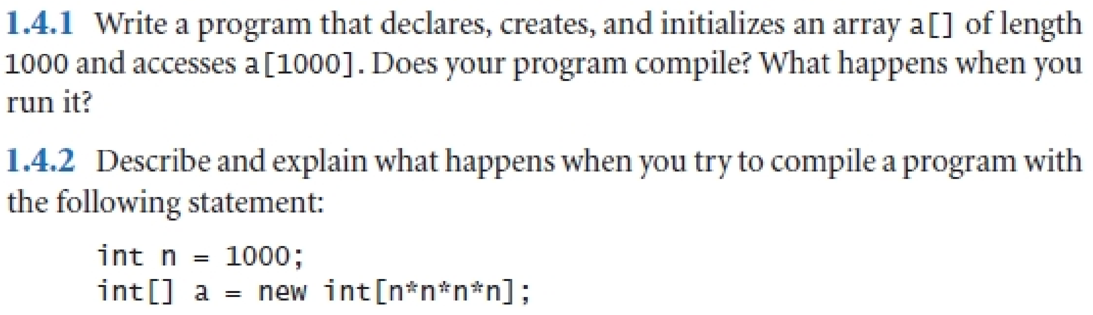
Exercises Three *(Sept. 25)*: *Selected* Exercises.

Try to solve the Following Problems as many as you can & submit your programs only to email program06@yeah.net before (including) Oct. 7.

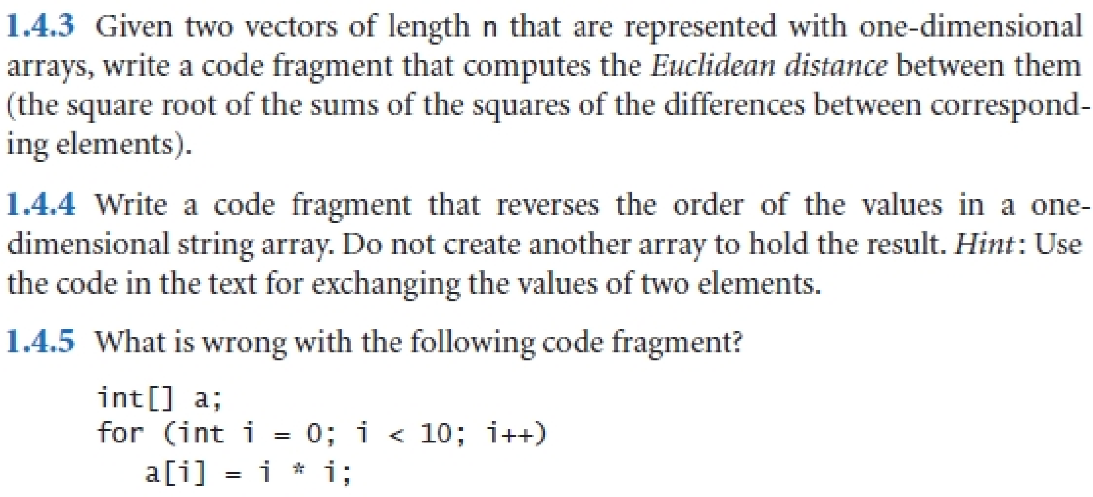
(Notice: The last submitting date for Exercises Two is delayed to Oct. 2.)



1.4.1 Write a program that declares, creates, and initializes an array a[] of length 1000 and accesses a[1000]. Does your program compile? What happens when you run it?

