COMPENG 2SI4

Lab 1 and 2

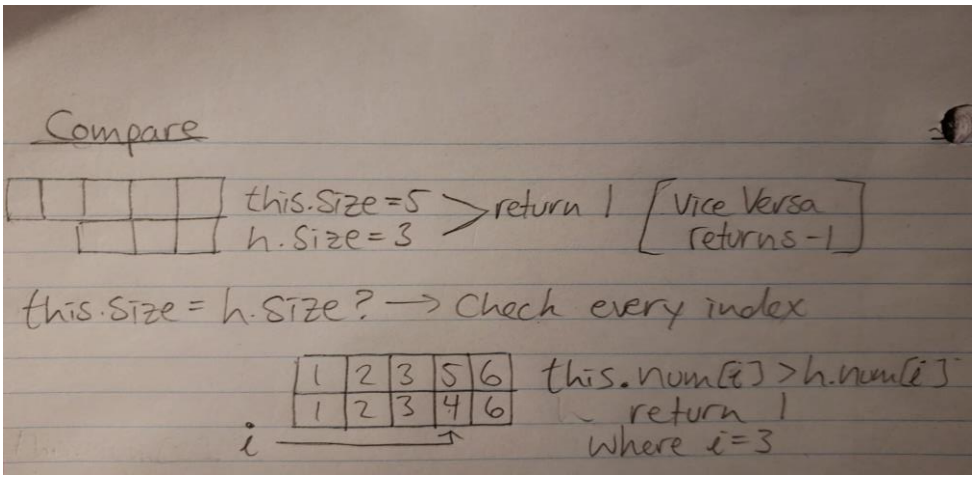
Helen

L03

1.0 Description of Data Structures and Algorithms

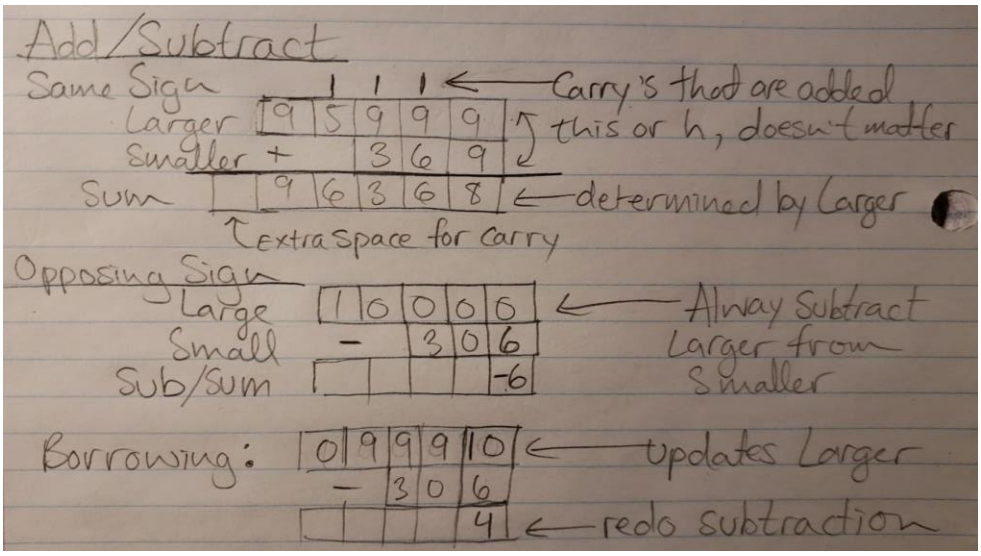
I implemented the HugeInteger class using arrays. For my contructor, I store the sign, the size and the array of the HugeInteger as instance fields so I can refer to them in any method of this class. For all methods (except compareTo), a function deepCopy was used to make a copy of this and h so they are not overwritten in memory.

