```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Tue Oct 15 12:38:16 CEST 2024
 */

package ex05.solution.task01;

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

/**
 * @author tbeeler
 *
 * BubbleSort. Two versions of the bubblesort for sorting integers.
 *
 */
public class BubbleSort {

  /**
   * First version: no optimization.
   *
   * @param <T>
   *          Type of elements to be sorted. Must be comparable.
   * @param sequence
   *          The sequence to be sorted.
   */
  public static <T extends Comparable<? super T>> void bubbleSort1(T[] sequence) {
    boolean bubbled = false;
    T temp;
    do {
      bubbled = false;
      for (int i = 1; i < sequence.length; i++) {
        if (sequence[i].compareTo(sequence[i - 1]) < 0) {
          // swap
          temp = sequence[i];
          sequence[i] = sequence[i - 1];
          sequence[i - 1] = temp;
          bubbled = true;
        }
      }
    } while (bubbled);
  }

  /**
   * Second version with slight optimization: The upper boundary is reduced by
   * one in every iteration (the biggest bubble is on top now).
   *
   * @param <T>
   *          Type of elements to be sorted. Must be comparable.
   * @param sequence
   *          The sequence to be sorted.
   */
  public static <T extends Comparable<? super T>> void bubbleSort2(T[] sequence) {
    boolean bubbled = false;
    T temp;
    int unsortedSize = sequence.length;
    do {
      bubbled = false;
      for (int i = 1; i < unsortedSize; i++) {
        if (sequence[i].compareTo(sequence[i - 1]) < 0) {
          // swap
          temp = sequence[i];
          sequence[i] = sequence[i - 1];
          sequence[i - 1] = temp;
          bubbled = true;
        }
      }
      unsortedSize--;
    } while (bubbled);
  }
```

```java

  public static void main(String args[]) throws Exception {
    int nSequence = 200;
    if (args.length > 0) {
      nSequence = Integer.parseInt(args[0]);
    }
    Integer[] s1 =
        new Random().ints(nSequence, 0, 100).boxed().toArray(Integer[]::new);
    Integer[] s2 = s1.clone();
    if (nSequence > 300) {
      System.out.println("Too many elements, not printing to stdout.");
    } else {
      Arrays.asList(s1).forEach(i -> System.out.print(i + ","));
      System.out.println();
    }
    System.out.print("Bubble sort 1...");
    long then = System.nanoTime();
    bubbleSort1(s1);
    long now = System.nanoTime();
    long d1 = now - then;
    System.out.println("done.");
    System.out.print("Bubble sort 2...");
    then = System.nanoTime();
    bubbleSort2(s2);
    now = System.nanoTime();
    long d2 = now - then;
    System.out.println("done.");
    if (nSequence > 300) {
      System.out.println("Too many elements, not printing to stdout.");
    } else {
      for (int i = 0; i < nSequence; i++) {
        if (s1[i] != s2[i]) {
          System.err.println("Sorting does not match!");
          System.exit(1);
        }
        System.out.print(s2[i] + ",");
      }
      System.out.println();
    }
    System.out.format(
        "Time bubble sort 1 :  Array-Size: %,7d       Time: %,7.1f ms\n",
        nSequence, d1 / 1_000_000.0);
    System.out.format(
        "Time bubble sort 2 :  Array-Size: %,7d       Time: %,7.1f ms\n",
        nSequence, d2 / 1_000_000.0);
  }
}
```

```

/* Session-Log:

$ java -Xint -Xms5m -Xmx5m ex05/solution/task01/BubbleSort
4,93,12,64,76,89,0,88,12,87,18,14,17,57,2,17,25,11,56,88,3,52,73,86,77,25,3,3,68,62,13
,70,62,26,70,35,92,62,61,52,74,53,38,53,19,55,96,14,93,36,55,43,42,21,44,79,26,98,65,4
4,13,94,35,78,57,8,76,58,97,7,5,15,42,98,76,98,71,19,75,3,76,65,33,20,7,59,30,57,86,44
,55,81,45,18,24,0,21,89,98,22,4,49,29,21,59,62,75,43,65,43,0,20,41,14,84,31,87,5,11,75
,86,31,31,60,74,77,25,16,21,35,60,34,59,95,54,25,42,53,34,98,25,98,21,20,13,55,25,36,6
7,16,33,94,61,43,66,83,19,55,89,82,90,43,29,13,63,61,32,40,3,71,98,30,51,29,44,96,56,7
1,60,20,69,42,54,50,88,60,52,29,24,61,76,77,43,74,6,5,85,68,61,94,
Bubble sort 1...done.
