algoritmos de ORDENACION

Ordenación Interna (en RAM)

Bubble Sort (Burbuja)

Idea: Compara pares adyacentes e intercambia si están mal.
Usar cuando: lista pequeña (<20) o para enseñanza.
No usar cuando: listas grandes (muy lento).
Complejidad: Mejor O(n), Peor O(n²).

```
void bubbleSort(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
```

Ejemplo

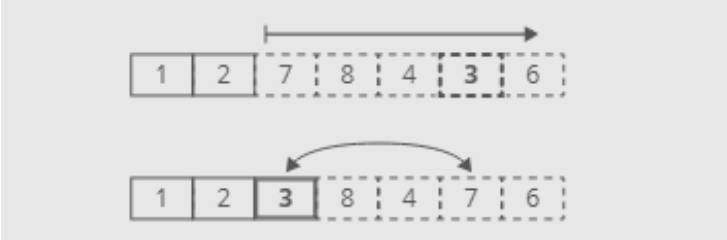
[5,3,8,4] → [3,5,8,4] → [3,5,4,8] → [3,4,5,8]

Selection Sort (Selección)

Idea: Buscar el mínimo, lo pone al inicio, y repite.
Usar cuando: intercambios son costosos y lista pequeña.
No usar cuando: velocidad es prioridad.
Complejidad: Siempre O(n²).

```
void selectionSort(int arr[]) {
    int n = arr.length;
    for (int i = 0; i < n-1; i++) {
        int min = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[min]) min = j;
        }
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
```

Ejemplo

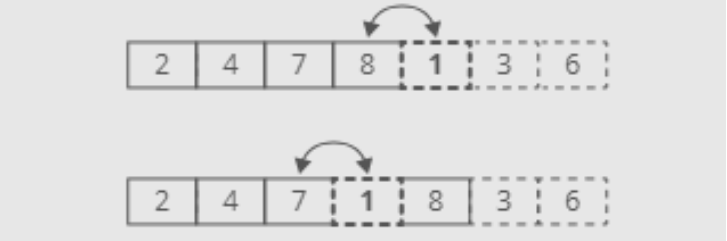


Insertion Sort (Inserción)

Idea: Inserta cada elemento en su posición en la parte ordenada.
Usar cuando: lista pequeña o casi ordenada.
No usar cuando: listas grandes y muy desordenadas.
Complejidad: Mejor O(n), Peor O(n²).

```
void insertionSort(int arr[]) {
    int n = arr.length;
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key)
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
```

Ejemplo



Ordenación Externa (Disco, datos grandes)

Mezcla Directa (Direct Merge Sort)

Algoritmo de ordenamiento externo que divide el archivo en particiones de tamaño fijo y las va mezclando en sucesivas pasadas hasta ordenar todo.

algoritmos de ORDENACION

Proceso resumido

Primera pasada:
- Leer clave por clave del archivo original (Fo).
- Distribuir alternadamente en F1 y F2 ↠ bloques de tamaño 1.
- Intercalar bloques de F1 y F2 en Fo ↠ secuencias de longitud 2.

Iteraciones sucesivas:
- Se duplican los tamaños de bloques en cada paso: 2, 4, 8, 16...
- Cada fusión genera bloques más grandes y ordenados.

1. Repetir hasta que el archivo original (Fo) quede completamente ordenado.

VENTAJA	DESVENTAJA
Fácil de implementar. No necesita algoritmos adicionales (trabaja directo con el archivo).	Ineficiente en la primera pasada Genera bloques de tamaño fijo sin aprovechar secuencias ya ordenadas.

Mezcla Equilibrada (Natural Merge Sort)

Optimización de la mezcla directa: en lugar de bloques fijos de 1, aprovecha secuencias ya ordenadas naturalmente

Proceso resumido

Primera iteración (mejora clave):
- Detectar secuencias ya ordenadas en el archivo (runs).
- Distribuirlas en F1 y F2.
- Así los bloques iniciales no tienen tamaño fijo de 1, sino que pueden ser más largos.

Iteraciones siguientes:
- Igual que mezcla directa: fusionar bloques entre F1 y F2, duplicando tamaños.

Repetir hasta que quede un archivo final completamente ordenado

VENTAJA	DESVENTAJA
Eficiente, usa pocos archivos, rápido en datos grandes.	Complejo, sin ventaja en datos pequeños, depende del algoritmo interno.

Algoritmo Polifásico

Variante optimizada de mezcla múltiple. Usa tres archivos auxiliares y fases sucesivas de intercalación para reducir accesos a disco.

Proceso resumido

Fase 1 – División y Ordenación:
- Leer bloques del archivo original (Fo).
- Ordenar cada bloque en RAM (algoritmo interno).
- Distribuir bloques ordenados en F1 y F2.

Fase 2 – Intercalación:
- Intercalar primer bloque de F1 con F2 ↠ guardar en Fo.
- Siguiente bloque de F1 y F2 ↠ guardar en F3.
- Repetir alternando salida entre Fo y F3.
- Repetir fases hasta que todo quede ordenado en Fo.

VENTAJA	DESVENTAJA
Eficiente, menos archivos, rápido en grandes datos.	Complejo, inútil en datos pequeños, depende del algoritmo interno.