CMSC 351: Integer Multiplication

Justin Wyss-Gallifent

March 29, 2022

1 Introduction

Suppose we have two n-digit numbers and wish to multiply them. What is the worst-case time complexity of this operation?

2 Schoolbook Multiplication

2.1 Method

The first and most obvious way to multiply two numbers is the way we learn in school. Here is an example. The carry digits are not shown:

		1	0	2
		2	5	7
		7	1	4
	5	1	0	
2	0	4		
2	6	2	1	4

2.2 Time Complexity

What is the time complexity of this algorithm? Well row i in the intermediate calculation equals digit b_i multiplied by each digit in A, (with a possible carry addition) so row i requires n operations. Since there are n rows there are n^2 digit multiplications (each with a possible carry addition). Without even worrying about the additions we're at $\Theta(n^2)$.

Could we do any better?

3 A Sneaky Approach

3.1 Two 2-Digit Numbers

Let $A = a_1a_0$ and $B = b_1b_0$ be the base-10 digit representations of two 2-digit numbers. In reality then $A = 10a_1 + a_0$ and $B = 10b_1 + b_0$ and the product is:

$$AB = (10a_1 + a_0)(10b_1 + b_0) = 100a_1b_1 + 10(a_1b_0 + a_0b_1) + a_0b_0$$

For now let's ignore the 100 and the 10 and consider that we have four multiplications. Can we do better?

Observe that:

$$a_1b_0 + a_0b_1 = (a_1 + a_0)(b_1 + b_0) - a_0b_0 - a_1b_1$$

So consequently the middle expression can actually be calculated using two multiplications that we've already done as well as one new one. In summary:

$$AB = 100a_1b_1 + 10\left[(a_1 + a_0)(b_1 + b_0) - a_0b_0 - a_1b_1\right] + a_0b_0$$

Of course you may observe that the newly required product $(a_1+a_0)(b_1+b_0)$ may itself be the product of two 2-digit numbers but for now let's just be satisfied that they're certainly smaller than the original two 2-digit numbers.

In addition in order to guarantee that we have actually obtained the digits of the product AB we will actually need to perform the multiplications by 100 and 10 and the resulting additions.

However multiplication by a 100 can be done using two decimal shifts (insert two zeros) and multiplication by 10 can be done using one decimal shift (insert one zero) and shifting by n digits is $\Theta(n)$ (insert n zeros).

Thus in total to calculate all of the required digits precisely we have:

- A total of 3 multiplications, a_0b_0, a_1b_1, and $(a_1 + a_0)(b_1 + b_0)$, two of which have half as many digits as the original two numbers and one is a significantly easier product.
- A total of 6 additions/subtractions of numbers with at most 4 digits.
- A total of $2 + 1 = 3$ decimal shifts.

3.2 Two 4-Digit Numbers

Suppose we wanted to calculate a product such as (1234)(5678). Ordinarily this would take 16 single-digit multiplications.

Instead let's write two 4-digit numbers as $A = A_1A_0$ and $B = B_1B_0$ where the A_i and B_i are pairs of digits. In reality then $A = 100A_1+A_0$ and $B = 100B_1+B_0$ and the product is:

$$AB = (100A_1 + A_0)(100B_1 + B_0) = 10000A_1B_1 + 100(A_1B_0 + A_0B_1) + A_0B_0$$

Again observe that:

$$A_1B_0 + A_0B_1 = (A_1 + A_0)(B_1 + B_0) - A_0B_0 - A_1B_1$$

So once again the middle term can be calculated using two multiplications that we've already done as well as one new one, and again the required new product is certainly simpler than the original one. In summary again:

$$AB = 10000A_1B_1 + 100\left[(A_1 + A_0)(B_1 + B_0) - A_0B_0 - A_1B_1\right] + A_0B_0$$

To calculate this out in order to obtain the digits we have:

- A total of 3 multiplications, A_0B_0, A_1B_1, and $(A_1 + A_0)(B_1 + B_0)$, two of which have half as many digits as the original two numbers and one is a significantly easier product.
- A total of 6 additions/subtractions of numbers with at most 8 digits.
- A total of $4 + 2 = 6$ decimal shifts.

