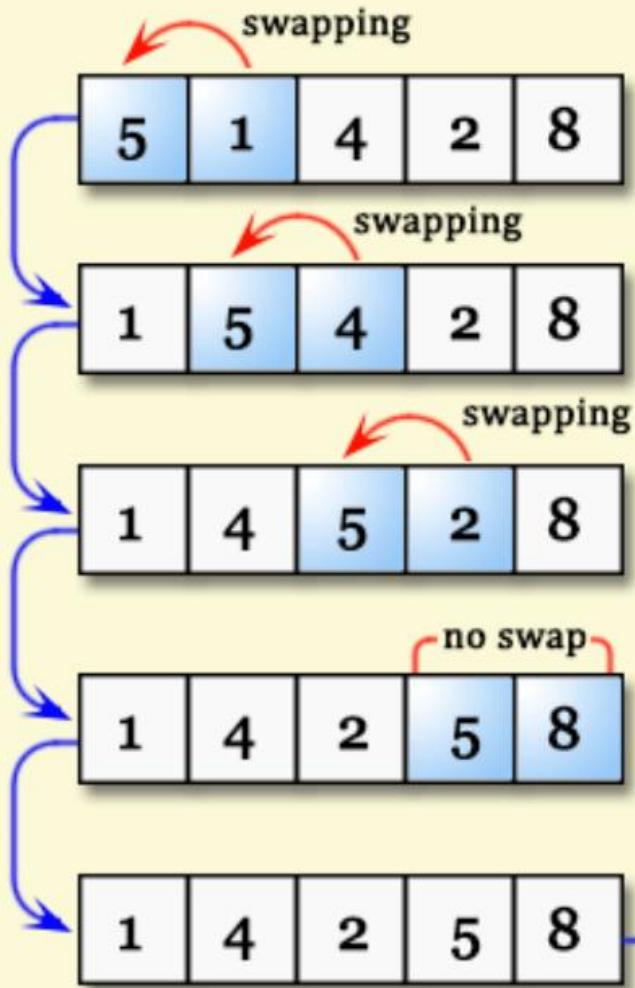
- Compara pares adyacentes y los intercambia si están en el orden incorrecto.
- Al final de cada iteración (pasada), el mayor elemento aparece al final del arreglo (“burbuja”). En general se requieren  $N-1$  pasadas para un arreglo de tamaño  $N$
- El algoritmo puede optimizarse teniendo en cuenta los intercambios. Si en una pasada no hay intercambio significa que el arreglo ya está ordenado.
  - ✓ Se suele utilizar una variable que indica si ocurre o no intercambios. En caso no ocurra intercambio se debe terminar la iteración.

## Ejemplo: algoritmo de burbuja

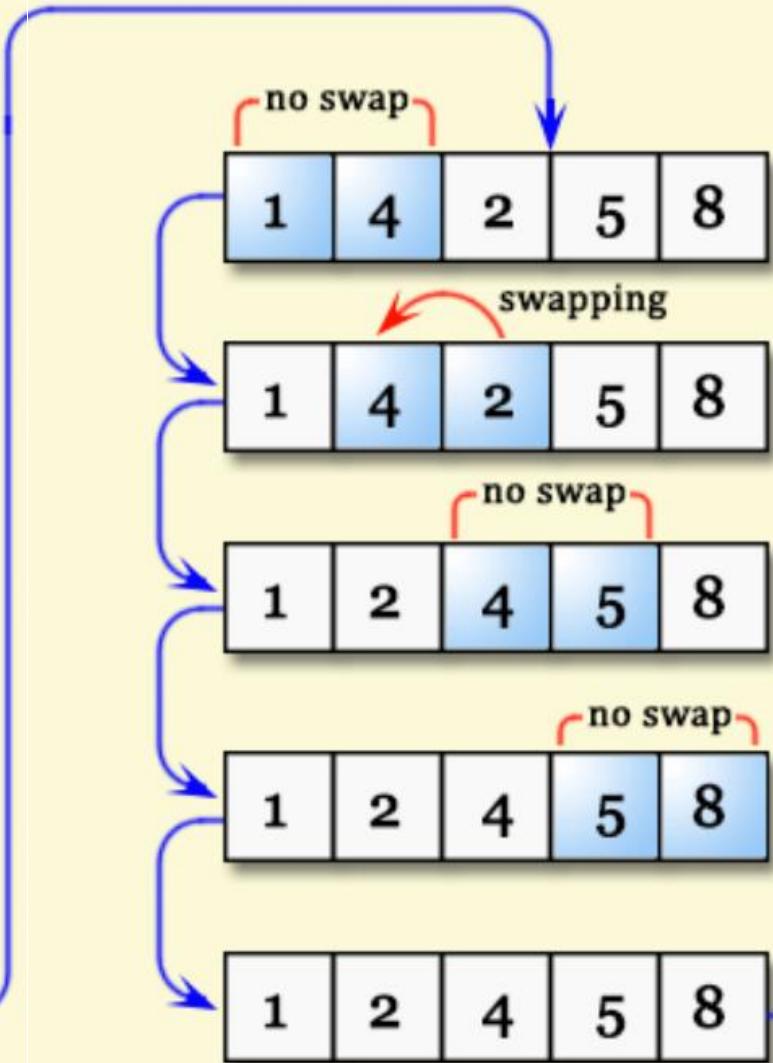
- Arreglo desordenado:

5	1	4	2	8
---	---	---	---	---

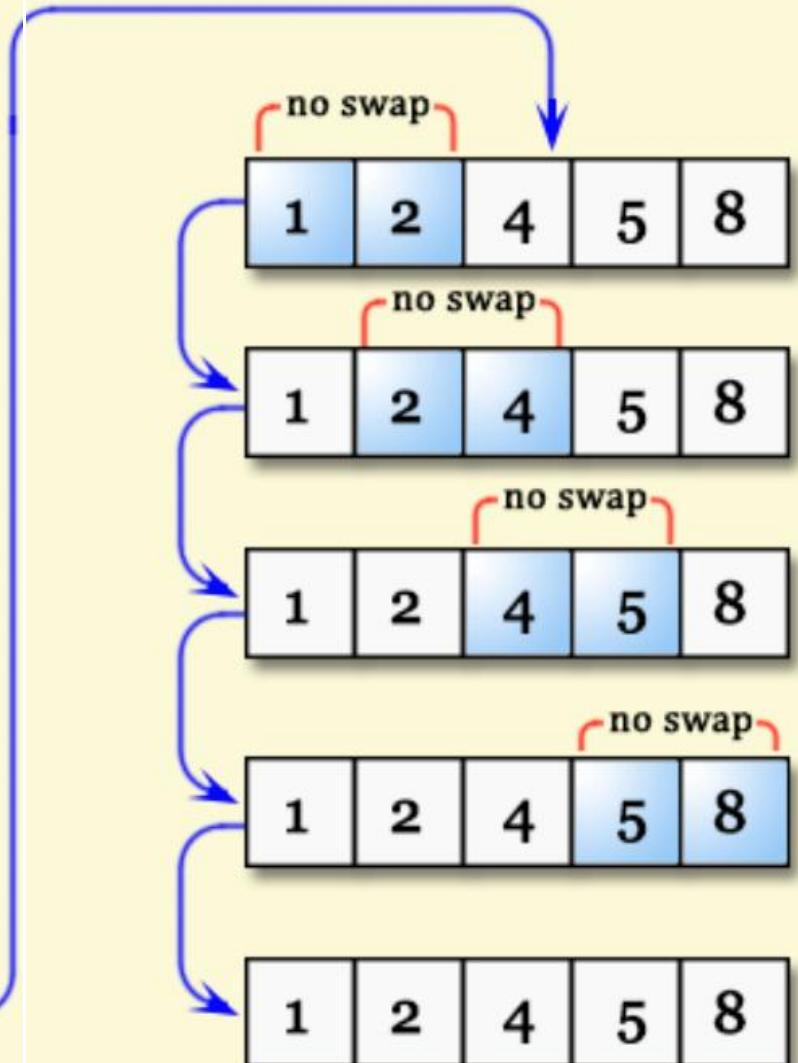
First Pass



Second Pass



Third Pass



# Ejercicios

- Implementar el algoritmo de burbuja. Utilice funciones.
- Implementar una variante que detecte si el arreglo ya está ordenado y termine antes de tiempo.
- Escribir una versión utilizando recursividad.
- Escribir una versión para cadenas

```
void burbuja(int arr[], int n){
    for(int i = 0; i < n - 1; ++i){
        for(int j = 0; j < n - 1 - i; ++j){
            if(arr[j] > arr[j+1]){
                swap(arr[j], arr[j+1]);
            }
        }
    }
}
```

# Algoritmo de Selección (Selection Sort)

- Encuentra el menor elemento del arreglo y lo intercambia con el primer elemento
- Buscar el siguiente menor elemento en el resto del arreglo y lo intercambia con el segundo elemento.
- En general, en cada pasada selecciona el menor elemento entre una posición  $i$  y el final del arreglo e intercambia el mínimo con el elemento de la posición  $i$ .
- Si el arreglo tiene  $N$  elementos, el proceso se repite  $N-1$  veces, ya que el último elemento no se compara.

