Number Theory

Number theory is the study of the properties of numbers, especially the positive integers. In this exercise, you will write a program that asks the user for a positive integer. Then, the program will investigate some properties of this number. As a guide for your implementation, a skeleton Java version of this program is available online as NumberTheory.java.

Your program should determine the following about the input number n:

1. See if it is prime.

2. Determine its prime factorization.

3. Print out all of the factors of n. Also count them. Find the sum of the proper factors of n. Note that a "proper" factor is one that is less than n itself. Once you know the sum of the proper factors, you can classify n as being abundant, deficient or perfect.

4. Determine the smallest positive integer factor that would be necessary to multiply n by in order to create a perfect square.

5. Verify the Goldbach conjecture: Any even positive integer can be written as the sum of two primes. It is sufficient to find just one such pair.

Here is some sample output if the input value is 360.

```
Please enter a positive integer n:  360
360 is NOT prime.

Here is the prime factorization of 360:
2 ^ 3
3 ^ 2
5 ^ 1

Here is a list of all the factors:
1 2 3 4 5 6 8 9 10 12 15 18 20 24 30 36 40 45 60 72 90 120 180 360
360 has 24 factors.

The sum of the proper factors of 360 is 810.
360 is abundant.

The missing factor to make 360 a perfect square is 10.

Result of Goldbach's conjecture:
360 = 7 + 353
```

Part 2: Divisor List

Let's suppose we wanted to help someone make a systematic study of positive integers. Instead of writing an interactive program about one integer, let's create a text file that summarizes data on many of them. That way, the properties of various numbers can be examined offline.

Create a new program. It can be called divisor.py or Divisor.java. It should do the following:

```
For each integer n from 1 to 1 million (or some other maximum)
   Find all the factors of n and put them into a list called L.
   Print a line to a text file containing:
      n,
      the size of L,
      the sum of L,
      and all the elements in L.
```

For example, the first 10 lines of the file should look like this:

```
1 1 1 1
2 2 3 1 2
3 2 4 1 3
4 3 7 1 2 4
5 2 6 1 5
6 4 12 1 2 3 6
7 2 8 1 7
8 4 15 1 2 4 8
9 3 13 1 3 9
10 4 18 1 2 5 10
```

With such an output file, we can scan it later to see which numbers are prime, perfect, abundant, etc.

Part 3: Highly Composite

Write a program that prints all of the highly composite numbers between 1 and 1 million. A highly composite number is a positive integer that has more divisors than any number smaller than it. In other words, as n increases, we continually want to set the bar higher on the number of divisors. In your output, print a line for each highly composite number you find. Each line should give the value of n and its number of divisors.

Connected with this – can you think of an efficient way to find the number of divisors n has?

Part 4: Inverse Pascal

Pascal's triangle is a two-dimensional sequence of positive integers defined as follows. In the following formula, n is the "row number" and r is the "column number," and these are both nonnegative integers.

C(n, r) =

- 0, if $r > n$ (This rule is only here to make sure we don't go outside the triangle.)
- 1, if $n = r$ or $r = 0$
- $C(n - 1, r) + C(n - 1, r - 1)$, otherwise

Pascal's triangle has many applications, and the numbers inside the triangle are often called "binomial coefficients." When you implement the Pascal's triangle formula above, it is not computationally efficient to use recursion directly as the definition indicates.

As you probably know, the number 1 appears at the beginning and end of every row of Pascal's triangle. And since n may be arbitrarily large, the number 1 therefore appears an infinite number of times in the triangle. However, each of the other positive integers appears only a finite number of times. We will look for these locations.

Write a program that determines the locations of all instances of the number k in Pascal's triangle, where k takes on the values 2 through 100, inclusive. Based on your output, which numbers appear more than three times?