ZiaUrRehman-bit final

1 contributor

482 lines (482 sloc) | 605 KB

Karatsuba Algorithm for Large Integer Multiplication

The Karatsuba algorithm is a fast multiplication algorithm that was developed by Anatoly Karatsuba in the 1960s.

It is a **'divide and conquer'** algorithm that reduces the multiplication of two n-digit numbers to three multiplications of n/2-digit numbers.

Divide each number into two halves, and then apply some given steps.

The Karatsuba algorithm has a time complexity of `O(n^log_2 3)`, which is faster than the traditional multiplication algorithm, which has a time complexity of `O(n^2)`.

This makes it more efficient for large numbers and has made it a popular choice for implementing multiplication in computer software.

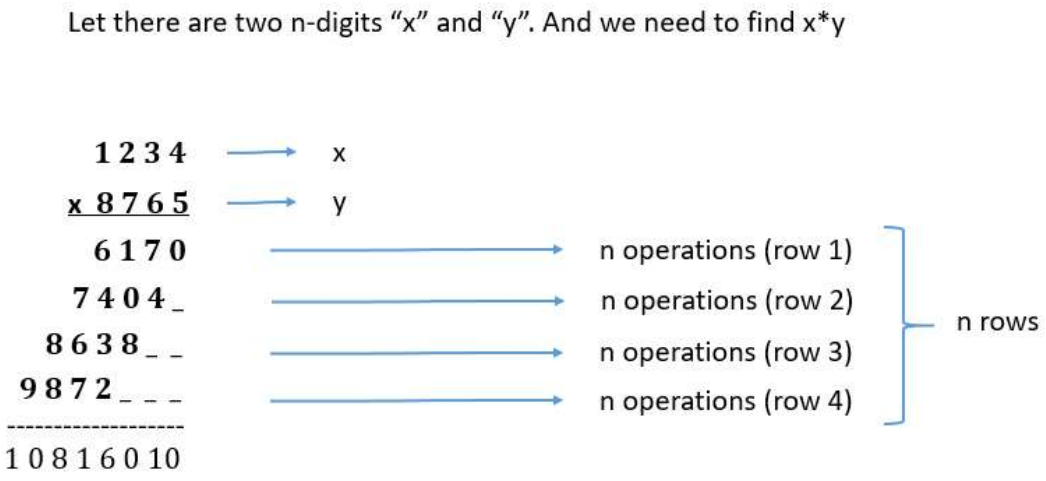
It is also used as a building block for more efficient multiplication algorithms, such as the `Toom-Cook` algorithm and the `Schönhage-Strassen` algorithm.

1. Why we need karatsuba Algorithm?

1. The conventional method used to multiply two numbers are not efficient in terms of time complexity for large numbers.

How?

2. Suppose that we have two n-digit numbers x and y as shown below, and we want to calculate the product of x and y.
3. In traditional multiplication method which is also known as Naive algorithm, we need to multiply one digit of "y" with all digits of "x" in each row.
4. Each digit multiplication is a single operation, so if "x" is n-digit number, then it means we are performing n-operations in each row.
5. So total number of operations for "n" rows would be n*n = n^2.
6. The total time required to multiply two n-digit numbers is O(n^2)



2. Implementing Naive Algorithm

In [7]:
```
def naive_multiplication(a, b):
```

```
    result = 0
    for i in range(1, b+1):
        result += a
    return result

# test the function
print(naive_multiplication(321, 545))  # should print 12
print(naive_multiplication(5, 5))  # should print 25
print(naive_multiplication(2, 7))  # should print 14
```

```
174945
25
14
```

