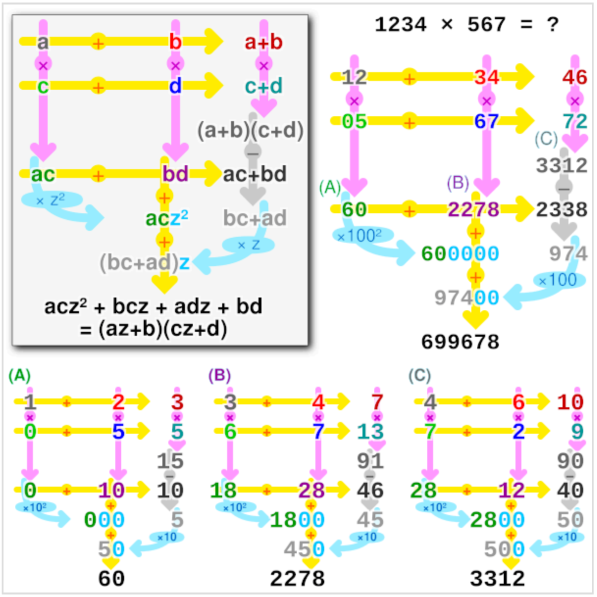
Wikipedia
The Free Encyclopedia

Karatsuba algorithm

Karatsuba algorithm	
Class	Multiplication algorithm

The **Karatsuba algorithm** is a fast multiplication algorithm for integers. It was discovered by Anatoly Karatsuba in 1960 and published in 1962.[1][2][3] It is a divide-and-conquer algorithm that reduces the multiplication of two n-digit numbers to three multiplications of $n/2$-digit numbers and, by repeating this reduction, to at most $n^{\log_2 3} \approx n^{1.58}$ single-digit multiplications. It is therefore asymptotically faster than the traditional algorithm, which performs n^2 single-digit products.



Karatsuba multiplication of az+b and cz+d (boxed), and 1234 and 567 with z=100. Magenta arrows denote multiplication, amber denotes addition, silver denotes subtraction and cyan denotes left shift. (A), (B) and (C) show recursion with z=10 to obtain intermediate values.

The Karatsuba algorithm was the first multiplication algorithm asymptotically faster than the quadratic "grade school" algorithm. The Toom–Cook algorithm (1963) is a faster generalization of Karatsuba's method, and the Schönhage–Strassen algorithm (1971) is even faster, for sufficiently large n.

History

The standard procedure for multiplication of two n-digit numbers requires a number of elementary operations proportional to n^2, or $O(n^2)$ in big-O notation. Andrey Kolmogorov conjectured that the traditional algorithm was *asymptotically optimal,* meaning that any algorithm for that task would require $\Omega(n^2)$ elementary operations.

In 1960, Kolmogorov organized a seminar on mathematical problems in cybernetics at the Moscow State University, where he stated the $\Omega(n^2)$ conjecture and other problems in the complexity of computation. Within a week, Karatsuba, then a 23-year-old student, found an algorithm that multiplies two n-digit numbers in $O(n^{\log_2 3})$ elementary steps, thus disproving the conjecture. Kolmogorov was very excited about the discovery; he communicated it at the next meeting of the seminar, which was then terminated. Kolmogorov gave some lectures on the Karatsuba result at conferences all over the world (see, for example, "Proceedings of the International Congress of Mathematicians 1962", pp. 351–356, and also "6 Lectures delivered at the International Congress of Mathematicians in Stockholm, 1962") and published the method in 1962, in the Proceedings of the USSR Academy of Sciences. The article had been written by Kolmogorov and contained two results on multiplication, Karatsuba's algorithm and a separate result by Yuri Ofman; it listed "A. Karatsuba and Yu. Ofman" as the authors. Karatsuba only became aware of the paper when he received the reprints from the publisher.[2]

Algorithm

Basic step

The basic principle of Karatsuba's algorithm is divide-and-conquer, using a formula that allows one to compute the product of two large numbers x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts. This basic step is, in fact, a generalization of a similar complex multiplication algorithm, where the imaginary unit i is replaced by a power of the base.

