

## **PARTE 1: Ingeniería inversa del JAR**

Realice la ingeniería inversa del archivo.jar utilizando la herramienta JD-GUI.

Clases identificadas:

BubbleSort, QuickSort, MergeSortDemo, SumArray, .App

Analisis de las clases:

Clase BubbleSort, Operaciones identificadas sobre arreglos:

Recorrido del arreglo mediante ciclos for anidados.

Comparacion de elementos consecutivos ( $\text{arr}[j] > \text{arr}[j + 1]$ ).

Intercambio de posiciones utilizando una variable temporal.

Uso de una variable boleana (swapped) para optimizar el algoritmo y finalizar anticipadamente si el arreglo ya esta ordenado.

Este algoritmo compara pares adyacentes y los intercambia si están en orden incorrecto, repitiendo el proceso hasta que el arreglo queda completamente ordenado.

Clase QuickSort, Operaciones identificadas sobre arreglos:

Selección de pivote

Recorrido del arreglo para comparar elementos con el pivote

Intercambio de elementos mediante variable temporal

División del arreglo en subproblemas.

Llamadas recursivas para ordenar las particiones izquierda y derecha

Clase MergeSortDemo, Operaciones identificadas sobre arreglos:

División del arreglo en dos mitades.

Creacion de arreglos auxiliares (Left y right).

Llamadas recursivas para ordenar cada mitad.

Comparación de elementos durante el proceso de fusión.

El método merge combina dos subarreglos ordenados en uno solo.

Clase SumArray, Operaciones identificadas:

Recorrido completo del arreglo

Acceso a posiciones mediante índice

Acumulación del valor en una variable auxiliar.

No implementa un algoritmo de ordenamiento, sino una operación aritmética sobre los elementos del arreglo.

A partir del proceso de ingeniería inversa se pudo identificar que el archivo JAR contiene implementaciones de tres algoritmos clásicos de ordenamiento:

BubbleSort, QuickSort, MergeSort.

Además, incluye una clase para operaciones básicas sobre arreglos(SumArray) y una clase principal (App).

Se observó el uso de estructuras fundamentales como:

Arreglos unidimensionales, Ciclos iterativos, condicionales, Recursividad.

## **PARTE 2: Ejercicio Algorítmico**

### **Pseudocódigo:**

Inicio

Si el tamaño del arreglo es menor que 2

    Mostrar "No es posible calcular el segundo mayor y menor"

Terminar

Iniciar:

    mayor = -infinito

    segundoMayor = -infinito

    menor = +infinito

    segundoMenor = +infinito

Para cada número en el arreglo hacer:

Si número > mayor entonces

    segundoMayor = mayor

    mayor = número

Sino si número > segundoMayor Y número ≠ mayor entonces

    segundoMayor = número

FinSi

Si número < menor entonces

    segundoMenor = menor

    menor = número

Sino si número < segundoMenor Y número ≠ menor entonces

    segundoMenor = número

FinSi

FinPara

Mostrar "Segundo mayor: ", segundoMayor

Mostrar "Segundo menor: ", segundoMenor

Fin

**Explicación del algoritmo:** El algoritmo funciona recorriendo el arreglo una sola vez.

Durante el recorrido mantiene actualizadas cuatro variables: mayor, segundoMayor, menor y segundoMenor.

Cada vez que se encuentra un número mayor al actual mayor, se actualiza el segundoMayor y luego el mayor.

De forma similar, cuando se encuentra un número menor al actual menor, se actualiza el segundoMenor y luego el menor.

De esta manera, no es necesario ordenar el arreglo, ya que los valores se actualizan dinámicamente en cada iteración.

**Análisis de complejidad:**

Complejidad en tiempo:

$O(n)$ :

Porque el arreglo se recorre una sola vez, donde  $n$  representa la cantidad de elementos.

Complejidad en espacio:

$O(1)$ :

Porque únicamente se utilizan cuatro variables adicionales, independientemente del tamaño del arreglo.