```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Tue Oct 15 12:37:47 CEST 2024
 */

package ex05.baseline.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) {

    // TODO Implement here...

    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

    // TODO Implement here...

    return null;
  }

  /**
   * 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);
  }

```

```java

  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;
    }
  }

  /**
   * Prints an int-Array to the Console.
   * @param array The int-Array.
   */
  static <T> void printArray(T[] array) {
    System.out.print("Array["+array.length+"]: ");
    for (T i: array) {
      System.out.print(i + " ");
    }
    System.out.println("");
  }


  /**
   * Verifies that sortedArray is a correctly sorted based on originalArray.
   * @param originalArray The original array.
   * @param sortedArray The sorted array, based on originalArray.
   *                      Can be shorter than originalArray.
   */
  static <T extends Comparable<? super T>> void verify(T[] originalArray,
      T[] sortedArray) {
    T[] originalSortedArray = newInstance(originalArray, sortedArray.length);
    System.arraycopy(originalArray, 0, originalSortedArray, 0, sortedArray.length);
    java.util.Arrays.sort(originalSortedArray);
    if ( ! java.util.Arrays.equals(originalSortedArray, sortedArray)) {
      try {Thread.sleep(200);} catch(@SuppressWarnings("unused") Exception e) {/*empty
*/}
      System.err.println("ERROR: wrong sorted!");
      System.exit(1);
    }
  }


}



/* Session-Log:

$ java -Xint -Xms100M -Xmx100M ex05/baseline/task02/MergeSort
Array[8]: 7 2 9 4 3 8 6 1
Array[8]: 1 2 3 4 6 7 8 9
Array-Size:   1,024       Time:    2.2 ms       Ratio to last: 0.0
Array-Size:   2,048       Time:    4.5 ms       Ratio to last: 2.0
Array-Size:   4,096       Time:    9.4 ms       Ratio to last: 2.1
Array-Size:   8,192       Time:   19.6 ms       Ratio to last: 2.1
Array-Size:  16,384       Time:   40.6 ms       Ratio to last: 2.1
Array-Size:  32,768       Time:   83.0 ms       Ratio to last: 2.0

*/
```

```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Tue Oct 15 12:37:47 CEST 2024
 */

package ex05.baseline.task02;
import static org.junit.Assert.assertArrayEquals;

import java.util.Arrays;
import java.util.Random;

import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class MergeSortJUnitTest {

  @Test
  public void test01() {
    Integer[] arr = {4, 1, 2, 3};
    sort(arr);
  }

  @Test
  public void test02() {
    Integer[] arr = {2, 4, 3, 1};
    sort(arr);
  }

  @Test
  public void test03() {
    Integer[] arr = {2, 1};
    sort(arr);
  }

  @Test
  public void test04() {
    Integer[] arr = {1, 2};
    sort(arr);
  }

  @Test
  public void test05() {
    Integer[] arr = {1};
    sort(arr);
  }

  @Test
  public void test06() {
    Integer[] arr = {};
    sort(arr);
  }

  @Test
  public void test07StressTest() {
    final int NUMBER_OF_TESTS = 50000;
    final int LENGTH = 128;
    for (int n = 0; n < NUMBER_OF_TESTS; n++) {
      Integer[] arr =
          new Random().ints(LENGTH, 0, 10).boxed().toArray(Integer[]::new);
      sort(arr);
    }
  }

  private void sort(Integer[] arr) {
    Integer[] clonedArr = arr.clone();
    Integer[] sortedArr = MergeSort.mergeSort(arr);
    verify(clonedArr, sortedArr);
  }

  @SuppressWarnings("static-method")
  private void verify(Integer[] orgArr, Integer[] sortedArr) {
    Integer[] sortedOrgArr = Arrays.copyOf(orgArr, orgArr.length);
    Arrays.sort(sortedOrgArr);
    assertArrayEquals(sortedOrgArr, sortedArr);
  }

}

```