1.1 Comparison



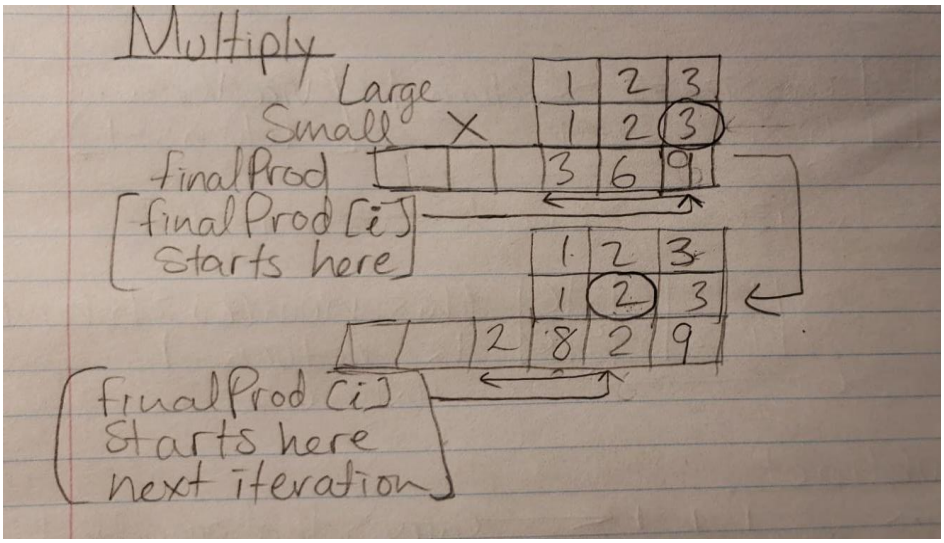
For my compareTo method, I also used another method called compareToMagnitude. compareToMagnitude will return 1 is this > h, -1 if this < h, and 0 if they are the exact number. It does this by comparing the size of the HugeIntegers (which is stored as a field), or looping through each index of this and h to find an instance where 1 index is greater than the other (if they are the same size). My compareTo method then checks the sign of both this and h. It will check if they have the same or opposing signs. If they have opposing signs, it will return 1 or -1 depending on which number is the positive one (regardless of magnitude). When they have the same sign, it will call my compareToMagnitude function. If they are both positive numbers, it will return the same value as my compareToMagnitude function. If they are both negatives, it will return the opposite value of my compare to magnitude function (because the larger number is technically the less negative one). It will return zero if they have the same sign and magnitude.

1.2 Addition/Subtraction



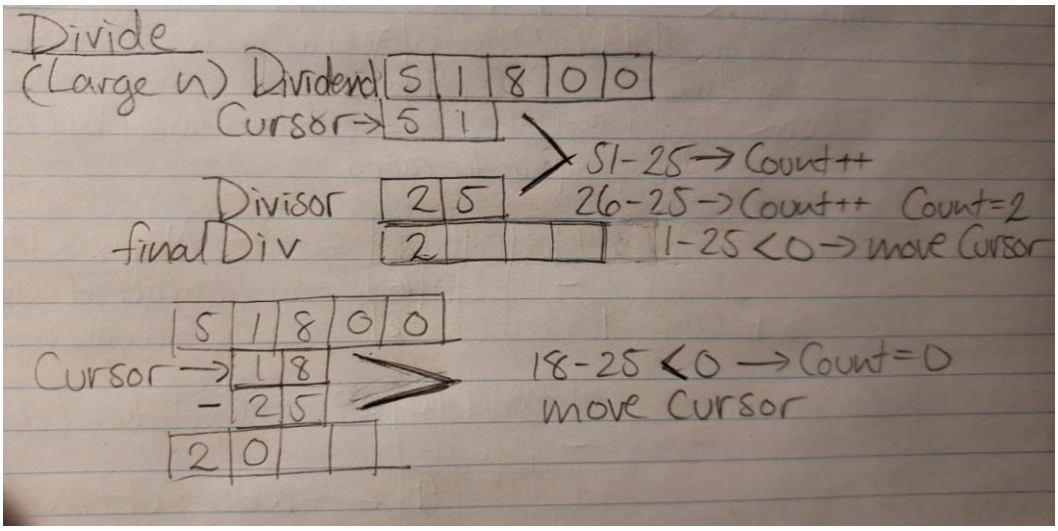
My addition function does the calculation of the subtraction as well. My Addition function is separated into 2 scenarios; if this and h have the same or opposing signs. It will also calculate which HugeInteger is larger in magnitude (using my compareToMagnitude method) in order to know how much space to allocate for the HugeInteger being returned (sum) and how to calculate the sign. The sign of sum is the same sign as both this and h if they have the same sign, and the sign of the larger number (in magnitude) if they have opposing signs. If they have the same sign, it iterates through both HugeIntegers using 2 for-loops (starting at the end of the arrays of both HugeIntegers) and adds the large HugeInteger to the smaller at i, plus any previous carry (initialized to zero) and stores it at sum. It does this because any numbers which have the same sign is adding more of it to itself. If they have a different sign you are essentially subtracting them. The second half of my method again uses 2 for loops and the concept of borrowing from a previous index, to subtract the smaller HugeInteger from the larger.

