



UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA

TAREA

ARREGLOS Y ALGORITMOS EN JAVA

MILDRED HANANI PINEDA PINEDA

0905-22-5811

PROGRAMACION 3

FEBRERO 2026

Parte 1: Ingeniería inversa del JAR

PASO 1

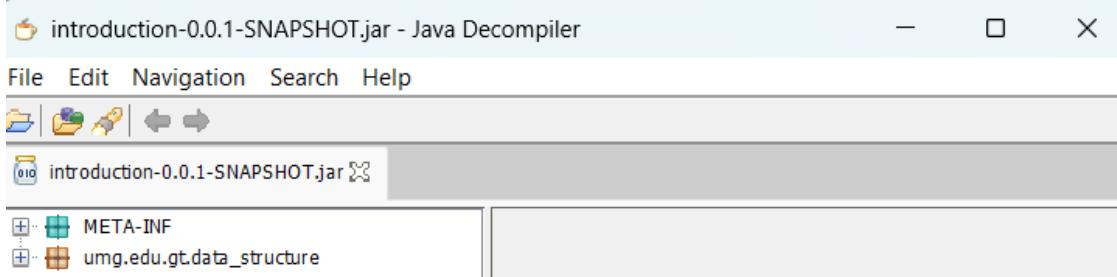
Herramienta De Descompilacion



PASO 2

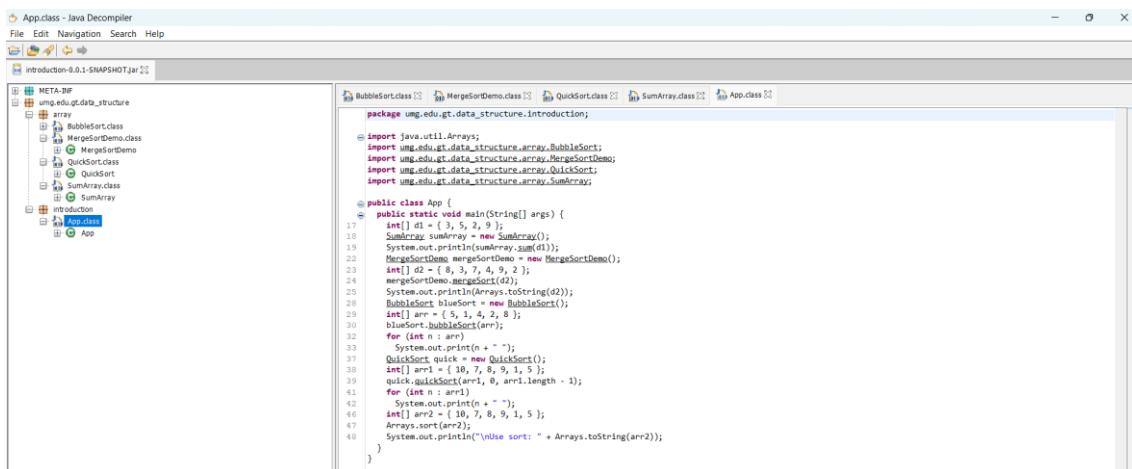
Abrir la Herramienta JD-GUI

Ir a FILE luego descargas y abrir el archivo .JAR



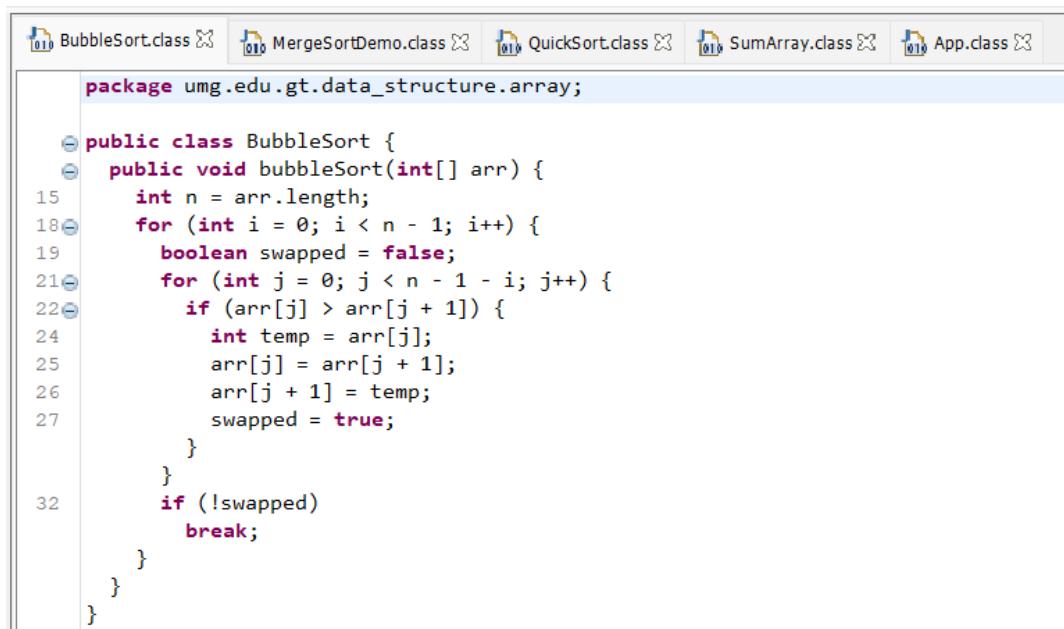
PASO 3

Abir todos los archivos .CLASS

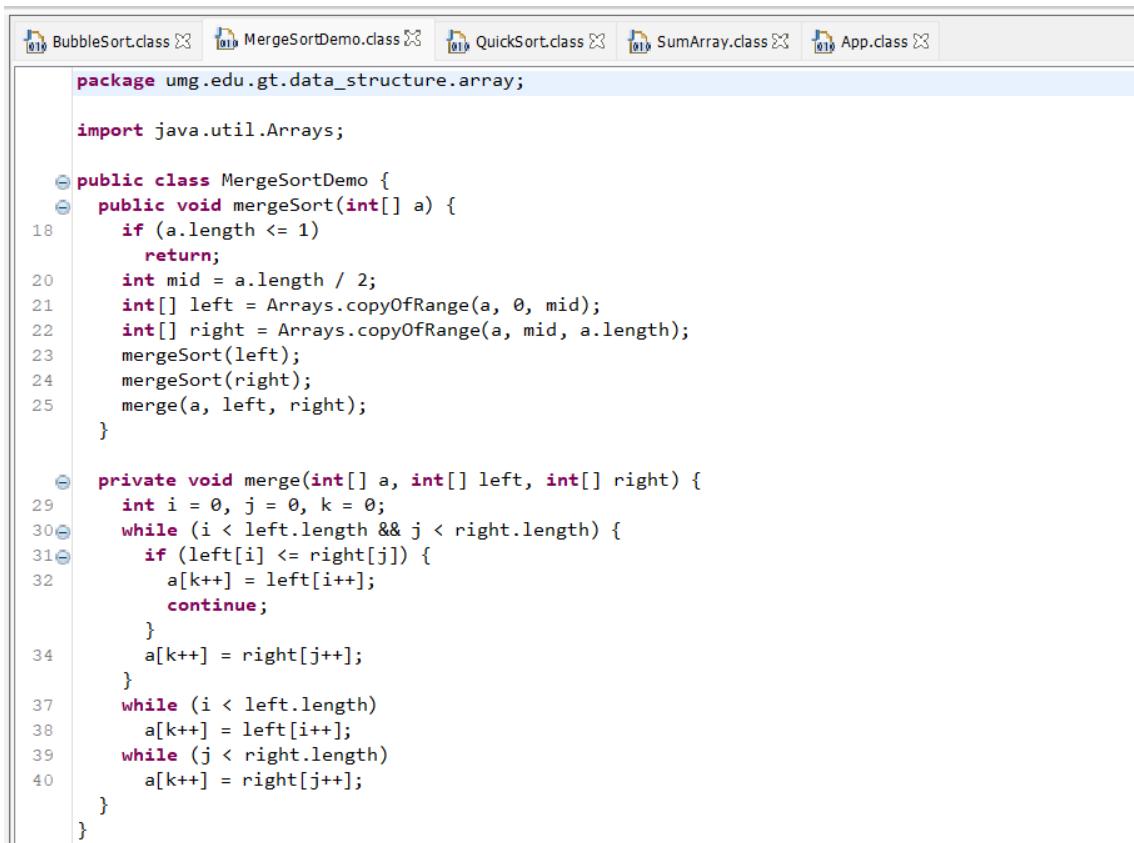


PASO 4

Mostrar Archivo BUBBLESORT.CLASS



Mostrar archivo MERGERSORTDEMO.CLASS



```

package umg.edu.gt.data_structure.array;

import java.util.Arrays;

public class MergeSortDemo {
    public void mergeSort(int[] a) {
        if (a.length <= 1)
            return;
        int mid = a.length / 2;
        int[] left = Arrays.copyOfRange(a, 0, mid);
        int[] right = Arrays.copyOfRange(a, mid, a.length);
        mergeSort(left);
        mergeSort(right);
        merge(a, left, right);
    }

    private void merge(int[] a, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) {
                a[k++] = left[i++];
                continue;
            }
            a[k++] = right[j++];
        }
