2SI4: Digital Signals

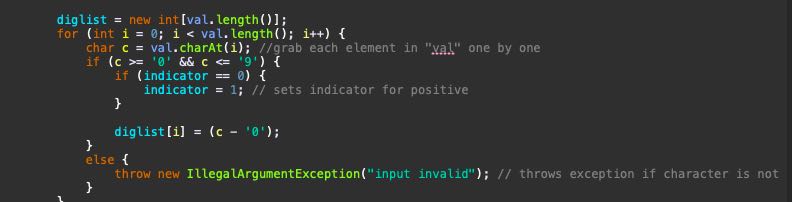
Lab #1

Instructor:

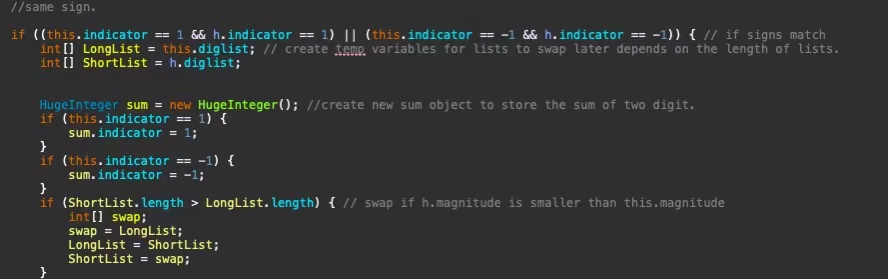
Tianze Zhang L01 – zhant22

As a future member of the engineering profession, the student is responsible for performing he required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by [Tianze Zhang, zhant22, 400208135]

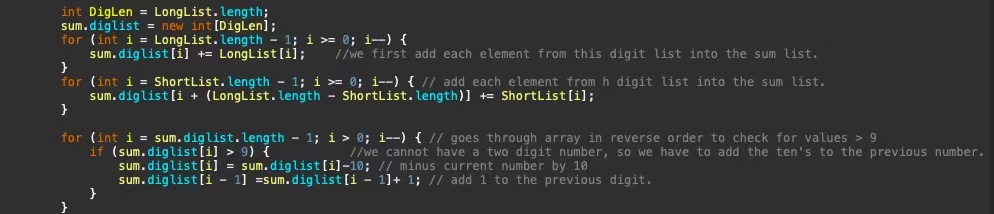
# Description of Data Structures and Algorithms

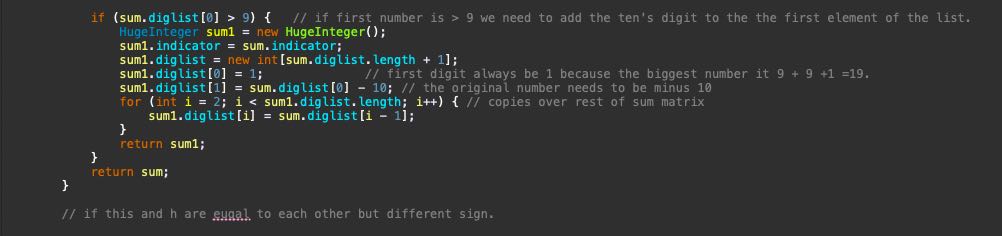
The data structure I used to implement the HugeInteger class and how you implemented each operations is arrays. For fundamental idea of “HugeInteger” is covert the integer type data into string type data and store them into the array. In order to use that, we first let user pass the string type data called Value. 

For addition Implementation, we first check if the sign of the value user input, if the sign is same, we find the array which has longer length using “.Length” method, and set this length as the length of the sum array we just create. Then we swap the temporary holder for the array, in this case, we always calculate the longer array as the base, no matter which array is longer.



After determine which one is the longer array, we starts to add up each element of the array backward. The reason we need to add from back is due to carry may occur when adding up two numbers.