1.3 Multiplication



My multiply method uses the standard multiplication algorithm, which is multiplication by parts and then adding them afterwards. Except my method doesn't add the multiplication parts at the end, it adds them as it moves along the finalProd HugeInteger array, thus saving time and complexity of the code. It'll loop through both arrays (starting at the last index for both) and multiply the larger HugeInteger by the smaller while simultaneously adding any carry and any previous number that was already stored in finalProd array at that index. For each time it changes an index in the smaller array, the place holder for the finalProd array knows to move to the next index to its left, thus making sure it is adding the multiplication parts at the right index. The sign of the finalProd is determined by a simple conditional statement which checks if this and h have opposing signs. If they do, we know it will be a negative finalProd.

1.4 Division



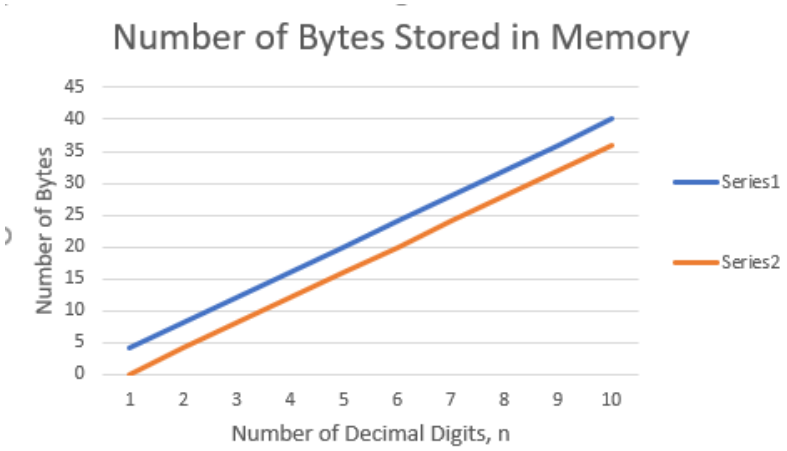
My division method is divided into separate cases. The first case it checks for is if we are dividing by zero. If we are dividing by zero it throws an exception, saying the division cannot be done. The next portion of the code checks to see if this and h are the same magnitude by calling compareToMagnitude. If they are the same magnitude it will return 1 (of type HugeInteger) and the sign be negative if this and h have opposing signs. The next portion checks to see if this is smaller than h. If it is it will return zero since a smaller number cannot be divided into a larger by integer division logic. The final portion of the code executes when this is larger than h. First my code will use two different methods of calculating division depending on whether this and h are large numbers. If this and h are small numbers, it will call the subtract function and just subtract h from this recursively till this becomes smaller than h. A counter keeps track of how many times this is done by adding 1 to itself every iteration. If this and n is relatively large, my function will store the first index values of this array in another variable called cursor, which is the same size as h. It will then proceed to subtract h from cursor till it no longer can while simultaneously counting how many times we have subtracted it. It will store this value at the corresponding index of the finalDiv array. When cursor becomes too small, it will move to the next index in this array and again subtract h from cursor.

