HugeIntegerClassDetails

Description of data structures and algorithms:

public HugeInteger(String val)throws IllegalArgumentException{

The HugeInteger(String val) function converts a string "val" to a integer array digits through the use of the string function charAt() for each character of the string. If the contains anything other than a number or '-'an exception will be thrown. This exception will also be thrown if '-' is not at the start of the string. If the string starts with '-'then the global integer variable Negative will be set to -1 otherwise it will be equal to 1.

}

public HugeInteger(int n) throws IllegalArgumentException{

The **HugeInteger(int n)** function is very similar to the other constructor but instead of receiving a String of numbers it is given a number (n) and generates a random number with n number of decimal places. The only exception that needs to be checked for this function is that n is greater than 0. If n is less than or equal to 0 an exception is thrown. If n is larger than zero a new array of integers is created of length n and each element is randomly chosen from 0 to 9 with the use of java.util.Random and the function nextInt(10), which randomly selects a number from zero to nine. At the end whether the random number is negative or not is also randomly selected using nextInt(2) in which 1 is negative and 0 is positive.

}

public HugeInteger add(HugeInteger h){

Addition between two negative numbers and too positive numbers is very simple you start at the end of each integer array (digits in my case) and add each element together plus the integer variable carry if the addition is greater than 10 carry is set to 1 and 10 is subtracted from the element otherwise carry is set to 0. I used an integer variable x to store each element addition then a temp integer array that would store the values in order. If the loop ends with carry equaling 1, 1 is added to the last element of temp. Finally leading zeros were removed, the integer array temp was transferred to a String named sum and a new HugeInteger was made with the string sum. If both numbers were negative the new HugeInteger .Negative was set to -1

Addition between a negative and positive number is a little more difficult. This requires to find the larger number between the two then subtracting the smaller number from the bigger number. This could have been done using the comparison function but as this was made first it has almost exact same method of finding the bigger number this process will be described in the description of compareTo().Once again the integer variable Carry and x are used for storage. If the numbers are equal a new Hugeinteger("0") is

made and returned. If a bigger number is found we start at the last element of both arrays and the smaller number is subtracted from the bigger number plus carry ,if borrowing is required carry is set to -1 and 10 is added to the element otherwise carry is set to 0 and temp[i] is set to x. Finally leading zeros were removed, the integer array temp was transferred to a String named sum and a new HugeInteger was made with the string sum. If the bigger number was negative the new hugeinteger is set to negative as well. The new HugeIntger is returned.

This function has safeguards emplace so that different number of decimals work together this is if the array would be out of bounds it is not considered in the addition.ie i>array.length where i is used to loop through each element.

Variables used:

Int Length: the larger of the 2 lengths / number of loops done for addition.

Int x: the value of the addition of each element + carry (not necessary)

int carry: either 1(carry over), 0(no Carry or borrow), or -1(Borrow)

int [] temp: Each element of the addition

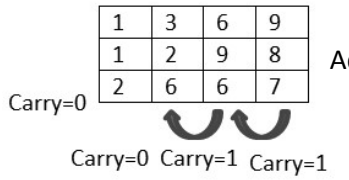
String sum: temp without leading zeros and the correct order (flipped temp)

bigger: store the bigger number ice bigger=1 **this** is bigger bigger=-1 **h** is bigger bigger=0 **this** and **h** are equal

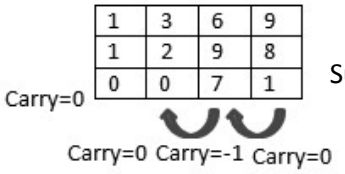
i = count for loops

j = count for loops

boolean Check a check for leading zeros



Addition



Subtraction

}

public HugeInteger subtract(HugeInteger h){

add(HugeInteger h) adds both negative and positive numbers therefore subtract only switches the sign of h and calls add.

}

public int compareTo(HugeInteger h){

CompareTo() checks the length of both HugeIntegers if one is bigger the appropriate value is returned. If they are the same length we loop through each element from the most significant digit to the least significant digit. Each element is compared if one is bigger the loop breaks and the appropriate value is returned. If they are equal the full loop completes and the value 0 is returned.

Variables:

Int bigger: store the bigger number ice bigger=1 **this** is bigger bigger=-1 **h** is bigger bigger=0 **this** and **h** are equal

Int Length: the larger of the 2 lengths / number of loops done for addition.

boolean Check a check for -0 compared to 0

j = count for loops

}

public String toString(){

Prints each element of the digits integer array and a negative sign if the number is negative.

