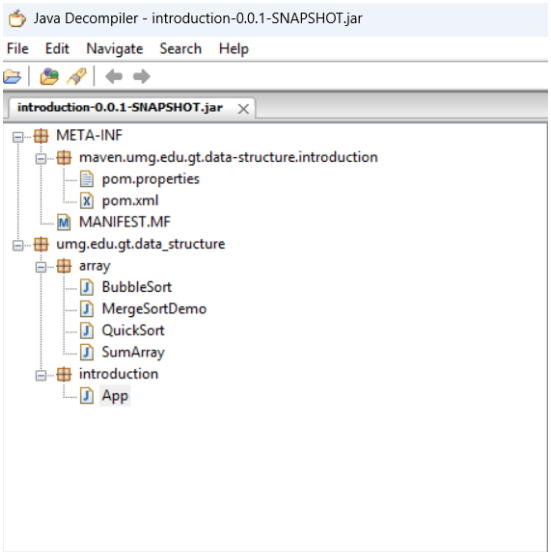
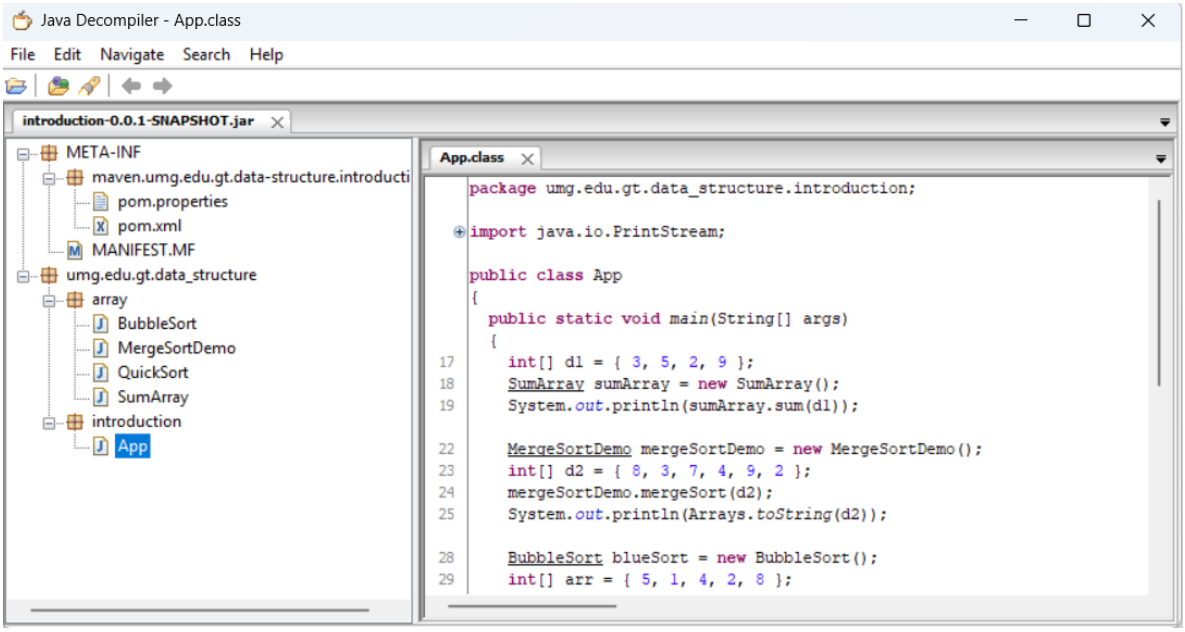
INGENIERIA A LA INVERSA



PROCESO DE DECOMPILACION



CLASES EXISTENTES

1. BubbleSort.class
2. MergeSortDemo
3. QuickSort
4. SumArray
5. App

QUÉ OPERACIONES SE REALIZAN SOBRE LOS ARREGLOS.

