Assignment objectives

The purpose of this assignment is to compare in some detail the performance of two algorithms for the same problem. One algorithm uses a straightforward, brute-force approach, and the other uses a customized algorithm, tailored to the specifics of the problem.

Introduction

The underlying problem is: You are given a list of N positive integers, and guaranteed that there will be *three and only three numbers* somewhere in the list which, when added together, will be an exact multiple of 37. Find these three numbers.

This problem can be solved easily via brute force. The time complexity of this algorithm is $O(N^3)$.

There is also a clever algorithm whose time complexity is O(N)! That's an exclamation mark, not a factorial!

In this assignment you will implement both algorithms and compare their performance both in number of operations performed and in actual running time.

More details

Start with the "starter code", a Java class called TripleFinder, in the Lab1 assignment folder.

Complete the two triple-finder algorithms as well as all the other required methods.

Add some more code to both triple-finder algorithms that tracks how much time (milliseconds) the algorithm takes to run, and how many *operations* the algorithm performs. We are defining "operation" for this assignment to mean the mod(%) operator; *i.e.*, you should count the number of times the algorithm performs a mod(%) operation.

Both of these values should be computed as the function is running, and then stored in private members within your class, so that they can be accessed later via getter methods.

After you have both algorithms working correctly, run your code on *all* the sample data files and gather the following data for each file:

- The number of items in the file (N)
- The number of mod(%) operations performed by the Brute force algorithm
- The time (in milliseconds) that the Brute Force algorithm took to run
- The number of mod(%) operations performed by the Clever algorithm
- The time (in milliseconds) that the Clever algorithm took to run

Finally, make three plots using desmos.com or another graphing tool.

1. A plot of brute force runtime vs. N for all the data files
    - N is the "x-axis" and bfRunTime is the "y-axis"
2. A plot of brute force operations vs. N for all the data files
3. A plot of clever operations vs. N for all the data files

The brute force algorithm

English description: Use three nested for-loops to run indices over every possible combination of three (different) locations in the array. Check each combination to see if the sum of the three referenced elements is a multiple of 37. If it is, you're done.

The clever algorithm

The algorithm relies on some basic principles of modular arithmetic, plus the fact that there is guaranteed to be *only one* valid set of 3 numbers in the input. We can spend some lab time discussing the details. For now, here is a summary of the algorithm:

- Declare an array with exactly 37 elements. Initialize all elements to -1.
- Loop over the input array, finding the congruence class of every number. Congruence class is the remainder of the number when divided by 37, i.e. num%37.
- If the congruence class of a number X is "i", then store X in the second array at position i, overwriting any previous number that was there.
- Do the same as "brute force" algorithm, but on the *second* array.
- The triple that is found is the same one that was in the original input file!

A question to think about: If this uses the same brute force loops, how can it be only O(N)?

Expected results (triples)

Here are the triples found in all the sample data files. This information is to help you while you're testing, to know if your code is working. NOTE: I will be running your code with some other data files besides these ones!

These ones are small enough for you to find the answer by hand:

```
tinydata0.txt, {11, 12, 14}
tinydata1.txt, {95, 60, 178}
tinydata2.txt, {21, 60, 30}
```

These ones take less than 1 second to run on my laptop (brute force algo):

```
data4.txt, {813577, 902305, 304318}
data5.txt, {359920, 352374, 249854}
zzz23.txt, {27401, 49751, 17568}
zzz151.txt, {61515, 3834, 44874}
zzz277.txt, {37428, 23407, 47945}
zzz384.txt, {15228, 4574, 30925}
```

These ones take from 1 second to 1 minute on my laptop (brute force algo):

```
zzz502.txt, {96480, 27477, 24524}
zzz1000.txt, {8494, 86196, 39361}
zzz1499.txt, {33210, 17709, 23118}
zzz2001.txt, {34875, 53969, 6283}
zzz2501.txt, {96665, 53636, 23007}
zzz3003.txt, {44310, 79351, 51127}
```

This one I estimated would take 70 hours on my laptop (brute force algo)! But the clever algo solves it in a few milliseconds. *I do not expect you to complete the brute force algorithm on this data file.*

