

15.10.2024 12:38:16

BubbleSort.java

Page 1/3

```

1  /*
2   * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3   * Version: Tue Oct 15 12:38:16 CEST 2024
4   */
5
6  package ex05.solution.task01;
7
8  import java.util.Arrays;
9  import java.util.Random;
10
11 /**
12  * @author tbeeler
13  *
14  * BubbleSort. Two versions of the bubblesort for sorting integers.
15  */
16
17 public class BubbleSort {
18
19     /**
20      * First version: no optimization.
21      *
22      * @param <T>
23      *      Type of elements to be sorted. Must be comparable.
24      * @param sequence
25      *      The sequence to be sorted.
26      */
27     public static <T extends Comparable<? super T>> void bubbleSort1(T[] sequence) {
28         boolean bubbled = false;
29         T temp;
30         do {
31             bubbled = false;
32             for (int i = 1; i < sequence.length; i++) {
33                 if (sequence[i].compareTo(sequence[i - 1]) < 0) {
34                     // swap
35                     temp = sequence[i];
36                     sequence[i] = sequence[i - 1];
37                     sequence[i - 1] = temp;
38                     bubbled = true;
39                 }
40             } while (bubbled);
41         }
42     }
43
44     /**
45      * Second version with slight optimization: The upper boundary is reduced by
46      * one in every iteration (the biggest bubble is on top now).
47      *
48      * @param <T>
49      *      Type of elements to be sorted. Must be comparable.
50      * @param sequence
51      *      The sequence to be sorted.
52      */
53     public static <T extends Comparable<? super T>> void bubbleSort2(T[] sequence) {
54         boolean bubbled = false;
55         T temp;
56         int unsortedSize = sequence.length;
57         do {
58             bubbled = false;
59             for (int i = 1; i < unsortedSize; i++) {
60                 if (sequence[i].compareTo(sequence[i - 1]) < 0) {
61                     // swap
62                     temp = sequence[i];
63                     sequence[i] = sequence[i - 1];
64                     sequence[i - 1] = temp;
65                     bubbled = true;
66                 }
67             }
68             unsortedSize--;
69         } while (bubbled);
70     }

```

15.10.2024 12:38:16

BubbleSort.java

Page 2/3

```

71
72     public static void main(String args[]) throws Exception {
73         int nSequence = 200;
74         if (args.length > 0) {
75             nSequence = Integer.parseInt(args[0]);
76         }
77         Integer[] s1 =
78             new Random().ints(nSequence, 0, 100).boxed().toArray(Integer[]::new);
79         Integer[] s2 = s1.clone();
80         if (nSequence > 300) {
81             System.out.println("Too many elements, not printing to stdout.");
82         } else {
83             Arrays.asList(s1).forEach(i -> System.out.print(i + ", "));
84             System.out.println();
85         }
86         System.out.print("Bubble sort 1...");
87         long then = System.nanoTime();
88         bubbleSort1(s1);
89         long now = System.nanoTime();
90         long d1 = now - then;
91         System.out.println("done.");
92         System.out.print("Bubble sort 2...");
93         then = System.nanoTime();
94         bubbleSort2(s2);
95         now = System.nanoTime();
96         long d2 = now - then;
97         System.out.println("done.");
98         if (nSequence > 300) {
99             System.out.println("Too many elements, not printing to stdout.");
100         } else {
101             for (int i = 0; i < nSequence; i++) {
102                 if (s1[i] != s2[i]) {
103                     System.err.println("Sorting does not match!");
104                     System.exit(1);
105                 }
106                 System.out.print(s2[i] + ", ");
107             }
108             System.out.println();
109         }
110         System.out.format(
111             "Time bubble sort 1 : Array-Size: %7d          Time: %7.1f ms\n",
112             nSequence, d1 / 1_000_000.0);
113         System.out.format(
114             "Time bubble sort 2 : Array-Size: %7d          Time: %7.1f ms\n",
115             nSequence, d2 / 1_000_000.0);
116     }
117 }
118
119
120

