

Description of Data Structure and Algorithms

I used strings to store the integers value. I picked this because it's the easiest way to store and also make changes. Also, because of the fact that we are already inputting a string to the constructor, it's easier to just keep it as a string rather than storing it as an array or linked list. For my HugelInteger class, I have three private members called hugeInt, size, and sign. HugelInt is a string of the integer inputted, excluding any 0 at the start before a non-zero integer and the negative sign if the number is negative. Size is how many digit the number is. In my constructor, I update the size of the number by going through the inputted string. For every 0 before a non-zero integer and a possible negative sign, I decrease the size by one so the size is a true representation of the length of the integer. Sign is a boolean value that defines if it's a positive or negative number. For positive, it is stored as true, and false for negative. So when I do my operations, I can just call sign and figure out the sign of the integer and result.

For addition, I did a basic addition digit by digit, so that no matter how big the integers are, they won't be out of range. Before actually doing addition, I first assume the second integer is bigger in size and store it in a variable call big, while also storing the first integer to variable small. But in order to make sure the second integer is actually bigger in size, I used the compare method to check. When the result of compare to is 1, which means the first integer is bigger, I will switch the value of big and small so that the variables are right. Then I start from the digit all the way from the right, and from there the addition of that one digit is executed. If the result of the digit addition is bigger than 10, then I subtract 10 away from it and put it into a variable "carry", which will be added to the next digit. In order to make it easier, I used the reverse function from stringbuilder so that instead of going through the integer from the back to front, I just compute it from left to right, and then reverse it back after addition is done. For the result, it was stored in a new string and goes through the HugelInteger constructor to make sure the outcome is a valid number without any problem. There are three special cases, which are adding a positive to a negative number, adding a negative to a positive number, or adding two negative numbers. For adding a positive to a negative number and adding a negative to a positive number, it is basically the positive number subtract the absolute value of the negative number, so I call the subtract function. But because it's a negative number, I have an absolute function that returns the absolute value of the number. And for adding two negative numbers, it is just two addition of the absolute value of the numbers but then setting the value of the result to be negative. Here is an example of $5628 + 74216$. As it is shown in the result, the addition was done one by one starting from the right, but because it was reversed at the start, the value shown before actually printing the result is 44897, which is backward of the result 79844.

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package Lab1_2;
7
8  /**
9   *
10   * @author Amos
11   */
12  public class Lab1_2 {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          HugeInteger int1 = new HugeInteger("5628");
20          HugeInteger int2 = new HugeInteger("74216");
21          HugeInteger int3 = int1.add(int2);
22          // HugeInteger int4 = int1.subtract(int2);
23          // HugeInteger int5 = int1.multiply(int2);
24          // HugeInteger int6 = int1.divide(int2);
25          System.out.println("Integer 1 =" + int1);
26          System.out.println("Integer 2 =" + int2);
27          System.out.println("Result =" + int3);
28          // System.out.println(int4);
29          // System.out.println(int5);
30          // System.out.println(int6);
31      }
32  }

```

lab1_2.Lab1_2 > main

```

run:
4
44
448
4489
44897
Integer 1 =5628
Integer 2 =74216
Result =79844
BUILD SUCCESSFUL (total time: 0 seconds)

```

For subtraction, it is the same basic idea from addition. I do subtraction of each digit starting from the right, and if the value is negative, I add ten, which is the complement of the result, and will output the correct value at that digit. And if ten was added to the digit, I subtract one from the next digit, so that the “borrowing” in math operation is done. For the special cases of subtracting a positive to a negative number and subtracting a negative to a positive number, it is actually the positive number adding the absolute value of the negative number, so addition is called instead to do the operation. And for subtracting two negative numbers, it is just the the absolute value of the second number subtracting the absolute value of the first number. So here is an example of 2036-2345. As it shows, the result prints it digit and reverse it back and add a negative sign because the first integer is smaller than the second one.

