COS 126 | Programming Exam 1 (Fall 2015)

Instructions. This exam is like a mini programming assignment. You will have 50 minutes to create several programs. Debug your code as needed. You may use your book, your notes, your code from programming assignments, the code on the COS 126 course website and the booksite, and you may read Piazza. No form of communication is permitted (e.g., talking, email, IM, texting, Facebook, etc.) during the exam.

Downloads. Before you begin, download the code templates and sample input files as instructed on the COS 126 Exams page. (You may need to refresh the page to see them.)

Grading. Your program will be graded on correctness, clarity (including comments), design, and efficiency. You will lose a substantial number of points if your program does not compile, does not follow the API we are expecting, or if it crashes on typical inputs.

Submitting code. Submit your programs via the "Submit" link on the Exams page.

This paper. In addition to submitting your code electronically, you must turn in this paper. Print your name, login, and precept below. Copy and sign the Honor Code pledge.

Discussing this exam. As you know, discussing the contents of this exam before solutions have been posted is a serious violation of the Honor Code.

NAME: _____

LOGIN: _____

PRECEPT: _____

"I pledge my honor that I have not violated the Honor Code during this examination."

_____

_____

SIGNATURE: _____

Programming Exam: Count Distinct Values

Part 1. Your task is to write programs that find the number of distinct values among the integers on standard input, assuming that the input is nonempty *and in sorted order*.

Your task. Add code to this template (the file `Count1.java` that you have downloaded):

```
public class Count1
{
    public static void main(String[] args)
    {
        int count = 1;
        int distinct = 1;

       // YOUR CODE HERE

    }
}
```

Your code must print the number of integers on standard input *and* the number of distinct values among those integers.

Example. Test your program with the file `testCount1tiny.txt`, which has 18 integers having six distinct values (`1`, `2`, `4`, `5`, `6`, and `9`). Your program must behave as follows:

```
% more testCount1tiny.txt
1 1 1 1 2 2 2 2 4 4 4 4 5 5 6 6 9 9
% java Count1 < testCount1tiny.txt
6 distinct values among 18 integers
```

Write a single loop that uses `StdIn.readInt()` to read each integer one at a time, but *do not save them in an array*. To compute the number of distinct values, add code to the loop to update `distinct` if the new value just read differs from the value read just before it. For this example, your code should start with `distinct` at `1`, increment it to `2` after reading the first `2`, increment it to `3` after reading the first `4`, and so forth.

Note. The assumption that the input is in sorted order is a very strong one, since input does not come that way in applications. We will be examining efficient methods for making this happen later in this course.

SUBMIT `Count1.java` AS INSTRUCTED ON THE COVER PAGE.

DO NOT PROCEED TO PART 2 UNTIL YOU HAVE DONE SO.

Part 2. Assume that all of the integers on standard input are nonnegative, *not necessarily in order*, and less than a value `M` given as the first command-line argument, Also assume that you have enough memory to store an array of `M` booleans.

Your task. Add code to this template (the file `Count2.java` that you have downloaded):

```
public class Count2
{
    public static void main(String[] args)
    {
      int M = Integer.parseInt(args[0]);
      boolean[] b = new boolean[M];

      // YOUR CODE HERE

    }
}
```

Again, your code must print the number of integers on standard input *and* the number of distinct values among those integers. Again, use `StdIn.readInt()` to read each integer one at a time, and do not save them in an array.

To compute the needed values, use two loops. First, write a loop that reads the integers one at a time from standard input, using the boolean array `b[]` to record which values have been seen: when you read a value `val`, set `b[val]` to `true`. *The first loop must also count the number of integers on standard input.* Second, write a loop that counts the number of `true` values in `b[]` (the number of distinct values in the input).

Example. Suppose that `M` is `10` and the input is the file `testDistinctA.txt`:

```
4 1 4 1 5 9 2 6 2 4 1 4 1 5 9 2 6 2
```

Since there are no 0s, 3s, 7s, or 8s in the input, the contents of `b[]` after the first loop should be

```
false true true false true true true false false true
```

and the second loop should count the 6 `true` values in `b[]`.

Test your program on the file `testCount2tiny.txt` (using `M` = 10) and `testCount2.txt` (using `M` = 1000). These files have the same values as the corresponding files in Part1, but they are not in sorted order.

SUBMIT `Count2.java` AS INSTRUCTED ON THE COVER PAGE.

Food for thought 1: You can use your programs to study various interesting properties of sequences of random integers. For example, how many different values are likely in a random sequence of *N* nonnegative integers all less than *N*?

Food for thought 2: Both of these solutions break down in real applications with huge sequences of unordered integers having a huge number of different values. Development of *cardinality estimation* algorithms that can provide good estimates under various assumptions without using much memory is an active research topic in computer science.

FOR REFERENCE ONLY. DO NOT TYPE THESE FILES—DOWNLOAD THEM!

Count1.java:

```
public class Count1
{
    public static void main(String[] args)
    {
        int count = 1;
        int distinct = 1;

        // YOUR CODE HERE

    }
}
```

Count2.java:

```
public class Count2
{
    public static void main(String[] args)
    {
       int M = Integer.parseInt(args[0]);
       boolean[] b = new boolean[M];

       // YOUR CODE HERE

    }
}
```

testCount1tiny.txt:

```
1 1 1 1 2 2 2 2 4 4 4 4 5 5 6 6 9 9
```

testCount2tiny.txt:

```
4 1 4 1 5 9 2 6 2 4 1 4 1 5 9 2 6 2
```

testCount1.txt:

```
0
1
3
3
4
4
5
.
.
.
```

testCount2.txt:

```
473
385
447
413
781
564
526
 .
 .
 .
```