UNBC Laboratory Assignment
Computer Science 200—Fall 2021

Timing Big Integer Multiplication

Due Date:

This assignment is due Monday, 2021-09-20 *by the beginning of lecture*.

Stopwatches

Create a `StopWatch` class that has the methods shown in Figure 1 on page 3. Note that the action methods have return type `StopWatch` rather than CPSC 101-style return type `void`. This alternate return style allows one to write something like

```
StopWatch sx = new StopWatch().reset().start() ;
```

Use `System.nanoTime()` to provide the timing information. Be aware that although `System.nanoTime()` returns values in nanoseconds, the actual high-precision timer in Java may well be coarser.

Read the "Instructions on Plotting and Timing" on Casperson's web-site.

`BigIntegers`

Read about the `java.math.BigInteger` class. One important fact about `BigIntegers` is that you *cannot* treat the running time of operations as constants independent of the size of the integers.

⇒ Write a program that uses your `StopWatch` clas from above, and the `java.math.BigInteger` class to measure the average time for the following operations:

- Measure the *average* time to add two n-bit big integers, where n varies between 10^3 and 10^8.
- Measure the *average* time to multiply two n-bit big integers, where n varies between 10^3 and 10^8.

⇒ Plot your data. (See the "Instructions on Plotting and Timing".) Provide separate graphs for the addition data and the multiplication data.

⇒ Comment on the mathematical nature of the running times. For instance, if you increase the problem size by a factor of 100, does the running time increase by the same factor? What do you suspect the functional relation between problem size and running time is?

Comments on Implementation

1. The problem-size range 10^3 to 10^8 seems appropriate for my laptop. You may have to adjust the range to be appropriate for the machine on which you perform your tests.

2. For small problem sizes, the running time that you are measuring may be small compared to the precision of `System.nanoseconds()`. To accommodate this, you may need to use techniques like starting your stopwatch, running multiple calculations, and then stopping your stop-watch.

3. Older versions of JAVA have less efficient algorithms for the `java.math.BigInteger` class. If you have access to an old version of JAVA, please use it. The data are more interesting.

Method	*Meaning*
attributes	
`public double elapsed()`	Returns the elapsed CPU time *in seconds* at the time of the call.
actions	
`public StopWatch start()`	Starts the stop watch. Has no effect if the stop watch is already started. Does not reset the time. Returns `this`.
`public StopWatch stop()`	Stops the stop watch. Has no effect if the stop watch is already stopped. Does not reset the time. Returns `this`.
`public StopWatch reset()`	Resets the elapsed time to zero. Neither starts nor stops the stop watch. Returns `this`.
Constructors	
`public StopWatch()`	Creates a new `StopWatch`, which is initially stopped with zero elapsed time.

Figure 1: Properties and Actions of `StopWatch`es