

14.9.2019 18:12:49

## BinarySearchArrayTest.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sat Sep 14 18:03:13 CEST 2019
4  */
5
6  package uebung01.as.aufgabe04;
7
8  import java.util.ArrayList;
9  import java.util.Random;
10
11 public class BinarySearchArrayTest {
12
13     protected ArrayList<Integer> arrayList;
14
15     public BinarySearchArrayTest() {
16         arrayList = new ArrayList<Integer>();
17     }
18
19     public void clear() {
20         arrayList = new ArrayList<Integer>();
21     }
22
23     public void generateTree(int nodes) {
24         for (int i: new Random().ints(nodes, 0, Integer.MAX_VALUE).toArray()) {
25             if (arrayList.size() == 0)
26                 arrayList.add(i);
27             else
28                 add(0, arrayList.size() - 1, i);
29         }
30     }
31
32     /**
33      * Adds 'content' recursively into the ArrayList by applying a Binary-Search.
34      *
35      * @param lower The lower bound (inclusive) of the range where to insert the content
36      *
37      * @param upper The upper bound (inclusive) of the range where to insert the content
38      *
39      * @param content The number to insert into the ArrayList.
40      */
41     public void add(int lower, int upper, int content) {
42         // TODO Implement here...
43     }
44
45     public boolean verify(int size, boolean exiting) {
46         int arrayListSize = arrayList.size();
47         if (arrayListSize != size) {
48             System.err.println("ERROR: bad size: " + arrayListSize);
49             if (exiting) {
50                 System.exit(1);
51             } else {
52                 return false;
53             }
54         }
55         int lhs = Integer.MIN_VALUE;
56         boolean failure = false;
57         for (int i = 0; i < arrayList.size(); i++) {
58             int rhs = arrayList.get(i);
59             if (lhs > rhs) {
60                 System.out.format("ERROR: wrong order at [%d]: %d > %d\n", i, lhs, rhs);
61                 failure = true;
62                 break;
63             }
64             lhs = rhs;
65         }

```

14.9.2019 18:12:49

## BinarySearchArrayTest.java

Page 2/2

```

66     if (failure) {
67         if (arrayListSize < 20) {
68             System.out.println(arrayList);
69         }
70         if (exiting) {
71             System.exit(2);
72         } else {
73             return false;
74         }
75     }
76     return true;
77 }
78
79 public static void main(String[] args) {
80     System.out.println("ARRAYLIST based TEST");
81     System.out.println("Please be patient, the following operations may take some time
82     ...");
83     final int BEGINSIZE = 10000;
84     final int TESTRUNS = 100;
85     final int VARYSIZE = 10;
86
87     BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();
88     double avgTime = 0;
89     long startTime;
90     for (int i = 0; i < TESTRUNS; i++) {
91         binarySearchArray.clear();
92         startTime = System.currentTimeMillis();
93         int size = BEGINSIZE + i * VARYSIZE;
94         binarySearchArray.generateTree(size);
95         avgTime = ((avgTime * i) + (System.currentTimeMillis() - startTime))
96             / (i + 1);
97         binarySearchArray.verify(size, true);
98     }
99     System.out.println("Test successful, result is as follows:");
100     System.out.println("Average time for generation is: " + avgTime + " ms");
101 }
102
103 }
104
105
106 /* Session-Log:
107
108 ARRAYLIST based TEST
109 Please be patient, the following operations may take some time...
110 Test successful, result is as follows:
111 Average time for generation is: 5.16ms
112
113 */
114

```

14.9.2019 18:05:49

## BinarySearchArrayJUnitTest.java

Page 1/2

```

1  /*
2  * HSR - Uebungen 'Algorithmen & Datenstrukturen 2'
3  * Version: Sat Sep 14 18:03:13 CEST 2019
4  */
5
6  package uebung01.as.aufgabe04;
7
8  import static org.junit.Assert.assertTrue;
9
10 import java.util.Arrays;
11 import java.util.List;
12 import java.util.Random;
13 import java.util.stream.Collectors;
14
15 import org.junit.Before;
16 import org.junit.FixMethodOrder;
17 import org.junit.Test;
18 import org.junit.runners.MethodSorters;
19
20 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
21 public class BinarySearchArrayJUnitTest {
22
23     // Stress-Test:
24     private static final int NUMBER_OF_TESTS = 10_000;
25     private static final int MIN_SIZE = 1;
26     private static final int MAX_SIZE = 32;
27     private static final int LOWER_BOUND = 0; // inclusive
28     private static final int UPPER_BOUND = 10; // inclusive
29
30     BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();
31
32     @Before
33     public void setUp() {
34         binarySearchArray.clear();
35     }
36
37     @Test
38     public void test_1() {
39         fillBinarySearchArray(Arrays.asList(1, 2));
40         assertTrue(binarySearchArray.verify(2, false));
41     }
42
43     @Test
44     public void test_2() {
45         fillBinarySearchArray(Arrays.asList(2, 1));
46         assertTrue(binarySearchArray.verify(2, false));
47     }
48
49     @Test
50     public void test_3() {
51         fillBinarySearchArray(Arrays.asList(1, 1));
52         assertTrue(binarySearchArray.verify(2, false));
53     }
54
55     @Test
56     public void test_4() {
57         fillBinarySearchArray(Arrays.asList(1, 2, 3));
58         assertTrue(binarySearchArray.verify(3, false));
59     }
60
61     @Test
62     public void test_5() {
63         fillBinarySearchArray(Arrays.asList(3, 2, 1));
64         assertTrue(binarySearchArray.verify(3, false));
65     }

```

14.9.2019 18:05:49

## BinarySearchArrayJUnitTest.java

Page 2/2

```

66
67     @Test
68     public void test_6() {
69         fillBinarySearchArray(Arrays.asList(3, 1, 2));
70         assertTrue(binarySearchArray.verify(3, false));
71     }
72
73     @Test
74     public void test_7() {
75         fillBinarySearchArray(Arrays.asList(1, 1, 1));
76         assertTrue(binarySearchArray.verify(3, false));
77     }
78
79     @Test
80     public void test_StressTest() {
81         new Random().ints(NUMBER_OF_TESTS, MIN_SIZE, MAX_SIZE + 1).forEach(size -> {
82             List<Integer> list = new Random()
83                 .ints(size, LOWER_BOUND, UPPER_BOUND + 1).boxed()
84                 .collect(Collectors.toList());
85             System.out.println(list);
86             binarySearchArray.clear();
87             fillBinarySearchArray(list);
88             System.out.println(binarySearchArray.arrayList());
89             assertTrue(binarySearchArray.verify(list.size(), false));
90         });
91     }
92
93     private void fillBinarySearchArray(List<Integer> list) {
94         for (int i: list) {
95             if (binarySearchArray.arrayList.size() == 0) {
96                 binarySearchArray.arrayList.add(i);
97             } else {
98                 binarySearchArray.add(0, binarySearchArray.arrayList.size() - 1, i);
99             }
100         }
101     }
102
103 }

```