```java
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Wed Oct 10 14:42:19 CEST 2018
 */

package uebung05.as.aufgabe01;

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) {
    // TODO Implement here...
  }

  /**
   * 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) {
    // TODO Implement here...
  }
```

```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 uebung05/ml/aufgabe01/BubbleSort
40,82,87,53,91,58,63,61,49,73,61,1,80,92,99,3,84,46,16,52,29,98,87,63,93,70,40,56,54,8
4,9,84,96,43,5,0,13,55,90,33,66,47,85,18,99,97,33,69,62,90,60,17,74,3,74,6,55,22,16,35
,14,50,96,57,70,42,20,76,85,42,9,55,6,75,11,77,65,81,66,99,70,56,4,34,34,16,26,33,98,5
9,33,0,18,84,34,3,99,41,37,54,54,78,47,75,54,69,11,12,92,99,69,95,38,89,3,99,81,68,75,
84,60,71,37,57,26,67,30,4,72,69,27,39,77,95,49,79,2,29,45,73,86,35,12,52,35,73,8,3,84,
20,83,96,16,15,54,36,51,21,5,49,63,82,26,9,69,30,55,32,91,95,46,6,91,30,60,4,38,3,21,8
0,78,87,36,60,49,39,87,15,4,49,30,48,13,35,26,86,50,54,64,37,
Bubble sort 1...done.
Bubble sort 2...done.
0,0,1,2,3,3,3,3,3,3,4,4,4,4,5,5,6,6,6,8,9,9,9,11,11,12,12,13,13,14,15,15,16,16,16,16,1
7,18,18,20,20,21,21,22,26,26,26,26,27,29,29,30,30,30,30,32,33,33,33,33,34,34,34,35,35,
35,35,36,36,37,37,37,38,38,39,39,40,40,41,42,42,43,45,46,46,47,47,48,49,49,49,49,49,50
,50,51,52,52,53,54,54,54,54,54,54,55,55,55,55,56,56,57,57,58,59,60,60,60,60,61,61,62,6
3,63,63,64,65,66,66,67,68,69,69,69,69,69,70,70,70,71,72,73,73,73,74,74,75,75,75,76,77,
77,78,78,79,80,80,81,81,82,82,83,84,84,84,84,84,84,85,85,86,86,87,87,87,87,89,90,90,91
,91,91,92,92,93,95,95,95,96,96,96,97,98,98,99,99,99,99,99,99,
Time bubble sort 1 :  Array−Size:     200       Time:    12.5 ms
Time bubble sort 2 :  Array−Size:     200       Time:     6.9 ms

*/
```

```
/*
 * HSR − Uebungen ’Algorithmen & Datenstrukturen 2’
 * Version: Wed Oct 10 14:45:53 CEST 2018
 */

package uebung05.as.aufgabe01;

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);
  }

  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);
  }

}

```