2.0 Theoretical Analysis of Running Time and Memory Requirement

2.1 Memory Required

My constructor stores the string of n digits as an array of integers, where each index/integer is one digit. Assuming an integer allocates 4 bytes in memory, the formula for the number of bytes

required to store my integer is 4*n (Series 1 in diagram). If it is a negative number, then the formula is 4*(n-1) (Series 2 in diagram).



2.2 Running Time and Extra Memory

2.2.0 Compare

This method calls compareToMagnitude, which at worst checks n integers if this and h are the same size. Therefore, it's big-Theta worst case is n. CompareTo method has no extra memory needed for execution.

2.2.1 Addition/Subtraction

Since my subtraction function calls my addition function, they both have the same worst and average case. My method, whether adding or subtraction, will use a for loop to move through the larger array. Big-Theta average is therefore n. In the worst-case scenario, it will have to go through the while loop which will borrow from every previous index in the larger array. Therefore big-Theta worst case is n^2.

2.2.2 Multiplication

My multiplication method uses 2 for loops to iterate through this and h. Therefore, its big-Theta average and worst case is n^2.

2.2.3 Division

My division method is divided into 2 scenarios, n really small and n really large. Both scenarios have big-Theta average time complexity n ,because they call the subtract function in order to implement. However If n is really large, big-Theta worst case will be 10*n because it will go through every index and subtract around at most n amount of times.

3.0 Test Procedure

My test class that I handed in during lab time covered all possible test cases. For a detailed analysis you can look into the test class handed in. All my outputs were found to meet the

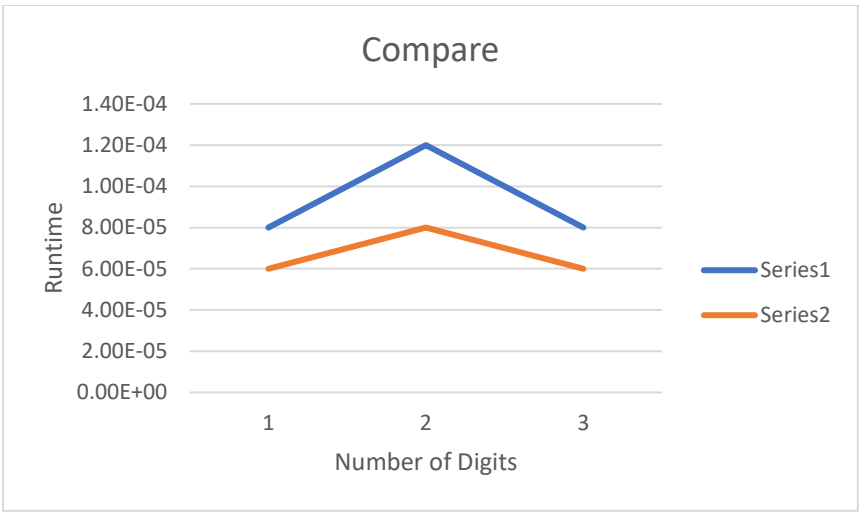
specifications required. I found I had the most difficulty in debugging my division method because I originally could not think of a simple and time efficient method of implementing division using arrays, so some of my outputs took too long to calculate when presenting the lab. I had to change my division method for the report in order to test all the methods properly. The only input conditions I originally could not check were really large numbers divided by really small numbers for my division class, as the running time was too large.