In [1]:

```
def multiplication(X, Y):
    # convert numbers into string
    x = str(X)
    y = str(Y)
    result = 0
    # looping over y
    for i in range(len(y)):
        carry = 0  # Intermediate carry
        inter_res = ""  # Intermediate result
        # looping over x
        for j in range(len(x) - 1, -1, -1):
            # intermediate multiplication of each digit and addition of carry
            num = int(y[i]) * int(x[j]) + carry
            # if intermediate multiplication is of two digits and j>0
            #   then second digit is appended to intermediate result
            #   and first digit is stored as carry
            if num > 9 and j > 0:
                inter_res = str(num % 10) + inter_res
                carry = num // 10
            # else the digit is append to intermediate result
            # And assign carry as zero
            else:
                inter_res = str(num) + inter_res
                carry = 0
        # Adding the intermediate results
        result *= 10
        result += int(inter_res)
    return result
```

3. Time Complexity of Naive Algorithm

Note that this is not the most efficient way to multiply two numbers, as it has a time complexity of O(n), where n is the value of y.

However, the function also uses the addition operation, which has a time complexity of O(n). Since the function is using both a loop and the addition operation, the overall time complexity is O(n^2).

In [4]:

```
from datetime import datetime

currentTime = datetime.now()
TimeInMicroSecBefore = currentTime.microsecond
print(TimeInMicroSecBefore)

multiplication(1231212, 2121221)

endTime = datetime.now()
TimeInMicroSecAfter = endTime.microsecond
print(TimeInMicroSecAfter)

totalTime = TimeInMicroSecAfter - TimeInMicroSecBefore

print(totalTime)
```

```
500769
503008
2239
```

In [8]:

```
import time
```

```
start = time.time()
result = naive_multiplication(10000032300000,100323000)
end = time.time()

print(f"Multiplication Result: {result}")
print('Time taken for ' +  str((end - start))
 + ' seconds')
```

```
Multiplication Result: 1003233240432900000000
Time taken for 8.632903575897217 seconds
```

In [10]:

```
import time

start = time.time()
result = naive_multiplication(89898989898,187878780999880)
end = time.time()

print(f"Multiplication Result: {result}")
print('Time taken for ' +  str((end - start))
 + ' seconds')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In [10], line 4
      1 import time
      3 start = time.time()
----> 4 result = naive_multiplication(89898989898,187878780999880)
      5 end = time.time()
      7 print(f"Multiplication Result: {result}")

Cell In [7], line 4, in naive_multiplication(a, b)
      2 result = 0
      3 for i in range(1, b+1):
----> 4     result += a
      5 return result

KeyboardInterrupt:
```

4. Steps of Karatsuba Algorithm for Large Integer Multiplication

Here, $x = 1234$, $y = 8765$, $b = 10$ (Decimal Base System), $n = 4$ (digits)

$x = \underbrace{12}_{X_H}\ \underbrace{34}_{X_L}$ $\quad y = \underbrace{87}_{Y_H}\ \underbrace{65}_{Y_L}$

Steps

1. Calculate $x_h * y_h$ ⟶ $S1$
2. Calculate $x_l * y_l$ ⟶ $S2$
3. Calculate $(x_l + x_h) * (y_l + y_h)$ ⟶ $S3$
4. Calculate $(S3 - S2 - S1)$ ⟶ $S4$
5. Calculate $S1 * (b^n) + S4 * \left(b^{\frac{n}{2}}\right) + S2$ ⟶ ***Result***

$x_h = 12$, $x_l = 34$, $y_h = 87$, $y_l = 65$, $b = 10$, $n = 4$

1. Calculate $S1 = x_h * y_h = 12 * 87 = 1044$ ⟶ $R1$
2. Calculate $S2 = x_l * y_l = 34 * 65 = 2210$ ⟶ $R2$

3. Calculate S3 $= (x_l + x_h) * (y_l + y_h) = (34 + 12) * (65 + 87) = 6992$ ⟶ $R3$
4. Calculate S4 $= (S3 - S2 - S1) = 6992 - 2210 - 1044 = 3738$
5. Calculate $S1 * (b^n) + S4 * \left(b^{\frac{n}{2}}\right) + S2 = 1044 * (10^4) + 3738 * (10^{\frac{4}{2}}) + 2210$

