Aim: To provide hands-on experience with using singly linked list, algorithm design and complexity analysis.

Details: In this programming assignment, you are required to perform ***large positive integer arithmetic operations: addition, multiplication and factorial, using a singly linked list.***

For modern computers, the largest integer number they can operate using a ***64-bit long*** type is:
9,223,372,036,854,775,807 - total 19 digits.

To handle larger numbers of arbitrary digits, we need to use Java's BigInteger class. BigInteger class uses an array as the internal data store. In this assignment, you are ***NOT*** allowed to use the BigInteger class nor any Java library, but to design your own algorithms and implement programming code to perform large positive integer arithmetic operations. And we will use singly linked list as the data store.

Examples

Addition:

3141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128475648233786783165271201909145648566923460348610454326482133936072602491412737245870066063155881748815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861173819326117931051185480744623799627495673518857527248912279381830119491 + 2718281828459045235360287471352662497757247093699959574966967627724076630353547594571382178525166427427466391932003059921817413596629043572900334295260595630738132328627943490763233829880753195251019011573834187930702154089149934884167509244761460668082264800168477411853742345442437107539077744992069551702761838606261331384583000752044933826560297606737113200709328709127443747047230696977209310141692836819025515108657463772111252389784425056953696770785449969967946864454905987931636889230098793 = 5859874482048838473822930854632165381954416493075065395941912220031893036639756593199417003867283495409614478445285366568911258206179625804625693703389076748188416431329882011868793474503702150181400976002645163596635600217625583117954292410032662577227913367091937760464196672090650501146337994133343276289768444921849506266103921714874187918275661974629703560720659239676264213568614843929150821751125894089391274700610555958228634322396218162072244845247829932717514216340658712822864827413110742

Multiplication:

3141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128475648233786783165271201909145648566923460348610454326482133936072602491412737245870066063155881748815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861173819326117931051185480744623799627495673518857527248912279381830119491 × 2718281828459045235360287471352662497757247093699959574966967627724076630353547594571382178525166427427466391932003059921817413596629043572900334295260595630738132328627943490763233829880753195251019011573834187930702154089149934884167509244761460668082264800168477411853742345442437107539077744992069551702761838606261331384583000752044933826560297606737113200709328709127443747047230696977209310141692836819025515108657463772111252389784425056953696770785449969967946864454905987931636889230098793 = 8539734222673567065463550869546574495034888535765114961879601130179228611157330807572563869710473943913774942511677467646321187590696023990618363453790704145420215994889633428527467000466877660930727112903935074804010557270403486273039986565406441661792292857137082163744129761684711725446723184203407516578730205067079994720762989679643737139009008398707852206330482980353846401731530019782362767702580357412559720551726398986173449590926124122896807645827854205431632157954195102617533261393270911938173189473257740456381155151842163457273203266961962968100477774056264513586245506579374650896392308628024869246347076767703130508428699685203041224786126335755248791258397060829962955411742641036052642779108003964415591177796028771999650128921313207851422859374331722669282978008091376770803386816691949151867168671608691578494423861761530548270671517300325179795264912813107799495542731427446300956525607946484154600181256827723124910158877299179427239566728952250408905276832679976323569477557

Factorial (100! = 100 × 99 × 98 × … × 3 × 2 × 1):

=

9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000000000000000000000

Tips: It is a lot more efficient to handle large integer if we store the digits in the singly linked list in a reverse order, see below. To begin with, we use small numbers as test bed.

Input:

first integer 98765 : [9]<-[8]<-[7]<-[6]<-[5]<-[head]
second integer 123 : [1]<-[2]<-[3]<-[head]
factorial 7

Output:

Addition:
large integer 98888 : [9]<-[8]<-[8]<-[8]<-[8]<-[head]
Multiplication:
large integer 12148095 : [1]<-[2]<-[1]<-[4]<-[8]<-[0]<-[9]<-[5]<-[head]
Factorial:
large integer 5040 : [5]<-[0]<-[4]<-[0]<-[head]

Assignment Details (Total 100 marks):

Part 1) Programming [70 marks]

You are given the following java files.