Variables:

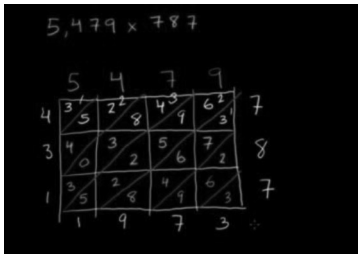
String temp Store the array for temporary time.

}

public HugeInteger multiply(HugeInteger h){

Multiply uses a lattice multiplication style which you can see below. This required a nested for loop which multiplies each element and adds them plus the carry to the appropriate temp element. This process starts at the bottom right of the picture depicted below. Carry is calculated by dividing the current element of temp by 10 resulting in a value from 0 to 9.

Variables used:



boolean Check a check for leading zeros

int carry: 0-9 element multiplication/10

int [] temp: Each element of the multiplication

String sum: temp without leading zeros and the correct order (flipped temp)

bigger: store the bigger number ice bigger=1 **this** is bigger bigger=-1 **h** is bigger bigger=0 **this** and **h** are equal

i = count for loops

}

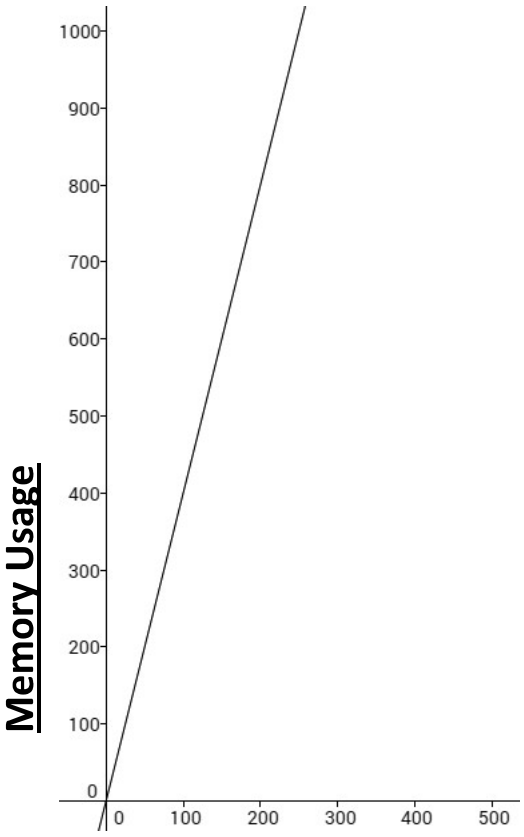
Theoretical analysis of running time and memory requirement:

Memory Usage:

$Integer = 4\ bytes\ \therefore Memory\ usage = 4 \times (n+1)\ where\ n\ is\ the\ size\ of\ the\ Intger\ array.$

n is from the integer array and the 1 is from the integer Negative

Figure:



Addition:	**Subtraction:**	**Multiplication:**	**Comparison:**
Worst Case: n^2	Worst Case: n^2	Worst Case: n^2	Worst Case: n
Average Case: n^2	Average Case: n^2	Average Case: n^2	Average Case: 1

ddition Subtraction and Comparison there was a single loop has the value of n, but both addition and subtraction input the array into a string which makes the function run at n^2. While there was a nested loop in multiplication plus the addition to a string makes multiply equal to n^3;

Test procedure:

The same number of decimals, different decimals, numbers greater than and less than 2^{63}, positive and negative numbers for both "this" and "h". Also numbers with carries and without a carry were tested. In addition the runtime test was done with all the numbers being printed and 20 random cases for each function were compared to the actual result and were correct.

Test Cases:

Special Case:

(0 & -0)

Errors:

"Helloworld" "-25-56"

Same # decimals > 2^{63}:

10223372036854775808, 10223372036854775808

-10223372036854775808, 10223372036854775808

10223372036854775808, -10223372036854775808

-10223372036854775808, -10223372036854775808

Different # decimals > 2^{63}:

1004433758368537899580 8, 10223372036854775808

-1004433758368537899580 8, 10223372036854775808

1004433758368537899580 8, -10223372036854775808

-1004433758368537899580 8, -10223372036854775808

Same # decimals < 2^{63}:	**Different # decimals < 2^{63}:**
25, 50	625, 99
-25, 50	-625, 99
25, -50	625, -99
-25, -50	-625, -99

All solutions to the above numbers are correct for each function. Also an exception is thrown if an invalid input is used. I.e. HugeInteger("Hello World"); would throw an exception.