```

13  /*
14  *
15  * @author Amos
16  */
17  public class Lab1_2 {
18
19      /**
20       * @param args the command line arguments
21       */
22      public static void main(String[] args) {
23          // TODO code application logic here
24          HugeInteger int1 = new HugeInteger("2036");
25          HugeInteger int2 = new HugeInteger("2345");
26          // HugeInteger int3 = int1.add(int2);
27          HugeInteger int4 = int1.subtract(int2);
28          // HugeInteger int5 = int1.multiply(int2);
29          // HugeInteger int6 = int1.divide(int2);
30          System.out.println("Integer 1 =" + int1);
31          System.out.println("Integer 2 =" + int2);
32          System.out.println("Result =" + int4);
33          // System.out.println(int4);
34          // System.out.println(int5);
35          // System.out.println(int6);
36
37          // HugeInteger huge1, huge2, huge3;
38          // int hi;
39          // long startTime, endTime;
40          // double runTime=0;
41          // int n=10000, MAXNUMINTS = 100, MAXRUN=300;
42          // for (int numInts=0; numInts < MAXNUMINTS; numInts++)
43          //     huge1 = new HugeInteger(n); //creates a random
44          //     huge2 = new HugeInteger(n); //creates a random
45          //     startTime = System.currentTimeMillis();
46      }
47  }

```

lab1_2.Lab1_2 > main > int2

```

run:
9
90
903
9030
Integer 1 =2036
Integer 2 =2345
Result =-309
BUILD SUCCESSFUL (total time: 0 seconds)

```

For multiplication, I ran a nested for loops. I realize at the start the doing repeated addition will work, but it is not efficient at all. How the nested for loop works is that I realize when you do

multiplication, we basically take the digit all the way on the right of the first, multiply it with the each digit of the second integer, then add them together. And once you went through the first time and move to the next digit of the first number, the second round will be the same but multiplied by 10, or moved one digit to the left and added a zero there. So I can run a nested for loop, store all the values in an array, and then do addition of each row once the operation is done. And when the loop is done running and all addition is done, I check if any of the values are bigger than 10. If they are bigger, I find the remainder after dividing by 10, and store that to the digit of that position. Also, for the value at that position, I do an integer division by 10, store that as "carry", and add it to the next digit I look at. The special case here is the signs of the integers. I use if statements to check the signs of the two integers, and return the correct sign. If they are the same sign(positive and negative), the result is positives. If they are a different sign, the result is negative. Here is an example of 72×16 , and as it show, it do digit by digit, then gives

```

11  */
12  public class Lab1_2 {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          HugeInteger int1 = new HugeInteger("72");
20          HugeInteger int2 = new HugeInteger("16");
21          // HugeInteger int3 = int1.add(int2);
22          // HugeInteger int4 = int1.subtract(int2);
23          HugeInteger int5 = int1.multiply(int2);
24          // HugeInteger int6 = int1.divide(int2);
25          System.out.println("Integer 1 =" + int1);
26          System.out.println("Integer 2 =" + int2);
27          System.out.println("Result =" + int5);
28          // System.out.println(int4);
29          // System.out.println(int5);
30          // System.out.println(int6);
31
32          //
33          //         HugeInteger huge1, huge2, huge3;
34          //         int h1;
35          //         long startTime, endTime;
36          //         double runTime=0;
37          //         int n=10000, MAXNUMINTS = 100, MAXRUN=300;
38          //         for (int numInts=0; numInts < MAXNUMINTS; numInts++)
39          //             huge1 = new HugeInteger(n); //creates a random
40          //             huge2 = new HugeInteger(n); //creates a random
41          //             startTime = System.currentTimeMillis();

```

lab1_2.Lab1_2 2 main int2

Output - Lab1_2 (run)

