Karatsuba Algorithm

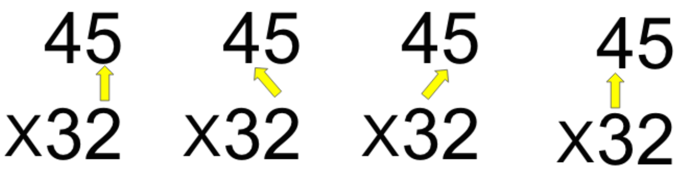
The **Karatsuba algorithm** is a fast multiplication algorithm that uses a divide and conquer approach to multiply two numl
The naive algorithm for multiplying two numbers has a running time of $\Theta\left(n^2\right)$ while this algorithm has a running time o
$\Theta\left(n^{\log_2 3}\right) \approx \Theta\left(n^{1.585}\right)$. Being able to multiply numbers quickly is very important. Computer scientists often consider multiplication to be a constant time $O(1)$ operation, and this is a reasonable simplification for smaller numbers; but for l numbers, the actual running times need to be factored in, which is $O\left(n^2\right)$. The point of the Karatsuba algorithm is to bre large numbers down into smaller numbers so that any multiplications that occur happen on smaller numbers. Karatsuba used to multiply numbers in all base systems (base-10, base-2, etc.).

Contents

Naive Multiplication Algorithm
Karatsuba Algorithm
Implementation of the Karatsuba Algorithm
Complexity of Karatsuba
See Also
References

Naive Multiplication Algorithm

The naive way to multiple numbers is commonly taught in elementary school.



Grade school multiplcation takes four multiplication steps

Here's the naive multiplication algorithm to multiply two n-bit numbers, x and y that are in base b.

Divide each number into two halves, the high bits H and the low bits L :

$$x = x_H b^{\frac{n}{2}} + X_L, \quad y = y_H b^{\frac{n}{2}} + Y_L.$$

Multiply the two numbers together:

$$\begin{aligned} xy &= \left(x_H b^{\frac{n}{2}} + X_L\right) \times \left(y_H b^{\frac{n}{2}} + Y_L\right) \\ &= x_H y_H b^n + (x_H y_L + x_L y_H) b^{\frac{n}{2}} + x_L y_L. \end{aligned}$$

These formulas describe what is going on in the image above--the familiar grade-school way of multiplying numbers. has four multiplications: $x_H y_H b^n$, $(x_H y_L) b^{\frac{n}{2}}$, $(x_L y_H) b^{\frac{n}{2}}$, and $x_L y_L$, which has a running time of $O(n^2)$.

EXAMPLE

Let's use this method to multiply the base-10 numbers 1234 and 8765 :

$$\begin{aligned} x &= x_H b^{\frac{n}{2}} + X_L \\ &= 12 \times 10^2 + 34 \end{aligned}$$

$$\begin{aligned} y &= y_H b^{\frac{n}{2}} + Y_L \\ &= 87 \times 10^2 + 65 \end{aligned}$$

$$\begin{aligned} xy &= \left(x_H b^{\frac{n}{2}} + X_L\right) \times \left(y_H b^{\frac{n}{2}} + Y_L\right) \\ &= (12 \times 10^2 + 34) \times (87 \times 10^2 + 65) \\ &= x_H y_H b^n + (x_H y_L + x_L y_H) b^{\frac{n}{2}} + x_L y_L \\ &= 12 \times 87 \times 10^4 + (12 \times 65 + 34 \times 87) \times 10^2 + (65 \times 34) \\ &= 10,816,010. \end{aligned}$$

Karatsuba Algorithm

The Karatsuba algorithm decreases the number of subproblems to three and ends up calculating the product of two n-b numbers in $\Theta\left(n^{\log_2 3}\right)$ time--a vast improvement over the naive algorithm.

To multiply two n-bit numbers, x and y, the Karatsuba algorithm performs three multiplications and a few additions, anc on smaller numbers that are roughly half the size of the original x and y.

Here's how the Karatsuba method works to multiply two n-bit numbers x and y which are in base b.

