

Capturas de JUnit tests

- **Radix sort:**

```
J RadixSortTest.java > RadixSortTest
1  import static org.junit.jupiter.api.Assertions.*;
2  import org.junit.jupiter.api.Test;
3
4  public class RadixSortTest {
5      @Test
6      void testSort() {
7          RadixSort<Integer> radixSort = new RadixSort<>();
8          Integer[] array = {170, 45, 75, 90, 802, 24, 2, 66};
9          Integer[] expected = {2, 24, 45, 66, 75, 90, 170, 802};
10
11          assertEquals(expected, radixSort.sort(array));
12      }
13
14      @Test
15      void testEmptyArray() {
16          RadixSort<Integer> radixSort = new RadixSort<>();
17          Integer[] array = {};
18          Integer[] expected = {};
19
20          assertEquals(expected, radixSort.sort(array));
21      }
22
23      @Test
24      void testSingleElement() {
25          RadixSort<Integer> radixSort = new RadixSort<>();
26          Integer[] array = {42};
27          Integer[] expected = {42};
28
29          assertEquals(expected, radixSort.sort(array));
30      }
31  }
```

- **Quick sort:**

```
J QuickSortTest.java > ...
1  import static org.junit.jupiter.api.Assertions.*;
2  import org.junit.jupiter.api.Test;
3
4  public class QuickSortTest {
5
6      @Test
7      public void testSort() {
8          QuickSort<Integer> quickSort = new QuickSort<>();
9          Integer[] array = {34, 7, 23, 32, 5, 62};
10         Integer[] expected = {5, 7, 23, 32, 34, 62};
11         assertEquals(expected, quickSort.sort(array));
12     }
13
14     @Test
15     public void testSortWithEmptyArray() {
16         QuickSort<Integer> quickSort = new QuickSort<>();
17         Integer[] array = {};
18         Integer[] expected = {};
19         assertEquals(expected, quickSort.sort(array));
20     }
21
22     @Test
23     public void testSortWithSingleElement() {
24         QuickSort<Integer> quickSort = new QuickSort<>();
25         Integer[] array = {42};
26         Integer[] expected = {42};
27         assertEquals(expected, quickSort.sort(array));
28     }
29 }
```

- **Data Generator**

```

J DataGeneratorTest.java > DataGeneratorTest > testGenerateData()
1 import static org.junit.jupiter.api.Assertions.*;
2 import org.junit.jupiter.api.Test;
3 import java.io.File;
4
5 public class DataGeneratorTest {
6
7     @Test
8     public void testGenerateData() {
9         // Crear una instancia de DataGenerator
10        DataGenerator generator = new DataGenerator();
11        String filename = "test_data.txt";
12
13        generator.generateData(filename);
14
15
16
17        File file = new File(filename);
18        assertTrue(file.exists(), "El archivo no se creó correctamente");
19
20
21        assertTrue(file.length() > 0, "El archivo está vacío");
22
23
24        file.delete();
25    }
26 }

```

- **Data Reader:**

```

J DataReaderTest.java > DataReaderTest > testReadData()
1 import static org.junit.jupiter.api.Assertions.*;
2 import org.junit.jupiter.api.Test;
3 import java.io.FileWriter;
4 import java.io.File;
5 import java.util.List;
6 import java.util.ArrayList;
7
8 @SuppressWarnings("unused")
9 public class DataReaderTest {
10     @Test
11     public void testReadData() {
12         String filename = "test_read_data.txt";
13         try (FileWriter writer = new FileWriter(filename)) {
14             writer.write(str:"170\n45\n75\n90\n802\n24\n2\n66\n");
15         } catch (Exception e) {
16             e.printStackTrace();
17         }
18
19
20         List<List<Integer>> data = DataReader.readData(filename);
21
22         assertEquals(1, data.size());
23         assertEquals(8, data.get(index:0).size());
24
25         // Verifica Los valores
26         List<Integer> dataset = data.get(index:0);
27         assertEquals(new Integer[]{170, 45, 75, 90, 802, 24, 2, 66}, dataset.toArray());
28
29         new File(filename).delete();
30     }
31
32     @Test
33     public void testReadDataWithMultipleDatasets() {
34         String filename = "test_multiple_datasets.txt";
35         try (FileWriter writer = new FileWriter(filename)) {
36             writer.write(str:"Dataset size: 3\n170\n45\n75\nDataset size: 2\n90\n802\n");
37         } catch (Exception e) {
38             e.printStackTrace();
39         }
40
41         List<List<Integer>> data = DataReader.readData(filename);
42
43         assertEquals(2, data.size());
44         assertEquals(new Integer[]{170, 45, 75}, data.get(index:0).toArray());
45         assertEquals(new Integer[]{90, 802}, data.get(index:1).toArray());
46         new File(filename).delete();
47     }
48 }

