INTEGER MULTIPLICATION

**Problem Statement** Multiplication of two n digit numbers using common grade school algorithm takes n^2 multiplication operations. Design a sub quadratic time algorithm for multiplying two n digit integers.

For 2 integers A and B, use the following approach:

A ∗B = (A1 ∗B1)10^n+ (A1 ∗B2 + A2 ∗B1) 10^n/2 + A2 ∗B2

1.**Explanation of Design Strategy**

A common approach for multiplying two numbers is to follow the technique that we have studied in school. One by one take all bits of second number and multiply it with all bits of first number. Finally add all multiplications. Unfortunately, a straightforward adaptation of the standard method for multi-plying integers, as taught in elementary school, results in a method for multiplying two n-bit integers that runs in O(n2) time

The other approach is a sub quadratic time algorithm is **Divide and Conquer**, we can multiply two integers in less time complexity. We divide the given numbers in two halves. Let the given numbers be A and B.

For simplicity let us assume that n is even

A ∗B = (A1 ∗B1)10^n+ (A1 ∗B2 + A2 ∗B1) 10^n/2 + A2 ∗B2

For simplicity let us assume that n is even

A = A1\*10n/2 + A2 [A1 contains leftmost and A2 contains rightmost n/2 bits of A]

B = B2\*10n/2 + B2 [B1 contains leftmost and B2 contains rightmost n/2 bits of B]

The product of AB can be written as following.

AB = (A1\*10^n/2 + A2) \* (B1\*10^n/2 + A2)

= 10^n \*A1\*B1 + 10^n/2\*(A1\*B2 + A2\*B1) + A2\*B2

In formula, there are four multiplications of size n/2, so we basically divided the problem of size n into four sub-problems of size n/2. But that doesn’t help because solution of recurrence T(n) = 4T(n/2) + O(n) is O(n^2). The solution is implemented in **Karatsuba.py**.

The tricky part of this algorithm is to change the middle two terms to some other form so that only one extra multiplication would be sufficient. The following is tricky expression for middle two terms.

A1\*B2 + A2\*B1=(A1+A2) \* (B1+B2)-A1\*B1-A2\*B2

So, the final value of AB becomes

=10^n \*A1\*B1 + 10^n/2\*((A1+A2) \* (B1+B2)-A1\*B1-A2\*B2) + A2\*B2

With above modification, the recurrence becomes T(n) = 3T(n/2) + O(n) and solution of this recurrence is O(n1.59) that is approximately equals to O(n). This has improved the efficiency of algorithm using.

This code is implemented in **Karatsuba\_withLessComplexity.py**

To handle the different length case, we append 0’s in the beginning by using **zfill()** method of python. To handle odd length, we put *floor (n/2)* bits in left half and *ceil(n/2)* bits in right half. So, the expression for AY changes to following.

=10^(2ceil(n/2) \* A1\*B1 + 10^(ceil(n/2)) \* ((A1+A2)\*(B1+B2)-A1\*B1-A2\*B2) + A2\*B2

This algorithm is called **Karatsuba algorithm**.

2. Done in python file and explained above in point 1.

In formula, there are four multiplications of size n/2, so we basically divided the problem of size n into four sub-problems of size n/2. But that doesn’t help because solution of recurrence T(n) = 4T(n/2) + O(n) is O(n^2).

AB=10^n \*A1\*B1 + 10^n/2\*(A1\*B2 + A2\*B1) + A2\*B2

There are four recursion A1\*B1,A1\*B2,A2\*B1 and A2\*B2 as following

*A1\_B1 = numberMultiplication(A1, B1) # recursive call to multiply A1 and B1 part*

*A1\_B2 = numberMultiplication(A1, B2) # recursive call to multiply A1 and B2 part*

*A2\_B1 = numberMultiplication(A2, B1) # recursive call to multiply A2 and B1 part*

*A2\_B2 = numberMultiplication(A2, B2) # recursive call to multiply A2 and B2 part*

The tricky part of this algorithm is to change the middle two terms to some other form so that only one extra multiplication would be sufficient. The following is tricky expression for middle two terms.

A1\*B2 + A2\*B1=(A1+A2) \* (B1+B2)-A1\*B1-A2\*B2

So, the final value of AB becomes

=10^n \*A1\*B1 + 10^n/2\*((A1+A2) \* (B1+B2)-A1\*B1-A2\*B2) + A2\*B2

With above modification, the recurrence becomes T(n) = 3T(n/2) + O(n) and solution of this recurrence is O(n1.59) that is approximately equals to O(n). This has improved the efficiency of algorithm using.

*A1\_B1 = numberMultiplication(A1, B1) # recursive call to multiply A1 and B1 part  
A2\_B2 = numberMultiplication(A2, B2) # recursive call to multiply A2 and B2 part  
P = numberMultiplication(int(A1) + int(A2), int(B1) + int(B2)) ) # recursive call to multiply A1+A2 and B1+B2 part*

3**. “Multiplying big integers has applications to data security, where big integers are used in encryption schemes.”. Substantiate the above statement with one example.**

A common computation in many cryptographic systems is the multiplication of large integers. For instance, real-world uses of the RSA cryptosystem often involve 1024-bit, 2048-bit, or even 3072-bit keys; hence, RSA encryption and decryption with such keys involves the multiplication of large integers having these bit lengths. In fact, the U.S. National Institute for Standards and Technology (NIST) has argued that to achieve a high level of security for the RSA cryptosystem one should use 15,360-bit keys. Thus, from an algorithmic viewpoint, improving the running time for integer multiplication can result in faster and more secure cryptographic protocols involving large integers.