CCPS616 Lab 2 – Dividing and Conquering Multiplication

Preamble

In this lab you will write a C++ program that performs large integer multiplication. Your implementation must be from the ground up without linking any third-party libraries. Your lab should be completed in a single cpp file called **mult.cpp**, and should compile at the command line very simply as follows:

```
g++ -o lab2 mult.cpp lab2.cpp
```

Lab Description

In this lab you will implement several algorithms for performing large integer multiplication.

First, implement the so called "Schoolbook" $O(n^2)$ method that goes digit by digit. Your function should be called `mult4()`, and it will accept two strings (**`std::string`**) of decimal digits as input. Keep in mind that the whole point of implementing an algorithm like this is for use on integers too large to be represented by existing integer data types (pretend Python's integers don't exist). Thus, we use character arrays (or C++ strings) to represent our large integers precisely.

Second, implement Karatsuba's algorithm that performs multiplication in $O(n^{1.585})$. Call this function `mult3a()`. Its input arguments should take the same form as `mult4()`. This implementation should recurse all the way down to n = 1.

Third, implement a modified version of Karatsuba's algorithm that recurses only *so long as the result of a multiplication would overflow an unsigned 64-bit integer* (**`unsigned long long int`** in C/C++). This should be a very simple modification to the version above. Call this function `mult3b()`. ***Hint:*** *Base the overflow decision on the number of digits in the operands*.

In all cases, you may assume the input integer strings are positive. You may also assume that the length of each string is a power of two (512, 1024, 2048, etc.). This is not required by the algorithm, but it does make certain implementation details easier.

Testing

To test your implementations, the tester provided will randomly generate strings of 2^k digits. Sizes include {512, 1024, 2048, 4096}. Much like Lab #1, do not modify the tester. Your code should be placed in mult.cpp.

The tester will validate the correctness of your multiplication functions by ensuring that for the same input, mult4(), mult3a(), and mult3b() all return the exact same result. If there's a discrepancy, it means something is going wrong in at least one of your functions.

Results

The tester will write out results to the terminal (stdout) in a clean, easy to read format. A sample output for my own model solution is below:

```
CCPS616 - Lab 2 - Alex Ufkes

Sanity check #1: Example from the slides
  3563474256143563 * 8976558458718976 = 31987734976412811376690928351488
  mult4...  PASSED!
  mult3a... PASSED!
  mult3b... PASSED!

Sanity check #2: Small strings tested against primitive *
  m4  passed 100000/100000 (100%)
  m3a passed 100000/100000 (100%)
  m3b passed 100000/100000 (100%)

mult4()    512 digits: .......... Avg: 12ms
mult3a()   512 digits: .......... Avg: 38ms
mult3b()   512 digits: .......... Avg: 6ms
           512 digits: PASSED
mult4()   1024 digits: .......... Avg: 50ms
mult3a()  1024 digits: .......... Avg: 142ms
mult3b()  1024 digits: .......... Avg: 22ms
          1024 digits: PASSED
mult4()   2048 digits: .......... Avg: 178ms
mult3a()  2048 digits: .......... Avg: 328ms
mult3b()  2048 digits: .......... Avg: 64ms
          2048 digits: PASSED
mult4()   4096 digits: .......... Avg: 689ms
mult3a()  4096 digits: .......... Avg: 972ms
mult3b()  4096 digits: .......... Avg: 188ms
          4096 digits: PASSED
```

Notes

You will almost certainly notice that mult3a() performs much worse than mult4() until **n** gets *very* large. When you implement mult3b() correctly, however, you should start to see it pull ahead of the schoolbook mult4() when **n** exceeds 256 digits (perhaps 512 or 1024, depending on how good your implementation is).

Do not despair if you are unable to achieve the expected relative efficiencies of these three algorithms. Karatsuba's algorithm can be tricky to implement, and even trickier to optimize. Your lab will not be marked on this basis. You will be marked on the correctness of your implementation, not its running time.

Submission

Submit your source file (**mult.cpp**) on D2L.

This lab may be submitted individually, or in groups of up to **two**. If you submit as a group, be sure to include both names in the D2L submission as well as your source code.

"You do one half of the assignment I'll do the other half and then we'll join them together"

(If you're working in partners, don't do this)