Let x and y be represented as n-digit strings in some base B. For any positive integer m less than n, one can write the two given numbers as

$$x = x_1 B^m + x_0,$$
$$y = y_1 B^m + y_0,$$

where x_0 and y_0 are less than B^m. The product is then

$$\begin{aligned} xy &= (x_1 B^m + x_0)(y_1 B^m + y_0) \\ &= x_1 y_1 B^{2m} + (x_1 y_0 + x_0 y_1) B^m + x_0 y_0 \\ &= z_2 B^{2m} + z_1 B^m + z_0, \end{aligned}$$

where

$$\begin{aligned} z_2 &= x_1 y_1, \\ z_1 &= x_1 y_0 + x_0 y_1, \\ z_0 &= x_0 y_0. \end{aligned}$$

These formulae require four multiplications and were known to Charles Babbage.[4] Karatsuba observed that xy can be computed in only three multiplications, at the cost of a few extra additions. With z_0 and z_2 as before and $z_3 = (x_1 + x_0)(y_1 + y_0)$, one can observe that

$$\begin{aligned} z_1 &= x_1 y_0 + x_0 y_1 \\ &= (x_1 + x_0)(y_0 + y_1) - x_1 y_1 - x_0 y_0 \\ &= z_3 - z_2 - z_0. \end{aligned}$$

Thus only three multiplications are required for computing z_0, z_1 and z_2.

Example

To compute the product of 12345 and 6789, where B = 10, choose m = 3. We use m right shifts for decomposing the input operands using the resulting base (B^m = *1000*), as:

12345 = **12** · *1000* + **345**
6789 = **6** · *1000* + **789**

Only three multiplications, which operate on smaller integers, are used to compute three partial results:

z_2 = **12 × 6** = 72
z_0 = **345 × 789** = 272205
z_1 = (**12** + **345**) **×** (**6** + **789**) − z_2 − z_0 = 357 **×** 795 − 72 − 272205 = 283815 − 72 − 272205 = 11538

We get the result by just adding these three partial results, shifted accordingly (and then taking carries into account by decomposing these three inputs in base *1000* as for the input operands):

> result = $z_2 \cdot (B^m)^2 + z_1 \cdot (B^m)^1 + z_0 \cdot (B^m)^0$, i.e.
> result = $72 \cdot 1000^2 + 11538 \cdot 1000 + 272205 =$ **83810205**.

Note that the intermediate third multiplication operates on an input domain which is less than two times larger than for the two first multiplications, its output domain is less than four times larger, and base-*1000* carries computed from the first two multiplications must be taken into account when computing these two subtractions.

Recursive application

If n is four or more, the three multiplications in Karatsuba's basic step involve operands with fewer than n digits. Therefore, those products can be computed by recursive calls of the Karatsuba algorithm. The recursion can be applied until the numbers are so small that they can (or must) be computed directly.

In a computer with a full 32-bit by 32-bit multiplier, for example, one could choose $B = 2^{31}$ and store each digit as a separate 32-bit binary word. Then the sums $x_1 + x_0$ and $y_1 + y_0$ will not need an extra binary word for storing the carry-over digit (as in carry-save adder), and the Karatsuba recursion can be applied until the numbers to multiply are only one digit long.

Time complexity analysis

Karatsuba's basic step works for any base B and any m, but the recursive algorithm is most efficient when m is equal to $n/2$, rounded up. In particular, if n is 2^k, for some integer k, and the recursion stops only when n is 1, then the number of single-digit multiplications is 3^k, which is n^c where $c = \log_2 3$.

Since one can extend any inputs with zero digits until their length is a power of two, it follows that the number of elementary multiplications, for any n, is at most $3^{\lceil \log_2 n \rceil} \leq 3n^{\log_2 3}$.

Since the additions, subtractions, and digit shifts (multiplications by powers of B) in Karatsuba's basic step take time proportional to n, their cost becomes negligible as n increases. More precisely, if $T(n)$ denotes the total number of elementary operations that the algorithm performs when multiplying two n-digit numbers, then

$$T(n) = 3T(\lceil n/2 \rceil) + cn + d$$

for some constants c and d. For this recurrence relation, the master theorem for divide-and-conquer recurrences gives the asymptotic bound $T(n) = \Theta(n^{\log_2 3})$.

It follows that, for sufficiently large n, Karatsuba's algorithm will perform fewer shifts and single-digit additions than longhand multiplication, even though its basic step uses more additions and shifts than the straightforward formula. For small values of n, however, the extra shift and add operations may make it run slower than the longhand method.