LargeIntegerDemo.java	- the driver program for reading data and performing: addition, multiplication and factorial.
LargeIntegerNode.java	- the node of a linked list that stores 1 digit.
LargeIntegerLL.java	- the linked list that provides the functionality of multiplication and printing.

Your task is to modify ***LargeIntegerLL.java***, and to fill in the content of the following functions.

`public LargeIntegerLL add(LargeIntegerLL secondNumber)` [30 marks]
This function `add` computes the addition of two large integers `(this, secondNumber)` and returns the result in a new linked list `(result)`.

`public LargeIntegerLL mul(int digit)` [15 marks]
Given a digit `(digit)`, the function `mul` computes the multiplication of the large integer and the digit, and returns the result in a new linked list `(result)`.

`public LargeIntegerLL mul(LargeIntegerLL secondNumber)` [15 marks]
This function `mul` computes the multiplication of two large integers `(this, secondNumber)` and return the results in a new linked list `(result)`.

`public static LargeIntegerLL factorial(int number)` [10 marks]
This function `factorial` computes the factorial of a given number `(number)` and returns the result in a new linked list `(result)`. You can assume that the given number **K** is **≤ 2000**.

Note:
Tip1: you are **not** allowed to modify any other files.

Tip2: You should prepare and test your program to work. Use the provided test case, example:

```
> javac *.java
> java –cp . LargeIntegerDemo < input.txt
First Number: …
…
```

Use "`< input.txt`" to redirect input from input.txt, save you from copy/pasting and making unnecessary errors.

Tip 3: Verify the correctness of your program. For example:

```
> java –cp . LargeIntegerDemo < input.txt > output.txt
> diff output.txt result.txt
>
```

Use "`> output1.txt`" to pipe your result into output.txt.
Use "`diff result.txt output.txt`" to check your program's output. `diff` is a program to find "difference" in two files. If there is nothing returned like above, your result is correct. The marker will use diff to check your output.

Tip 4: The marker will use different test cases to check your program. You should test your program with more test cases.

Part 2) Complexity Analysis [30 marks]

A) Discuss the complexity of the `add` function (your algorithm) in Big-O notation, suppose the two input large integers consists of N digits respectively. [15 marks]
B) Discuss the complexity of the `mul` function in Big-O notation, suppose the two input large integers consists of M and N digits respectively. [10 marks]
C) Discuss the complexity of the `factorial` function in Big-O notation in terms of the ***value*** of input number K (Note: not the number of digits in K). [5 marks]

Tip 5: Use these facts for Part 2C)
- The number of digits in K is on the order of O (log K). For example, if K = 10, there are $O(\log_{10}(10) + 1) = 2$ digits in K.
- The number of digits in K! grows on the order of O (K log K).

Release Date: 23 February, 2016
Due Date: 15 March, 2016, 11:00 (**Zero** marks for **late submission**, in line with College regulation.)

Submission:
You should submit the following to the Blackboard assignment collection page:

1. Part 1: ***ONE*** working java source file, ***clearly commented***: LargeIntegerLinkedList.java
The marker will use the ***submitted*** file and compile against the two ***given*** files :
LargeIntegerNode.java and LargeIntegerAdditionDemo.java
If your program does not work and the marker is unable to understand your code, you will score no marks in Part1.

2. A written report (***pdf only***) for Part2.

Compulsory Oral Viva:
A 5-minute oral viva, to be scheduled on 16 March 2016 (Wed afternoon 1pm-3:30pm), time slot TBA, at Instructor's office.
To speed things up, you are advised to bring your own laptop to demonstrate your working java program.
Each student is given a maximum of 5-minute. Strict time limitation is enforced to ensure no delay to subsequent vivas.

Purpose of the oral viva:
- To show your understanding of the program (you actually did the work),
- To show your understanding on complexity analysis, and
- To provide you early feedback.

If you do not attend the viva, there will be a cap of 60% of your CW marks.

Tip 6: Do ensure you have submitted the correct java source file to blackboard. Check your submission after you have uploaded your file.
Tip 7: How do you achieve full marks? Read the assessment matrix.

Feedback
Marks and sample solution will be provided 2 weeks after the submission deadline.