PA 1

Due Date: Friday 2 February 2024 by 11:59 PM

General Guidelines.

The instructions given below describe the required methods in each class. You may need to write additional methods or classes that will not be directly tested but may be necessary to complete the assignment.

In general, you are not allowed to import or use any additional classes in your code without explicit permission from your instructor!

Note: It is okay to use the Math class if you want.

Project Overview

In this assignment, you will implement solutions to four problems. The main goals are

- to make sure you know how to use lectura and turnin
- to refresh your memory of Java
- to get you thinking about how to solve problems both correctly and efficiently
- to do some mathematical investigation regarding moduli and prime numbers

Part 0. Before you start coding.

Before you start coding, it is a good idea to make sure you have a current account on lectura and you know your password. You should also go through the process of transferring files to lectura to make sure you know how to do it. The following are resources that may help with this process.

- mac to lectura tutorial
- lectura instructions for pc
- CS Help Desk (in case you need help with account information)
- TAs & instructors (We can walk you through the process in office hours or SI.)

Part 1. (9 points)

Later in this course, we will discuss hashtables, and one thing we will discuss is how to manage the table to avoid collisions. One of the parts of that is understanding the relationship between prime numbers, composite numbers, and modular arithmetic. In this part, you will investigate this relationship experimentally. Please note that there will likely be a homework question later that will ask you to look at the results of this experiment.

The modulus function (denoted by `%` in Java) is essentially a remainder function. For example, if you divide 10 by 3, you get a quotient of 3 with a remainder of 1. That means that $10 \% 3 = 1$.
You may recall from previous courses that any pair of integers *a* and *b* can be related such that there exist *q* and *r* so that: $a = qb + r$. In the example above, we would write $10 = 3 * 3 + 1$. The modulus function gives us the value of *r*. That is, given that $a = qb + r$, we can say that $a \% b = r$.

Modular arithmetic will be used in hashing in Unit 4, and there will be some discussion of the value of using a prime number (a number whose only factors are 1 and itself) as a modulus rather than a composite number (a number that has at least one factor other than 1 and itself). In this part of the project, you will write a program to help illustrate what we will discuss later.

The following are things you will need to do:

- Read and parse integers from a file.
- Maintain a count of the results of the modular arithmetic.

You are provided with *Part1.java*, and several text files used for testing. In *Part1.java,* you are provided with a main method that helps test your code, as well as some helping functions that are used in the main method. You may change this code if needed, as none of the provided methods are used in grading (though the grading code is very similar to what you see in *Part1.java*). The one method that you must implement in *Part1.java* is described below. Please be sure not to change the method signature as that may affect testing and grading. In general, if your method works with the test code that is provided, you are likely to do fine with the grading.

The required method.
`public static int[] getCounts(String fn, int m)`

- *fn* is the name of the input file
- *m* is the modulus

In this method, you will need to open the file and read the contents, which contain integers, each on a separate line. You may read the file however you choose (and import classes for that as necessary), but I suggest you use *checkCounts* as a model for how to read and process the contents. (The required classes for that are already imported.) As you read the integers from the input file, you should calculate the remainder $r = x \% m$ (where *x* is the integer from the file and *m* is the modulus parameter). Note that the possible values of *r* will be 0 to *m-1.* You should maintain an array of size *m* that keeps track of the number of times each possible remainder occurs. The following shows an example of how the method should work.

input.txt

```
78
23
10
5
20
9
```

```
Part1.getCounts("input.txt", 5):
This call should read each number x in the file and calculate x % 5,
counting the number of times each one occurs. The result is an array
with the counts:

output: [3|0|0|2|1]

Explanation: 10, 5, and 20 all produce the result 0, so the count at
at that index is 3; 78 and 23 produce 3, so the count there is 2; 9
produces 4, so that count is 1; the other counts are 0
```

Part 2. (13.5 points)

In Part 2, you must implement a method that reads in a file of integers (each integer is on a separate line) and maintains and returns an Array of the current highest *M* numbers.

Two major requirements of this is that **you may not use more than O(M) space** and **you may not use any sorting method.*** This means that you must maintain the array of integers as an Array of size *M* and use no additional data structures. You should think through the following questions before you start implementing.

- What do you do with a new integer if the *M*-sized array is not yet full?
- What do you do with a new integer if the *M*-sized array is full?

- **You should not use any sorting method in your solution.**
- **You are not allowed to import any classes other than those used for reading the file.**
- **You are not allowed to use any data structures except the M-sized array (and variables for individual values).**

- **You must use the Array.java class that is provided, and do not change it because it will not be submitted with your code. It is required because it is a wrapper class for an array that counts the number of times you access the array. This is how we test efficiency.**

You are provided with *Part2.java* and several text files used for testing. *Part2.java* contains a main method and several helper methods used for testing. Like *Part1.java*, you may change this code if you want, but make sure it still compiles. The grading code is similar but is separate and only calls the required method that you are to implement.

The required method.

`public static Array getTop(String fn, int m)`

- *fn* is the name of the input file
- *m* is the number of items that should be returned in the Array

In this method, you will need to open the file and read the contents, which contain integers, each on a separate line. Although you *may* read the file and process the contents in another way, I suggest you use *checkVals* as a model for how to read and process the contents. (The required classes are already imported.) You should have an Array of size *M* (**you are not allowed to use an array any bigger than this!**) that *always contains the highest M values that have been read up to that point.* Note that this array should be maintained *as you read in the numbers* and that you will not know how many numbers are coming.

