Fast Integer Multiplication Algorithms

Benjamin Hillard

Euler Circle

June 2022

Time Complexity with Multiplication Algorithms

Big O notation shows the limiting behavior of a function when an input approaches a certain value, typically infinity.
For example, if I had an algorithm that required $3n^3 + 6n^2 - \log_6 n$ steps, the big O notation for the limiting behavior as $n \to \infty$ would be simply $O(n^3)$.
To measure the speed of multiplication algorithms, big O notation is used and is represented in terms of the number of digits in the factors, which will be called n.

Multiplication Algorithm

The standard algorithm or long multiplication is the most well-known multiplication algorithm.
It is performed by writing the two factors down, multiplying every digit from one factor by every digit from the other, and getting the sum of all the products to get the final product.
This algorithm has a time complexity of $O(n^2)$, as the number of operations is $\approx 2n^2$ depending on what you count.

New Integer Multiplication Algorithms

Long multiplication was the most efficient method to multiply any two integers since time immemorial, until new algorithms were found.
1960: Karatsuba algorithm: $O(n^{\log_2 3}) \approx O(n^{1.585})$.
1963: Toom-Cook algorithm: $O(n^{\log_{10} 5/\log_{10} 3}) \approx O(n^{1.465})$.
1970: The Schönhage–Strassen algorithm: $O(n \log n \log\log n)$.
In the same paper a limit of the lowest time complexity of $O(n \log n)$ is conjectured.
2020: The Harvey-van der Hoeven algorithm: $O(n \log n)$.

The Karatsuba Algorithm

I will now go into a explanation of how to use the Karatsuba algorithm and why it works. It has three main steps, which are to

1. split,
2. evaluate and
3. sum all parts.

Split

The algorithm will be performed in base b with the integer factors l and m. First split the numbers into two parts evenly in terms of digits in positional notation, where l_0 and m_0 are the first $\lceil n/2 \rceil$ digits of l and m from the right, and l_1 and m_1 are the first $\lfloor n/2 \rfloor$ digits of l and m from the left.
For example, for the factors $l = 1234$ and $m = 5678$, $n = 4$ and $l_1 = 12$, $l_0 = 34$, $m_1 = 56$, and $m_0 = 78$.

Evaluate

First, let $z_0 = l_0 m_0$ and $z_2 = l_1 m_1$, which in a typical implementation would be done using the Karatsuba algorithm or the standard algorithm, depending on the situation and size of the factors. Next, find the sums $(l_0 + l_1)$ and $(m_0 + m_1)$, and multiply them to get $(l_0 + l_1)(m_0 + m_1)$
In the example from before, $z_0 = 12 * 56 = 672$ and $z_2 = 34 * 78 = 2652$. Then $(l_0 + l_1)(m_0 + m_1) = (12 + 34)(56 + 78) = (46)(134) = 6164$

Evaluate

From here, let $z_1 = (l_0 + l_1)(m_0 + m_1) - (l_0 m_0) - (l_1 m_1)$. Now,

$$(l * m) = z_2 * b^n + z_1 * b^{n/2} + z_0.$$

Again, in the example $z_1 = 6164 - 2652 - 672 = 2840$. Then, base $b = 10$ and $n = 4$, so

$$\begin{aligned}
(1234 * 5678) &= 672 * 10^4 + 2840 * 10^2 + 2652 \\
(1234 * 5678) &= 6720000 + 284000 + 2652 \\
(1234 * 5678) &= 7006652.
\end{aligned}$$

And there's our product. Ta-da!
Note that all multiplication operations performed would be performed likely using the Karatsuba algorithm, but was skipped in the example.

Run-time

The algorithm will perform recursively by splitting a single multiplication into three smaller multiplications, where each factor has at most half the digits. If $n = 2^k$ for some integer k, the algorithm will run at most k times, and split down into 3^k single-digit multiplication operations. And for any number, leading zeros can be placed in front of the number without changing it's value in order to give it $n = 2^k$ for some k. For any factors where $n \leq 2^k$, the number of single-digit multiplication operations with be at most 3^k. Note that $k = \log_2 n$, so if $T(n)$ is the number of single-digit multiplication operations required to multiply any two factors using the Karatsuba algorithm, then $T(n) = 3^{\log_2 n}$. $T(n)$ can also be written as $n^{\log_2 3}$ or $n^{1.585}$.

Practicality

A final note: in a practical sense, the standard algorithm is still very strong and is used to multiply integers of a visible scale. In most implementations, there are multiple different algorithms used based on the size, so when multiplying two large integers it may use several different algorithms while breaking down the multiplication. For example, the GNU C library uses seven different algorithms to multiply similar-size integers based on the size of factors.
For all implementations the practical thresholds to different algorithms change based on different operating systems and hardware.