Importantly note that the three multiplications can essentially be done by applying the method for two 2-digit numbers.

3.3 Generalized

This approach will then extend to two 8-digit numbers, two 16-digit numbers, and so on. In general if $A = A_1A_0$ and $B = B_1B_0$ where the A_i and B_i are two n-digit numbers where (for simplicity) we'll say n is even then we can write:

$$AB = 10^n(A_1B_1) + 10^{n/2}\left[(A_1 + A_0)(B_1 + B_0) - A_0B_0 - A_1B_1\right] + A_0B_0$$

Then we can reduce finding the digits of AB to:

- A total of 3 multiplications, A_0B_0, A_1B_1, and $(A_1 + A_0)(B_1 + B_0)$, two of which have half as many digits as the original two numbers and one is a significantly easier product.
- A total of 6 additions/subtractions of numbers with at most $2n$ digits.
- A total of $n + n/2$ decimal shifts.

4 Karatsuba Method Theory

This leads to the following generalized observation. Suppose A and B are both n digit numbers. Calculation of the digits of the multiplication AB can be done using three multiplications involving numbers with essentially half as many digits and then $\Theta(n)$ worth of addition and shifts.

Thus if $T(n)$ is the time complexity for multiplying A and B then $T(n)$ satisfies:

$$T(n) = 3T(n/2) + \Theta(n)$$

The Master Theorem with $a = 3$, $b = 2$ and $c = 1$ tells us that since $1 < \log_2 3$ that:

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{\lg 3}) \approx \Theta(n^{1.5849625})$$

This is significantly faster than $\Theta(n^2)$, especially for large n.

For smaller n of course it depends upon the actual specifics of the time requirements.

Note 4.0.1. We're playing fast and loose with powers of 2, half-sizes and so on, but this is just to keep the explanation tidy and avoid fiddly cases and floor and ceiling functions. The result still holds with those details added, it's just far harder to understand.

Note 4.0.2. As we'll see in the actual Python implementation for small numbers the savings (in single-digit multiplications) are practically nonexistent. For large numbers they're very noticeable.

5 Karatsuba Method Implementation

5.1 Recursion Note 1

The implementation is actually sneakier than we might suspect. Recall the original case where $A = a_1a_0$ and $b = b_1b_0$ we noticed that the product $(a_1 + a_0)(b_1 + b_0)$ could potentially be a product of two 2-digit numbers. When applying the Karatsuba method we use recursion even here if necessary to ensure the reduction of everything to single-digit products.

So for example if $A = 76$ and $B = 48$ then we have:

$$(76)(48) = 100(7)(4) + 10\left[(7+6)(4+8) - (7)(4) - (6)(8)\right] + (6)(8)$$

Note the central term $(13)(12)$. We apply recursion again:

$$(13)(12) = 100(1)(1) + 10\left[(1+3)(1+2) - (1)(1) - (3)(2)\right] + (3)(2)$$

Now the central term involves the multiplication of two 1-digit numbers.

5.2 Recursion Note 2

In the case where one of A and B has one digit then even if the other has more than one, at this point it's certainly $\Theta(n)$ to simply multiply. Consequently the actual implementation of Karatsuba's algorithm uses this as the base case in the recursion so we've done this in the pseudocode too.

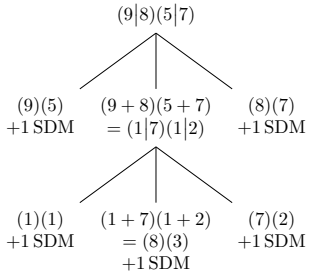
5.3 Splitting Note

Since it's entirely possible that A and B have differing numbers of digits we split based upon the shortest one in order to guarantee that both can, in fact, be split. Additionally we split from the right side (the units digit) to ensure that the decimal shifting (multiplication by powers of 10) works appropriately.