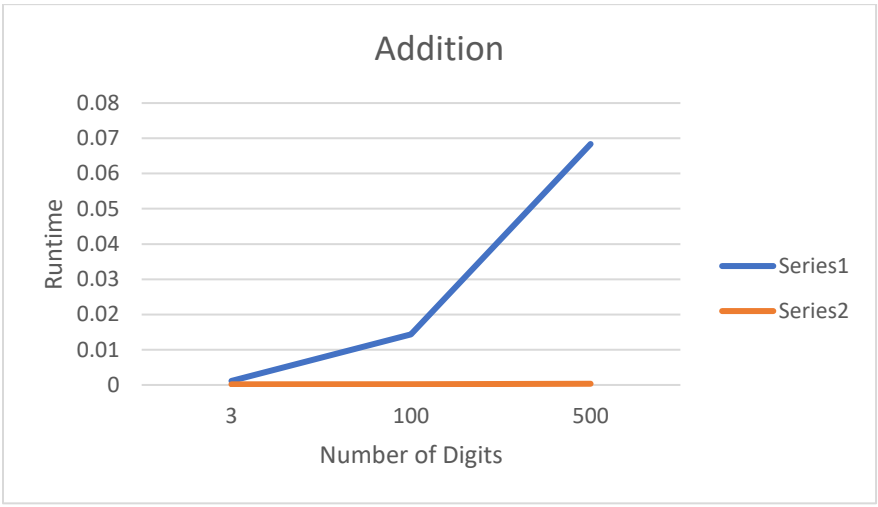
4.0 Experimental Measurement, Comparison and Discussion

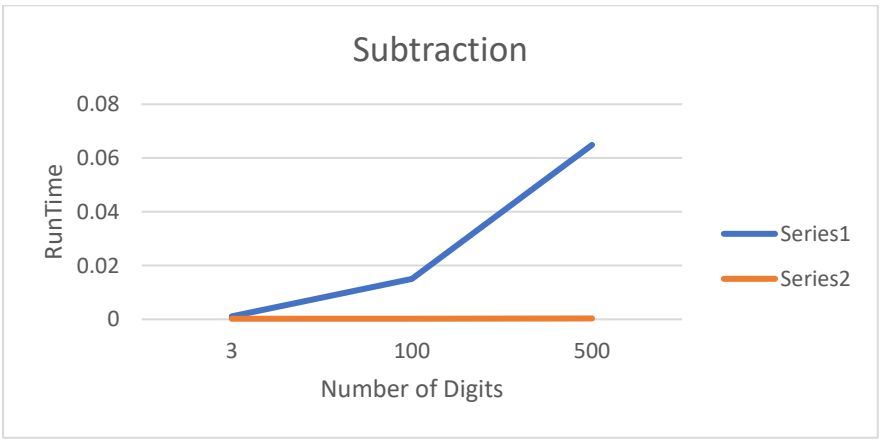
The values of the parameters used in this lab are given in the table below. They were measured using the sample code given in the lab requirements.

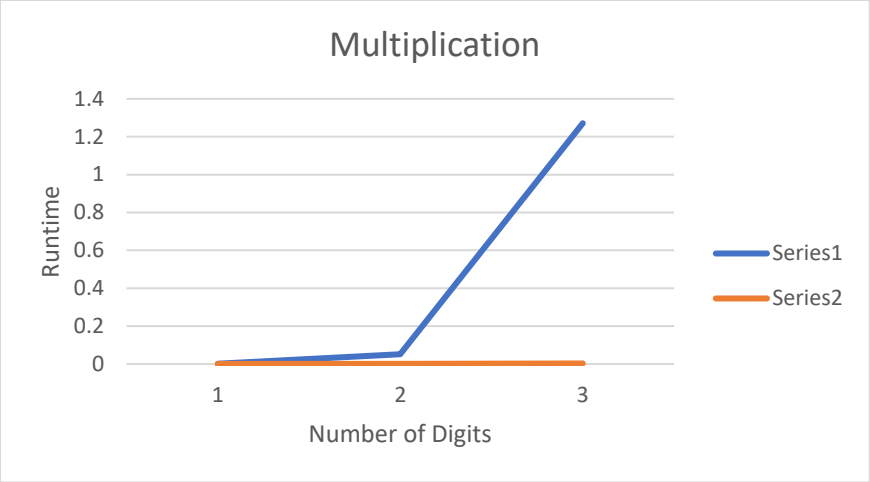
n	HugeInteger				
	Compare	Addition	Subtraction	Multiply	Divide
3	8.00E-05	0.00114	0.0011	0.00142	0.00138
100	1.20E-04	0.01432	0.01502	0.052179	0.05679
500	8.00E-05	0.068379	0.064879	1.27144	1.25268
	BigInteger				
	Compare	Addition	Subtraction	Multiply	Divide
3	6.00E-05	2.20E-04	2.20E-04	2.80E-04	2.20E-04
100	8.00E-05	2.00E-04	2.20E-04	5.20E-04	4.60E-04
500	6.00E-05	3.40E-04	3.20E-04	2.74E-03	2.74E-03
	Theoretical				
	Compare	Addition	Subtraction	Multiply	Divide
3	3.00E-04	9.00E-04	9.00E-04	9.00E-04	9.00E-04
100	1.00E-02	1.00E+00	1.00E+00	1.00E+00	1.00E+00
500	5.00E-02	2.50E+01	2.50E+01	2.50E+01	2.50E+01