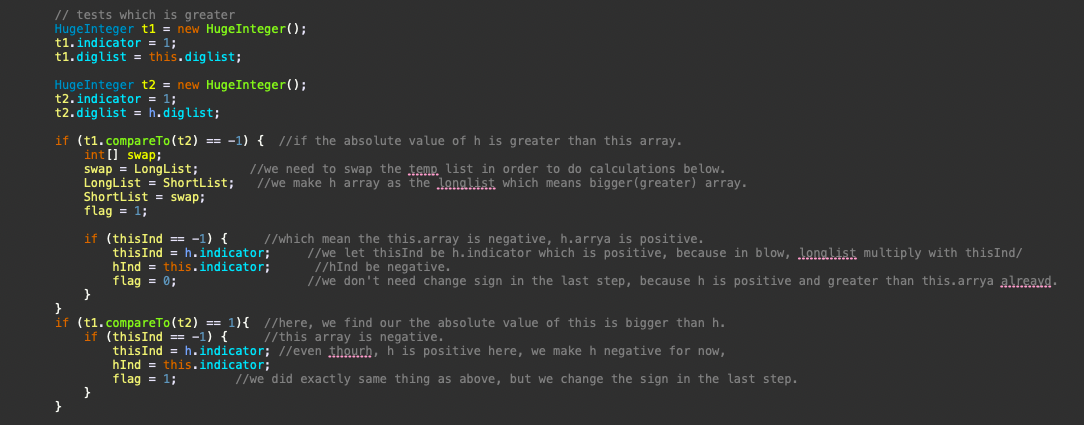
If there is a carry here, we minus 10 in the current number and add 1 into the previous position number in the sum array. Because the biggest number we can get is 9 + 9 +1 =19, this method will work. 

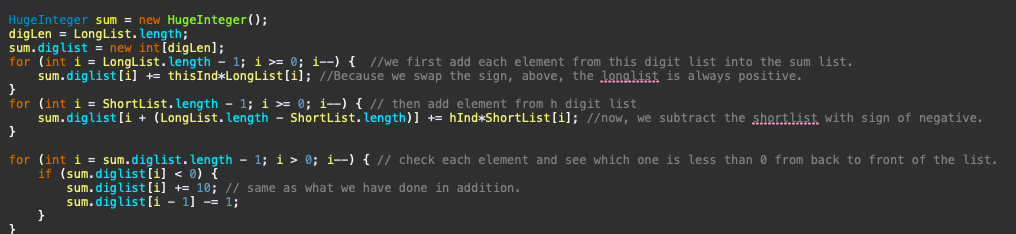
After we add up all the digit in our two array, we need to check if the first digit is bigger than 10, if that’s the case, we increase the length of our sum array by 1, and mins 10 in the first number, put a “1” as the very first digit of our sum array. After that, we can retune the sum array. 

Next, we have a special case: the two array have opposite sign but same anything else, in this case, we return zero. 

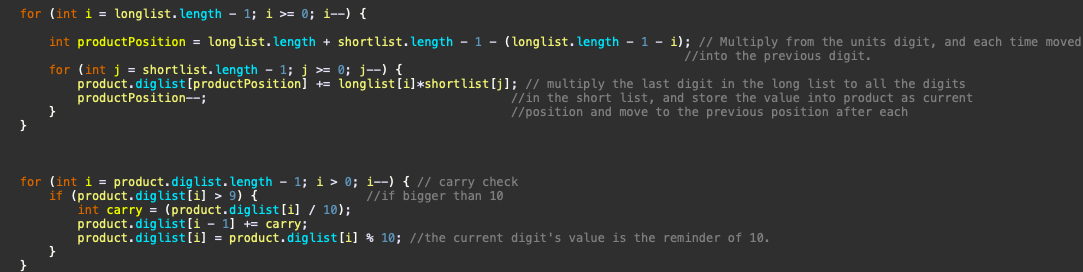
For comparison, we first compare the sign of two array, then we compare the length of each array. If the length of two array is identity, then we compare each element in the array. 

For subtraction, we first covert the sign of our two array in to positive, the compare them using the compare method I mentioned above, in this way, we find which array has bigger absolute value. if array h is greater, we make it as the longlist. Then, if the sign of this array is negative, we make it positive, and make h array negative, namely, we swap the sign. If array this is greater, and negative, we do all the process same as above, but we make flag = 1, which means we are going to change the sign in the last step.



After that, we start to adding each element in both array into our new sum array backward. In the last, we check if there is any digit is less than 0, if there is, we add 10 in the current digit, and -1 in the previous digit. 

