# Lab 3-4 – Report

2SI4

## Dates:

- Started: 02/03/2015 – Monday 2, 2015
- Halfway: 10/02/2015 – Tuesday 10, 2015
- Finished: 31/03/2015 – Tuesday 31, 2015

## Sources:

- I referred to the lecture slides

## Description of Algorithms:

The data structure I used to implement HugeInteger was with the use of arrays. When the input was a string I had it converted to an array with the same numbers and the first element in the array was always the sign of the number. Using int arrays made the math easier to do in the lab. For this lab there were additional methods that were used such add getSize() and getHugeInt(). These methods were used to make the size of the array and the array itself public.

- Addition:
    - This method was passed 2 huge numbers and the method was supposed to add the 2 numbers and return a huge number of type HugeInteger. First what this method does is it checks to see if the signs are the same. If the signs are not that same that means that the 2 numbers are going to be subtracted. So the method makes the sign the same for both of the huge numbers and calls the subtract method. If both the signs are the same then addition will be done between the numbers. Inorder to do the addition the method first finds the bigger and the smaller of the 2 numbers. This is done using the compare method. After is knows which is the bigger number of the 2 and which is the smaller it stores the arrays with the appropriate labels. Now to add the 2 numbers what this method does is it uses the variable temp to store the addition of each individual digit in both the numbers with a carry if there is one and gets the remainder divided by 10. So this way temp is only a single digit number. It then stores temp in sum which is a string created to old the final answer. After this is calculates the carry, which is the addition of the digits from both numbers and the carry divided by 10. Since temp2, which stores the carry, is of type int, the carry will be a whole number. This is done until one of the arrays has run out of numbers. After that it adds the remaining numbers in the bigger array with the carry appropriately. After all this is done and there still is a carry then the carry is added to the beginning of sum. After the method has the sum it creates a new

HugeInteger with this sum and checks to see if what sign should be in the beginning of the array. If both numbers had positives then it passes a positive otherwise it passes a negative. After all this the method returns the new HugeInteger.

- Adding 129 + 18
  - temp2 = 0
  - temp = (9+8+0)%10 = 7
  - sum = "7"
  - temp2 = (9+8+0)/10 = 1
  - …
  - return 147

- Subtraction:
  - This method was passed 2 huge numbers and the method was supposed to subtract the 2 numbers and return a huge number of type HugeInteger. This method is very similar to add. It starts of the same way. It starts by checking to see if the sign of both the numbers is the same. If it's not the same then it makes the signs the same and it calls the add method and the numbers are going to be added together. If the signs are the same then this method compares the 2 numbers calling the compare method and appropriately stores the number in the big array and the small array. If both the numbers are the same then this method just returns a new HugeInteger with just a 0 because that would be the difference. After this method gets the bigger and the smaller array it uses the variable temp. temp stores the subtraction between each digit in both the arrays. If temp is below 0 then the method subtracts one from the number to the left of the current number in bigger array and adds 10 to the current number in big array and then once again subtracts the numbers. temp is then stored in diff, which just like sum is a string that stores the difference of the 2 numbers. This is done until one array has run out of numbers then diff just stores the remaining values in the bigger array. After the method has a string representation of the difference of the 2 numbers it creates a new HugeInteger using diff. After all this, the method checks to see what the sign of this new HugeInteger should be. If both numbers were positive and this array was bigger, the answer would be a positive. If this array was smaller, the answer would be negative. If both signs were negative then the opposite would occur.
  - Subtracting 1122 – 33
    - temp = 2-3 = -1 < 0
    - 1113 – 33
    - temp = 3-3 = 0
    - diff = "0"
    - …
    - return 1089
- Multiplication:
  - This method was passed 2 huge numbers and the method was supposed to multiply the 2 numbers and return a huge number of type HugeInteger. This method starts be

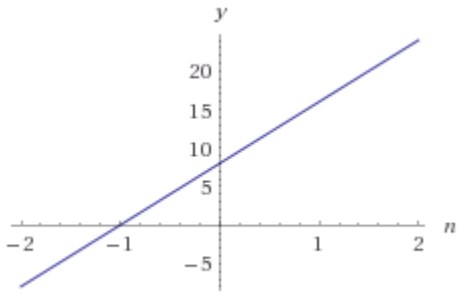
making a product HugeInteger and just setting it to 0. Then it checks to see which number is bigger and stores the numbers appropriately in the correct labeled arrays. Then the method uses 2 for loops. The first one goes through the smaller array and the second one goes through the bigger array. Basically the loop takes a number from the smaller array and multiplies it by every number in the bigger array. The multiplication follows the same process to store the number as add and subtract. So it uses holdnum to store the product of one number in small array with a number in big array and that product added with the carry is stored in tempstrnum, which is a string representation of the final answer. It then calculates the new carry the same way add did. After the second loop is done going through each number in the array and there is still a carry then the carry is added to tempstrnum. After the carry has been added the method creates a new HugeInteger with tempstrnum and adds it to the product. It then sets all the value in tempstrnum to 0 and moves to the next number in the smaller array and does everything over again. After the product has been calculated the method then checks to see what the sign should be. It checks to see if both the arrays have the same sign, if so then the final product is going to have a positive otherwise it will have a negative.