# Ejemplo: Algoritmo de Selección

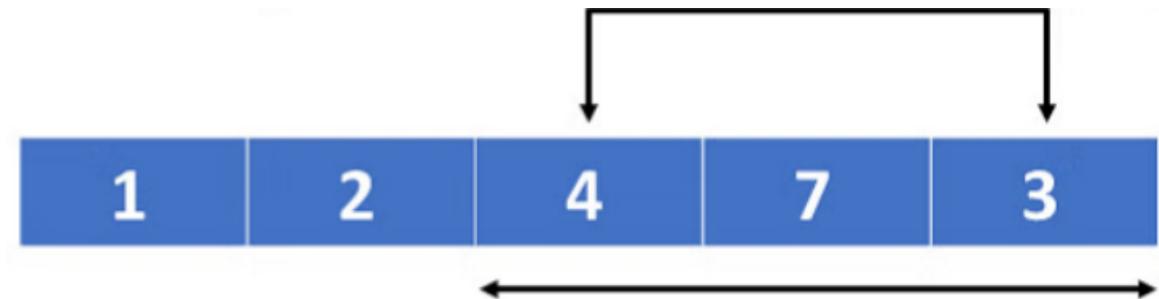
- Arreglo desordenado:

7	1	4	2	3
---	---	---	---	---

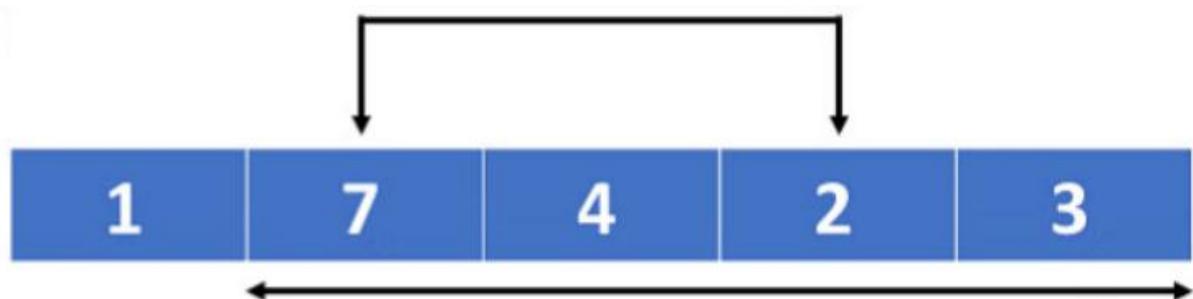
### Pasada 1



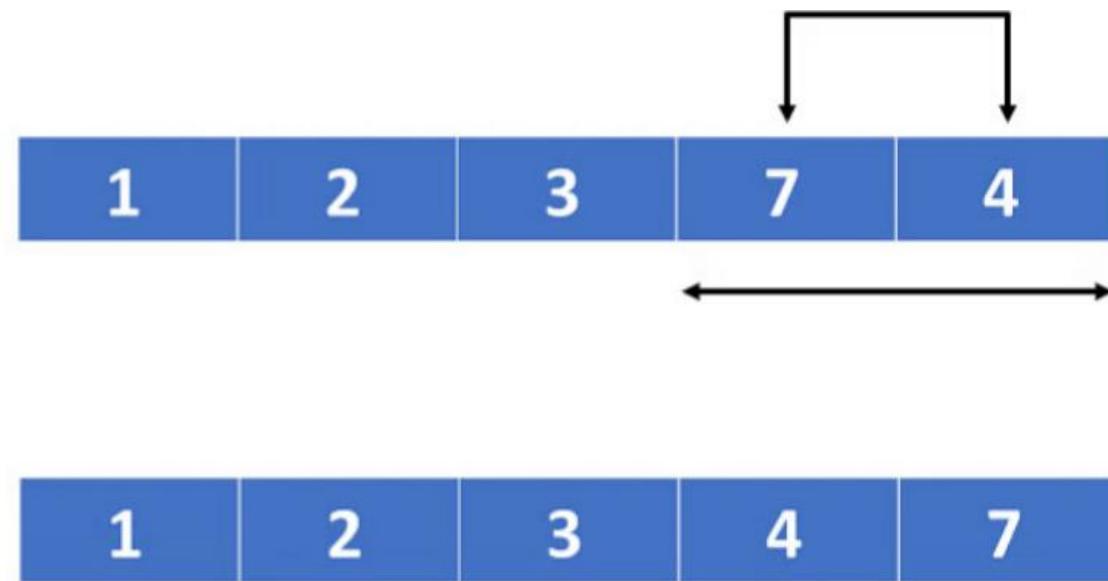
### Pasada 3



### Pasada 2



### Pasada 4

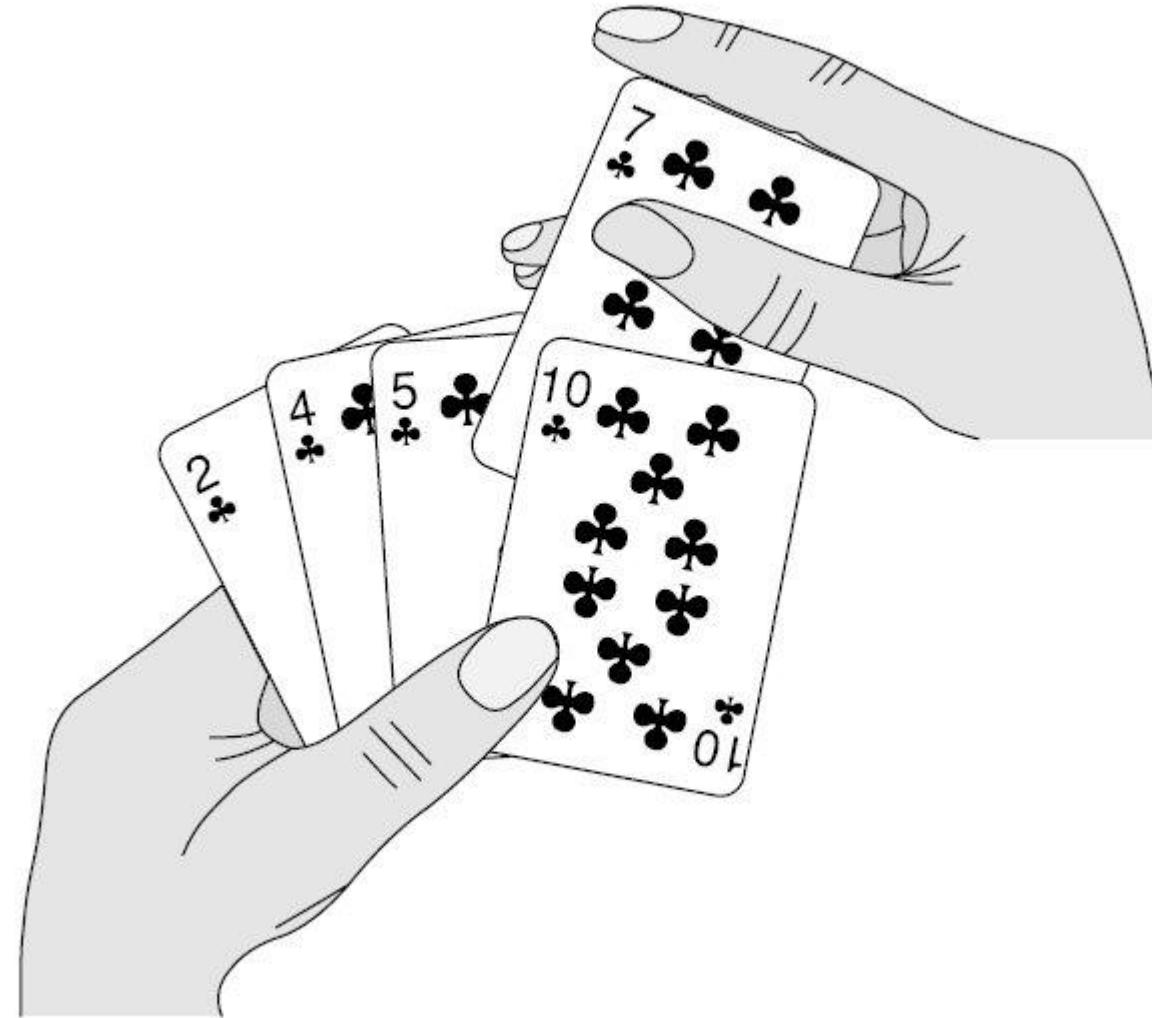


# Ejercicios

- Escribir una función que implemente el algoritmo de selección
- Escribir una versión recursiva del algoritmo.
- Escribir una versión para cadenas.
- Verificar los cambios que deben realizarse para un ordenamiento decreciente.

