```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Sun Sep 15 17:54:57 CEST 2024
 */

package ex01.baseline.task04;

import java.util.ArrayList;
import java.util.Random;

public class BinarySearchArrayTest {

  protected ArrayList<Integer> arrayList;

  public BinarySearchArrayTest() {
    arrayList = new ArrayList<>();
  }

  public void clear() {
    arrayList = new ArrayList<>();
  }

  public void generateTree(int nodes) {
    for (int i: new Random().ints(nodes, 0, Integer.MAX_VALUE).toArray()) {
      if (arrayList.size() == 0)
        arrayList.add(i);
      else
        add(0, arrayList.size() - 1, i);
    }
  }

  /**
   * Adds 'content' recursively into the ArrayList by applying a Binary-Search.
   *
   * @param lower The lower bound (inclusive) of the range where to insert the content
.
   * @param upper The upper bound (inclusive) of the range where to insert the content
.
   * @param content The number to insert into the ArrayList.
   */
  public void add(int lower, int upper, int content) {

    // TODO Implement here...

  }

  public boolean verify(int size, boolean exiting) {
    int arrayListSize = arrayList.size();
    if (arrayListSize != size) {
      System.err.println("ERROR: bad size: " + arrayListSize);
      if (exiting) {
        System.exit(1);
      } else {
        return false;
      }
    }
    int lhs = Integer.MIN_VALUE;
    boolean failure = false;
    for (int i = 0; i < arrayList.size(); i++) {
      int rhs = arrayList.get(i);
      if (lhs > rhs) {
        System.out.format("ERROR: wrong order at [%d]: %d > %d\n", i, lhs, rhs);
        failure = true;
        break;
      }
      lhs = rhs;
    }
```

```java
    if (failure) {
      if (arrayListSize < 20) {
        System.out.println(arrayList);
      }
      if (exiting) {
        System.exit(2);
      } else {
        return false;
      }
    }
    return true;
  }

  public static void main(String[] args) {
    System.out.println("ARRAYLIST based TEST");
    System.out.println("Please be patient, the following operations may take some time
...");
    final int BEGINSIZE = 10000;
    final int TESTRUNS = 100;
    final int VARYSIZE = 10;

    BinarySearchArrayTest binarySearchArray = new BinarySearchArrayTest();
    double avgTime = 0;
    long startTime;
    for (int i = 0; i < TESTRUNS; i++) {
      binarySearchArray.clear();
      startTime = System.currentTimeMillis();
      int size = BEGINSIZE + i * VARYSIZE;
      binarySearchArray.generateTree(size);
      avgTime = ((avgTime * i) + (System.currentTimeMillis() - startTime))
          / (i + 1);
      binarySearchArray.verify(size, true);
    }

    System.out.println("Test successful, result is as follows:");
    System.out.println("Average time for generation is: " + avgTime + " ms");
  }

}


/* Session-Log:

ARRAYLIST based TEST
Please be patient, the following operations may take some time...
Test successful, result is as follows:
Average time for generation is: 5.16ms

*/
```

```java
/*
 * OST - Uebungen 'Algorithmen & Datenstrukturen (AlgDat)'
 * Version: Sun Sep 15 17:54:57 CEST 2024
 */

package ex01.baseline.task04;

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);
      }
    }
  }

}
```