6 Tree Diagrams

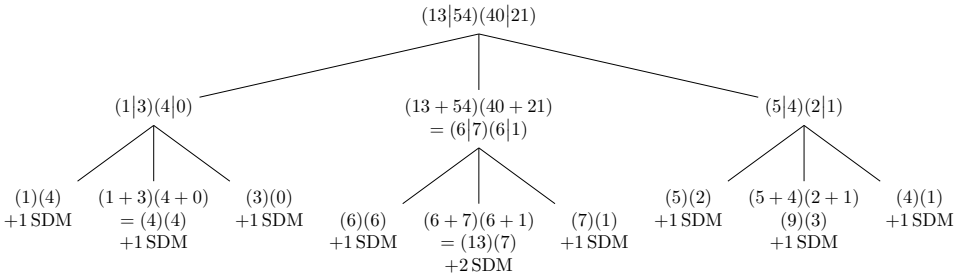
A tree diagram can succinctly show how Karatsuba's Algorithm plays out in terms of the breaking down to single-digit multiplications.

Example 6.1. Consider the product $(98)(57)$. This tree breaks down all the products until one of the numbers is single-digit:



From here we can see that there are 5 single-digit multiplications. This example is actually slower than schoolbook multiplication, which would require $(2)(2) = 4$ single-digit multiplications. ■

Example 6.2. Consider the product $(1354)(4021)$. This tree breaks down all the products until one of the numbers is single-digit:



From here we can see that there are 10 single-digit multiplications. This example is faster than schoolbook multiplication, which would require $(4)(4) = 16$ single-digit multiplications.

■

7 Pseudocode

The pseudocode can be a bit confusing because it needs to manage two numbers with differing numbers of digits, it needs to manage finding a split location and split from the right, and it needs to manage the powers of 10 correctly.

```
\\ A,B are the list representations of numbers.
function karatsuba(A,B)
    if either A or B is single-digit
        return(A*B)
    else
        sp = floor((minimum number of digits in A,B)/2)
        A1,A0 = split A, sp digits from the right
        B1,B0 = split B, sp digits from the right
        k1 = karatsuba(A1,B1)
        k2 = karatsuba(A1+A0,B1+B0)
        k3 = karatsuba(A0,B0)
        // The powers of 10 should be thought of as shifts.
        r = 10^(2*sp)*k1 + 10^(sp)*(k2-k3-k1) + k3
        return(r)
    end
end
```

8 Thoughts, Problems, Ideas

1. Draw a tree diagram for Karatsuba's Algorithm applied to (12)(34) and use it to count the number of SDMs.

2. Draw a tree diagram for Karatsuba's Algorithm applied to (98)(76) and use it to count the number of SDMs.

3. Draw a tree diagram for Karatsuba's Algorithm applied to (1201)(2231) and use it to count the number of SDMs.

4. Draw a tree diagram for Karatsuba's Algorithm applied to (345)(12231) and use it to count the number of SDMs. This is a little trickier than the previous few because of the different integer lengths. Trust the pseudocode!

5. Let $A = a_2a_1a_0$ and $B = b_2b_1b_0$ be the base-10 digit representations of two 3-digit numbers. Formally then $A = 100a_2 + 10a_1 + a_0$ and $B = 100b_2 + 10b_1 + b_0$.

   (a) Evaluate the product AB and collect the result into the form:

   $$10000(T1) + 1000(T2) + 100(T3) + 10(T4) + (T5)$$

   We'll say that the $T1$,... are the terms. How many different products arise in all of the terms together?

   (b) Rewrite the terms $T2$ and $T4$ in such a way as to reduce the number of total multiplications which are necessary to evaluate the product down to 7.

   (c) Explain (not even pseucode) how this might lead to a recursive algorithm for multiplication much like the Karatsuba Algorithm.

   (d) Write down the recurrence relation for this algorithm and solve it using the Master Theorem. Is it faster or slower than Karatsuba?

6. We observed that if n is even and $A = A_1A_0$ and $B = B_1B_0$ are two n-digit numbers that the special product $(A_1 + A_0)(B_1 + B_0)$ might not be a product of two $n/2$-digit numbers, thereby making the recurrence relation $T(n) = 3T(n/2) + \Theta(n)$ feel a little iffy. We glossed over this but let's patch it a bit.