1.4.2 Describe and explain what happens when you try to compile a program with the following statement:

```
int n = 1000;
int[] a = new int[n*n*n*n];
```

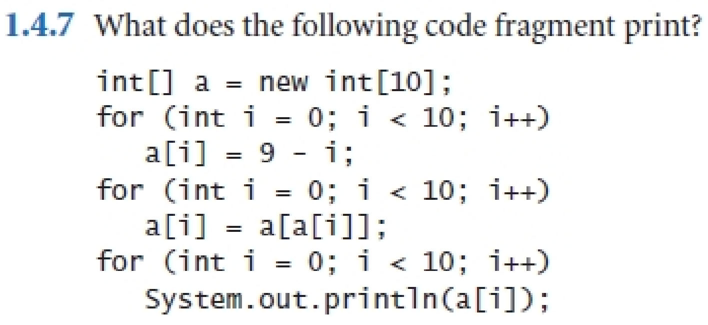


1.4.3 Given two vectors of length n that are represented with one-dimensional arrays, write a code fragment that computes the *Euclidean distance* between them (the square root of the sums of the squares of the differences between corresponding elements).

1.4.4 Write a code fragment that reverses the order of the values in a one-dimensional string array. Do not create another array to hold the result. *Hint*: Use the code in the text for exchanging the values of two elements.

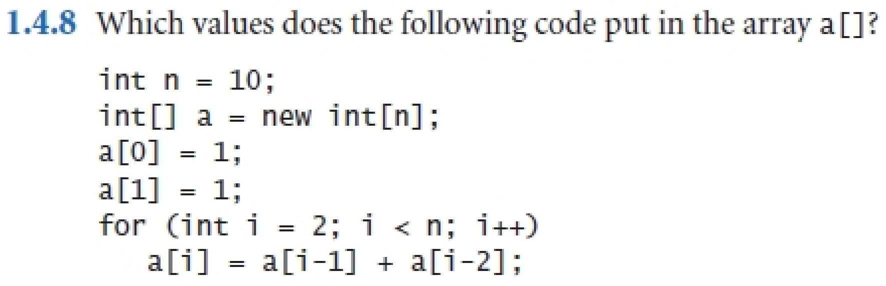
1.4.5 What is wrong with the following code fragment?

```
int[] a;
for (int i = 0; i < 10; i++)
   a[i] = i * i;
```



1.4.7 What does the following code fragment print?

```
int[] a = new int[10];
for (int i = 0; i < 10; i++)
   a[i] = 9 - i;
for (int i = 0; i < 10; i++)
   a[i] = a[a[i]];
for (int i = 0; i < 10; i++)
   System.out.println(a[i]);
```



1.4.8 Which values does the following code put in the array a[]?

```
int n = 10;
int[] a = new int[n];
a[0] = 1;
a[1] = 1;
for (int i = 2; i < n; i++)
   a[i] = a[i-1] + a[i-2];
```

1.4.9 What does the following code fragment print?

```
int[] a = { 1, 2, 3 };
int[] b = { 1, 2, 3 };
System.out.println(a == b);
```

1.4.10 Write a program `Deal` that takes an integer command-line argument `n` and prints `n` poker hands (five cards each) from a shuffled deck, separated by blank lines.

1.4.22 *Dice simulation.* The following code computes the exact probability distribution for the sum of two dice:

```
int[] frequencies = new int[13];
for (int i = 1; i <= 6; i++)
   for (int j = 1; j <= 6; j++)
      frequencies[i+j]++;

double[] probabilities = new double[13];
for (int k = 1; k <= 12; k++)
   probabilities[k] = frequencies[k] / 36.0;
```

The value `probabilities[k]` is the probability that the dice sum to `k`. Run experiments that validate this calculation by simulating `n` dice throws, keeping track of the frequencies of occurrence of each value when you compute the sum of two uniformly random integers between 1 and 6. How large does `n` have to be before your empirical results match the exact results to three decimal places?

2.1.1 Write a static method `max3()` that takes three `int` arguments and returns the value of the largest one. Add an overloaded function that does the same thing with three `double` values.

2.1.2 Write a static method `odd()` that takes three `boolean` arguments and returns `true` if an odd number of the argument values are `true`, and `false` otherwise.

2.1.3 Write a static method `majority()` that takes three `boolean` arguments and returns `true` if at least two of the argument values are `true`, and `false` otherwise. Do not use an `if` statement.

