Complexity Analysis on Integer Multiplication

Asimakis Kydros
SRN: 3881
asimakis@csd.auth.gr

Aristotle University of Thessalonica, CSD

March 2023

Introduction

Multiplication is an extremely common operation. Like addition, very rarely will one not use it when designing an algorithm. In order to keep computations fast, optimizing this operation should be a primary goal.

Many types of multiplication exist: real, complex, matrix. Here, we will examine Integer Multiplication, an important sub-field, useful in cryptography, number theory and computer science in general.

Traditional Addition

Let's backtrack a bit before diving into multiplication. Consider how we add by hand:

$$\begin{array}{ll} & 1234 \\ + & 5678 \\ \text{carry} & 0011 \\ \hline & 6912 \end{array}$$

We add each integer element-by-element, and perform some carry operations if necessary. Overall, this procedure is of $\mathcal{O}(n)$ steps. It is equally easy to see that any method for addition has to be of $\Omega(n)$ steps, as both integers need to be read in their entirety. Thus, the school-grade method for adding two integers is an **optimal algorithm**.

Long Multiplication

Perhaps the same is true for multiplication? Let's consider traditional multiplication (denoted *Long Multiplication* from then on):

$$
\begin{array}{rr}
 & 123 \\
\times & 456 \\
\hline
 & 738 \\
 & 615 \\
+ & 492 \\
\text{carry} & 01100 \\
\hline
 & 56088
\end{array}
$$

Claiming this procedure is optimal isn't as easy or obvious as in the case of addition; here, the steps are of $\mathcal{O}(n^2)$, while the method is still of $\Omega(n)$, at least theoretically.

Kolmogorov Conjecture

While the above predicament is true, historically, every civilization had developed a method for multiplication independently, and every single one of them is of $\mathcal{O}(n^2)$. It would be easy to assume, therefore, that this is also the true lower bound.

Andrei Kolmogorov certainly theorized as much, as signified by his famous conjecture:

Multiplication of two N-digit integers requires $\Omega(N^2)$ *steps*

Romantically at least, it made some sense that a simple, rudimentary procedure would be the optimal solution for both addition and multiplication.

Karatsuba's method

Anatolii Karatsuba would shatter this conjecture about a week later, as he attended the very seminar where Kolmogorov made his announcement and took interest.
The method he employed is very simple and stems from the following observations:

1. Assuming two 2-digit integers of base m
   $$x = x_1 m + x_0 \text{ and } y = y_1 m + y_0$$
   their product gives
   $$xy = x_1 y_1 m^2 + (x_1 y_0 + x_0 y_1) m + x_0 y_0$$
   which requires 4 sub-multiplications.
2. $x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$

Thus it holds in general that

$$xy = x_1 y_1 m^2 + [(x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0] m + x_0 y_0$$

which requires only 3 sub-multiplications.

Karatsuba's method

This is significant, as by splitting each integer in two parts and recursively using this trick, the complexity drops greatly.
Armed with this knowledge, Karatsuba designed the following algorithm:

$karatsuba(x, y, m)$:
    if (x *or* y *is a digit*) *then return* xy
    $pivot \leftarrow \lfloor max\{|x|_m, |y|_m)\}/2 \rfloor$
    $x_1, x_0 \leftarrow split(x, pivot)$
    $y_1, y_0 \leftarrow split(y, pivot)$
    $A \leftarrow karatsuba(x_1, y_1, m)$
    $C \leftarrow karatsuba(x_0, y_0, m)$
    $B \leftarrow karatsuba(x_1 + x_0, y_1 + y_0, m) - A - C$
    return $Am^{2pivot} + Bm^{pivot} + C$

The above procedure recursively calls itself thrice with half the input. By using the Master Theorem, we arrive at

$$T(n) \approx 3T(\frac{n}{2}) + \Theta(n) \Rightarrow T(n) = \Theta(n^{log_2 3}) \approx \Theta(n^{1.58})$$