For the multiplication method, we first check if any of the two array is equals to zero, if that’s the case, we return zero.

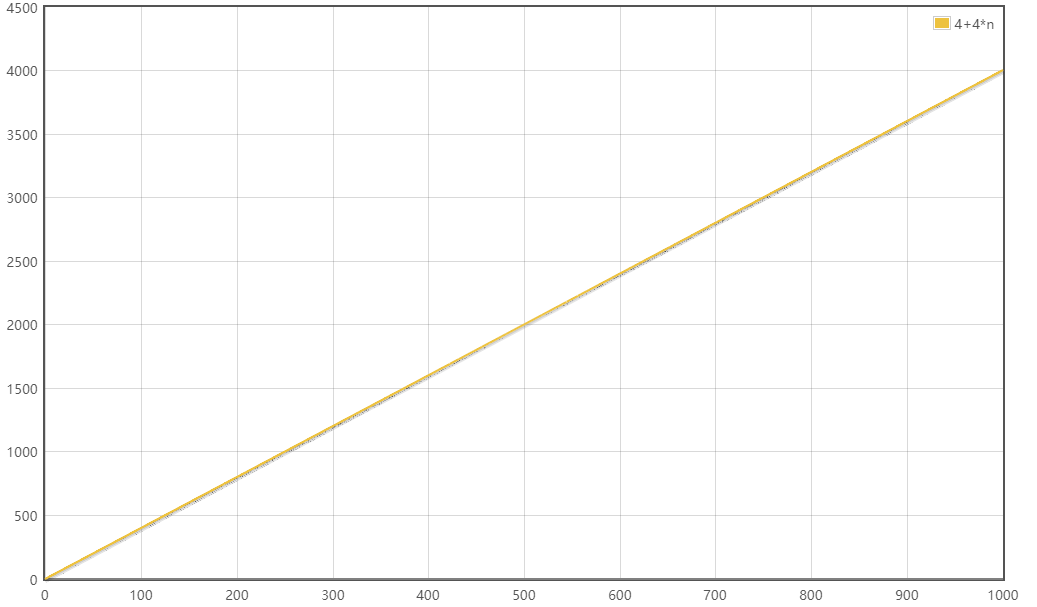
We create new object called product to store the answer, then we compare the length of two array, put the longer array into integer array LongList. Then we start to multiply each element in two list using for loop. 

After that, we eliminate problem such as single digits bigger than zero, leading zero, then we can safely return the product.

# Theoretical Analysis of Running Time and Memory Requirement

In order to using my HugeInteger class, we need (4+4\*n) amount of memory. 4 bytes for the sign and 4 bytes for each integer. Therefore, if we have n amount of integer, the memory required is 4\*n.

Memory requirement



x-axis : number of digits in n

y-axis : memory requirement in bytes

The running time for adding and subtracting are both theta(n) = n, as subtracting is just adding two array with different sign at the front. For adding and subtracting, the operation take constant time, so the running time is T(n)=1, for “for loop”, the running complexity is theta(n)=n, as it always relative to the length of the array which is n. therefore, the total running time is T(n)=C1n+C2, the running complexity is theta(n)=n in both average and worst case for both adding and subtracting function. Because we are adding two array together, the extra bytes needed for the return is 2\*(4+4n).

The running for multiplication operation is theta(n) = n^2, because there is nested for loop in the function, each for loop has running time of T(n)=n, with nested loop, the total running time is T(n)=c1n^2+c2n+c3. Running complexity is theta(n)=n^2 in both average and worst case. Due to the nested for loop, the memory required for the return of our HugeInteger is 4+4\*(2\*n).

The running time for compareTo function is theta(n)=1 for average case and theta(n)=n fir worst case. For average case, the function give returns after constant operations; T(n)=c, complexity theta(n)=1. For worst, the function need a for loop to compare every single element in both array; T(n)=c1n+c2, complexity theta(n)=n. the comparTo function does not return any HugeInteger, so there is no extra memory needed.