```

run:
2
25
251
2511
Integer 1 =72
Integer 2 =16
Result =1152
BUILD SUCCESSFUL (total time: 0 seconds)

```

the result of the actual answer at the end

$ \begin{array}{r} 1234 \\ \times 345 \\ \hline 34022343120 \end{array} $	<ul style="list-style-type: none"> - $m(0+0) = 4 \times 5 = 20$ - $m(0+1) = 4 \times 4 = 16$ - $m(0+2) = 4 \times 3 = 12$ - $m(1+0) = 15$ - $m(1+1) = 12$ - $m(1+2) = 9$ - $m(2+0) = 10$ - $m(2+1) = 8$ - $m(2+2) = 6$ - $m(3+0) = 5$ - $m(3+1) = 4$ - $m(3+2) = 3$
--	---

For division, It is not the most efficient and best method, but due to the time limit I had, I just did a repeated subtraction. So I use the subtract function to subtract the second integer from the second one in a while loop, and having a counter to count how many times it ran through. Once the while loop is done running, the counter is the answer of that division, so I just runs it through the HugeInteger constructor and give the result. For the signs of division, it is the same idea as multiplication, where same signs gives you positive and different signs gives you negative. Here is an example of -17/8. Because 17 can only be divided by 8 twice, the loop only ran twice and output a negative number because the signs of the two integers are different.

```

11  */
12  public class Lab1_2 {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          HugeInteger int1 = new HugeInteger("-17");
20          HugeInteger int2 = new HugeInteger("8");
21          //
22          HugeInteger int3 = int1.add(int2);
23          //
24          HugeInteger int4 = int1.subtract(int2);
25          //
26          HugeInteger int5 = int1.multiply(int2);
27          //
28          HugeInteger int6 = int1.divide(int2);
29          System.out.println("Integer 1 = " + int1);
30          System.out.println("Integer 2 = " + int2);
31          System.out.println("Result = " + int6);
32          //
33          System.out.println(int4);
34          //
35          System.out.println(int5);
36          //
37          System.out.println(int6);
38
39          //
40          HugeInteger huge1, huge2, huge3;
41          int hi;
42          //
43          long startTime, endTime;
44          double runTime=0;
45          //
46          int n=10000, MAXNUMINTS = 100, MAXRUN=300;
47          for (int numInts=0; numInts < MAXNUMINTS; numInts++){
48              huge1 = new HugeInteger(n); //creates a random integer
49              huge2 = new HugeInteger(n); //creates a random integer
50              startTime = System.currentTimeMillis();
51          }
52      }
53  }

```

Output - Lab1_2 (run) **main**

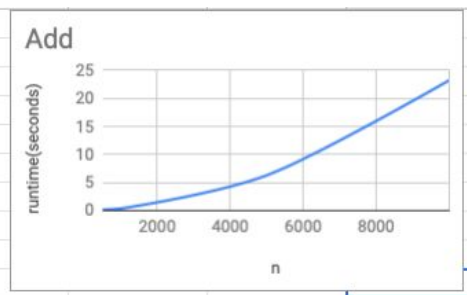
```

run:
1
2
Integer 1 =-17
Integer 2 =8
Result =-2
BUILD SUCCESSFUL (total time: 0 seconds)