If an error had occurred it would be difficult to debug considering there are so many test cases. This was not an issue because the program is correct for all possibilities.

There are infinite many inputs that have not been checked but it is assumed beyond a reasonable doubt that the program would work for them.

Experimental measurements, comparison and discussion:

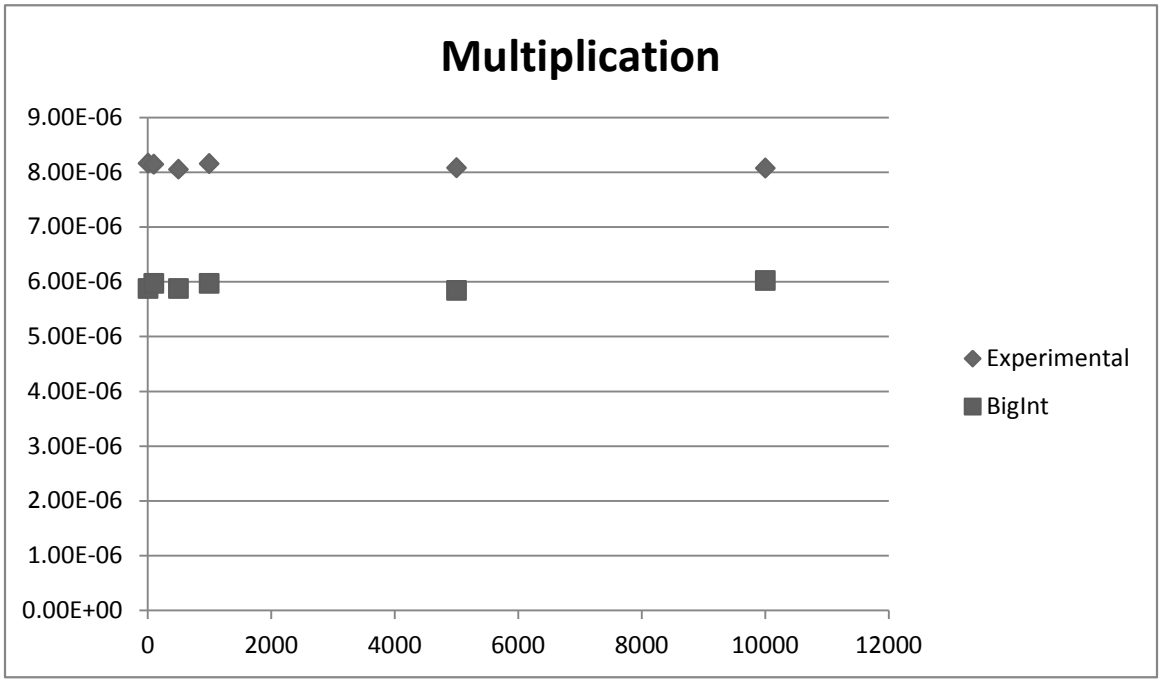
Test Method:

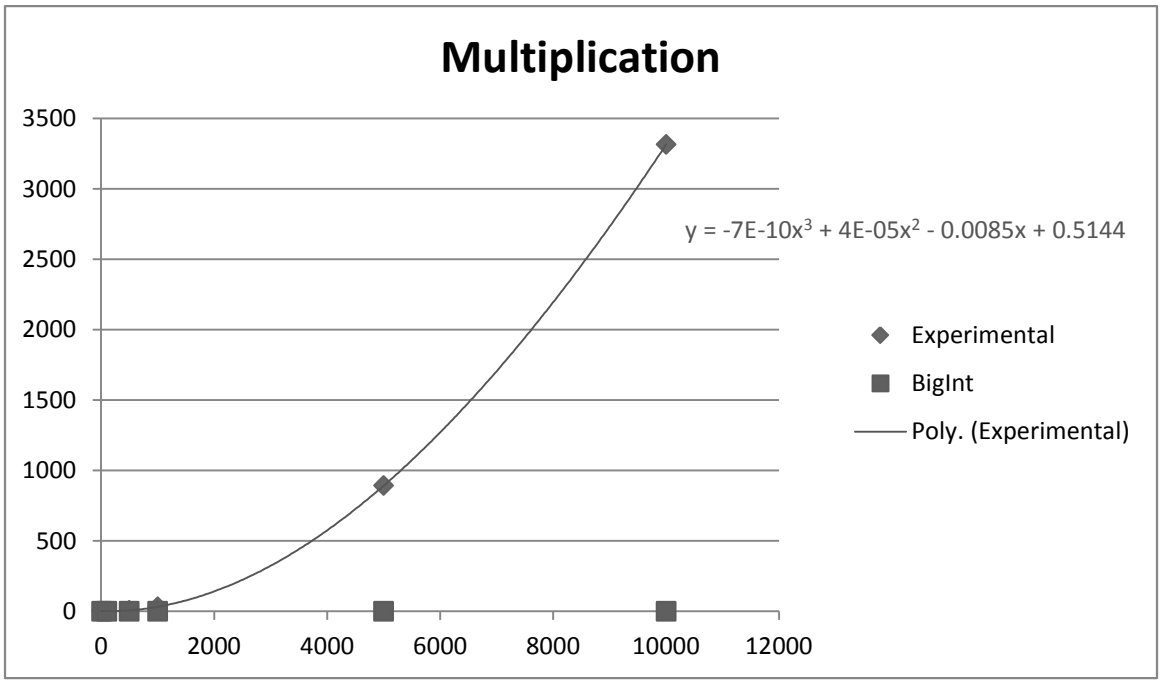
The data here was tabulated using a variation on the test code provided in the lab document. The time required to run each function was measured and recorded.

