Kaitlyn Stumpf
CSC231, Fall 2015
10/2/2015

Problem 2: When performing arithmetic, is using BigIntegers slower, or faster than using ints, and how much faster or slower? Please include supporting evidence in your answer.

To answer this question, I took the arithmetic problem written in Java and used to compare Java, C++, Nasm, and Python in a previous class exercise and created two (slightly, consistently modified) versions, one where the numbers were stored as BigIntegers and another where the numbers were stored as ints. I compiled and ran both programs ssh-ed to the Aurora server. Below is a copy of the code from both programs, and my results after running the two.

kaitlynstumpf — ssh 231a-ao@aurora.smith.edu — 90×44

File Edit Options Buffers Tools Java Help

```
/**
 * BigintTest.java
 * @author Kaitlyn Stumpf
 */

import java.math.BigInteger;


class BigintTest {

    public static void main( String[] args ) {
        BigInteger a = new BigInteger("0");
        BigInteger b = new BigInteger("2");
        BigInteger c = new BigInteger("3");
        long noIterations = 100000000l;
        long start = System.currentTimeMillis(), end;

        for ( int i=0; i<noIterations; i++ )
            a = a.add(b.multiply( b ).add( c ));

        end = System.currentTimeMillis();
        System.out.println( "a = " + a );
        System.out.println( noIterations + " iterations, "
                          + (end-start) + " milliseconds ==> "
                          + (end-start)*1000000/noIterations + " ns/iteration" );
    }
}
```

kaitlynstumpf — ssh 231a-ao@aurora.smith.edu — 90×44

File Edit Options Buffers Tools Java Help

```
/**
 * IntsTest.java
 * @author Kaitlyn Stumpf
 */
class IntsTest {

    public static void main( String[] args ) {
        int a = 0;
        int b = 2;
        int c = 3;
        long noIterations = 100000000l;
        long start = System.currentTimeMillis(), end;

        for ( int i=0; i<noIterations; i++ )
            a += b*2 + c;

        end = System.currentTimeMillis();
        System.out.println( "a = " + a );
        System.out.println( noIterations + " iterations, "
                          + (end-start) + " milliseconds ==> "
                          + (end-start)*1000000/noIterations + " ns/iteration" );
    }
}
```

```
[231a-ao@aurora ~/hw5 $ java BigintTest
a = 700000000
100000000 iterations, 12756 milliseconds ==> 127 ns/iteration
[231a-ao@aurora ~/hw5 $ java IntsTest
a = 700000000
100000000 iterations, 94 milliseconds ==> 0 ns/iteration
231a-ao@aurora ~/hw5 $
```

As expected, the code containing the class BigIntegers took a much longer time (12,756ms) to execute, whereas the code working with integers took only 94ms. Integers in Java are always 64 bits and unsigned. BigIntegers, however, act as an array, in that their size depends on the size of the number stored within. If the int became overflowed, it would still be working (incorrectly) within its 64 bit parameters, whereas BigInteger would continue to accommodate the growing number, slowing performance speeds but providing a more reliable result.

According to the Javamex BigInteger tutorial found at (http://www.javamex.com/tutorials/math/BigDecimal_BigInteger_performance.shtml) BigInteger's add() and subtract() methods also "scale linearly as a function of the number of digits in the (larger) number being added", while the performance of its multiply() method scales exponentially. What sets BigInteger and int apart is simply the amount of space allocated to contain the numerical values being worked with.