Example.

input.txt

```
78
23
10
5
20
9
```

`Part2.getTop("input.txt", 3):`

This call should read each number *x* in the file and maintain a list of the highest 3 values that have been read up to that point. **The order of the result does not matter as it will be sorted during testing.**

```
output: [78|23|20]
```

Explanation: The first three values (78, 23, and 10) would be put into the array, but after that, the new values must be considered differently. 5, for example, would not be put into the array because it is smaller than 78, 23, and, most importantly, 10. 20, on the other hand, would be added to the array and, specifically, would replace 10.

Part 3. (15 points)

Problem Statement: **Given an array of integers *A* and an integer *m*, determine the maximum product of *m* consecutive integers in the array.**

Example:
A = [3, 1, 6, 2, 8, 9, 3, 4], m = 3
Answer: 216 (which is the product of 8, 9, and 3)

Guidelines:

- Implement your solution in a class called *Part3.java* with the following method signature:
    - `public static int maxProduct(Array a, int m)`
    - The method takes the Array and the integer and returns the maximum product.
- You are not allowed to use any additional data structures.
- For full credit, your solution should be no worse than *O(N+M)* where *N* is the number of elements in *A* and *M* is *m*. You might also want to reduce the constants in front of those as much as possible because a solution that is *O(N+M)* could still include some inefficient things.
- Note that the Array class is a wrapper class for an array. You must use this class as is. Do not change it because you will not be submitting it. We use this to count the accesses in order to measure the efficiency of your solution.

Part 4. (15 points)

Problem Statement: **Given an array of integers *A*, rearrange *A* so that all the 0's are pushed to the end of the array. All other values should stay in the same relative order.**
Examples:
A = [2, 6, 1, 9, 0, 2]
Result: [2, 6, 1, 9, 2, 0]
A = [0, 0, 6, 1, 0, 9, 8]

Result: [6, 1, 9, 8, 0, 0, 0]
A = [3, 1, 4, 1, 5, 9, 2]
Result: [3, 1, 4, 1, 5, 9, 2]

Guidelines:

- Implement your solution in a class called *Part2.java* with the following method signature:
  - `public static void pushZeroes(Array a)`
  - `The method takes the array and pushes all 0's to the end of the array.`
- You are not allowed to use any additional data structures.
- For full credit, your solution should be no worse than *O(N)* where *N* is the number of elements in *A*. You might also want to reduce the constant in front of the *N* as much as possible because a solution that is *O(N)* could still include some inefficient things.

Testing & Submission Procedure.

This time, test code is provided for you, but keep in mind that the test code may not be exhaustive and catch all possible errors. The test code for Parts 2-4 also compares your access counts to the expected ones, so you should get a good idea of whether or not your efficiency is okay. Again, you should not rely solely on the test cases as your code will also be checked manually. Make sure your code runs with the test code *as it is*. Since you won't be submitting the test code file, your submission cannot depend on any changes you make to that file. The same is true for the *Array.java* file. You can change it for yourself for testing purposes, but we will use the original one when testing, so make sure your code works with that.

Note on Efficiency Testing. Unfortunately, there is no foolproof way to autograde for efficiency. The tests provided are meant to give you a reasonable idea of whether your solutions fits within the expected efficiency level and what is possible. Ultimately, we can always check your code manually if necessary (we do some of this already) to verify that the autograder is returning reasonable results.

Your code must compile and run on lectura, and it is up to you to check this before submitting.
To test your code on lectura:

- Transfer all the files (including test files) to lectura.

- Run *javac *.java* to compile all the java files. (You can also use *javac* with an individual java file to compile just the one file: *javac ArrList.java*)
- Run *java <filename>* to run a specific java file. (Example: *java ArrListTest.java*)
- I recommend running the second command on the file with the .java extension as it catches certain errors that won't be caught if you run it without the .java extension. (e.g. *java ArrListTest.java* vs. *java ArrListTest*).

After you are confident that your code compiles and runs properly on lectura, you can submit the required files using the following **turnin** command. Please do not submit any other files than the ones that are listed.

```
    turnin csc345pa1 Part1.java Part2.java Part3.java Part4.java
```

Upon successful submission, you will see this message:

```
    Turning in:
          Part1.java – ok
          Part2.java – ok
          Part3.java – ok
          Part4.java – ok
    All done.
```

Note: Only properly submitted projects are graded! No projects will be accepted by email or in any other way except as described in this handout. If you are worried about your submission, feel free to ask us to check that it got into the right folder.

Grading.

Please note that the total points for the autograding in this project is 52.5, which means that there are a few extra points. Although these do count as "extra" for the PAs, your PA total will still be capped at 250.

In addition to the auto-grading, the following lists items we could potentially take points off for.

- Code does not compile or does not run on lectura (up to 100% of the points).
- Bad Coding Style (up to 5 points) – this is generally only deducted if your code is hard to read, which could be due to bad documentation, lack of helpful comments, bad indentation, repeating code instead of abstracting out methods, writing long methods instead of smaller, more reusable ones, etc.
- Not following directions (up to 100% of the points depending on the situation)
- Late Submission (5 points per day)--see the policy in the syllabus for specifics
- Inefficient code (up to 50% of the points)