   (a) Show that in the $n = 2$ case that even if both $a_1 + a_0$ and $b_1 + b_2$ are not single-digit numbers that the product $(a_1 + a_0)(b_1 + b_0)$ can be calculated using one single-digit multiplication and $\Theta(n)$ additions and decimal shifts. [10 pts]

   (b) Generalize this argument in the following sense: Show that even if both A_1+A_0 and B_1+B_0 are not $n/2$-digit numbers that the product $(A_1 + A_0)(B_1 + B_0)$ can be calculated using one $n/2$-digit multiplication and $\Theta(n)$ additions and decimal shifts, thereby rendering the recurrence relation accurate. [10 pts]

9 Python Code and Output

Code:

```
import random
import math

numdig = 4
AA = random.randint(10**(numdig-1),10**(numdig)-1)
BB = random.randint(10**(numdig-1),10**(numdig)-1)
mcounter = 0

AA = 3451
BB = 45862

def karatsuba(A,B,indent):
    global mcounter
    #print(indent*' ' + 'Pair = ' + str(A) + ' ' + str(B))
    if A<10 or B<10:
        mcounter = mcounter + (len(str(A))*len(str(B)))
        print(indent*' ' + 'Base Product = ' + str(A*B))
        return(A*B)
    else:
        AS = [int(i) for i in str(A)]
        BS = [int(i) for i in str(B)]
        sl = min(len(AS),len(BS)) // 2
        A1 = int(''.join(map(str,AS[0:len(AS)-sl])))
        A0 = int(''.join(map(str,AS[len(AS)-sl:])))
        B1 = int(''.join(map(str,BS[0:len(BS)-sl])))
        B0 = int(''.join(map(str,BS[len(BS)-sl:])))
        print(indent*' ' + 'Recurse to (A1,B1) = ('+str(A1)+','+str(B1)+')')
        k1 = karatsuba(A1,B1,indent+2)
        print(indent*' ' + 'Recurse to (A1+A0,B1+B0) = ('+str(A1)+'+'+str(A0)+','+str
        k2 = karatsuba(A1+A0,B1+B0,indent+2)
        print(indent*' ' + 'Recurse to (A0,B0) = ('+str(A0)+','+str(B0)+')')
        k3 = karatsuba(A0,B0,indent+2)
        k = (10**(2*sl))*k1 + (10**(sl))*(k2-k3-k1) + k3
        print(indent*' ' + 'Product = ' + str(10**(2*sl)) + '*' + str(k1) + '+' + str
        return(k)

print('Start (A,B) = ('+str(AA)+','+str(BB)+')')
p = karatsuba(AA,BB,2)

print('n = ' + str(numdig))
print('Result: ' + str(p))
print('Number of SDM: ' + str(mcounter))
print('Note that '+str(numdig)+'^lg(3) = ' + str(numdig**math.log(3,2)))
print('Direct Python Calculation: ' + str(AA*BB))
print('Would Take SDM: ' + str(numdig**2))
```

Output with two 2-digit numbers. Observe this takes exactly the same number of SDM as schoolbook multiplication and a bit less than twice $n^{\lg 3}$:

```
Start (A,B) = (95,96)
  Recurse to (A1,B1) = (9,9)
    Base Product = 81
  Recurse to (A1+A0,B1+B) = (9+5,9+6) = (14,15)
    Recurse to (A1,B1) = (1,1)
      Base Product = 1
    Recurse to (A1+A0,B1+B) = (1+4,1+5) = (5,6)
      Base Product = 30
    Recurse to (A0,B0) = (4,5)
      Base Product = 20
    Product = 100*1+10*(30-20-1)+20 = 210
  Recurse to (A0,B0) = (5,6)
    Base Product = 30
  Product = 100*81+10*(210-30-81)+30 = 9120
n = 2
Result: 9120
Number of SDM: 5
Note that 2^lg(3) = 3.0000000000000004
Direct Python Calculation: 9120
Would Take SDM: 4
```

