```java
import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber2;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Program with implementation of {@code NaturalNumber} secondary operation
 * {@code root} implemented as static method.
 *
 * @author Jeng Zhuang
 *
 */
public final class NaturalNumberRoot {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private NaturalNumberRoot() {
    }

    /**
     * Updates {@code n} to the {@code r}-th root of its incoming value.
     *
     * @param n
     *            the number whose root to compute
     * @param r
     *            root
     * @updates n
     * @requires r >= 2
     * @ensures n ^ (r) <= #n < (n + 1) ^ (r)
     */
    public static void root(NaturalNumber n, int r) {
        assert n != null : "Violation of: n is  not null";
        assert r >= 2 : "Violation of: r >= 2";

        // Initialize low and high for the interval
        NaturalNumber low = new NaturalNumber2(0);
```

```
38          NaturalNumber high = new NaturalNumber2(n);
39          high.increment(); // high = n + 1
40
41          // Initialize the result
42          NaturalNumber result = new NaturalNumber2(0);
43
44          // Interval halving algorithm
45          while (low.compareTo(high) < 0) {
46              NaturalNumber mid = new NaturalNumber2(low);
47              // Compute (high - low)
48              NaturalNumber temp = new NaturalNumber2(high);
49              temp.subtract(low);
50
51              // Compute (high - low) / 2
52              NaturalNumber two = new NaturalNumber2(2);
53              temp.divide(two);
54
55              // Compute mid = low + (high - low) / 2
56              mid.add(temp);
57
58              // Calculate mid^r
59              NaturalNumber midToR = new NaturalNumber2(mid);
60              midToR.power(r);
61
62              // Compare mid^r with n
63              if (midToR.compareTo(n) <= 0) {
64                  result.copyFrom(mid);
65                  low.copyFrom(mid);
66                  low.increment();
67              } else {
68                  high.copyFrom(mid);
69              }
70          }
71
72          // Update n to the result
73          n.copyFrom(result);
74
75      }
76
77      /**
```

```
 78      * Main method.
 79      *
 80      * @param args
 81      *            the command line arguments
 82      */
 83     public static void main(String[] args) {
 84         SimpleWriter out = new SimpleWriter1L();
 85
 86         final String[] numbers = { "0", "1", "13", "1024",
    "189943527", "0", "1", "13",
 87                 "4096", "189943527", "0", "1", "13", "1024",
    "189943527", "82", "82",
 88                 "82", "82", "82", "9", "27", "81", "243",
    "143489073", "2147483647",
 89                 "2147483648", "92233720368547758O7",
    "92233720368547758O8",
 90                 "6189700196426901374495621111",
    "16225927682921336339157801028812O",
 91                 "170141183460469231731687303715884105727" };
 92         final int[] roots = { 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 15, 15,
    15, 15, 15, 2, 3, 4,
 93                 5, 15, 2, 3, 4, 5, 15, 2, 2, 3, 3, 4, 5, 6 };
 94         final String[] results = { "0", "1", "3", "32", "13782",
    "0", "1", "2", "16",
 95                 "574", "0", "1", "1", "1", "3", "9", "4", "3", "2",
    "1", "3", "3", "3",
 96                 "3", "3", "46340", "46340", "2097151", "2097152",
    "4987896", "2767208",
 97                 "2353973" };
 98
 99         for (int i = 0; i < numbers.length; i++) {
100             NaturalNumber n = new NaturalNumber2(numbers[i]);
101             NaturalNumber r = new NaturalNumber2(results[i]);
102             root(n, roots[i]);
103             if (n.equals(r)) {
104                 out.println("Test " + (i + 1) + " passed: root(" +
    numbers[i] + ", "
105                         + roots[i] + ") = " + results[i]);
106             } else {
107                 out.println("*** Test " + (i + 1) + " failed:
```

```
    root(" + numbers[i] + ", "
108                          + roots[i] + ") expected <" + results[i] +
    "> but was <" + n
109                          + ">");
110             }
111         }
112
113         out.close();
114     }
115
116 }
117
```