# Test Procedure

Test case for constructor, the input test can be as followed:

* zero : 0000
* array of zero : 0000
* positive/negative zeros :+000 / -000
* string with leading zeros :0001234
* positive/negative string :1234/-1234
* string with sign in middle :12345-12345

Test case for adding, the input test can be as followed:

* zero + x : 0+0 / 0+1 / 0+(-1)
* positive + x : 123+0/ 123 + 123 / 123+(-123)
* negative + x: -123+0 / -123+123/ =123+(-123)

Test case for subtraction, the input test can be as followed:

* zero - x : 0-0 / 0-1 / 0-(-1)
* positive - x : 123-0/ 123 - 123 / 123 - (-123)
* negative - x: -123 - 0 / -123 - 123/ =123 - (-123)

Test case for multiplication, the input test can be as followed:

* zero \* x : 0\*0 / 0\*1 / 0\*(-1)
* positive\*- x : 123\*0/ 123 \* 123 / 123 \* (-123)
* negative \* x: -123 \* 0 / -123 \* 123/ =123 \* (-123)

Test case for comparison, the input test can be as followed:

Zero vs X : 0 VS 123 / 0 VS 0 / 0 VS (-123)

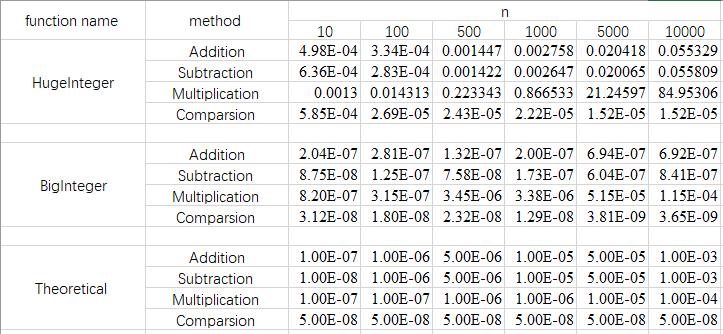
Positive VS X : 123 VS 123 / 123 VS 0 / 123 VS (-123)

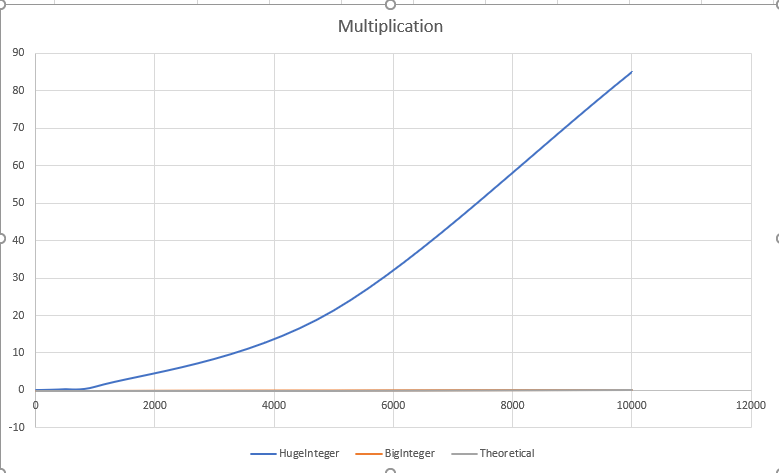
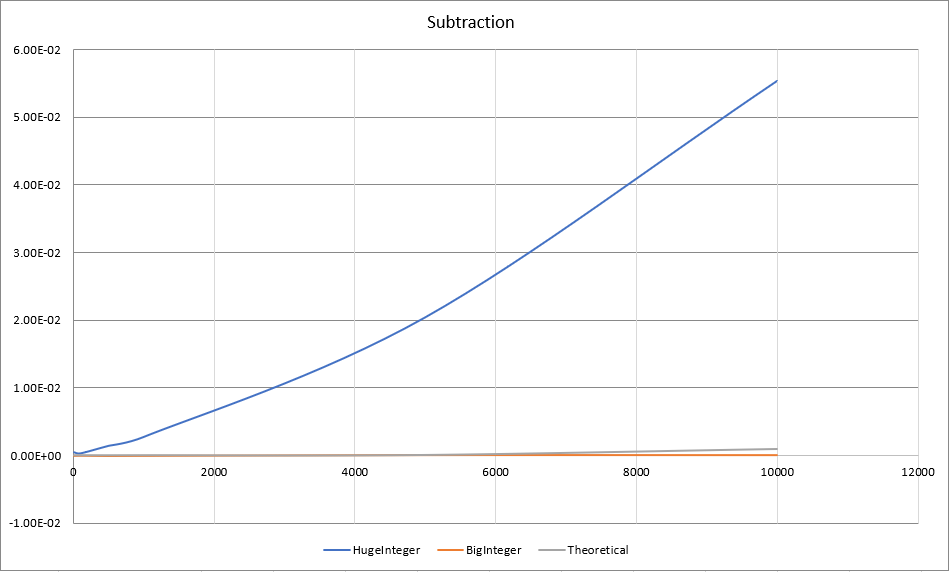
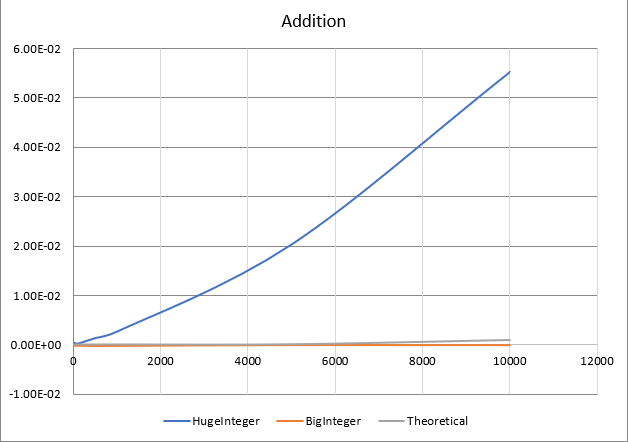
Negative VS X: -123 VS 123 / -123 VS 0 / -123 VS (-123)