Implementation

Here is the pseudocode for this algorithm, using numbers represented in base ten. For the binary representation of integers, it suffices to set BASE to a different number, usually a power of 2 in line with the size of the machine word that the computer can natively multiply.[5]

```
const BASE = 10

/* Count the size of num in BASE. For example, 12345 has a size of 5 in base 10 and a size of 2 in base 1024. */
function size_in_base(num)
    string_num = num.toString()
    return string_num.length()

/* Split a digit into its low "d" digits and its high digits. For example, split_at(12345, 3) will extract the 3 final
digits, giving: high=12, low=345. */
function split_at(num, d)
    hi =  num / (BASE ^ d)
    low = num % (BASE ^ d) /* remainder of division */
    return hi, low

function karatsuba(num1, num2)
    if (num1 < BASE or num2 < BASE)
        return num1 × num2 /* fall back to traditional multiplication */

    /* Calculates the size of the numbers. */
    m = max(size_in_base(num1), size_in_base(num2))
    m2 = floor(m / 2)
    /* m2 = ceil (m / 2) will also work */

    /* Split the digit sequences in the middle. */
    high1, low1 = split_at(num1, m2)
    high2, low2 = split_at(num2, m2)

    /* 3 recursive calls made to numbers approximately half the size. */
    z0 = karatsuba(low1, low2)
    z1 = karatsuba(low1 + high1, low2 + high2)
    z2 = karatsuba(high1, high2)

    return (z2 × BASE ^ (m2 × 2)) + ((z1 - z2 - z0) × BASE ^ m2) + z0
```

An issue that occurs with this implementation is that the computation of $(x_1 + x_0)$ and $(y_1 + y_0)$ for z_1 may result in overflow (will produce a result in the range $B^m \leq \text{result} < 2B^m$), which require a multiplier having one extra bit. This can be avoided by noting that

$$z_1 = (x_0 - x_1)(y_1 - y_0) + z_2 + z_0.$$

This computation of $(x_0 - x_1)$ and $(y_1 - y_0)$ will produce a result in the range of $-B^m < \text{result} < B^m$. This method may produce negative numbers, which require one extra bit to encode signedness, and would still require one extra bit for the multiplier. However, one way to avoid this is to record the sign and then use the absolute value of $(x_0 - x_1)$ and $(y_1 - y_0)$ to perform an unsigned multiplication, after which the result may be negated when both signs originally differed. Another advantage is that even though $(x_0 - x_1)(y_1 - y_0)$ may be negative, the final computation of z_1 only involves additions.

References

1. Karatsuba, A. A.; Ofman, Y. P. (1962). "Multiplication of Many-Digital Numbers by Automatic Computers" (https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=dan&paperid=26729&option_lang=eng). *Proceedings of the USSR Academy of Sciences* (in Russian). **145**: 293–294. "Translation in the academic journal *Physics-Doklady*, **7** (1963), pp. 595–596"
2. Karatsuba, A. A. (1995). "The Complexity of Computations" (http://www.ccas.ru/personal/karatsuba/divcen.pdf) (PDF). *Proceedings of the Steklov Institute of Mathematics*. **211**: 169–183. "Translation from Trudy Mat. Inst. Steklova, 211, 186–202 (1995)"
3. Knuth, Donald E. (1969). *The Art of Computer Programming*. Vol. 2, Seminumerical Algorithms (1st ed.). Reading, Massachusetts: Addison-Wesley. pp. xi+624. ISBN 978-0201038026.
4. Babbage, Charles (1864). "Chapter VIII – Of the Analytical Engine, Larger Numbers Treated". *Passages from the Life of a Philosopher* (https://archive.org/details/bub_gb_Fa1JAAAAMAAJ/page/n142). London: Longman Green. p. 125.

5. Weiss, Mark A. (2005). *Data Structures and Algorithm Analysis in C++* (3rd ed.). Boston: Addison-Wesley. p. 480. ISBN 0321375319.

External links

- Beaver, Jonathan. "Karatsuba's Algorithm for Polynomial Multiplication" (http://www.cs.pitt.edu/~kirk/cs1501/animations/Karatsuba.html). *University of Pittsburgh – CS1501*.
- Bernstein, Daniel J. (2001-08-11). "Multidigit multiplication for mathematicians" (https://cr.yp.to/papers/m3.pdf) (PDF). *cr.yp.to*. Retrieved 2025-10-06. "Covers Karatsuba and many other multiplication algorithms."
- Weisstein, Eric W. "Karatsuba Multiplication" (https://mathworld.wolfram.com/KaratsubaMultiplication.html). *MathWorld*.
- Karatsuba's Multiplication Trick Summarised in 1 Minute (https://www.youtube.com/watch?v=LCY4dnm88oI) on YouTube
- How a Russian student invented a faster multiplication method (https://www.youtube.com/watch?v=cCKOl5li6YM) on YouTube