```

- Insertion Sort:**

The screenshot shows an IDE with a file named `InsertionSortTest.java`. The code defines a `InsertionSortTest` class with three test methods: `shouldBeAbleToSortIntegerByInsertion()`, `shouldReturnEmptyArrayWhenInputIsEmpty()`, and `shouldReturnSameArrayWhenInputHasOneElem()`. Each test method creates an `InsertionSort` object, sets an input array, and asserts the result. The IDE's `TEST RESULTS` pane shows three successful tests. The `Test Runner for Java` pane lists the same three tests with green checkmarks.

```
1 import static org.junit.jupiter.api.Assertions.*;
2 import org.junit.jupiter.api.Test;
3
4 public class InsertionSortTest {
5
6     @Test
7     public void shouldBeAbleToSortIntegerByInsertion() {
8         InsertionSort<Integer> insertionSort = new InsertionSort<>();
9         Integer[] input = {5, 2, 9, 1, 5, 6, 1};
10        Integer[] expected = {1, 1, 2, 5, 5, 6, 9};
11
12        Integer[] result = insertionSort.sort(input);
13        assertEquals(expected, result);
14    }
15
16    @Test
17    public void shouldReturnEmptyArrayWhenInputIsEmpty() {
18        InsertionSort<Integer> insertionSort = new InsertionSort<>();
19        Integer[] input = {};
20        Integer[] expected = {};
21
22        Integer[] result = insertionSort.sort(input);
23        assertEquals(expected, result);
24    }
25
26    @Test
27    public void shouldReturnSameArrayWhenInputHasOneElem() {
28        InsertionSort<Integer> insertionSort = new InsertionSort<>();
29        Integer[] input = {199};
30        Integer[] expected = {199};
31
32        Integer[] result = insertionSort.sort(input);
33        assertEquals(expected, result);
34    }
35 }
```

TESTS 7,shouldBeAbleToSortIntegerByInsertion(InsertionSortTest)
XTESTE 7,shouldBeAbleToSortIntegerByInsertion(InsertionSortTest)
RUNTIME:222

Test Runner for Java
✔ shouldBeAbleToSortIntegerByInsertion()
✔ shouldReturnEmptyArrayWhenInputIsEmpty()
✔ shouldBeAbleToSortIntegerByInsertion()
✔ shouldReturnSameArrayWhenInputHasOneElem()
✔ shouldNotAlterAlreadySortedArray()

- Gnome Sort:**

The screenshot shows an IDE with a file named `GnomeSortTest.java`. The code defines a `GnomeSortTest` class with three test methods: `shouldBeAbleToSortIntegerByGnome()`, `shouldReturnEmptyArrayWhenInputIsEmpty()`, and `shouldReturnSameArrayWhenInputHasOneElem()`. Each test method creates a `GnomeSort` object, sets an input array, and asserts the result. The IDE's `TEST RESULTS` pane shows three successful tests. The `Test Runner for Java` pane lists the same three tests with green checkmarks.

```
1 import static org.junit.jupiter.api.Assertions.*;
2 import org.junit.jupiter.api.Test;
3
4 public class GnomeSortTest {
5
6     @Test
7     public void shouldBeAbleToSortIntegerByGnome() {
8         GnomeSort<Integer> gnomeSort = new GnomeSort<>();
9         Integer[] input = {5, 2, 9, 1, 5, 6, 1};
10        Integer[] expected = {1, 1, 2, 5, 5, 6, 9};
11
12        Integer[] result = gnomeSort.sort(input);
13        assertEquals(expected, result);
14    }
15
16    @Test
17    public void shouldReturnEmptyArrayWhenInputIsEmpty() {
18        GnomeSort<Integer> gnomeSort = new GnomeSort<>();
19        Integer[] input = {};
20        Integer[] expected = {};
21
22        Integer[] result = gnomeSort.sort(input);
23        assertEquals(expected, result);
24    }
25
26    @Test
27    public void shouldReturnSameArrayWhenInputHasOneElem() {
28        GnomeSort<Integer> gnomeSort = new GnomeSort<>();
29        Integer[] input = {199};
30        Integer[] expected = {199};
31
32        Integer[] result = gnomeSort.sort(input);
33        assertEquals(expected, result);
34    }
35 }
```

TESTS 7,shouldReturnEmptyArrayWhenInputIsEmpty(GnomeSortTest)
XTESTE 7,shouldReturnEmptyArrayWhenInputIsEmpty(GnomeSortTest)
RUNTIME:262

Test Runner for Java
✔ shouldBeAbleToSortIntegerByGnome()
✔ shouldNotAlterAlreadySortedArray()
✔ shouldReturnEmptyArrayWhenInputIsEmpty()
✔ shouldReturnSameArrayWhenInputHasOneElem()
✔ shouldBeAbleToSortIntegerByGnome()

- Merge Sort:**

MergeSortTest.javaMergeSort.java

src > MergeSortTest.java

```
3 public class MergeSort<T extends Comparable<T>> implements IGenericSort<T> {
12 private void mergeSort(T[] A, int lo, int hi) {
14     int mid = lo + (hi - lo) / 2;
15     mergeSort(A, lo, mid);
16     mergeSort(A, mid + 1, hi);
17     merge(A, lo, mid, hi);
18 }
19
20 private void merge(T[] A, int lo, int mid, int hi) {
21     T[] B = Arrays.copyOfRange(A, lo, hi + 1);
22     int i = 0;
23     int j = mid - lo + 1;
24     int k = lo;
25
26     while (i <= mid - lo && j <= hi - lo) {
27         if (B[i].compareTo(B[j]) <= 0) {
28             A[k++] = B[i++];
29         } else {
30             A[k++] = B[j++];
31         }
32     }
33
34     while (i <= mid - lo) {
35         A[k++] = B[i++];
36     }
37
38     while (j <= hi - lo) {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS COMMENTS

%TESTS 7,shouldReturnEmptyArrayWhenInputIsEmpty(MergeSortTest)

%TESTE 7,shouldReturnEmptyArrayWhenInputIsEmpty(MergeSortTest)

%RUNTIME187

Test Runner for Java

✓ shouldBeAbleToSortIntegerByGnome()

✓ shouldBeAbleToSortIntegerByInsertion()

✓ shouldNotAlterAlreadySortedArray()

✓ shouldReturnEmptyArrayWhenInputIsEmpty()

✓ shouldReturnSameArrayWhenInputHasOneElement()

> 1 older results