```java
1   /*
2    * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3    * Version: Sun Sep 15 17:57:14 CEST 2024
4    */
5
6   package ex01.solution.task04;
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<>();
17    }
18
19    public void clear() {
20      arrayList = new ArrayList<>();
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      if (lower == upper) { // we found the insert-position
41        if (content >= arrayList.get(lower)) {
42          arrayList.add(lower+1, content);
43        } else {
44          arrayList.add(lower, content);
45        }
46        return;
47      }
48      // we have to search further:
49      int middle = (lower + upper) / 2;
50      if (content > arrayList.get(middle)) {
51        add(middle+1, upper, content);
52      } else {
53        add(lower, middle, content);
54      }
55    }
56
57    public boolean verify(int size, boolean exiting) {
58      int arrayListSize = arrayList.size();
59      if (arrayListSize != size) {
60        System.err.println("ERROR: bad size: " + arrayListSize);
61        if (exiting) {
62          System.exit(1);
63        } else {
64          return false;
65        }
66      }
```

```java
67      int lhs = Integer.MIN_VALUE;
68      boolean failure = false;
69      for (int i = 0; i < arrayList.size(); i++) {
70        int rhs = arrayList.get(i);
71        if (lhs > rhs) {
72          System.out.format("ERROR: wrong order at [%d]: %d > %d\n", i, lhs, rhs);
73          failure = true;
74          break;
75        }
76        lhs = rhs;
77      }
78      if (failure) {
79        if (arrayListSize < 20) {
80          System.out.println(arrayList);
81        }
82        if (exiting) {
83          System.exit(2);
84        } else {
85          return false;
86        }
87      }
88      return true;
89    }
90
91    public static void main(String[] args) {
92      System.out.println("ARRAYLIST based TEST");
93      System.out.println("Please be patient, the following operations may take some time
    ...");
94      final int BEGINSIZE = 10000;
95      final int TESTRUNS = 100;
96      final int VARYSIZE = 10;
97
98      BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();
99      double avgTime = 0;
100     long startTime;
101     for (int i = 0; i < TESTRUNS; i++) {
102       binarySearchArray.clear();
103       startTime = System.currentTimeMillis();
104       int size = BEGINSIZE + i * VARYSIZE;
105       binarySearchArray.generateTree(size);
106       avgTime = ((avgTime * i) + (System.currentTimeMillis() - startTime))
107           / (i + 1);
108       binarySearchArray.verify(size, true);
109     }
110
111     System.out.println("Test successful, result is as follows:");
112     System.out.println("Average time for generation is: " + avgTime + " ms");
113   }
114
115  }
116
117
118  /* Session-Log:
119
120  ARRAYLIST based TEST
121  Please be patient, the following operations may take some time...
122  Test successful, result is as follows:
123  Average time for generation is: 5.16ms
124
125  */
126
```

```java
1   /*
2    * OST – Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
3    * Version: Sun Sep 15 17:57:14 CEST 2024
4    */
5
6   package ex01.solution.task04;
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    }
```

```java
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 }
```