Bubble sort 2...done.
0,0,0,2,3,3,3,3,3,4,4,5,5,5,6,7,7,8,11,11,12,12,13,13,13,13,14,14,14,15,16,16,17,17,18
,18,19,19,19,20,20,20,20,21,21,21,21,21,22,24,24,25,25,25,25,25,25,26,26,29,29,29,29,3
0,30,31,31,31,32,33,33,34,34,35,35,35,36,36,38,40,41,42,42,42,42,43,43,43,43,43,43,44,
44,44,44,45,49,50,51,52,52,52,53,53,53,54,54,55,55,55,55,55,56,56,57,57,57,58,59,59,59
,60,60,60,60,61,61,61,61,61,62,62,62,62,63,64,65,65,65,66,67,68,68,69,70,70,71,71,71,7
3,74,74,74,75,75,75,76,76,76,76,76,77,77,77,78,79,81,82,83,84,85,86,86,86,87,87,88,88,
88,89,89,89,90,92,93,93,94,94,94,95,96,96,97,98,98,98,98,98,98,98,
Time bubble sort 1 :  Array-Size:     200       Time:     6.8 ms
Time bubble sort 2 :  Array-Size:     200       Time:     3.9 ms



$ for i in 1024 2048 4096
> do
>  java -Xint -Xms5m -Xmx5m ex05/solution/task01/BubbleSort $i
> done
Too many elements, not printing to stdout.
Bubble sort 1...done.
Bubble sort 2...done.
Too many elements, not printing to stdout.
Time bubble sort 1 :  Array-Size:   1,024       Time:   251.9 ms
Time bubble sort 2 :  Array-Size:   1,024       Time:   134.6 ms
Too many elements, not printing to stdout.
Bubble sort 1...done.
Bubble sort 2...done.
Too many elements, not printing to stdout.
Time bubble sort 1 :  Array-Size:   2,048       Time:   971.9 ms
Time bubble sort 2 :  Array-Size:   2,048       Time:   490.1 ms
Too many elements, not printing to stdout.
Bubble sort 1...done.
Bubble sort 2...done.
Too many elements, not printing to stdout.
Time bubble sort 1 :  Array-Size:   4,096       Time: 3,705.2 ms
Time bubble sort 2 :  Array-Size:   4,096       Time: 1,987.8 ms

*/
```

```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Tue Oct 15 12:38:16 CEST 2024
 */

package ex05.solution.task01;

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 BubbleSortJUnitTest {

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

  @Test
  public void test02() {
    Integer[] arr = {2, 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 = 10000;
    final int LENGTH = 100;
    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();
    BubbleSort.bubbleSort1(arr);
    verify(clonedArr, arr);
    arr = clonedArr.clone();
    BubbleSort.bubbleSort2(arr);
    verify(clonedArr, arr);
  }

  @SuppressWarnings("static-method")
  private void verify(Integer[] orgArr, Integer[] sortedArr) {
    Integer[] sortedOrgArr = new Integer[orgArr.length];
    System.arraycopy(orgArr, 0, sortedOrgArr, 0, orgArr.length);
    Arrays.sort(sortedOrgArr);
    assertArrayEquals(sortedOrgArr, sortedArr);
  }

}

```