```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Tue Oct 15 12:39:35 CEST 2024
 */

package ex05.solution.task02;

import java.lang.reflect.Array;
import java.util.Random;

public class MergeSort {

  /**
   * Sorts an Array with the Merge-Sort Algorithm.
   * Precondition: Length must be 2^x.
   * @param s Sequence (Array) to be sorted.
   * @return The sorted Sequence (Array).
   */
  public static <T extends Comparable<? super T>> T[] mergeSort(T[] s) {
    int n = s.length;
    if (n > 1) {
      Partitions<T> s12 = partition(s, n/2);
      T[] s1 = mergeSort(s12.s1);
      T[] s2 = mergeSort(s12.s2);
      s = merge(s1, s2);
    }
    return s;
  }

  record Partitions<T>(T[] s1, T[] s2) {}

  static <T> Partitions<T> partition(T[] s, int length) {
    T[] s1 = newInstance(s, length);
    T[] s2 = newInstance(s, length);
    System.arraycopy(s, 0, s1, 0, length);
    System.arraycopy(s, length, s2, 0, length);
    return new Partitions<T>(s1, s2);
  }

  /**
   * Merges the two Sequences (Arrays) 'a' and 'b' in ascending Order.
   * @param a Sequence A.
   * @param b Sequence B.
   * @return The merged Sequence.
   */
  static <T extends Comparable<? super T>> T[] merge(T[] a, T[] b) {
    T[] s = newInstance(a, a.length * 2);
    int ai = 0; // First Element in 'Sequence' A
    int bi = 0; // First Element in 'Sequence' B
    int si = 0; // Last Element in 'Sequence' S
    while (!(ai == a.length) && !(bi == b.length)) {
      if (a[ai].compareTo(b[bi]) < 0) {
        s[si++] = a[ai++];
      }
      else {
        s[si++] = b[bi++];
      }
    }
    while (!(ai == a.length)) {
      s[si++] = a[ai++];
    }
    while (!(bi == b.length)) {
      s[si++] = b[bi++];
    }
    return s;
  }
```



```java

  /**
   * Utility-Method to create a <T>-Array.
   *
   * @param array
   *          An Array with the same Type as the new one (only used to get the
   *          correct Type for the new Array).
   * @param length
   *          The Length of the new Array.
   * @return The new created Array.
   */
  @SuppressWarnings("unchecked")
  static <T> T[] newInstance(T[] array, int length) {
    return (T[]) Array.newInstance(array[0].getClass(), length);
  }


  public static void main(String[] args) {

    Integer[] array = {7, 2, 9, 4, 3, 8, 6, 1};
    Integer[] orginalArray = array.clone();
    printArray(array);

    array = mergeSort(array);

    printArray(array);
    verify(orginalArray, array);

    /* Makeing some Test to measure the Time needed of mergeSort().
     * Creating int-Arrays, beginning with Length of 2^minExponent
     * until the last Array with Length of 2^maxExponent.
     */
    final int minExponent = 10;
    final int maxExponent = 15;
    int n = (int)Math.round(Math.pow(2, maxExponent));
    array = new Integer[n];
    Random rand = new Random(0);    // a Random-Generator
    for (int i = 0; i < n; i++) {
      array[i] = rand.nextInt(101); // generating Numbers: 0..100
    }
    long lastTime = Long.MAX_VALUE;
    for (int exp = minExponent; exp <= maxExponent; exp++) {
      int len = (int)Math.round(Math.pow(2, exp));
      Integer[] arr = new Integer[len];
      final int MEASUREMENTS = 10;
      long minTime = Long.MAX_VALUE;
      for (int m = 0; m < MEASUREMENTS; m++) {
        System.arraycopy(array, 0, arr, 0, len);
        long start = System.nanoTime();
        arr = mergeSort(arr);
        long end = System.nanoTime();
        long time = end - start;
        if (time < minTime) {
          minTime = time;
        }
        verify(array, arr);
      }
      System.out.format("Array-Size: %,7d       Time: %,6.1f ms       "
                        + "Ratio to last: %2.1f\n",
                        len, (double) minTime / (long) 1e6,
                        (double) minTime / lastTime);
      lastTime = minTime;
    }
  }
```

```
131
132   /**
133    * Prints an int-Array to the Console.