2.1.4 Write a static method `eq()` that takes two `int` arrays as arguments and returns `true` if the arrays have the same length and all corresponding pairs of of elements are equal, and `false` otherwise.

2.1.5 Write a static method `areTriangular()` that takes three `double` arguments and returns `true` if they could be the sides of a triangle (none of them is greater than or equal to the sum of the other two). See Exercise 1.2.15.

2.1.7 Write a static method `sqrt()` that takes a `double` argument and returns the square root of that number. Use Newton's method (see Program 1.3.6) to compute the result.

2.1.9 Write a static method `lg()` that takes a `double` argument `n` and returns the base-2 logarithm of `n`. You may use Java's `Math` library.

2.1.10 Write a static method `lg()` that takes an `int` argument `n` and returns the largest integer not larger than the base-2 logarithm of `n`. Do *not* use the `Math` library.

2.1.11 Write a static method `signum()` that takes an `int` argument `n` and returns `-1` if `n` is less than `0`, `0` if `n` is equal to `0`, and `+1` if `n` is greater than `0`.

2.1.12 Consider the static method `duplicate()` below.

```
public static String duplicate(String s)
{
   String t = s + s;
   return t;
}
```

What does the following code fragment do?

```
String s = "Hello";
s = duplicate(s);
String t = "Bye";
t = duplicate(duplicate(duplicate(t)));
StdOut.println(s + t);
```

2.1.13 Consider the static method `cube()` below.

```
public static void cube(int i)
{
   i = i * i * i;
}
```

How many times is the following `for` loop iterated?

```
for (int i = 0; i < 1000; i++)
   cube(i);
```

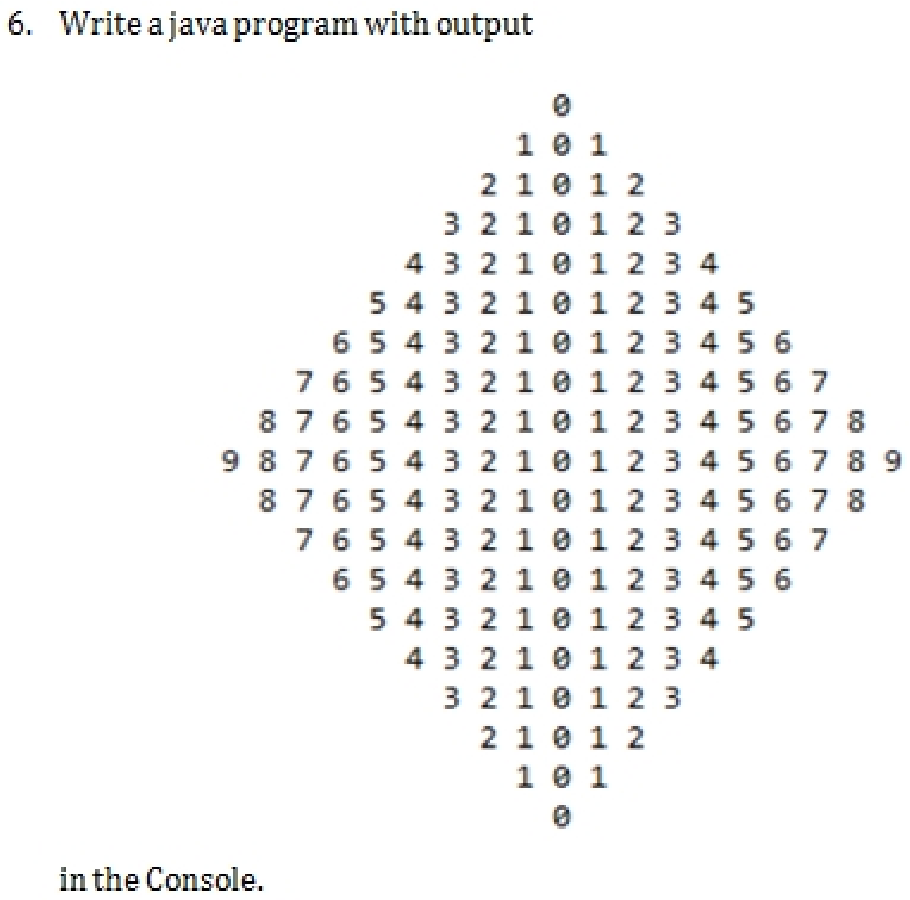
Answer: Just 1,000 times. A call to `cube()` has no effect on the client code. It changes the value of its local parameter variable `i`, but that change has no effect on the `i` in the `for` loop, which is a different variable. If you replace the call to `cube(i)` with the statement `i = i * i * i;` (maybe that was what you were thinking), then the loop is iterated five times, with `i` taking on the values `0`, `1`, `2`, `9`, and `730` at the beginning of the five iterations.