```

15.10.2024 12:38:16

BubbleSort.java

Page 3/3

```

121
122 /* Session-Log:
123
124 $ java -Xint -Xms5m -Xmx5m ex05/solution/task01/BubbleSort
125 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,
126 Bubble sort 1...done.
127 Bubble sort 2...done.
128 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
0,30,31,31,31,32,33,33,34,34,35,35,35,36,36,38,40,41,42,42,42,42,43,43,43,43,43,44,
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,
88,89,89,89,90,92,93,93,94,94,94,95,96,96,97,98,98,98,98,98,98,98,98,98,98,98,98,98,
129 Time bubble sort 1 : Array-Size: 200 Time: 6.8 ms
130 Time bubble sort 2 : Array-Size: 200 Time: 3.9 ms
131
132
133
134 $ for i in 1024 2048 4096
135 > do
136 > java -Xint -Xms5m -Xmx5m ex05/solution/task01/BubbleSort $i
137 > done
138 Too many elements, not printing to stdout.
139 Bubble sort 1...done.
140 Bubble sort 2...done.
141 Too many elements, not printing to stdout.
142 Time bubble sort 1 : Array-Size: 1,024 Time: 251.9 ms
143 Time bubble sort 2 : Array-Size: 1,024 Time: 134.6 ms
144 Too many elements, not printing to stdout.
145 Bubble sort 1...done.
146 Bubble sort 2...done.
147 Too many elements, not printing to stdout.
148 Time bubble sort 1 : Array-Size: 2,048 Time: 971.9 ms
149 Time bubble sort 2 : Array-Size: 2,048 Time: 490.1 ms
150 Too many elements, not printing to stdout.
151 Bubble sort 1...done.
152 Bubble sort 2...done.
153 Too many elements, not printing to stdout.
154 Time bubble sort 1 : Array-Size: 4,096 Time: 3,705.2 ms
155 Time bubble sort 2 : Array-Size: 4,096 Time: 1,987.8 ms
156
157 */
158
159
160
161

```

15.10.2024 12:38:16

BubbleSortJUnitTest.java

Page 1/2

```

1 /*
2  * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3  * Version: Tue Oct 15 12:38:16 CEST 2024
4  */
5
6 package ex05.solution.task01;
7
8 import static org.junit.Assert.assertEquals;
9
10 import java.util.Arrays;
11 import java.util.Random;
12
13 import org.junit.FixMethodOrder;
14 import org.junit.Test;
15 import org.junit.runners.MethodSorters;
16
17 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
18 public class BubbleSortJUnitTest {
19
20     @Test
21     public void test01() {
22         Integer[] arr = {3, 1, 2};
23         sort(arr);
24     }
25
26     @Test
27     public void test02() {
28         Integer[] arr = {2, 3, 1};
29         sort(arr);
30     }
31
32     @Test
33     public void test03() {
34         Integer[] arr = {2, 1};
35         sort(arr);
36     }
37
38     @Test
39     public void test04() {
40         Integer[] arr = {1, 2};
41         sort(arr);
42     }
43
44     @Test
45     public void test05() {
46         Integer[] arr = {1};
47         sort(arr);
48     }
49
50     @Test
51     public void test06() {
52         Integer[] arr = {};
53         sort(arr);
54     }
55

```

15.10.2024 12:38:16

BubbleSortJUnitTest.java

Page 2/2

```
55
56 @Test
57 public void test07StressTest() {
58     final int NUMBER_OF_TESTS = 10000;
59     final int LENGTH = 100;
60     for (int n = 0; n < NUMBER_OF_TESTS; n++) {
61         Integer[] arr =
62             new Random().ints(LENGTH, 0, 10).boxed().toArray(Integer[]::new);
63         sort(arr);
64     }
65 }
66
67 private void sort(Integer[] arr) {
68     Integer[] clonedArr = arr.clone();
69     BubbleSort.bubbleSort1(arr);
70     verify(clonedArr, arr);
71     arr = clonedArr.clone();
72     BubbleSort.bubbleSort2(arr);
73     verify(clonedArr, arr);
74 }
75
76 @SuppressWarnings("static-method")
77 private void verify(Integer[] orgArr, Integer[] sortedArr) {
78     Integer[] sortedOrgArr = new Integer[orgArr.length];
79     System.arraycopy(orgArr, 0, sortedOrgArr, 0, orgArr.length);
80     Arrays.sort(sortedOrgArr);
81     assertEquals(sortedOrgArr, sortedArr);
82 }
83
84 }
85
```