- Multiply 789 * 23
  - temp2 = 0
  - holdnum = 3*9 = 27
  - tempstrnum = ((27%10) + 0) + "" = "7"
  - temp2 = 27/10 = 2
  - holdnum = 3*8 = 24
  - tempstrnum = ((24%10) + 2) + "7" = "67"
  - temp2 = 24/10 = 2
  - …
  - return 18147

- Compare:
  - This method was passed 2 huge numbers and the method was supposed to compare the 2 numbers and return an integer stating the comparison of the 2 number. This method first compares the sizes of both the arrays. If this array has a bigger size that means there are more number and therefore it is bigger and vice versa with that array. If both the sizes are the same then this method compares each value in the array and returns the appropriate value, 1 if this <h, -1 if this>h, and 0 if this=h.
    - Compare 111 with 121
    - this = {+111} , h = {+121}
    - size of this and that are same
    - compare {+} with {+}
    - compare {1} with {1}
    - compare {1} with {2}
    - return 1

## Theoretical Analysis of Run Time and Memory Requirement:

HugeInteger uses 2 private variables one of which is an int array and the other is just an int. So therefore the memory used is (int(bits) + n*int(bits)) = (32 + n*32)/4.




- Addition:
  - This method has 2 huge numbers, m and n. The best case run time is Θ(1) and the best case is Θ(n). Therefore the average runtime is also Θ(n). The memory usage for this method is Θ = $n^2$.
- Subtraction:
  - This method is very similar to addition. There are 2 numbers passed, n and m. The best case run time is the same as before, Θ(1). The worst case run time is Θ(n), where n is the size of the bigger array. And the average run time is Θ(n). The memory usage is Θ = $n^2$.
- Multiplication:
  - This method just as before had 2 huge numbers passed to it. The best case un time is Θ(1). The worst case run time is Θ($n^2$) and therefore the average run time is Θ($n^2$). The memory usage is Θ = n.
- Compare:
  - This method just as before has 2 huge numbers passed. The best case run time is Θ(1). The worst case run time is Θ(n). Thus the average case run time is Θ(n). The memory usage is Θ = 0.

## Test Procedure:

To test the code there were many different cases. First to test it we checked with both a positive and a negative sign. Also we tested with an empty string to be passed which is supposed to return an error. Then I tested the random number generator. I passed a valid input and an invalid input. With the invalid input there was an error as expected.

To test addition I have 4 test cases. The first one adds 2 positive numbers with a carry at the end of the array. The second test adds 2 negative numbers and the output should be a negative number. Test 3 adds 2 numbers with different signs. The last test adds 0 to a huge number.

To test subtraction I have 7 test cases. The first test case subtracts 2 positive numbers with a negative difference. The second test case subtracts 2 positive numbers with a positive difference. Test case 3

subtracts 2 negative numbers with a positive difference. Test case 4 subtracts 2 negative numbers with a negative difference. Test 5 subtracts 2 numbers with different signs. Test 6 subtracts 0 from a number. Test 7 subtracts 2 numbers that are the same so the difference is going to be 0.