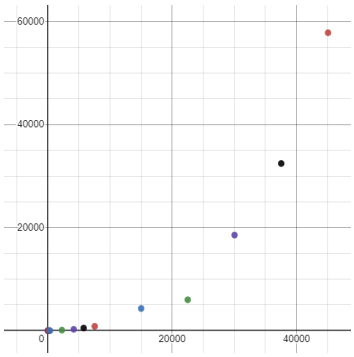
```
data70hrs.txt, {18521, 70138, 67814}
```

Expected results (data plots)

Here are what my data plots look like:

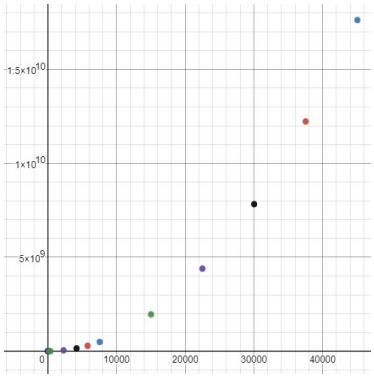
Plot1

Running time (millisecs) as a function of N, this is $O(N^3)$:



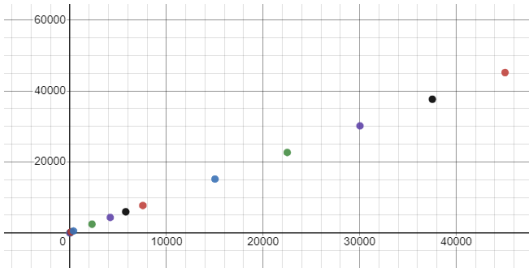
Plot2

Number of mod(%) operations of Brute Force algorithm as a function of N, this is also $O(N^3)$:



Plot3

Number of mod(%) operations of Clever algorithm as a function of N, this is O(N):



Specific code requirements

1. Do not change the name of the "starter code" class TripleFinder (filename TripleFinder.java).
2. Implement the following public methods that are "stubbed out" in the starter code. See comments in the code for full information.

```
public void findTripleBruteForce(Integer[] listOfIntegers)
public void findTripleClever(Integer[] listOfIntegers)
public Integer[] getTheTriple()
public long getBruteForceOperations()
public long getBruteForceRuntime()
public long getCleverOperations()
public long getCleverRuntime()
```

3. Do not change the function signature of any of the above methods.
4. You are free to declare other methods (if any) that you wish to use in the TripleFinder class.
5. Note that all of your methods may be called on one instance of your class object *multiple times with different data files.* The various getters should always return results of the last time that one of the triple-finder algorithms was called.
6. You will need code to read a data file and create an array of Integers (the object type of Integer, not the primitive int). You can put this code in TripleFinder or in your Main class. I will *not* be calling it (my code for testing your program has its own).
7. I shared some code to read the data file, which you can use or not, as you wish. It is available in the Lab1 assignment folder. (See "readArray.java".)

Tips

You'll need a main() method to test your code

You can put this in the TripleFinder class, or you can make a separate class whose purpose is basically to contain "main()". *I strongly recommend making the separate class*, since that is the same way that I will be testing/using your TripleFinder class. But you do *not* need to turn in your extra class.

How to time your code in Java

Use function `System.currentTimeMillis()`; it returns a `long` integer. Call it before the code you want to time, and then again after. Subtract the two results to get the total running time in milliseconds.

Submission information

Due date: As shown on Learning Hub.

Submit the following items to the drop box on Learning Hub:

- Your Java source code (TripleFinder.java)
- Images of your three data plots (screenshot is fine, but if you know a way to make SVG or some other graphic format directly, how cool would that be?)

- Please *do not zip* or otherwise archive your code or your submission. Plain Java and image files only.
- Please *do not zip* or include your entire IDE project directory.

Marking information

This lab is worth 20 points. 5 points are reserved for conformance with the COMP 3760 Coding Requirements (handout in Learning Hub "Content" section). Write your name and ID in your code comments!