```java
/*
 * HSR – Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Mon Oct 22 12:14:37 CEST 2018
 */

package uebung06.as.aufgabe01;

import java.awt.Point;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Random;

/**
 * InPlaceQuickSort from "Data Structures and Algorithms" implemented to use a
 * comparator instead. This allows the usage of multiple keys.
 *
 * @author tbeeler
 */
public class QuickSort {

  /**
   * @param <T>
   *          Type of elements to be sorted.
   * @param sequence
   *          The sequence to be sorted.
   * @param comp
   *          The comperator to be used.
   * @param a
   *          The leftbound of the part that shall be sorted.
   * @param b
   *          The rightbound of the part that shall be sorted.
   */
  public <T> void inPlaceQuickSort(T[] sequence, Comparator<T> comp, int a, int b) {

    // TODO Implement here...

  }
```



```java

  enum SequenceType {RANDOM, EQUAL, SORTED};

  public static void main(String[] args) {
    Comparator<Point> comp = new PointComparator();
    QuickSort qs = new QuickSort();
    Random random = new Random(0);
    int nSequence = 50;
    if (args.length > 0)
      nSequence = Integer.parseInt(args[0]);
    final Point s1[] = new Point[nSequence];
    for (int i = 0; i < s1.length; i++) {
      s1[i] = new Point((int)(random.nextDouble() * 100),
                        (int)(random.nextDouble() * 100));
    }
    printArray(s1);
    System.out.print("Quick sort...");
    long then = System.currentTimeMillis();
    qs.inPlaceQuickSort(s1, comp, 0, nSequence - 1);
    long now = System.currentTimeMillis();
    long d1 = now - then;
    System.out.println("done.");
    printArray(s1);
    System.out.println("Time quick sort [ms]: " + d1);

    System.out.println("\nRuntime:");
    final int INITIAL_SIZE = 64;
    final int NR_OF_DOUBLING = 4;
    for (SequenceType seqType: SequenceType.values()) {
      System.out.println("Sequence-Type: "+seqType);
      int arraySize = INITIAL_SIZE;
      long lastTime = Long.MAX_VALUE;
      for (int j = 0; j <= NR_OF_DOUBLING; j++) {
        final int MEASUREMENTS = 200;
        long minimalTime = Long.MAX_VALUE;
        Point[] s2 = new Point[arraySize];
        for (int m = 0; m < MEASUREMENTS; m++) {
          for (int i = 0; i < s2.length; i++) {
            switch (seqType) {
              case RANDOM:
                s2[i] = new Point((int)(Math.random() * 100),
                                  (int)(Math.random() * 100));
                break;
              case EQUAL:
                s2[i] = new Point(1, 1);
                break;
              case SORTED:
                s2[i] = new Point(i, i);
                break;
            }
          }
          long startTime = System.nanoTime();
          qs.inPlaceQuickSort(s2, comp, 0, arraySize - 1);
          long endTime = System.nanoTime();
          long time = endTime - startTime;
          if (time < minimalTime) {
            minimalTime = time;
          }
          Point[] test = s2.clone();
          Arrays.sort(test, comp);
          if (!Arrays.equals(s2, test)) {
            System.err.println("ERROR:");
            System.err.println(Arrays.toString(s2));
            System.err.println(Arrays.toString(test));
            System.exit(1);
          }
        }
```

```
105
106         System.out.format(
107           "Array-Size: %,6d   Time: %,7.0f us   Ratio to last: %2.1f\n",
108           arraySize, (double) minimalTime / (long) 1e3,
109           (double) minimalTime / lastTime);
110         lastTime = minimalTime;
111         arraySize = arraySize * 2;
112       }
113     }
114   }
115
116   private static void printArray(final Point[] array) {
117     if (array.length > 300) {
118       System.out.println("Too many elements, not printing to stdout.");
119     } else {
120       for (Point point: array) {
121         System.out.print("(" + point.x + "/" + point.y + "), ");
122       }
123       System.out.println();
124     }
125   }
126
127 }
128
129
130 class PointComparator implements Comparator<Point> {
131
132   /**
133    * Total order relation for points:
134    * p1 > p2 | p1.x > p2.x
135    * p1 > p2 | p1.x = p2.x && p1.y > p2.y
136    * p1 = p2 | p1.x = p2.x && p1.y = p2.y
137    * else p1 < p2
138    *
139    * @return p1 > p2  : +1,
140    *         p1 == p2 :  0,
141    *         p1 < p2  : -1
142    *
143    * @author tbeeler
144    */
145   public int compare(Point p1, Point p2) {
146
147     // TODO Implement here...
148
149     return 0;
150   }
151
152 }
```



```
153
154 /* Session-Log:
155
156 $ java -Xint -Xms10m -Xmx10m uebung06.as.aufgabe01.QuickSort
157
158 (73/24), (63/55), (59/33), (38/98), (87/94), (27/12), (14/2), (54/96), (10/62), (41/77
    ), (99/48), (74/73), (81/83), (52/89), (13/8), (97/72), (71/14), (46/0), (7/34), (33/8
    5), (97/86), (61/17), (21/85), (0/69), (77/71), (21/78), (94/1), (39/85), (78/99), (88
    /17), (96/72), (67/80), (44/46), (85/50), (99/96), (35/4), (7/2), (48/97), (98/76), (5
    0/25), (30/84), (5/1), (35/8), (85/0), (30/53), (91/27), (87/60), (90/4), (64/49), (50