2.1.14 The following *checksum* formula is widely used by banks and credit card companies to validate legal account numbers:

$$d_0 + f(d_1) + d_2 + f(d_3) + d_4 + f(d_5) + \dots \; = 0 \;(\text{mod } 10)$$

The d_i are the decimal digits of the account number and $f(d)$ is the sum of the decimal digits of $2d$ (for example, $f(7) = 5$ because $2 \times 7 = 14$ and $1 + 4 = 5$). For example, 17,327 is valid because $1 + 5 + 3 + 4 + 7 = 20$, which is a multiple of 10. Implement the function f and write a program to take a 10-digit integer as a command-line argument and print a valid 11-digit number with the given integer as its first 10 digits and the checksum as the last digit.

A Problem from the Spring Semester's Lab Assignment:



6. Write a java program with output

```
                  0
                1 0 1
              2 1 0 1 2
            3 2 1 0 1 2 3
          4 3 2 1 0 1 2 3 4
        5 4 3 2 1 0 1 2 3 4 5
      6 5 4 3 2 1 0 1 2 3 4 5 6
    7 6 5 4 3 2 1 0 1 2 3 4 5 6 7
  8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8
9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9
  8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8
    7 6 5 4 3 2 1 0 1 2 3 4 5 6 7
      6 5 4 3 2 1 0 1 2 3 4 5 6
        5 4 3 2 1 0 1 2 3 4 5
          4 3 2 1 0 1 2 3 4
            3 2 1 0 1 2 3
              2 1 0 1 2
                1 0 1
                  0
```

in the Console.

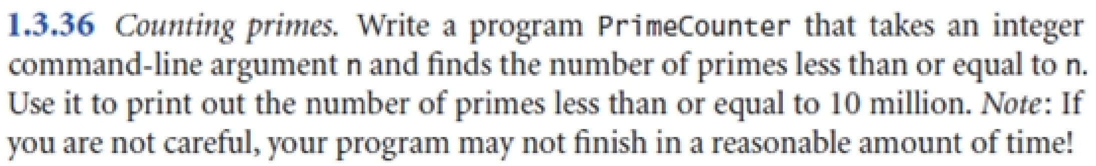
(Optional) Explore the time consumption of PrimeCounter (in Exercises 2, without array) and PrimeSeive given below with the following test frameworks:

```
import java.util.Date;
…
Date start = new Date();   // JDK 1.0+
…
Date end = new Date();
long timeInMS = end.getTime() – start.getTime();
```