```
void seleccion(int arr[], int n){
    for(int i = 0; i < n - 1; ++i){
        int indMin = i;
        for(int j = i + 1; j < n; ++j){
            if(arr[j] < arr[indMin]){
                indMin = j;
            }
        }
        swap(arr[i], arr[indMin]);
    }
}
```

# Algoritmo de Inserción (Insertion Sort)



## 4. Algoritmo de Inserción (Insertion Sort)

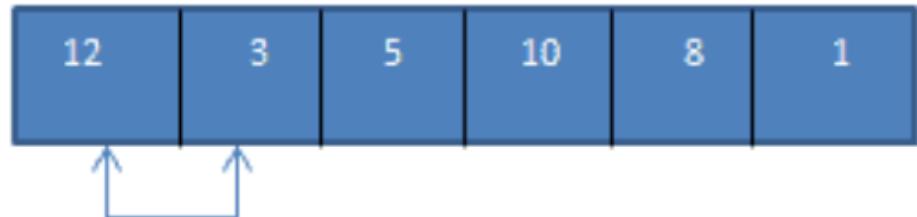
- El algoritmo se asemeja al hecho de insertar una carta en su posición correcta dentro de una lista que ya está ordenada.
- El primer elemento lista[0] se considera ordenado (una sola carta).
- Se inserta lista[1] en la posición correcta (delante o detrás de lista[0]) y así sucesivamente.
- En general, toma cada elemento y lo inserta en la posición correcta dentro de la parte ordenada.
- Para un arreglo de N elementos, deben hacerse N-1 pasadas, pues el primer elemento ya está ordenado.

# Ejemplo: Algoritmo de Inserción

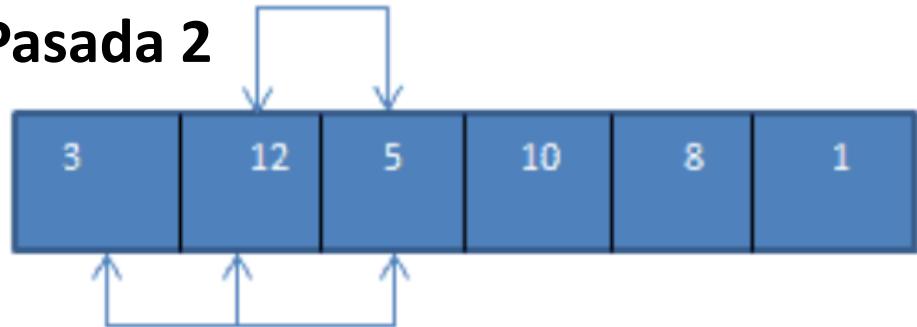
- Arreglo a desordenado:

12	3	5	10	8	1
----	---	---	----	---	---

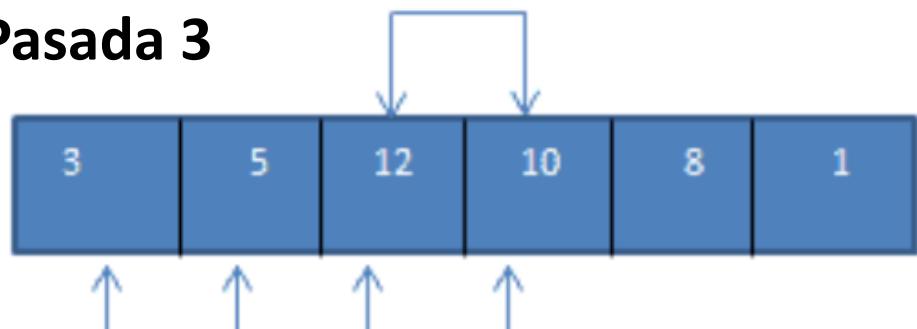
**Pasada 1**



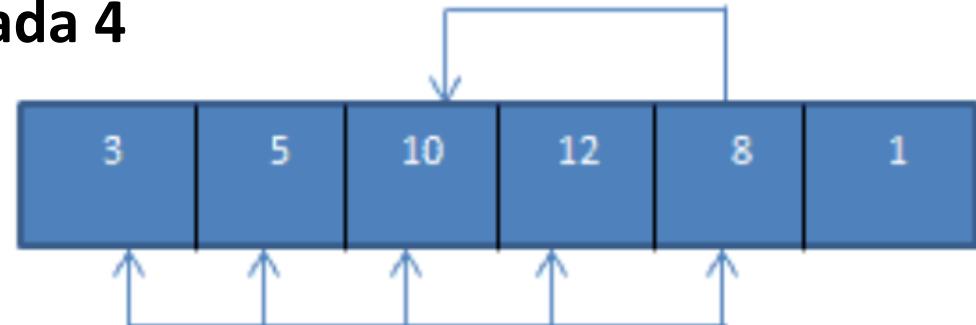
**Pasada 2**



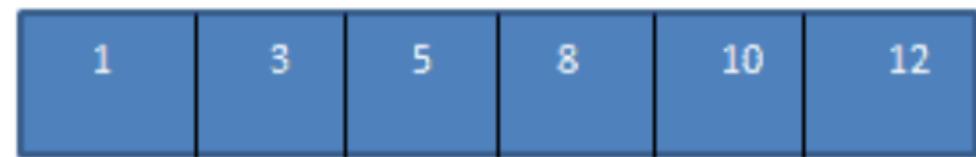
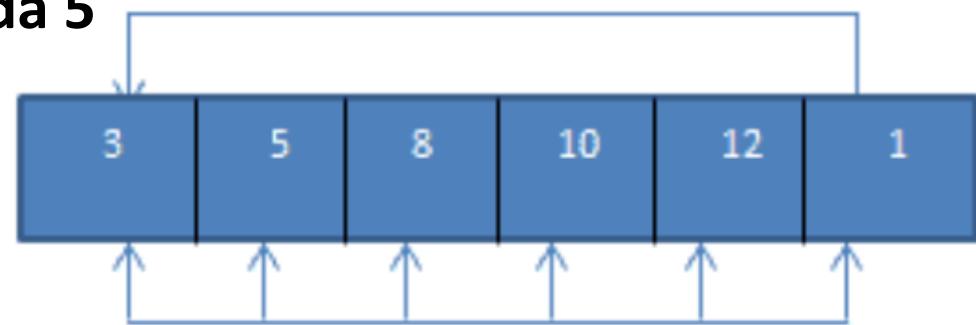
**Pasada 3**



**Pasada 4**



**Pasada 5**



# Ejercicios

- Utilice funciones para implementar el algoritmo de inserción
- Escribir una versión para cadenas.
- Verificar los cambios que deben hacerse para un ordenamiento decreciente.
- Analice una versión recursiva.

```
void insercion(int arr[], int n){
    for(int i = 1; i < n ; ++i){
        int temp = arr[i];
        int j = i-1;
        while(j>=0 && arr[j] > temp){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
}
```

