```java
1  /*
2   * OST – Uebungen 'Algorithmen & Datenstrukturen 2'
3   * Version: Wed Sep  9 18:48:52 CEST 2020
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    * @param upper The upper bound (inclusive) of the range where to insert the content
37    * @param content The number to insert into the ArrayList.
38    */
39   public void add(int lower, int upper, int content) {
40
41     // TODO Implement here...
42
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     }
```

```java
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     final int BEGINSIZE = 10000;
83     final int TESTRUNS = 100;
84     final int VARYSIZE = 10;
85
86     BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();
87     double avgTime = 0;
88     long startTime;
89     for (int i = 0; i < TESTRUNS; i++) {
90       binarySearchArray.clear();
91       startTime = System.currentTimeMillis();
92       int size = BEGINSIZE + i * VARYSIZE;
93       binarySearchArray.generateTree(size);
94       avgTime = ((avgTime * i) + (System.currentTimeMillis() - startTime))
95           / (i + 1);
96       binarySearchArray.verify(size, true);
97     }
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
```

```java
/*
 * OST – Uebungen 'Algorithmen & Datenstrukturen 2'
 * Version: Wed Sep  9 18:48:52 CEST 2020
 */

package uebung01.as.aufgabe04;

import static org.junit.Assert.assertTrue;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

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

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class BinarySearchArrayJUnitTest {

  // Stress-Test:
  private static final int NUMBER_OF_TESTS = 10_000;
  private static final int MIN_SIZE =  1;
  private static final int MAX_SIZE = 32;
  private static final int LOWER_BOUND =  0; // inclusive
  private static final int UPPER_BOUND = 10; // inclusive

  BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();

  @Before
  public void setUp() {
    binarySearchArray.clear();
  }

  @Test
  public void test_1() {
    fillBinarySearchArray(Arrays.asList(1, 2));
    assertTrue(binarySearchArray.verify(2, false));
  }

  @Test
  public void test_2() {
    fillBinarySearchArray(Arrays.asList(2, 1));
    assertTrue(binarySearchArray.verify(2, false));
  }

  @Test
  public void test_3() {
    fillBinarySearchArray(Arrays.asList(1, 1));
    assertTrue(binarySearchArray.verify(2, false));
  }

  @Test
  public void test_4() {
    fillBinarySearchArray(Arrays.asList(1, 2, 3));
    assertTrue(binarySearchArray.verify(3, false));
  }

  @Test
  public void test_5() {
    fillBinarySearchArray(Arrays.asList(3, 2, 1));
    assertTrue(binarySearchArray.verify(3, false));
  }
```

```java
  @Test
  public void test_6() {
    fillBinarySearchArray(Arrays.asList(3, 1, 2));
    assertTrue(binarySearchArray.verify(3, false));
  }

  @Test
  public void test_7() {
    fillBinarySearchArray(Arrays.asList(1, 1, 1));
    assertTrue(binarySearchArray.verify(3, false));
  }

  @Test
  public void test_StressTest() {
    new Random().ints(NUMBER_OF_TESTS, MIN_SIZE, MAX_SIZE + 1).forEach(size -> {
      List<Integer> list = new Random()
          .ints(size, LOWER_BOUND, UPPER_BOUND + 1).boxed()
          .collect(Collectors.toList());
      System.out.println(list);
      binarySearchArray.clear();
      fillBinarySearchArray(list);
      System.out.println(binarySearchArray.arrayList);
      assertTrue(binarySearchArray.verify(list.size(), false));
    });
  }

  private void fillBinarySearchArray(List<Integer> list) {
    for (int i: list) {
      if (binarySearchArray.arrayList.size() == 0) {
        binarySearchArray.arrayList.add(i);
      } else {
        binarySearchArray.add(0, binarySearchArray.arrayList.size() - 1, i);
      }
    }
  }

}
```