1. int 32bits -2,147,483,648 to 2,147,483,647  
   long 64bits -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
   1. For the plus I just used the normal pencil procedure since it’s easy to do this with the 2’s components.
   2. For subtract, I changed the sign of the second BigInt and then use the add method.
   3. For multiply, I used the booth algorithm.
   4. For factorial, I use a for-loop to call multiply.
2. I used the booth algorithm for multiply. It's better than paper/pencil version.

# Booth algorithm:

The example of x: 0010 \* y: 1111

x is 2, y is -1

-y: 0001

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| step | Upper | Lower | Multiplicand | Last digit of lower | Pre last digit of lower |  |
| 0 | 0000 | 0010 | 1111 | 0 | 0 | Nothing |
| 1 | 0000 | 0001 | 1111 | 1 | 0 | subtract |
| 1a | 0001 | 0001 | Make right shift | | | |
| 2 | 1000 | 1000 | 1111 | 0 | 1 | add |
| 2a | 0111 | 1000 | Make right shift | | | |
| 3 | 0011 | 1100 | 1111 | 0 | 0 | Nothing |
| 4 | 0001 | **1110** | 1111 | 0 | 0 | Nothing |

**Always make a right shift for upper and lower, the last digit of lower becomes the first digit of upper.**

**How many bits then how many steps.**

**Last digit of lower and Pre last digit of lower:**

00 and 11 do nothing.

01 add upper and y

10 subtract the y form upper.

1110 is -2. It’s the answer.

I think this is much more efficient than the paper/pencil version because it only has adding and shifting. The shifting is really fast since I only need to move the heads and tails.

1. I used the unit test java file to test my code. I meet the overflow bugs for the multiplying method and fix it by double the bits before compute.
2. It’s a good project to practice the 2’s components and binary.

I learnt the booth algorithm, which is very smart way to solve the multiply problem.

It is also a great practice to optimize codes. I tried a lot to make it run faster. Now, it can compute the factorial of 2000 in about 2 seconds