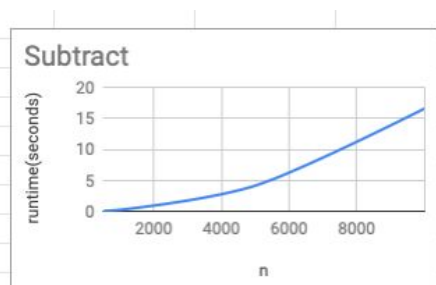
```

Run Time Results

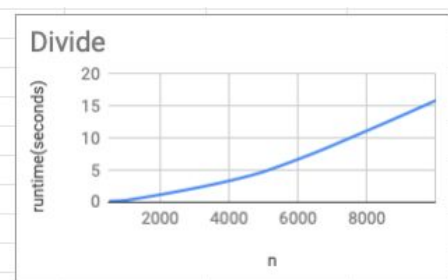
Add			
n	maxnumints	maxrun	runtime(seconds)
10	100	300	0.01513333333
100	100	300	0.03806666667
500	100	300	0.1435333333
1000	100	300	0.3456666667
5000	100	300	6.2675
10000	100	300	23.1988

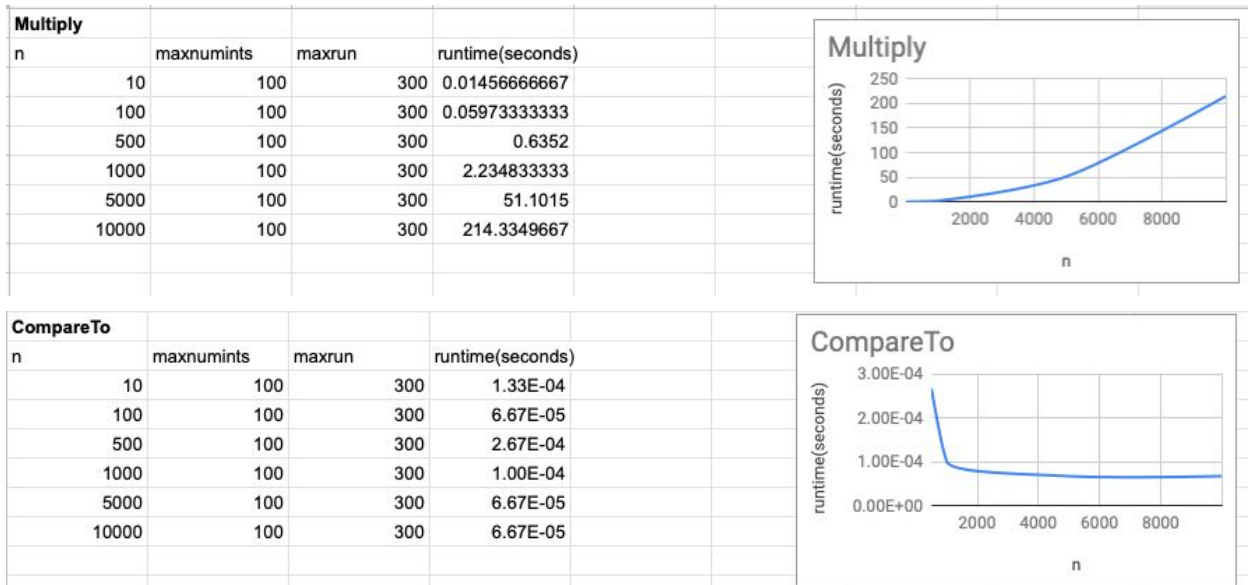


Subtract			
n	maxnumints	maxrun	runtime(seconds)
10	100	300	0.009033333333
100	100	300	0.0375
500	100	300	0.1378
1000	100	300	0.3152333333
5000	100	300	4.213033333
10000	100	300	16.61196667



Divide			
n	maxnumints	maxrun	runtime(seconds)
10	100	300	0.0073
100	100	300	0.0448
500	100	300	0.1555
1000	100	300	0.2928
5000	100	300	4.747533333
10000	100	300	15.87766667





As it shows in the runtime result, all operations have a close to linear runtime. There was one small problem during the runtime process, and that is the long build time. When doing multiply on 10000 digits, it took a total of 105 minutes to build after the laptop crashed on the previous try.

Discussion

In the first lab, I actually attempted to implement the lab in an array, but it wasn't a good choice as resizing an array is impossible, which makes it harder to create the functions as more has to be taken into account. Because of this, I changed my implementation method and used strings to store my integers and do operations on. If extra time was given, I would implement my divide function in another way. The current way of division is the easiest to implement, but the runtime is really slow as it has to keep calling the subtract function and do the operation one by one. I would have implemented it similar to multiply, by looking at digits one by one and do operations on it.