Create the following three subproblems where H represents the high bits of the number and L represents the lower b

- $a = x_H y_H$
- $d = x_L y_L$
- $e = (x_H + x_L)(y_H + y_L) - a - d$
- $xy = ab^n + eb^{\frac{n}{2}} + d.$

This only requires three multiplications and has the recurrence

$$3T\left(\frac{n}{2}\right) + O(n) = O\left(n^{\log 3}\right).$$

Karatsuba can be applied recursively to a number until the numbers being multiplied are only a single-digit long (the b case).

Divide and conquer techniques come in when Karatsuba uses recursion to solve subproblems--for example, if multiplic is needed to solve for a, d, or e before those variables can be used to solve the overall $x \times y$ multiplication.

EXAMPLE

Perform the following multiplication using the Karatsuba method: $1234 \times 4321.$[1]

Show Answer

Implementation of the Karatsuba Algorithm

Here is a Python implementation of the Karatsuba algorithm for base-10 numbers.

Python

```
from math import ceil, floor



def karatsuba(x,y):

    if x < 10 and y < 10:
        return x*y

    n = max(len(str(x)), len(str(y)))
    m = ceil(n/2)

    x_H  = floor(x / 10**m)
    x_L = x % (10**m)

    y_H = floor(y / 10**m)
    y_L = y % (10**m)


    a = karatsuba(x_H,y_H)
    d = karatsuba(x_L,y_L)
    e = karatsuba(x_H + x_L, y_H + y_L) - a - d

    return int(a*(10**(m*2)) + e*(10**m) + d)
```

Complexity of Karatsuba

To analyze the complexity of the Karatsuba algorithm, consider the number of multiplications the algorithm performs as function of n, $M(n)$. Recall that the algorithm multiplies together two n-bit numbers. If $n = 2^k$ for some k, then the alg recurses three times on $\frac{n}{2}$-bit number. The recurrence for this is

$$M(n) = 3M\left(\frac{n}{2}\right).$$

This takes care of the multiplications required for Karatsuba--now to consider the additions and subtractions. There are (additions and subtractions required for the algorithm. Therefore, the overall recurrence for the Karatsuba algorithm is[2]

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

Using the master theorem on the above recurrence yields that the running time of the Karatsuba algorithm is $\Theta\left(n^{\log_2 3}\right)$ $\Theta\left(n^{1.585}\right)$.

The Schönhage–Strassen algorithm and the Toom–Cook algorithm are other popular integer multiplication algorithms. T
Toom-Cook algorithm is faster, more generalized version of the Karatsuba algorithm that runs in $\Theta\left(n^{\frac{\log 5}{\log 3}}\right) \approx \Theta\left(n^{1.465}\right)$
Schönhage–Strassen algorithm is faster than both Karatsuba and Toom-Cook for very large n $\left(\text{on the order of } n > 2^{2^{15}}\right)$
runs in $O(n \log n \log \log n)$.[4]

See Also

- Schönhage–Strassen Algorithm
- Toom–Cook Algorithm

References

1. Demaine, E., Indyk, P., & Kellis, M. *Karatsuba's Algorithm*. Retrieved May 29, 2016, from http://courses.csail.mit.edu/6.006/spring11/exams/notes3-karatsuba
2. Babai, L. *Divide and Conquer: The Karatsuba–Ofman algorithm*. Retrieved May 30, 2016, from http://people.cs.uchicago.edu/~laci/HANDOUTS/karatsuba.pdf
3. , . *Toom–Cook multiplication*. Retrieved May 30, 2016, from https://en.wikipedia.org/wiki/Toom%E2%80%93Cook_multiplication
4. , . *Schönhage–Strassen algorithm*. Retrieved May 30, 2016, from https://en.wikipedia.org/wiki/Sch%C3%B6nhage%E2%80%93Strassen_algorithm

Cite as: Karatsuba Algorithm. *Brilliant.org*. Retrieved 09:51, October 14, 2022, from https://brilliant.org/wiki/karatsuba-algorithm/