Output with two 3-digit numbers. Observe this takes exactly the same number of SDM as schoolbook multiplication and again a bit less than twice $n^{\lg 3}$:

```
Start (A,B) = (840,240)
  Recurse to (A1,B1) = (84,24)
    Recurse to (A1,B1) = (8,2)
      Base Product = 16
    Recurse to (A1+A0,B1+B) = (8+4,2+4) = (12,6)
      Base Product = 72
    Recurse to (A0,B0) = (4,4)
      Base Product = 16
    Product = 100*16+10*(72-16-16)+16 = 2016
  Recurse to (A1+A0,B1+B) = (84+0,24+0) = (84,24)
    Recurse to (A1,B1) = (8,2)
      Base Product = 16
    Recurse to (A1+A0,B1+B) = (8+4,2+4) = (12,6)
      Base Product = 72
    Recurse to (A0,B0) = (4,4)
      Base Product = 16
    Product = 100*16+10*(72-16-16)+16 = 2016
  Recurse to (A0,B0) = (0,0)
    Base Product = 0
```

```
  Product = 100*2016+10*(2016-0-2016)+0 = 201600
n = 3
Result: 201600
Number of SDM: 9
Note that 3^lg(3) = 5.704522494691118
Direct Python Calculation: 201600
Would Take SDM: 9
```

Output with two 4-digit numbers. Observe this takes fewer SDM as schoolbook multiplicaion and again a bit less than twice $n^{\lg 3}$:

```
Start (A,B) = (7087,2600)
  Recurse to (A1,B1) = (70,26)
    Recurse to (A1,B1) = (7,2)
      Base Product = 14
    Recurse to (A1+A0,B1+B) = (7+0,2+6) = (7,8)
      Base Product = 56
    Recurse to (A0,B0) = (0,6)
      Base Product = 0
    Product = 100*14+10*(56-0-14)+0 = 1820
  Recurse to (A1+A0,B1+B) = (70+87,26+0) = (157,26)
    Recurse to (A1,B1) = (15,2)
      Base Product = 30
    Recurse to (A1+A0,B1+B) = (15+7,2+6) = (22,8)
      Base Product = 176
    Recurse to (A0,B0) = (7,6)
      Base Product = 42
    Product = 100*30+10*(176-42-30)+42 = 4082
  Recurse to (A0,B0) = (87,0)
    Base Product = 0
  Product = 10000*1820+100*(4082-0-1820)+0 = 18426200
n = 4
Result: 18426200
Number of SDM: 10
Note that 4^lg(3) = 9.000000000000002
Direct Python Calculation: 18426200
Would Take SDM: 16
```

Summary with two 10-digit numbers. Observe this takes fewer SDM as schoolbook multiplicaion and again a bit less than twice $n^{\lg 3}$:

```
Pair = 8035207000 9075773597
...
n = 10
Result: 72925719537029579000
Number of SDM: 64
Note that 10^lg(3) = 38.45585757936911
Direct Python Calculation: 72925719537029579000
Would Take SDM: 100
```

Summary with two 20-digit numbers. Observe this takes fewer SDM as schoolbook multiplicaion and again a bit less than twice $n^{\lg 3}$:

```
Pair = 49521157366056646229 93687401978021091533
...
n = 20
Result: 463950857657058918509964521779580827905 7
Number of SDM: 222
Note that 20^lg(3) = 115.36757273810733
Direct Python Calculation: 463950857657058918509964521779580827905 7
Would Take SDM: 400
```

Very brief summary with two 100-digit numbers (numbers not listed):

```
n = 100
Result: ...
Number of SDM: 2721
Note that 100^lg(3) = 1478.8529821647205
Direct Python Calculation: ...
Would Take SDM: 10000
```

Very brief summary with two 1000-digit numbers (numbers not listed):

```
n = 1000
Result: ...
Number of SDM: 98486
Note that 1000^lg(3) = 56870.559662951775
Direct Python Calculation: ...
Would Take SDM: 1000000
```

In each case observe that that the number of SDM is a bit less than twice $n^{\lg 3}$ which aligns with our assertion that Karatsuba is $\Theta(n^{\lg 3})$.