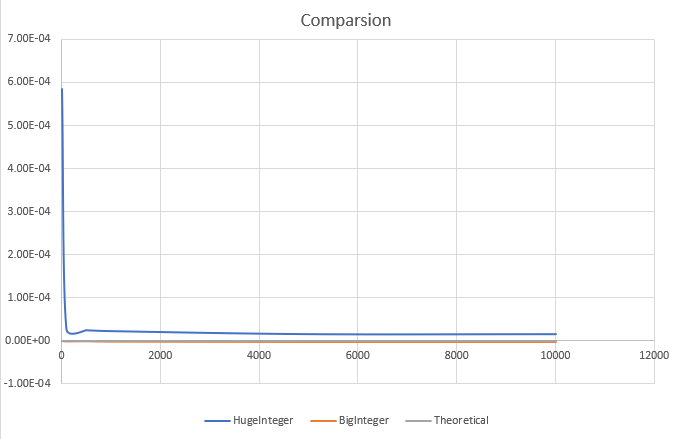
All the test I mentioned above is processed with successful result. I believe I listed all the possible case that HugeInteger class may encounter with. In the process of debugging nothing to special stand out, believe most of bugs have been eliminated during the process of running the testcases from instructor provided on Avenue to Learn.

# Experimental Measurement, Comparison and Discussion

The running time for each operation is tested by the code listed in the lab manual. With N set to be 10, 100, 500, 1000, 5000,10000 respectively for each test run. MAXNUMMINT is set to be 100, MAXRUN for adding, subtraction, comparison is set to be 10000, 100 for multiplication due to much higher level of running time required(T(n)=n^2) .







# Discussion of Results and Comparison

The theoretical calculation correspond to my calculation in a pretty consistent manner. My experiment make sense according to the chart I have here, as we can see here, the HugeInteger calss is significant time consuming than java owned BigInteger class, BigInteger running time matches the theoretical running time in all operations. While for HugeInteger, as the parameter N increasing, the running time increasing in a rapid speed.

One way to increasing the performance of HugeInteger is to implement the correctness of each array as the calculation is process. In my current code, problem like remove leading one, calculate carries are done after the calculation is done, if I can implement those array correctness code with the calculation code at the same time, in this way, the running time and complexity will improve, this change will also reduce the extra memory we need since, correct the array always need a new array in order to store the corrected array.