```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 David Park
 *
 */
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 to 1 and high to n for the binary search
        NaturalNumber low = new NaturalNumber2(0);
        NaturalNumber high = new NaturalNumber2(n);
        high.increment();
        // Temporary variable to hold the mid point from search
        NaturalNumber mid = new NaturalNumber2(0);
        // Constant value of 2 for dividing the search range
        NaturalNumber two = new NaturalNumber2(2);

        NaturalNumber difference = new NaturalNumber2(high);
        difference.subtract(low);

        // Binary search loop to find the r-th root of n
        while (difference.compareTo(new NaturalNumber2(1)) > 0) {
            // Calculate mid as the average of low and high
            mid.copyFrom(low);
            mid.add(high);
            mid.divide(two);

            // Prepare to calculate mid raised to the power of r
            NaturalNumber root = new NaturalNumber2(1); // Start with root = 1
            for (int i = 0; i < r; i++) {
```

```java
                // Multiply root by mid, r times to simulate mid^r
                root.multiply(mid);
            }

            // If mid^r is less than or equal to n, adjust low to search upper half
            if (root.compareTo(n) <= 0) {
                low.copyFrom(mid);
            } else {
                // If mid^r is greater than n, adjust high to search lower half
                high.copyFrom(mid);
            }
            difference.copyFrom(high);
            difference.subtract(low);
        }
        n.copyFrom(low);
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleWriter out = new SimpleWriter1L();

        final String[] numbers = { "0", "1", "13", "1024", "189943527", "0",
                "1", "13", "4096", "189943527", "0", "1", "13", "1024",
                "189943527", "82", "82", "82", "82", "82", "9", "27", "81",
                "243", "143489073", "2147483647", "2147483648",
                "9223372036854775807", "9223372036854775808",
                "618970019642690137449562111",
                "162259276829213363391578010288127",
                "170141183460469231731687303715884105727" };
        final int[] roots = { 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 15, 15, 15, 15, 15,
                2, 3, 4, 5, 15, 2, 3, 4, 5, 15, 2, 2, 3, 3, 4, 5, 6 };
        final String[] results = { "0", "1", "3", "32", "13782", "0", "1", "2",
                "16", "574", "0", "1", "1", "1", "3", "9", "4", "3", "2", "1",
                "3", "3", "3", "3", "3", "46340", "46340", "2097151", "2097152",
                "4987896", "2767208", "2353973" };

        for (int i = 0; i < numbers.length; i++) {
            NaturalNumber n = new NaturalNumber2(numbers[i]);
            NaturalNumber r = new NaturalNumber2(results[i]);
            root(n, roots[i]);
            if (n.equals(r)) {
                out.println("Test " + (i + 1) + " passed: root(" + numbers[i]
                        + ", " + roots[i] + ") = " + results[i]);
            } else {
                out.println("*** Test " + (i + 1) + " failed: root("
                        + numbers[i] + ", " + roots[i] + ") expected <"
                        + results[i] + "> but was <" + n + ">");
            }
        }
        out.close();
    }
}
```