$= 10816010$

- *You can observe that, in above mentioned steps multiplication occures in first three steps, means algorithm recurses three times on* $\frac{n}{2}$ *digit number, and there are* $O(n)$ *additions and subtractions required.*
- *Recurence relation* $= \boldsymbol{T(n) = 3\,T\left(\frac{n}{2}\right) + n}$

5. Pseudo Code of Karatsuba Algorithm for Large Integer Multiplication

```
function karatsuba(x, y)
    if x < 10 or y < 10
      return x*y
    m = max(number of digits in x, number of digits in y)
    m2 = m / 2
    xh = most significant m2 digits of x
    xl = least significant m2 digits of x
    yh = most significant m2 digits of y
    yl = least significant m2 digits of y

    s1 = karatsuba(xh, yh)
    s2 = karatsuba(xl, yl)
    s3 = karatsuba(xl+xh , yl+yh)
    s4 = s3 - s2 - s1

    return s1*(10^(m*2)) + s4*(10**m)+s2
```

$O(1)$

$T(n/2)$

$T(n/2)$

$T(n/2)$

$O(n)$

$$\boldsymbol{T(n) = 3T\left(\frac{n}{2}\right) + n}$$

6. Finding Time Complexity of Karatsuba Algorithm

Prove: $\boldsymbol{T(n) = 3 * T\left(\frac{n}{2}\right) + n \quad \rightarrow \quad O(n^{\log_2 3})}$

$$T(n) = 3\left[3T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n) = 9T\left(\frac{n}{4}\right) + 3\left(\frac{n}{2}\right) + n$$

$$T(n) = 9\left[3T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 3\left(\frac{n}{2}\right) + n$$

→K=3

$$T(n) = 27T\left(\frac{n}{8}\right) + 9\left(\frac{n}{4}\right) + 3\left(\frac{n}{2}\right) + n$$

$$T(n) = 3^3T\left(\frac{n}{2^3}\right) + 9\left(\frac{n}{4}\right) + 3\left(\frac{n}{2}\right) + n$$

Take LCM

→K=3

$$\boldsymbol{T(n) = 3^kT\left(\frac{n}{2^k}\right) + \frac{(2^{k+1}+3)n}{2^{k-1}}}$$

Now consider $n = 2^k$, *and* $k = \log_2 n$

$$\boldsymbol{T(n) = 3^{\log_2 n}T\left(\frac{n}{n}\right) + \frac{(2^k * 2^1 + 3)n}{2^k * 2^{-1}}}$$

$$\boldsymbol{T(n) = 3^{\log_2 n}T(1) + \frac{2(n * 2 + 3)n}{n}}$$

$$\boldsymbol{T(n) = 3^{\log_2 n} * 1 + \frac{(4n + 6)n}{n}}$$

$$\boldsymbol{T(n) = 3^{\log_2 n} + 4n + 6}$$

$$\therefore T\left(\frac{n}{2}\right) = 3T\left(\frac{n}{4}\right) + \frac{n}{2}$$

$$\therefore T\left(\frac{n}{4}\right) = 3T\left(\frac{n}{8}\right) + \frac{n}{4}$$

$$\therefore T\left(\frac{n}{8}\right) = 3T\left(\frac{n}{16}\right) + \frac{n}{8}$$