        while (i < left.length)
            a[k++] = left[i++];
        while (j < right.length)
            a[k++] = right[j++];
    }
}

```

Parte 2: Ejercicio algorítmico

solucion_parte2.txt

SOLUCIÓN PARTE 2: SEGUNDO MAYOR Y SEGUNDO MENOR EN UN ARREGLO
ALGORITMO EN PSEUDOCÓDIGO

1. Inicializar:

```

max1 = -∞ (o Integer.MIN_VALUE en Java)
max2 = -∞ (o Integer.MIN_VALUE en Java)
min1 = +∞ (o Integer.MAX_VALUE en Java)
min2 = +∞ (o Integer.MAX_VALUE en Java)

```

2. Para cada elemento "num" en el arreglo:

a. Actualizar máximos:

- Si num > max1:
 - * max2 = max1
 - * max1 = num
- Si num < max1 Y num > max2:
 - * max2 = num

b. Actualizar mínimos:

- Si num < min1:
 - * min2 = min1
 - * min1 = num
- Si num > min1 Y num < min2:
 - * min2 = num

3. Retornar (max2, min2)**## EXPLICACIÓN BREVE**

Este algoritmo funciona porque actualiza continuamente las cuatro variables clave (`max1`, `max2`, `min1`, `min2`) mientras recorre el arreglo una sola vez.

- Para los valores máximos:

- * `max1` siempre contiene el valor más alto encontrado hasta el momento
- * `max2` se actualiza cuando encontramos un nuevo máximo (`max1`) o cuando encontramos un valor que es menor que `max1` pero mayor que `max2`

- Para los valores mínimos:

- * `min1` siempre contiene el valor más bajo encontrado hasta el momento
- * `min2` se actualiza cuando encontramos un nuevo mínimo (`min1`) o cuando encontramos un valor que es mayor que `min1` pero menor que `min2`

La clave está en el orden de las comparaciones. Primero verificamos si el número actual es mayor/menor que los valores principales (`max1/min1`), y luego verificamos si es candidato para ser el segundo valor (`max2/min2`).

Este enfoque garantiza que encontramos el segundo mayor y segundo menor sin necesidad de ordenar el arreglo, cumpliendo con la restricción de realizar un solo recorrido.

ANÁLISIS DE COMPLEJIDAD**### Complejidad Temporal: O(n)**

- El algoritmo recorre el arreglo exactamente una vez, procesando cada elemento con operaciones de tiempo constante.
- Para cada elemento, realizamos un número fijo de comparaciones (4 en total: 2 para máximos y 2 para mínimos).
- Por lo tanto, el tiempo total es proporcional al tamaño del arreglo: $O(n)$.

Complejidad Espacial: O(1)

- El algoritmo utiliza un espacio adicional constante: solo 4 variables (`max1`, `max2`, `min1`, `min2`).
- No importa el tamaño del arreglo de entrada, el espacio adicional utilizado siempre es el mismo.
- Por lo tanto, la complejidad espacial es $O(1)$.