    /52),
159 Quick sort...done.
160 (0/69), (5/1), (7/2), (7/34), (10/62), (13/8), (14/2), (21/78), (21/85), (27/12), (30/
    53), (30/84), (33/85), (35/4), (35/8), (38/98), (39/85), (41/77), (44/46), (46/0), (48
    /97), (50/25), (50/52), (52/89), (54/96), (59/33), (61/17), (63/55), (64/49), (67/80),
     (71/14), (73/24), (74/73), (77/71), (78/99), (81/83), (85/0), (85/50), (87/60), (87/9
    4), (88/17), (90/4), (91/27), (94/1), (96/72), (97/72), (97/86), (98/76), (99/48), (99
    /96),
161 Time quick sort [ms]: 0
162
163 Runtime:
164 Sequence-Type: RANDOM
165 Array-Size:     64   Time:     109 us   Ratio to last: 0.0
166 Array-Size:    128   Time:     256 us   Ratio to last: 2.3
167 Array-Size:    256   Time:     569 us   Ratio to last: 2.2
168 Array-Size:    512   Time:   1,294 us   Ratio to last: 2.3
169 Array-Size:  1,024   Time:   2,853 us   Ratio to last: 2.2
170 Sequence-Type: EQUAL
171 Array-Size:     64   Time:      19 us   Ratio to last: 0.0
172 Array-Size:    128   Time:      37 us   Ratio to last: 2.0
173 Array-Size:    256   Time:      74 us   Ratio to last: 2.0
174 Array-Size:    512   Time:     148 us   Ratio to last: 2.0
175 Array-Size:  1,024   Time:     296 us   Ratio to last: 2.0
176 Sequence-Type: SORTED
177 Array-Size:     64   Time:     307 us   Ratio to last: 0.0
178 Array-Size:    128   Time:   1,180 us   Ratio to last: 3.8
179 Array-Size:    256   Time:   4,638 us   Ratio to last: 3.9
180 Array-Size:    512   Time:  18,467 us   Ratio to last: 4.0
181 Array-Size:  1,024   Time:  74,368 us   Ratio to last: 4.0
182
183 */
184
```

```java
/*
 * HSR − Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Mon Oct 22 12:14:37 CEST 2018
 */

package uebung06.as.aufgabe01;

import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertTrue;

import java.awt.Point;
import java.util.Arrays;
import java.util.Comparator;
import java.util.stream.IntStream;

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

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class QuickSortJUnitTest {

  private QuickSort qs = new QuickSort();
  private Comparator<Point> comp = new PointComparator();

  @Test
  public void test01() {

    // Test−Points(x*,y*):
    int[] x = { 7, 5, 5, 1, 5, 3 };
    int[] y = { 7, 6, 5, 9, 4, 3 };
    // Sorted:
    int[] xSorted = { 1, 3, 5, 5, 5, 7 };
    int[] ySorted = { 9, 3, 4, 5, 6, 7 };

    test(x, y, xSorted, ySorted);
  }


  @Test
  public void test02() {

    // Test−Points(x*,y*):
    int[] x = { 1, 2, 3 };
    int[] y = { 1, 2, 3 };
    // Sorted:
    int[] xSorted = { 1, 2, 3 };
    int[] ySorted = { 1, 2, 3 };

    test(x, y, xSorted, ySorted);
  }

  @Test
  public void test03() {

    // Test−Points(x*,y*):
    int[] x = { 3, 2, 1 };
    int[] y = { 3, 2, 1 };
    // Sorted:
    int[] xSorted = { 1, 2, 3 };
    int[] ySorted = { 1, 2, 3 };

    test(x, y, xSorted, ySorted);
  }
```



```java

  @Test
  public void test04() {

    // Test−Points(x*,y*):
    int[] x = { 2, 2 };
    int[] y = { 2, 2 };
    // Sorted:
    int[] xSorted = { 2, 2 };
    int[] ySorted = { 2, 2 };

    test(x, y, xSorted, ySorted);
  }

  @Test
  public void test05StressTest() {
    IntStream.range(1, 200).forEach(
      len −> IntStream.range(0, len).forEach(i −> {
        Point[] sequence = new Point[len];
        IntStream.range(0, sequence.length).forEach(
          j −> sequence[j] = new Point((int) (Math.random() * len/2),
                                       (int) (Math.random() * len/2)));

        qs.inPlaceQuickSort(sequence, comp, 0, sequence.length − 1);

        Point minValuePoint = new Point(Integer.MIN_VALUE, Integer.MIN_VALUE);
        Arrays.stream(sequence).reduce(minValuePoint, (a, b) −> {
          assertTrue(a.getX() <= b.getX());
          if (a.getX() == b.getX()) {
            assertTrue(a.getY() <= b.getY());
          }
          return b;
        });
      })
    );
  }


  private void test(int[] x, int[] y, int[] xSorted, int[] ySorted) {
    Point[] sequence = new Point[x.length];
    IntStream.range(0, x.length).forEach(
      i −> sequence[i] = new Point(x[i], y[i]));
    System.out.println(Arrays.toString(sequence));
    Point[] sequenceSorted = new Point[ySorted.length];
    IntStream.range(0, xSorted.length).forEach(
      i −> sequenceSorted[i] = new Point(xSorted[i], ySorted[i]));

    qs.inPlaceQuickSort(sequence, comp, 0, sequence.length − 1);

    assertArrayEquals(sequenceSorted, sequence);
  }

}
```