1. BubbleSort.class

Lectura de elementos

Se leen dos posiciones del arreglo para compararlas:
arr[j] y arr[j + 1].

Intercambio de valores

Si están desordenados, se intercambian las dos posiciones:
arr[j] arr[j + 1].

Recorrido del arreglo

El arreglo se recorre varias veces usando ciclos for para repetir las comparaciones.

2. MergeSortDemo

División del arreglo

Se parte el arreglo original en dos subarreglos usando:
Arrays.copyOfRange(...).

Llamadas recursivas

Cada subarreglo se vuelve a dividir hasta quedar de 1 elemento.

Lectura y comparación de elementos

En el método merge, se leen elementos de los subarreglos left y right para compararlos.

Escritura en el arreglo final

Se copian los elementos (el menor primero) al arreglo original a.

Copia de elementos sobrantes

Si uno de los subarreglos se termina antes, se copian al final todos los elementos restantes del otro subarreglo.

3. QuickSort

Comparar elementos con el pivote.

Intercambiar para dejar menores a la izquierda y mayores a la derecha.
Colocar el pivote en su posición final.
Dividir en dos subarreglos y repetir recursivamente.

4. **SumArray**

Recorrer el arreglo

Se utiliza un ciclo for para ir elemento por elemento de nums.
Leer cada valor
En cada vuelta se toma el valor actual del arreglo (n).
Acumular la suma
Cada valor leído se agrega a un total:
total += n.
Devolver el resultado
Se regresa la suma total de todos los elementos del arreglo.

5. **App**

Suma un arreglo (d1).
Ordena distintos arreglos con MergeSort, BubbleSort, QuickSort y con Arrays.sort.
Imprime los resultados en consola

QUÉ ALGORITMOS DE ORDENAMIENTO SE UTILIZAN.

En el proyecto se usan tres algoritmos de ordenamiento, cada uno con una forma distinta de organizar los datos:

Bubble Sort

Compara elementos vecinos y los intercambia si están desordenados.

Es simple, pero lento para arreglos grandes.

Merge Sort

Divide el arreglo en partes pequeñas, las ordena por separado y luego las une.

Es rápido y muy eficiente en arreglos grandes.

Quick Sort

Elige un pivote y separa el arreglo en elementos menores y mayores.

Es uno de los métodos más rápidos en la práctica.

Además, se usa también el método interno de Java:

Arrays.sort()

Ordena automáticamente usando un algoritmo optimizado del sistema.

Parte 2: Ejercicio algorítmico

INICIO

Si el tamaño del arreglo es menor que 2

    Mostrar "No hay suficientes elementos"

    Terminar

Inicializar variables:

    mayor = -∞

    segundoMayor = -∞

    menor = +∞

    segundoMenor = +∞

Para cada número n en el arreglo hacer:

    // Actualizar mayor y segundoMayor

    Si n > mayor entonces

        segundoMayor = mayor

        mayor = n

    Sino si n > segundoMayor Y n < mayor entonces

        segundoMayor = n

    // Actualizar menor y segundoMenor

    Si n < menor entonces

        segundoMenor = menor

        menor = n

    Sino si n < segundoMenor Y n > menor entonces

        segundoMenor = n

Mostrar segundoMayor

Mostrar segundoMenor

FIN

EXPLICACIÓN DE POR QUÉ EL ALGORITMO FUNCIONA

El algoritmo funciona porque mientras recorre el arreglo una sola vez, va actualizando cuatro variables importantes:

- Mayor: Guarda el número más grande encontrado hasta el momento.
- Segundo Mayor: Guarda el segundo número más grande.
- Menor: Guarda el número más pequeño encontrado.
- Segundo Menor: Guarda el segundo número más pequeño.

Cada vez que se analiza un número:

- Si es más grande que el mayor actual, se actualizan ambos valores.
- Si no es el mayor pero es mayor que el segundoMayor, se actualiza solo el segundoMayor.
- Lo mismo se hace para el menor y segundoMenor.

