CMSC 441 Project 1: Karatsuba's Method C++ Implementation

Abstract:

I will describe the theoretical time complexity of Karatsuba's method, versus that of traditional 'schoolbook' multiplication. I will discuss the theoretical 'breakpoint' between efficiency of schoolbook and karatsuba efficiency. I will then discuss my implementation and the empirical results.

Background:

In America (and I'm sure many other countries), we learn multiplication at a fairly young age. Beyond memorizing simple tables of single digit multiplication and simple rules about multiplying by 10 (base), we learn the FOIL method, or simply the schoolbook mode of multiplication. This involves taking a multiplication of two complex numbers and breaking it into (4) smaller chunks which can be more easily multiplied, and then summed to find the final product. Karatsuba's method breaks the larger multiplication into 3 multiplications, 4 additions, two subtractions and two shifts. Since multiplications are certainly more costly than additions, at a certain point (digit size) Karatsuba's method becomes the more efficient method.

Algorithm Description:

The algorithm I used for finding the product uses the following variables:

Base β
m digits
For finding the product of two integers *a* and *b*,
A_1 = the more significant half of *a*
A_0 = the least significant half of *a*
B_1 = the more significant half of *b*
B_0 = the least significant half of *b*
$z_1 = A_1 * B_1$
$z_3 = A_0 * B_0$
$z_2 = (A_1 + A_0) * (B_1 + B_0) - z_1 - z_3$
Product p = $z_1\beta^{2m} + z_2\beta + z_3$

My Implementation:

First I began with some checks (the base case)
KARATSUBA(*A*, *B*, *n*):

- If *A* and *B* are not the same length, pad the lesser with zeroes to match the digit length, *n*

- Split *A* into two arrays *A1 and A0*, do the same to *B*
- Call KARATSUBA(*A1, B1, n*/2), setting the result equal to temporary variable *z1*
- Call KARATSUBA(*A0*, *B0*, *n*/2), setting the result equal to temporary variable *z3*
- Calculate and store the sums of (*A0* + *A1*) and (*B0* + *B1*) separately.
- Call KARATSUBA on the sums with *n*/2 and store the result in temporary variable *z2*
- Subtract *z1* and *z3* from *z2*
- Finally, apply appropriate shifts and return the combined result

Discussion of Running Time:

The ‘traditional’, or ‘pen-and-paper’ algorithm involves three recursive calls, four additions, two subtractions and two shifts. The additions/subtractions can be considered constant time operations. The recurrence relation then becomes $3T(n/2) + 6$. By the master theorem, we see that the running time is $O(n^{lg3})$. In my version however, I use a for loop that iterates to do the shifts, causing an *n* time operation. This for loop also however, replaces two of the additions. So the recurrence relation is ultimately $3T(n/2) + 5 + n$. By the master theorem again, we see that:

$a = 3$, $b = 2$, $f(n) = 5 + n$.

This is case 2:

If $\backepsilon$ = ~.58

$5 + n = \Theta(n^{lg3 - \backepsilon})$,

So therefore the running time is $\Theta(n^{lg3} \lg n)$.

This is unfortunately worse than the ‘pen-and-paper’ version of Karatsuba’s method, but is still asymptotically better than the $O(n^2)$ run-time of the ‘schoolbook’ method.

Theoretical Results:

Based on the expected run-time the ‘breaking point’ between where karatsuba’s method requires less operations is about $n > 91$.

$90^2 = 8100$

90^{lg3} * lg90 ~ 8124

$91^2 = 8281$

91^{lg3} * lg90 ~ 8288

$92^2 = 8464$

90^{lg3} * lg90 ~ 8453

$93^2 = 8649$

90^{lg3} * lg90 ~ 8620

Empirical Results:
I am not sure why my results are a few orders of magnitude off from the expected result. If I had to venture a guess, I would say memory management.

Conclusion:
It is clear that the Karatsuba's method is more efficient for numbers of a certain size. My algorithm proved to be faster than that of the traditional method.