Karatsuba Algorithm for Large Integer Multiplication

Monday, January 2, 2023 12:57 PM

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?

1. 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.
2. 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.
3. 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.
4. So total number of operations for "n" rows would be n*n = n^2.
5. The total time required to multiply two n-digit numbers is O(n^2)

Let there are two n-digits "x" and "y". And we need to find x*y

```
    1 2 3 4   →  x
  x 8 7 6 5   →  y
    6 1 7 0   →  n operations (row 1)
  7 4 0 4 _   →  n operations (row 2)
8 6 3 8 _ _   →  n operations (row 3)
9 8 7 2 _ _ _ →  n operations (row 4)
-------------------
1 0 8 1 6 0 1 0
```

n rows

2. Implementing Naive Algorithm

```python
def naive_multiplication(a, b):
    result = 0
    for i in range(1, b+1):
        result += a
    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).

```python
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)
```

```python
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')
```

4. Steps of Karatsuba Algorithm for Large Integer Multiplication

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

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

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)          → T(n/2)
    s2 = karatsuba(xl, yl)          → T(n/2)
    s3 = karatsuba(xl+xh , yl+yh)   → T(n/2)
    s4 = s3 – s2 – s1
    return s1*(10^(m*2)) + s4*(10**m)+s2
```

Lines from "if x < 10 or y < 10" to "yl = least significant m2 digits of y": $O(1)$

Lines "s4 = s3 – s2 – s1" and "return …": $O(n)$

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

6. Finding Time Complexity of Karatsuba Algorithm

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

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

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

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

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

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

→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

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

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

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

→K=3

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

Replace 3 with k

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

→K=3

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

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

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

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

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

7. Implementing Karatsuba Algorithm

```python
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)
```

9. Calculating Execution Time of Karatsuba Algorithm

```python
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')
```

8. Dry run with example

$$x = 12, \qquad y = 456$$

***karatsubaAlgorithm*(12, 456)**

1. if x < 10 and y < 10: return x*y, **(12<10 & 456<10) False**
2. n = max(len(str(x)), len(str(y))), **(n = 3)**
3. m = int(math.ceil(float(n)/2)), **(m = 3/2 = 1.5 = 2) → m = 2**
4. xh = int(math.floor(x / 10**m)), **(xh = 12/10^2 = 12/100 = 0.1 = 0) → xh = 0**
5. xl = int(x % (10**m)), **(xl = 12 % 10^2 = 12 % 100 = 12) → xl = 12**
6. yh = int(math.floor(y / 10**m)), **(yh = 456/10^2 = 456/100 = 4.5 = 4) → yh = 4**
7. yl = int(y % (10**m)), **(yl = 456 % 10^2 = 456 % 100 = 56) → yl = 56**
8. s1 = ***karatsubaAlgorithm*(xh = 0, yh = 4)**, s1 = 0
   1. if x < 10 and y < 10: return x*y, **(0<10 & 4<10) True**
      return 0*4 = 0
9. s2 = ***karatsubaAlgorithm*(xl = 12, yl = 56)**, s2 = 672
10. s3 = ***karatsubaAlgorithm*(xl + xh = 12, yl + yh = 60)**, s3 = 720

    ***karatsubaAlgorithm*(12, 60)**
    1. **(12<10 & 60<10) False**
    2. **(n = 2)**
    3. **(m = 2/2 = 1) → m = 1**
    4. **(xh = 12/10^1 = 12/10 = 1.2 = 1) → xh = 1**
    5. **(xl = 12 % 10^1 = 12% 10 = 2) → xl = 2**
    6. **(yh = 60/10^1 = 60/10 = 6) → yh = 6**
    7. **(yl = 60% 10^1 = 60% 10 = 0) → yl = 0**
    8. s1 = ***karatsubaAlgorithm*(1, 6)**, s1 = 6
    9. s2 = ***karatsubaAlgorithm*(2, 0)**, s2 = 0
    10. s3 = ***karatsubaAlgorithm*(3, 6)** s3 = 18
    11. s4 = s3 − s2 − s1 = 18 − 0 − 6 = 12
    12. return = $s1(10^{2m}) + s4(10^{m}) + s2$
        $= 6(10^{2(1)}) + 12(10^{1}) + 0 = 720$
11. s4 = s3 − s2 − s1 = 720 − 672 − 0 = 48
12. **return** = $s1(10^{2m}) + s4(10^{m}) + s2$
    $= 0(10^{2(2)}) + 48(10^{2}) + 672 =$ **5472**

***karatsubaAlgorithm*(12, 56)**

1. if x < 10 and y < 10: return x*y, **(12<10 & 56<10) False**
2. n = max(len(str(x)), len(str(y))), **(n = 2)**
3. m = int(math.ceil(float(n)/2)), **(m = 2/2 = 1) → m = 1**
4. xh = int(math.floor(x / 10**m)), **(xh = 12/10^1 = 12/10 = 1.2 = 1) → xh = 1**
5. xl = int(x % (10**m)), **(xl = 12 % 10^1 = 12 % 10 = 2) → xl = 2**
6. yh = int(math.floor(y / 10**m)), **(yh = 56/10^1 = 56/10 = 5.6 = 5) → yh = 5**
7. yl = int(y % (10**m)), **(yl = 56 % 10^1 = 56 % 10 = 6) → yl = 6**
8. s1 = ***karatsubaAlgorithm*(xh = 1, yh = 5)**, s1 = 5
   1. if x < 10 and y < 10: return x*y, **(1<10 & 5<10) True**
      return 1*5 = 5
9. s2 = ***karatsubaAlgorithm*(xl = 2, yl = 6)**, s2 = 12
   1. if x < 10 and y < 10: return x*y, **(2<10 & 6<10) True**
      return 2*6 = 12
10. s3 = ***karatsubaAlgorithm*(xl + xh = 3, yl + yh = 11)**, s3 = 33

    ***karatsubaAlgorithm*(3, 11)**
    1. **(3<10 & 11<10) False**
    2. **(n = 2)**
    3. **(m = 2/2 = 1) → m = 1**
    4. **(xh = 3/10^1 = 3/10 = 0.3 = 0) → xh = 0**
    5. **(xl = 3 % 10^1 = 3% 10 = 3) → xl = 3**
    6. **(yh = 11/10^1 = 11/10 = 1.1 = 1) → yh = 1**
    7. **(yl = 11% 10^1 = 11% 10 = 1) → yl = 1**
    8. s1 = ***karatsubaAlgorithm*(0, 1)**, s1 = 0
    9. s2 = ***karatsubaAlgorithm*(3, 1)**, s2 = 3
    10. s3 = ***karatsubaAlgorithm*(3, 2)** s3 = 6
    11. s4 = s3 − s2 − s1 = 6 − 3 − 0 = 3
    12. return = $s1(10^{2m}) + s4(10^{m}) + s2$
        $= 0(10^{2(1)}) + 3(10^{1}) + 3 = 33$
11. s4 = s3 − s2 − s1 = 33 − 12 − 5 = 16
12. **return** = $s1(10^{2m}) + s4(10^{m}) + s2$
    $= 5(10^{2(1)}) + 16(10^{1}) + 12 =$ **672**