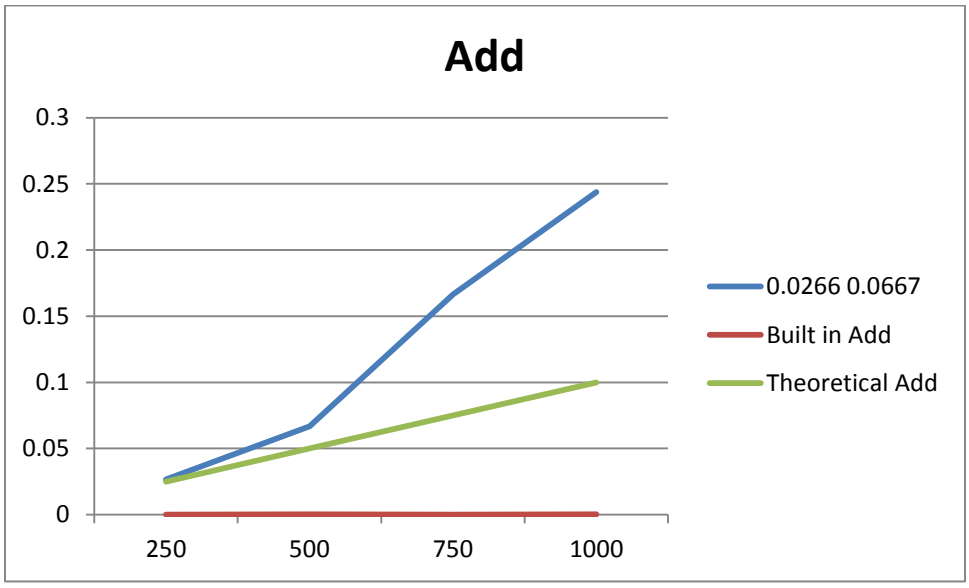
For multiply I have 4 test cases. The first one multiplies 2 positive numbers. The second one multiplies 2 negative numbers. The third one multiplies 2 numbers with different signs. And the last test case multiplies a number with 0.

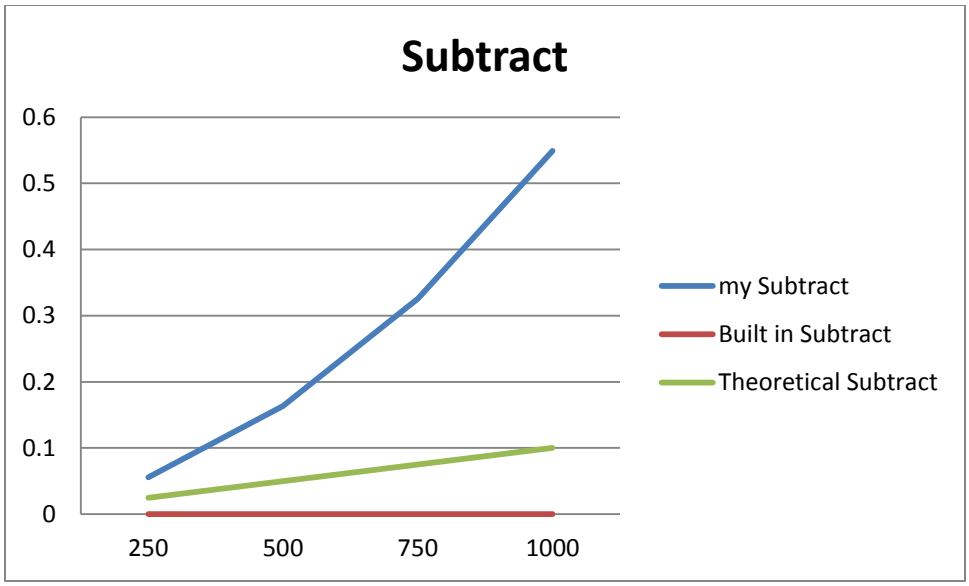
The test for compare uses 3 test cases. The first one compares 2 numbers that are the same. The second one compares 2 numbers where this number is saller than h. The last case tests 2 numbers where this is bigger than h.

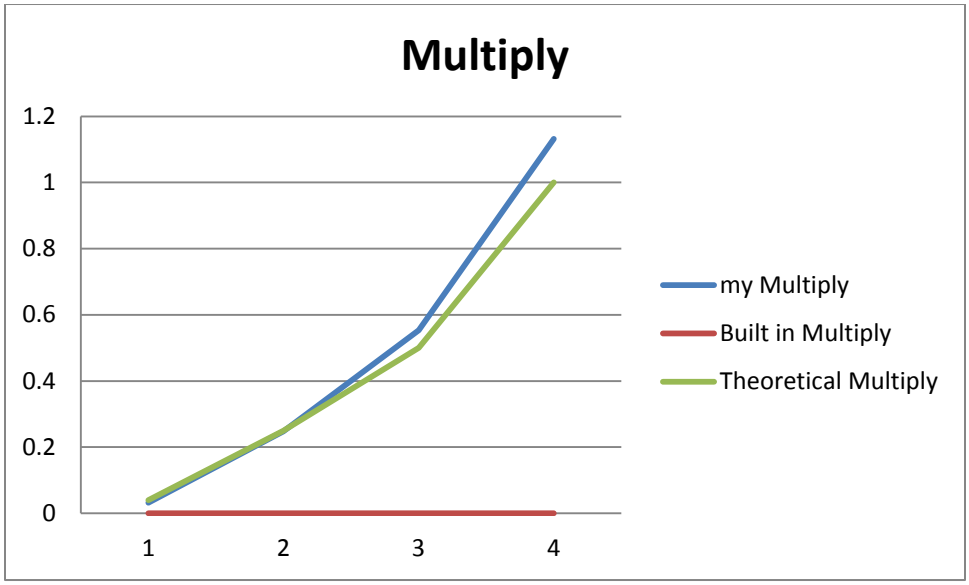
## Experimental Measurement, Comparison and Discussion:

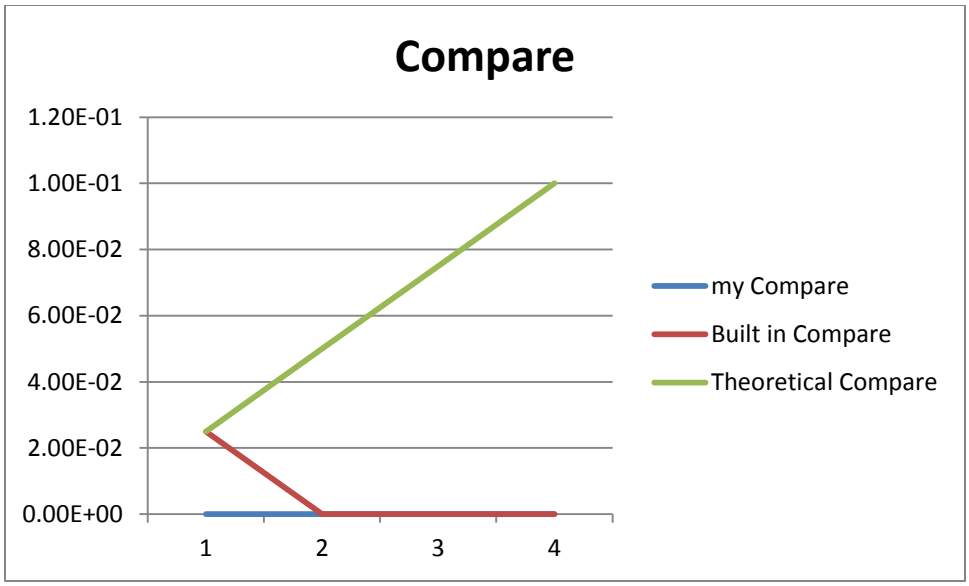
The code to compute the run time was given to us. Values for MAXRUN = 500, MAXNUMINTS = 100.

| Method | 250 | 500 | 750 | 1000 |
|---|---|---|---|---|
| My Add | 0.0266 | 0.0667 | 0.16636 | 0.2438 |
| Built in Add | 0 | 3e-4 | 3e-9 | 3e-4 |
| Theoretical Add | 0.025 | 0.05 | 0.075 | 0.1 |
| My Subtract | 0.05572 | 0.16362 | 0.32568 | 0.54884 |
| Built in Subtract | 0 | 0 | 0 | 0 |
| Theoretical Subtract | 0.025 | 0.05 | 0.075 | 0.1 |
| My Multiply | 0.03192 | 0.2481 | 0.5538 | 1.1322 |
| Built in Multiply | 3e-6 | 3.2e-6 | 3.2e-6 | 3.2e-4 |
| Theoretical Multiply | 0.04 | 0.25 | 0.5 | 1 |
| My Compare | 3.2e-6 | 3.2e-12 | 3.2e-16 | 3.2e-22 |
| Built in Compare | 3.9e-6 | 3.2e-12 | 3.9e-16 | 3.2e-22 |
| Theoretical Compare | 0.025 | 0.05 | 0.075 | 0.1 |




Subtract
0.6
0.5
0.4
0.3
0.2
0.1
0
250
500
750
1000
my Subtract
Built in Subtract
Theoretical Subtract


Multiply
1.2
1
0.8
0.6
0.4
0.2
0
1
2
3
4
my Multiply
Built in Multiply
Theoretical Multiply




## Discussion of Results and Comparison:

My theoretical calculations are very similar to my measured run time. This makes sense. Due to the calculation being almost the exact same as the measured results. My measured results for my methods are similar to that of the ones for the built in function. My codes just run a little slower than the built in ones. This is expected because my code cannot replicate the built in java methods. To make my code faster I would probably try using linked lists rather than just the normal int arrays. Or I would do the whole thing in strings, this way I wouldn't need to go back and forth between string and int. In conclusion my code did everything that was asked of us however it did not do it in the most efficient way.