Nested For-Loops and BRUTE FORCE

Problem: List all numbers between 1 and 10000000 which are both square numbers and cubic numbers.

Brute-force solution:

```
public static void main(String[] args) {

    for (int i = 0; i < 1000000; i++) {
        if ( isASquare(i) && isACube(i) )
            System.out.println(i);
        }
    }
}

public static ___________ isASquare(int n) {
    double a = Math.sqrt(n);
    return ((int)a == a);
}

public static ___________ isACube(int n) {
    double a= Math.cbrt(n);
    return ((int)a == a);
}
```

```
pseudo-code

loop from 1 to 1000000:
    if i is a cube and a square
        then display i.
```

About brute force search

You can think of the problem above as a search problem: We are searching for numbers that have certain properties (or that pass certain tests). In the above problem, it's numbers which are both squares and cubes.

If you knew an equation you could use to generate such numbers directly, that would be best. (In other words, plug 1 in to your equation and it gives you the first number that's both a square and a cube. Plug 2 in and it gives you the next one).

Finding such solutions can be tricky. So the most obvious, naive thing you can do is guess and check. Just test **every possibility** to see if it works. That's what a brute force search does.

When brute force isn't so bad:

(1) If your test is relatively fast to run.
(2) If the number of things you have to check is relatively small.

Nested Loops and Brute Force Search

Often you will need nested for-loops for your brute-force search. In particular, you will need them if there's more than one parameter which can vary in your problem. For example, in the problem above we were testing *single numbers* to see if they were what we wanted. That means we only needed one loop: to loop through all the possible numbers.

What if we wanted to find integer solutions to the equation: $y^2 = 3x^3 - 3$? Here we would need to test *pairs* of numbers. E.g. Does {x = 1, y = 1} work? Does {x = 1, y = 2} work? Does {x = 2, y = 1} work? To do this, you'll need a loop to search all possible x values. Then **for each** x-value, you'll want to check all possible y-values.

Brute-force solution

```
public static void main(String[] args) {
    int counter = 0;

    for (long x = 1; i < 1000000; x++) {

        for (long y = 1; y < 1000000; y++ ) {

            if ( satisfiesEquation(x, y) ) {
                System.out.print(x + ", " + y);
                counter++;
            }

        }

    }

    System.out.println("There were " + counter + "solutions");
}

public static boolean satisfiesEquation(long x, long y) {
    return (y*y == 3*x*x*x - 3);
}
```

General structure

```
for (long x = 1; i < 1000000; x++) {

   for (long y = 1; y < 1000000; y++ ) {

       if ( satisfiesEquation(x, y) ) {
          counter++;
       }

    }

}
```

```
for ( each x-values ) {

    for ( each y-values) {

       do the check;

    }

}
```

Here the inside loop will run through all the y-values with each repeat of the outter loop. So, the first time through when x = 1, the inner loop will check for y is 1, y is 2, y is .... y is 1000000 -- all while x = 1. Then the outter loop will repeat and x = 2. The inner loop will run through all the y-values when x is 2. Then the outter loop will repeat and the inner loop will run through all y-values when x = 3, and so on.

Problem Set

1. Find all integers satisfying the equation: $x^2 + y^2 = 1000$

```
Pseudo-code:
loop i from 1 to 1000
    loop j from 1 to 1000
        if i^2 + j^2 is 1000
            display i and j.
```

2. How many integer lattice points are inside (or on the perimeter) of a circle radius r, centered on the origin?

(a) If I draw a circle with radius r centered at the origin, there will be a bunch of integer-coordinate points that lie inside it. I want you to create a method which will use a brute-force search to count how many. [Hint: draw a picture and think about what the smallest and largest x and y values you want to check are. Think about what would be a mathematical test to check whether or not a point was inside the circle or not.]

(b) Now use your method to create a table of values and display it:

```
Radius          Number of lattice points
0               0
1               5
2               ..
...             ...
100             ...
```

3. S. Ramanujan was an Indian mathematician who became famous for his intuition about numbers. When the English mathematician G. H. Hardy came to visit him 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! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways."

Write a program that prompts the user to enter a positive integer, and prints out all integers less than or equal to that number which can be expressed as the sum of two cubes in two different ways .

In other words, a number x is a solution to this problem if $x = a^3 + b^3 = c^3 + d^3$ for distinct positive integers a, b, c, and d. Use four nested for loops.

4. Euler's sum of powers conjecture. In 1769 Euler generalized Fermat's Last Theorem and conjectured that it is impossible to find three 4th powers whose sum is a 4th power, or four 5th powers whose sum is a 5th power, etc. The conjecture was disproved in 1966 by exhaustive computer search.

Disprove the conjecture by finding positive integers a, b, c, d, and e such that $a^5 + b^5 + c^5 + d^5 = e^5$. Write a program that prompts the user for an integer N and exhaustively searches for all such solutions with a, b, c, d, and e less than or equal to N.

(Note: No counterexamples are known for powers greater than 5, but you can join EulerNet, a distributed computing effort to find a counterexample for sixth powers.)

***5. Prime Generating Functions**

Ok experts. Prime numbers are a huge deal. No one knows a way to predict exactly where the primes will occur (though we can say some probabilistic things about their density). Some of the most important unsolved problems in mathematics are related to this sort of question.

The quadratic equation $x^2 + x + 17$ is called a *prime generating function* because it generates 17 prime numbers for values of x from 0 to 16! In other words, if you plug in any integer from 0 to 16, this function will evaluate to a different prime number.

This is amazing and awesome!

Without the use of a computer, mathematician Leonard Euler discovered a quadratic equation which generates 40 primes. Let's use the computer to find other prime generating functions.

Your Task: Do a brute force search of all quadratic equations of the form $x^2 + ax + b$ with $a, b < 5{,}000$. For each quadratic equation, evaluate it for values of x from x = 0 to whenever the first non-prime is generated.

Display any equation which generates 10 or more primes.

Recommendations. Use some methods here. You'll definitely want an is_prime(num) method. You probably want a more general method like, result(a, b, x) which returns $x^2 + ax + b$, which you can then test for primality. Put it all in some loops and you're good!

* * *

Homework: Abundant Numbers

Definition: An abundant number is a number whose proper divisors sum to a number greater than it. For example, the proper divisors of the number 24 are 1, 2, 3, 4, 6, 8, and 12, which sum to 36. Since 36 (the sum of divisors) is larger than 24 (the original number), we say that 24 is an abundant number.

(1) Complete this method to test if a number is abundant.

```
// returns true if n is abundant.

public static ____________________ isAbundant(int n) {
  long sum = 0;
    for( _____________________) {        // Loop through all possible divisors of n

      if (_________________________________) {        // is it a divisor?

                                                      // if so, accumulate it into
                                                      //   the ongoing sum

      }
   }

   return (sum > n);                      // if the sum > n, n is abundant.
}
```

(2) Write a loop which uses this method to find the smallest abundant number between 1 and 100000.

(3) Write a loop which uses this method to find the smallest ODD abundant number between 1 and 100000.

(4) Time for some more brute force. Write a double for-loop which will find two abundant numbers that sum to 20162. Here's some inefficient psuedo-code. (If you can make a more efficient version of this, do it!)

```
loop through all #'s from 1 to whatever
        loop through all #'s from 1 to whatever
                test if the two numbers are both abundant
                        test if they sum to the target
                                if so, display them.
```