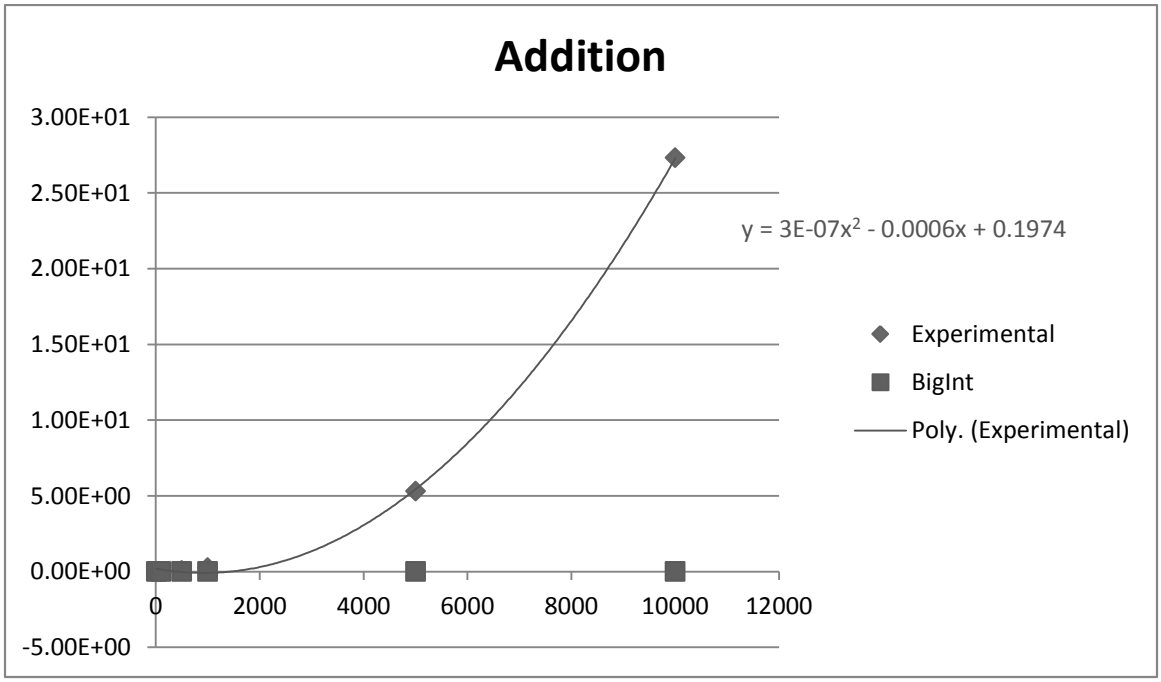
Table:

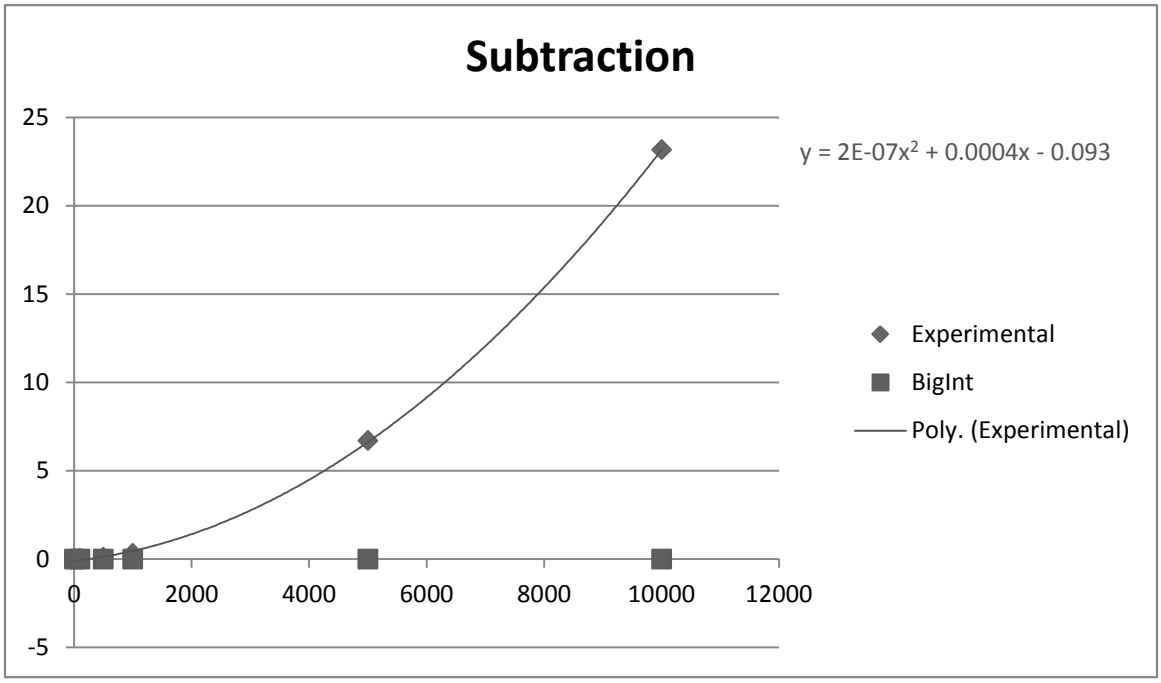
Runtime	Addition	Subtraction	Multiplication	Comparison	Digits(n)
Experimental	9.1234999999E-4	0.0012578249999	0.002253549	8.15869E-5	10
BigInt	4.39293333334E-5	5.816749998E-5	5.931450000E-5	5.87633E-6	10
Experimental	0.0083919999998	0.0087677	0.2303466666	8.14065E-6	100
BigInt	6.3455E-5	6.6735E-5	1.2808071428E-4	5.96783E-6	100
Experimental	0.06350066666665	0.094205333333	7.035799999	8.04775E-6	500
BigInt	8.21639999999E-5	8.7958024691E-5	8.1874117647E-4	5.87426E-6	500
Experimental	0.252162	0.303455	31.66219999	8.15369E-6	1000
BigInt	1.1670444442E-4	1.141979999E-4	0.0025087799995	5.9688E-6	1000
Experimental	5.31170000001	6.68806	892.085	8.0856E-6	5000
BigInt	4.66833333326E-4	4.955541666E-4	0.0543964999999	5.84139E-6	5000
Experimental	27.3301666666	23.155	3314.51	8.0698E-6	10000
BigInt	9.828299E-4	0.001010154545	0.1645687499	6.02325E-6	10000

Figures:


Multiplication
9.00E-06
8.00E-06
7.00E-06
6.00E-06
5.00E-06
4.00E-06
3.00E-06
2.00E-06
1.00E-06
0.00E+00
0
2000
4000
6000
8000
10000
12000
Experimental
BigInt


Multiplication
3500
3000
2500
2000
1500
1000
500
0
0
2000
4000
6000
8000
10000
12000
y = -7E-10x³ + 4E-05x² - 0.0085x + 0.5144
Experimental
BigInt
Poly. (Experimental)





Discussion of results and comparison:

As you can tell from the graphs above there are many improvements that could be applied to my code to run at a faster and less intensive manner. The first thing that would improve my runtime is a byte array instead of an integer array to store the hugeInt. Another improvement that could be made is by using a Boolean for the negative sign instead of an integer. There are also a lot of redundant/ unnecessary variables my code. For example I use the variable x in addition to temporally store the next integer to be entered into the array but I could simply modify the array as the integers are added.