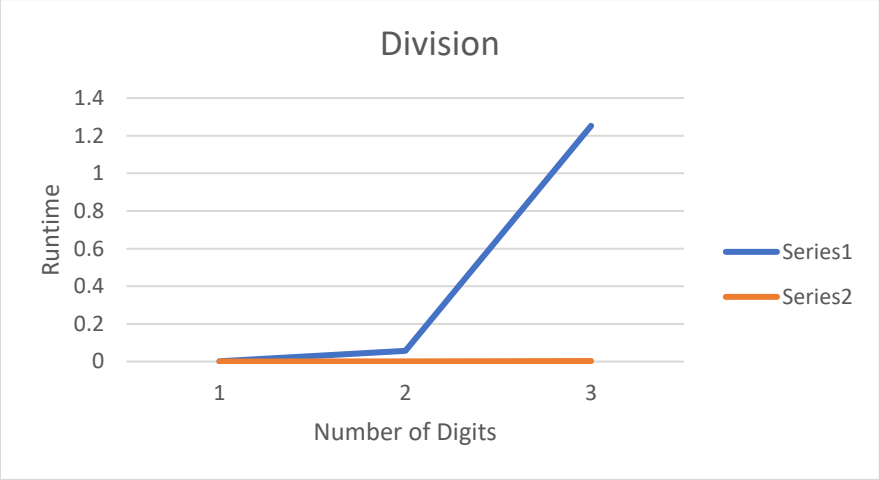
NOTE: The theoretical values were not plotted due to the values being too large to have an accurate scale on the graph. This could be due to the large values of n chosen, where the experimental values will diverge from the theoretical. **Series 1** (blue) is my HugeInteger class, and **series 2** (orange) is the BigInteger class.


Compare
1.40E-04
1.20E-04
1.00E-04
8.00E-05
6.00E-05
4.00E-05
2.00E-05
0.00E+00
Runtime
1
2
3
Number of Digits
Series1
Series2


Addition
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
Runtime
3
100
500
Number of Digits
Series1
Series2


Subtraction
0.08
0.06
0.04
0.02
0
RunTime
3
100
500
Number of Digits
Series1
Series2





5.0 Discussion of Results and Comparison

From my calculations, my theoretical results do not match my experimental. This could be due to the fact that they have a different leading constants. My theoretical constants could be larger than what experimentally results. The experimental results make sense because they match the big-Theta running time complexities that were determined at the beginning of the report.

My HugeInteger class was also found to be less efficient compared to the BigInteger class (as expected), for every method. You can see from the experimental values that my code takes longer to run compared to the BigInteger class.

Given extra time, I would go back and try to simplify my code by either removing unnecessary or repetitive lines of code. I would also try to research into implementing better algorithms for my divide and multiply methods because I feel they were the most complicated and inefficient. I would also try to implement my code using strings, or some other data structure that takes up less memory.

6.0 Sources:

Michael Skells

Nathan Best

All other source code was conceived by me.