Como cada elemento se evalúa solo una vez, se cumple la restricción del problema.

ANÁLISIS DE COMPLEJIDAD

El arreglo se recorre una sola vez.

Si el arreglo tiene n elementos, el número de operaciones depende de n.

Por lo tanto:

Tiempo = O(n)

Es lineal porque crece proporcionalmente al tamaño del arreglo.

Solo se utilizan 4 variables adicionales:

mayor

segundoMayor

menor

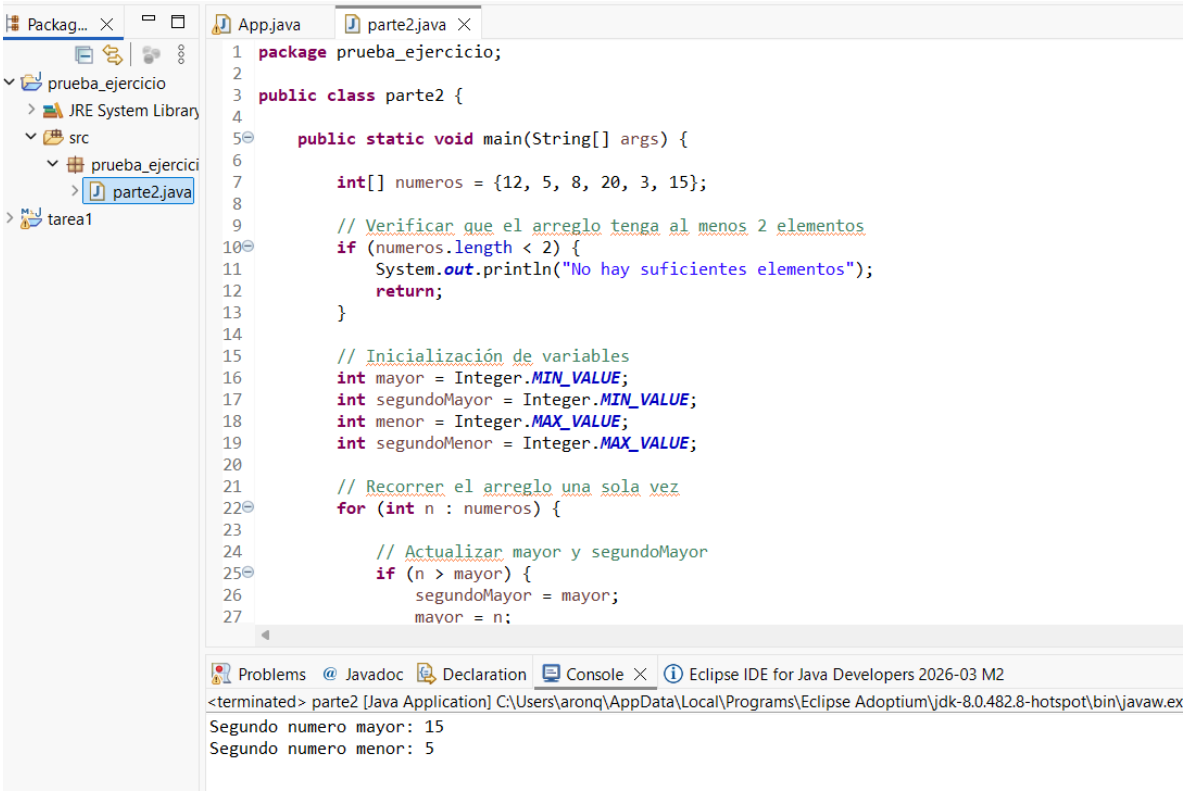
segundoMenor

No importa el tamaño del arreglo, siempre se usa la misma cantidad de memoria extra.

Por lo tanto:

Espacio = O(1)

Es constante.



Quise probar que el algoritmo funcionaba y lo trabaje con java en eclipse adjunto muestras.