134    * @param array The int-Array.
135    */
136   static <T> void printArray(T[] array) {
137     System.out.print("Array["+array.length+"]: ");
138     for (T i: array) {
139       System.out.print(i + " ");
140     }
141     System.out.println("");
142   }
143
144
145   /**
146    * Verifies that sortedArray is a correctly sorted based on originalArray.
147    * @param originalArray The original array.
148    * @param sortedArray The sorted array, based on originalArray.
149    *                    Can be shorter than originalArray.
150    */
151   static <T extends Comparable<? super T>> void verify(T[] originalArray,
152       T[] sortedArray) {
153     T[] originalSortedArray = newInstance(originalArray, sortedArray.length);
154     System.arraycopy(originalArray, 0, originalSortedArray, 0, sortedArray.length);
155     java.util.Arrays.sort(originalSortedArray);
156     if ( ! java.util.Arrays.equals(originalSortedArray, sortedArray)) {
157       try {Thread.sleep(200);} catch(@SuppressWarnings("unused") Exception e) {/*empty
    */}
158       System.err.println("ERROR: wrong sorted!");
159       System.exit(1);
160     }
161   }
162
163
164 }
165
166
167
168 /* Session-Log:
169
170 $ java -Xint -Xms100M -Xmx100M ex05/solution/task02/MergeSort
171 Array[8]: 7 2 9 4 3 8 6 1
172 Array[8]: 1 2 3 4 6 7 8 9
173 Array-Size:   1,024       Time:    2.2 ms       Ratio to last: 0.0
174 Array-Size:   2,048       Time:    4.5 ms       Ratio to last: 2.0
175 Array-Size:   4,096       Time:    9.4 ms       Ratio to last: 2.1
176 Array-Size:   8,192       Time:   19.6 ms       Ratio to last: 2.1
177 Array-Size:  16,384       Time:   40.6 ms       Ratio to last: 2.1
178 Array-Size:  32,768       Time:   83.0 ms       Ratio to last: 2.0
179
180 */
```



```
1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Tue Oct 15 12:39:35 CEST 2024
4   */
5
6  package ex05.solution.task02;
7  import static org.junit.Assert.assertArrayEquals;
8
9  import java.util.Arrays;
10 import java.util.Random;
11
12 import org.junit.FixMethodOrder;
13 import org.junit.Test;
14 import org.junit.runners.MethodSorters;
15
16 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
17 public class MergeSortJUnitTest {
18
19   @Test
20   public void test01() {
21     Integer[] arr = {4, 1, 2, 3};
22     sort(arr);
23   }
24
25   @Test
26   public void test02() {
27     Integer[] arr = {2, 4, 3, 1};
28     sort(arr);
29   }
30
31   @Test
32   public void test03() {
33     Integer[] arr = {2, 1};
34     sort(arr);
35   }
36
37   @Test
38   public void test04() {
39     Integer[] arr = {1, 2};
40     sort(arr);
41   }
42
43   @Test
44   public void test05() {
45     Integer[] arr = {1};
46     sort(arr);
47   }
48
49   @Test
50   public void test06() {
51     Integer[] arr = {};
52     sort(arr);
53   }
54
55   @Test
56   public void test07StressTest() {
57     final int NUMBER_OF_TESTS = 50000;
58     final int LENGTH = 128;
59     for (int n = 0; n < NUMBER_OF_TESTS; n++) {
60       Integer[] arr =
61           new Random().ints(LENGTH, 0, 10).boxed().toArray(Integer[]::new);
62       sort(arr);
63     }
64   }
```

```
65
66    private void sort(Integer[] arr) {
67      Integer[] clonedArr = arr.clone();
68      Integer[] sortedArr = MergeSort.mergeSort(arr);
69      verify(clonedArr, sortedArr);
70    }
71
72    @SuppressWarnings("static-method")
73    private void verify(Integer[] orgArr, Integer[] sortedArr) {
74      Integer[] sortedOrgArr = Arrays.copyOf(orgArr, orgArr.length);
75      Arrays.sort(sortedOrgArr);
76      assertArrayEquals(sortedOrgArr, sortedArr);
77    }
78
79  }
80
```