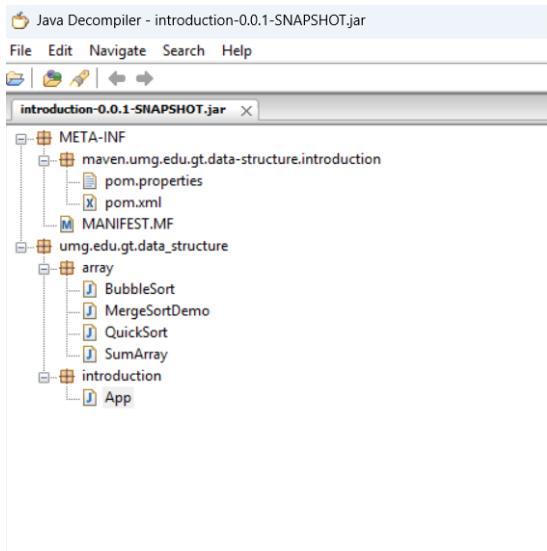


INGENIERIA A LA INVERSA



PROCESO DE DECOMPILEACION

The screenshot shows the Java Decomiler interface with the title bar "Java Decomiler - App.class". The menu bar includes File, Edit, Navigate, Search, and Help. Below the menu is a toolbar with icons for Open, Save, Print, and others. The main window displays the decompiled code for the class "App" located in the file "introduction-0.0.1-SNAPSHOT.jar". The code is as follows:

```
package umg.edu.gt.data_structure.introduction;  
import java.io.PrintStream;  
  
public class App  
{  
    public static void main(String[] args)  
    {  
        int[] d1 = { 3, 5, 2, 9 };  
        SumArray sumArray = new SumArray();  
        System.out.println(sumArray.sum(d1));  
  
        MergeSortDemo mergeSortDemo = new MergeSortDemo();  
        int[] d2 = { 8, 3, 7, 4, 9, 2 };  
        mergeSortDemo.mergeSort(d2);  
        System.out.println(Arrays.toString(d2));  
  
        BubbleSort blueSort = new BubbleSort();  
        int[] arr = { 5, 1, 4, 2, 8 };  
    }  
}
```

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

Iniciar 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:

$$\text{Tiempo} = O(n)$$

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:

$$\text{Espacio} = O(1)$$

Es constante.

The screenshot shows the Eclipse IDE interface. On the left is the Package Explorer view, which displays a project structure with a package named 'prueba_ejercicio' containing a source folder 'src' with a file named 'parte2.java'. The central part of the interface is the Java editor, showing the following code:

```
1 package prueba_ejercicio;
2
3 public class parte2 {
4
5     public static void main(String[] args) {
6
7         int[] numeros = {12, 5, 8, 20, 3, 15};
8
9         // Verificar que el arreglo tenga al menos 2 elementos
10        if (numeros.length < 2) {
11            System.out.println("No hay suficientes elementos");
12            return;
13        }
14
15        // Inicialización de variables
16        int mayor = Integer.MIN_VALUE;
17        int segundoMayor = Integer.MIN_VALUE;
18        int menor = Integer.MAX_VALUE;
19        int segundoMenor = Integer.MAX_VALUE;
20
21        // Recorrer el arreglo una sola vez
22        for (int n : numeros) {
23
24            // Actualizar mayor y segundoMayor
25            if (n > mayor) {
26                segundoMayor = mayor;
27                mayor = n;
```

Below the editor is the Console view, which shows the output of the application's execution:

```
<terminated> parte2 [Java Application] C:\Users\aronq\AppData\Local\Programs\Eclipse Adoptium\jdk-8.0.482.8-hotspot\bin\javaw.exe
Segundo numero mayor: 15
Segundo numero menor: 5
```

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