Or

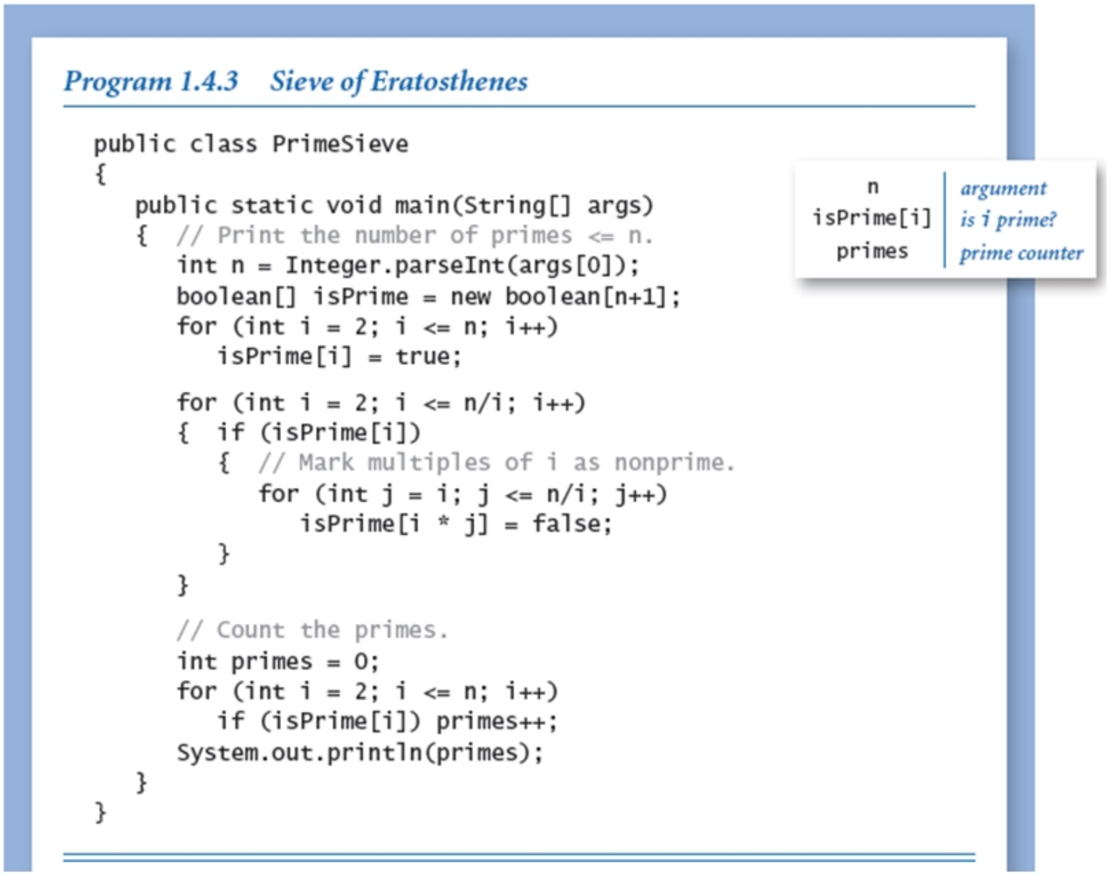
```
import java.time.Instant;
import java.time.Duration;
Instant start = Instant.now();   // JDK 8.0+
…
Instant end = Instant.now();
long timeInMS = Duration.between(start, end).toMillis();
```

Problem & Program in Exercises 2 :



1.3.36 *Counting primes.* Write a program `PrimeCounter` that takes an integer command-line argument `n` and finds the number of primes less than or equal to `n`. Use it to print out the number of primes less than or equal to 10 million. *Note*: If you are not careful, your program may not finish in a reasonable amount of time!

Primes Counting through the Seive of Eratoshenes:



Program 1.4.3 Sieve of Eratosthenes

```
public class PrimeSieve
{
   public static void main(String[] args)
   {  // Print the number of primes <= n.
      int n = Integer.parseInt(args[0]);
      boolean[] isPrime = new boolean[n+1];
      for (int i = 2; i <= n; i++)
         isPrime[i] = true;

      for (int i = 2; i <= n/i; i++)
      {  if (isPrime[i])
         {  // Mark multiples of i as nonprime.
            for (int j = i; j <= n/i; j++)
               isPrime[i * j] = false;
         }
      }

      // Count the primes.
      int primes = 0;
      for (int i = 2; i <= n; i++)
         if (isPrime[i]) primes++;
      System.out.println(primes);
   }
}
```

`n`	*argument*
`isPrime[i]`	*is i prime?*
`primes`	*prime counter*