Project 1

The four fundamental operations of arithmetic for large integer.

Data Structure

A class called 'lnumber' was made to package the operations of +, -, *, /. And each operation defined in the class is overloaded from +, -, *, /.

Four constructor function are created in the class for different case when doing calculate. The private attributes of 'lnumber' contain 12 main items as following:

{

char* number_: a char pointer pointing the input char array which store the large number.
int length_: indicate the length of the number. For processing convenience, the length of input number array may be not the valid number length. For example, '123' may be stored as '1230' where the length_ is set as 4.
int zero_bit_: operation between two number may produce the result like '00123' where the '00' is not desired when outputting the result or make it as participant for another operation. So zero_bit_ indicate the length of unnecessary zeros and number_ will be set as number_ + zero_bit_.
bool out_symbol_: indicate the positive\negative of number.
short pattern_: the operation - is designed to utilize operation +, so there are different cases for operation + like positive number plus positive number, positive number plus negative number (same as -). Pattern indicates the cases.
char* quotient\remainder_: when doing dividing, they store the quotient and remainder respectively.
int length_q\length_r: indicate the length of q\r respectively.
int e_length_: cause FFT is utilized when doing multiple operation, so each number should be complemented with zero to meet the acquisition of FFT. e_length_ indicate the length after complementation.
char* input_: cause number_ may be set as number_ + zero_bit_, when free the array, delete[] number_ doesn't work anymore, so input_ is created to constantly point the input array and can do delete[] input_ for freeing memory instead.

}

Algorithm Design Procedure

I simulate the human computation process when design the + and -. The test time for 600K digits +\- 600K digits under release version are 3 ms.

For * and /, I choose start from a simple case where I make * directly call the + to do the multiple and make / call the -. The complexity of both are O(N^2) and under the release version doing 600K digits multiple 600K digits will cost me more than a half hour.

Cause the it is unacceptable time consuming, I tried FFT to do *. By utilizing FFT, the multiple operation can be seen as polynomial convolution and can be transferred into frequency domain with Fourier Transfer and simply do element wise dot production then doing reverse Fourier Transfer. The complexity can be reduced to Nlog(N) technically.

My 1st version FFT code is attached in the class 'lnumber' as a friend function. By utilizing it, operation of 600K * 600K only cost 7 seconds under release version. By analyzing the code with the 'performance and diagnostics' in visual studio 2013, I found many time waste on allocating new space for array. So I did some optimization in my 2st version of FFT which make several array utilize same space in RAM and the test time reduce to 3.9 seconds.

Still, Because of architecture of my FFT, I have to allocate new space for each recurrence and 70% of time lying on it. So my final performance for 600K digits * 600K digits is around 3.9 seconds.

Because it's more complicated to utilize FFT for /, I choose to remain it as the origin version which doing some trick based on naïve human computing method. It will cost 30min plus to do 600K digits / 300K digits.

Project 2

prime

To find a prime, I apply a quick version of seize method to do it. Seize method is well known and it's not necessary to describe it. The key point in quick version of seize method is that it eliminates repeating problem to some extent. For example, if 15 is seized by 3, it won't be seized by 5 again. Also, for space saving, only odd number are stored in the array. The test time for find the 100000000 st prime is around 17 seconds.