**Performance**

*The purpose of this assignment is to give you practice writing efficient programs and analyzing their running time.*

1. **Inversions.** Suppose that a music site wants to compare your song preferences to those of a friend. One approach is to have you and your friend each rank a set of *n* songs and *count* the number of pairs of songs (*i*, *j*) for which you prefer *i* to *j* but your friend prefers *j* to *i*. When the count is low, the preferences are similar.

More generally, given an array of integers, a pair of elements a[i] and a[j] are *inverted* if i < j and a[i] > a[j]. For example, the array a[] has 1 inversion and the array b[] has 4 inversions.



Write a program Inversions.java that implements the following API:

**public class Inversions {**

// Return the number of inversions in the permutation a[].

**public static long count(int[] a)**

// Return a permutation of length n with exactly k inversions.

**public static int[] generate(int n, long k)**

// Takes an integer n and a long k as command-line arguments,

// and prints a permutation of length n with exactly k inversions.

**public static void main(String[] args)**

**}**

Here is some more information about the required behavior:

* + *Permutations.* A *permutation* of length *n* is an integer array of length *n* that contains each of the *n* integers 0, 1, ..., *n*–1 exactly once.
  + *Output format.* The main() method should print the permutation of length *n* to standard output as a sequence of *n* integers, separated by whitespace, all on one line.
  + *Performance.* The count() method should take time proportional to n2 in the worst case. The generate() method should take time proportional to n in the worst case.
  + *Corner cases.* You may assume that the arguments to generate() satsify n≥0 and 0≤k≤1/2n(n−1); this guarantees the existence of a permutation of length *n* with exactly *k* inversions.

Here are a few sample executions:

~/Desktop/performance> java-introcs Inversions 10 0

0 1 2 3 4 5 6 7 8 9

~/Desktop/performance> java-introcs Inversions 10 1

0 1 2 3 4 5 6 7 9 8

~/Desktop/performance> java-introcs Inversions 10 45

9 8 7 6 5 4 3 2 1 0

~/Desktop/performance> java-introcs Inversions 10 20

9 8 0 1 2 3 7 4 5 6

*Counting inversions arise in a number of applications, including sorting, voting theory, collaborative filtering, rank aggregation, and non-parametric statistics.*

1. **Ramanujan numbers.** When the English mathematician G. H. Hardy came to visit the Indian mathematician Srinivasa Ramanujan in the hospital one day, Hardy remarked that the number of his taxi was 1729, a rather dull number. To which Ramanujan replied, *No, Hardy! No, Hardy! It is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways.*

An integer *n* is a *Ramanujan number* if can be expressed as the sum of two positive cubes in two different ways. That is, there are four distinct positive integers *a*, *b*, *c*, and *d* such that n=a3+b3=c3+d3. For example 1729=13+123 =93+103.

Write a program Ramanujan.java that takes a long integer command-line argument *n* and prints true if it is a Ramanujan number, and false otherwise. To do so, organize your program according to the following public API:

**public class Ramanujan {**

// Is n a Ramanujan number?

**public static boolean isRamanujan(long n)**

// Takes a long integer command-line arguments n and prints true if

// n is a Ramanujan number, and false otherwise.

**public static void main(String[] args)**

**}**

Here are a few sample executions:

~/Desktop/performance> java-introcs Ramanujan 1729

true

~/Desktop/performance> java-introcs Ramanujan 3458

false

~/Desktop/performance> java-introcs Ramanujan 4104

true

~/Desktop/performance> java-introcs Ramanujan 216125

true

~/Desktop/performance> java-introcs Ramanujan 9223278330318728221

true

*Your program should take time proportional to*n1/3*in the worst case. It should be fast enough to process any 64-bit long integer in a fraction of a second.*

1. **Maximum square submatrix.** Given an *n*-by-*n* matrix of 0s and 1s, find a contiguous square submatrix of maximum size that contains only 1s. To do so, organize your program according to the following public API:

**public class MaximumSquareSubmatrix {**

// Returns the size of the largest contiguous square submatrix

// of a[][] containing only 1s.

**public static int size(int[][] a)**

// Reads an n-by-n matrix of 0s and 1s from standard input

// and prints the size of the largest contiguous square submatrix

// containing only 1s.

**public static void main(String[] args)**

**}**

Here is some more information about the required behavior:

* + *Size.* The *size* of a square submatrix is its number of rows (or columns). You may assume that argument to the size() method is a square matrix containing only 0s and 1s.
  + *Contiguous.* The square submatrix must be *contiguous*—the row indicies must be consecutive and the column indices must be consecutive.
  + *Performance.* The size() method should take time proportional to n2 in the worst case. Significant partial credit for solutions that take time proportional to n3 or n4.
  + *Input format.* Standard input will contain a positive integer *n*, followed by *n* lines, with each line containing *n* 0s and 1s, separated by whitespace.

Here are a few sample executions:

~/Desktop/performance> cat square6.txt

6

0 1 1 0 1 1

1 1 0 1 0 1

0 1 1 1 0 1

1 1 1 1 1 0

1 1 1 1 1 1

0 0 0 0 1 1

~/Desktop/performance> java-introcs MaximumSquareSubmatrix < square6.txt

3

~/Desktop/performance> cat square7.txt

7

0 1 1 0 1 1 1

1 1 0 1 1 1 1

0 1 1 1 1 1 1

1 1 1 1 1 0 1

1 1 1 1 1 1 0

1 1 1 1 0 1 1

1 1 1 1 0 1 1

~/Desktop/performance> java-introcs MaximumSquareSubmatrix < square7.txt

4

*The maximum square submatrix problem is related to problems that arise in databases, image processing, and maximum likelihood estimation. It is also a popular technical job interview question.*

**Submission.** Submit a .zip file containing Inversions.java, Ramanujan.java, and MaximumSquareSubmatrix.java. You may not call library functions except those in the java.lang (such as Long.parseLong() and Math.cbrt()) and stdlib.jar (such as StdIn.readInt()). Use only Java features that have already been introduced in the course.