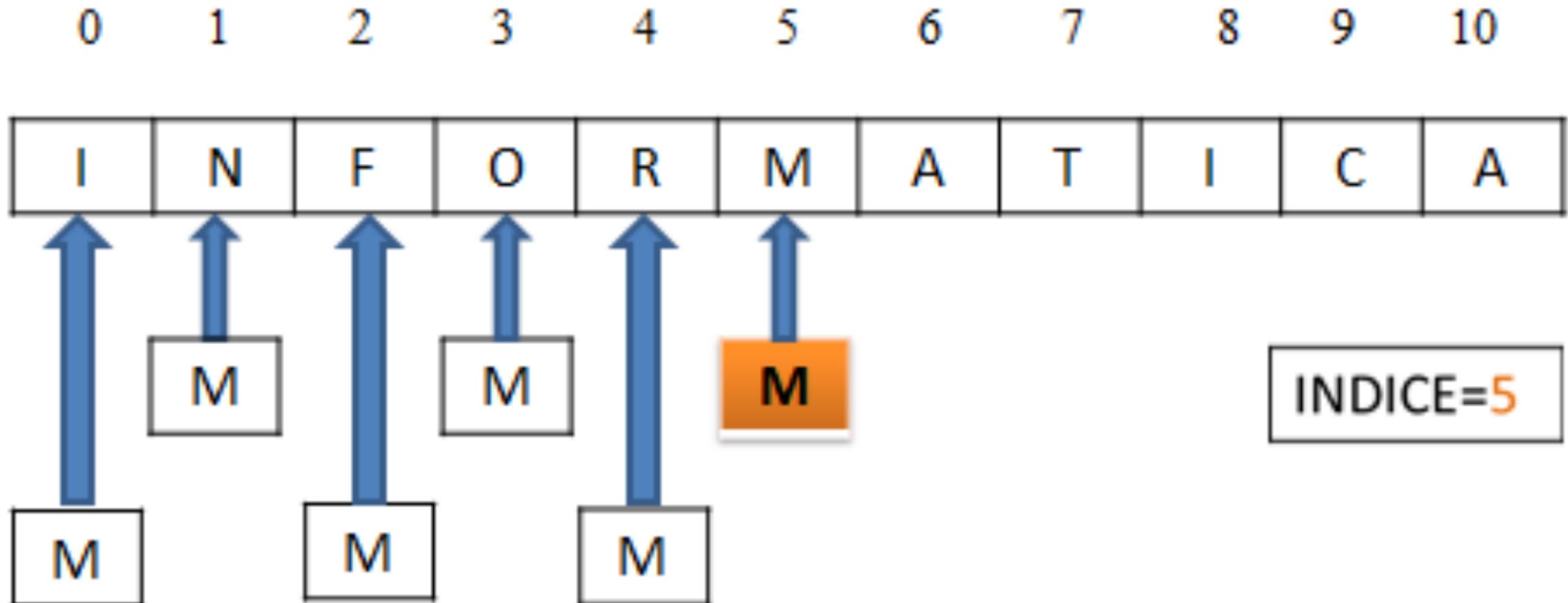
## 2. Búsqueda

- Es una de las operaciones más frecuentes en informática:
  - ✓ Permite recuperar datos de grandes volúmenes de información de manera eficiente.
  - ✓ Las aplicaciones requieren respuestas rápidas para búsquedas en bases de datos y archivos. Ejemplo: App de una tienda en línea, bibliotecas digitales.
  - ✓ Se usa en detección de patrones, reconocimiento de imágenes y en sistemas de seguridad para verificar identidades o detectar amenazas.
- **Ejercicio:** Investigar problemas reales en el que una búsqueda rápida sea crucial. ¿Cómo mejorarías la eficiencia de la búsqueda en ese caso?

# Algoritmo de Búsqueda Lineal

- Recorre el arreglo comparando uno a uno cada elemento hasta encontrar el valor buscado.
- Se comporta bien con arreglos pequeños o desordenados.

# Búsqueda lineal



# Ejercicios

- Implemente el algoritmo de búsqueda lineal utilizando funciones
- Implementar una versión recursiva del algoritmo de búsqueda lineal.
- Modificar la función para contar cuántas veces aparece un número en un arreglo.
- Escribir una versión para cadenas.
- Implementar un programa que permita encontrar la última ocurrencia de un determinado carácter en una cadena

```
int busquedaLineal(int arr[], int n, int valor){
    for(int i = 0; i < n; ++i){
        if(valor == arr[i]){
            return i;
        }
    }
    return -1;
}
```

# Ejercicios

- ¿ Cómo generar números aleatorios flotantes en el intervalo  $[0, 1]$ ? ¿En general en un intervalo la forma  $[a, b]$ ? .  
Genere arreglos con elementos aleatorios del tipo double.
- Realice la implementación modular de los algoritmos de ordenamiento y búsqueda que veremos en el curso (debe dividir su código en funciones bien definidas y distribuirlo en varios archivos para lograr una mayor organización y reutilización del código).

En la función principal debe implementar un menú de opciones donde el usuario pueda elegir qué algoritmo utilizar.

# Resumen

- Un algoritmo de ordenamiento reorganiza los elementos de una lista en orden creciente o decreciente. (burbuja, selección, inserción)
- Un algoritmo de búsqueda encuentra un elemento dentro de una estructura de datos. (búsqueda lineal)
- La elección de un algoritmo depende de su **eficiencia**, la cantidad de datos, del contexto en que se aplican, de su estabilidad (mantiene el orden relativo de los elementos iguales)
- Modifique los algoritmos con otros tipos de datos.