$$9\left(\frac{n}{4}\right) + 3\left(\frac{n}{2}\right) + n$$

$$\frac{9n + 6n + 4n}{4}$$

$$\frac{19n}{4}$$

→K=3

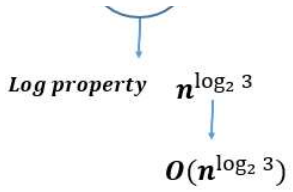
$2^{3+1} = 2^4 + 3 = 19$

$2^{3-1} = 2^2 = 4$

Replace 3 with k

$2^{k+1} + 3 = 19$

$2^{k-1} = 4$

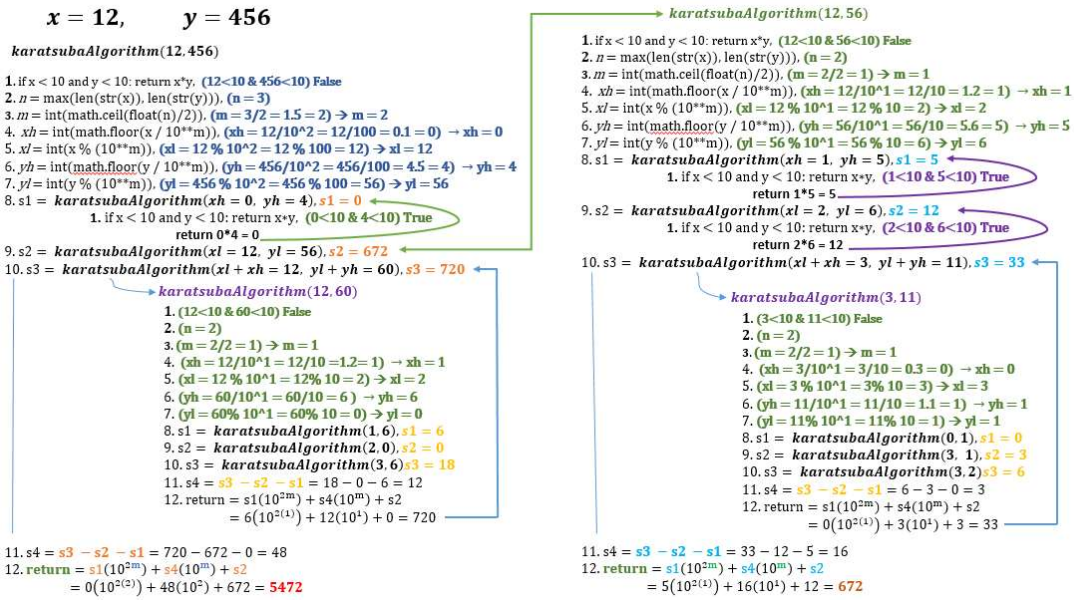


7. Implementing Karatsuba Algorithm

In [5]:

```
import math             # this library gives us ceil() and floor() function

def karatsubaAlgorithm(x, y):

    if x < 10 and y < 10:       # this is base case (when x and y remains single digit number)
        return x*y

    n = max(len(str(x)), len(str(y)))   # finding maximum number of digit in both number
    m = int(math.ceil(float(n)/2))      # number of digit for dividing numbers

    xh = int(math.floor(x / 10**m))     # dividing x into first half
    xl = int(x % (10**m))               # dividing x into second half

    yh = int(math.floor(y / 10**m))     # dividing y into first half
    yl = int(y % (10**m))               # dividing y into second half

    s1 = karatsubaAlgorithm(xh, yh)               # first recurrence (Step 1)
    s2 = karatsubaAlgorithm(xl, yl)               # second recurrence (Step 2)
    s3 = karatsubaAlgorithm(xh + xl, yh + yl)     # third recurrence (Step 3)

    s4 = s3 - s2 - s1                   # calculating s4 (Step 4)

    return int(s1*(10**(m*2)) + s4*(10**m)+s2)
```

8. Dry run with example



9. Calculating Execution Time of Karatsuba Algorithm

In [12]:

```
import time

start = time.time()
result = karatsubaAlgorithm(89898989898,187878780999880)
end = time.time()

print(f"Multiplication Result: {result}")
print('Time taken for ' +  str((end - start))
 + ' seconds')
```

```
Multiplication Result: 168901126351567664592122 40
Time taken for 0.00775146484375 seconds
```

In [13]:

```
from datetime import datetime

currentTime = datetime.now()
TimeInMicroSecBefore = currentTime.microsecond
print(TimeInMicroSecBefore)

karatsubaAlgorithm(1231212, 2121221)

endTime = datetime.now()
TimeInMicroSecAfter = endTime.microsecond
print(TimeInMicroSecAfter)

totalTime = TimeInMicroSecAfter - TimeInMicroSecBefore

print(totalTime)
```

```
903892
908069
4177
```

9. Comparison of Naive and Karatsuba Algo. w.r.t. execution time