

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.

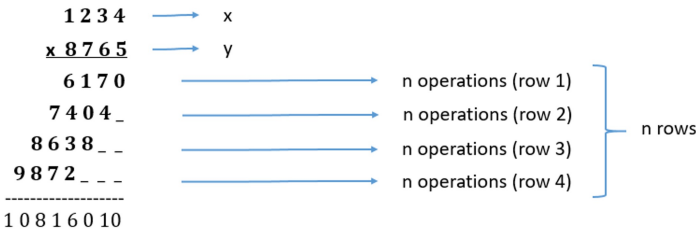
1. Why we need karatsuba Algorithm?

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

How?

- 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.
- 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.
- 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.
- So total number of operations for "n" rows would be $n * n = n^2$.
- 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 $x * y$



2. Implementing Naive Algorithm

```
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 $O(n^2)$.

```
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)
```

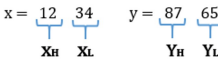
```
import time

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)



Steps

- Calculate $x_h * y_h$ → $S1$
- Calculate $x_l * y_l$ → $S2$
- Calculate $(x_l + x_h) * (y_l + y_h)$ → $S3$
- Calculate $(S3 - S2 - S1)$ → $S4$
- Calculate $S1 * (b^n) + S4 * (b^{\frac{n}{2}}) + S2$ → **Result**

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

- Calculate $S1 = x_h * y_h = 12 * 87 = 1044$ → $R1$
- Calculate $S2 = x_l * y_l = 34 * 65 = 2210$ → $R2$
- Calculate $S3 = (x_l + x_h) * (y_l + y_h) = (34 + 12) * (65 + 87) = 6992$ → $R3$
- Calculate $S4 = (S3 - S2 - S1) = 6992 - 2210 - 1044 = 3738$
- Calculate $S1 * (b^n) + S4 * (b^{\frac{n}{2}}) + S2 = 1044 * (10^4) + 3738 * (10^2) + 2210 = 10816010$

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

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
```

6. Finding Time Complexity of Karatsuba Algorithm

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

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

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

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

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

Take LCM

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

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

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

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

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

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

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

$\rightarrow K=3$

$\frac{9(\frac{n}{4}) + 3(\frac{n}{2}) + n}{4} \rightarrow \frac{9n + 6n + 4n}{4} \rightarrow \frac{19n}{4}$

$\rightarrow K=3$

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

Replace 3 with k

$\frac{2^{k+1} + 3}{2^{k-1}} = \frac{2^k + 3}{2^{k-1}} = 2 + \frac{3}{2^{k-1}}$

7. Implementing Karatsuba Algorithm

```
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

x = 12, 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

9. Calculating Execution Time of Karatsuba Algorithm

```
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')
```