Generalization by Toom and Cook

Now that the quadratic bound had been broken, many took interest in lowering the complexity even further. Andrei Toom would soon discover the "umbrella" method Toom-k, and Stephen Cook would improve it some years later.
The main idea of Toom-k is dissecting the two integers into polynomials and then multiplying those to get the product. Supposing two large integers $m_{k-1}m_{k-2}....m_1m_0$ and $n_{k-1}n_{k-2}....n_1n_0$ of base B, the following polynomials

$$m(x) = \sum_{i=0}^{k-1} m_i x^i, \; n(x) = \sum_{i=0}^{k-1} n_i x^i$$

give the desired product, when multiplied and for input B:

$$p(B) = m(B)n(B) = mn$$

Toom-k

In order for this to be faster than long multiplication, the selection of B and the computation of p have to be optimal. The key to achieving the first goal is choosing B as

$$B = b^j \ni j = max\{\lfloor \frac{\lfloor log_b m \rfloor}{k} \rfloor, \lfloor \frac{\lfloor log_b n \rfloor}{k} \rfloor\} + 1$$

where b is the original base of the integers and the equation limits the amount of their digits in base-B to at most k.
As for the second, we have to interpolate the coefficients of the product polynomial by computing 2k - 1 of its points from the two integer polynomials.

Toom-k

How this is done optimally is still a subject of debate, but it is ultimately dictated from the following relations:

$$\begin{bmatrix} m(0) \\ . \\ . \\ m(i) \\ . \\ . \\ m(\infty) \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & . & . & 0^{k-1} \\ . & & & & \\ . & & & & \\ i^0 & i^1 & . & . & i^{k-1} \\ . & & & & \\ . & & & & \\ 0 & 0 & . & . & 1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ . \\ . \\ . \\ m_{k-1} \end{bmatrix},$$

$$\begin{bmatrix} m(0)n(0) \\ . \\ . \\ m(i)n(i) \\ . \\ . \\ m(\infty)n(\infty) \end{bmatrix} = \begin{bmatrix} 0^0 & 0^1 & . & . & 0^{2k-2} \\ . & & & & \\ . & & & & \\ i^0 & i^1 & . & . & i^{2k-2} \\ . & & & & \\ . & & & & \\ 0 & 0 & . & . & 1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ . \\ . \\ . \\ p_{2k-2} \end{bmatrix}$$

Toom-3

The (2k - 1)×(2k - 1) matrix is called the **interpolation matrix** and it defines the entire instance of the algorithm.
For example, Toom-3, which is arguably the most popular version of Toom-k, uses the following interpolation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The optimization of Toom-3 has been attempted by many, including Marco Bodrato, who has given the following sequence:

Toom-3 by Bodrato

$calculate\_points(r, B)$:
$r_2, r_1, r_0 \leftarrow digits(m, B)$
$temp \leftarrow r_0 + r_2$
$R[0] \leftarrow r_0$
$R[1] \leftarrow temp + r_1$
$R[-1] \leftarrow temp - r_1$
$R[-2] \leftarrow 2(R[-1] + r_2) - r_0$
$R[\infty] \leftarrow r_2$
$return\ R$
$bodrato(m, n, B)$:
$if(m\ or\ n\ less\ than\ some\ overhead)\ then\ return\ mn$
$M \leftarrow calculate\_points(m, B)$
$N \leftarrow calculate\_points(n, B)$
$P[0] \leftarrow bodrato(M[0], N[0], B)$
$P[1] \leftarrow bodrato(M[1], N[1], B)$
$P[-1] \leftarrow bodrato(M[-1], N[-1], B)$
$P[-2] \leftarrow bodrato(M[-2], N[-2], B)$
$P[\infty] \leftarrow bodrato(M[\infty], N[\infty], B)$
$//interpolate\ and\ recompose$
$p_0 \leftarrow P[0]$
$p_4 \leftarrow P[\infty]$
$p_3 \leftarrow (P[-2] - P[-1])/3$
$p_1 \leftarrow (P[1] - P[-1])/2$
$p_2 \leftarrow P[-1] - P[0]$
$p_3 \leftarrow (p_2 - p_3)/2 + 2p_4$
$p_2 \leftarrow p_2 + p_1 - p_4$
$p_1 \leftarrow p_1 - p_3$
$return\ p_4 B^4 + p_3 B^3 + p_2 B^2 + p_1 B + p_0$

Toom-3 by Bodrato

The above algorithm uses the least possible amount of additions and basic multiplications, and invokes itself recursively five times with a third of the original input.
Applying the Master Theorem, we get

$$T(n) = 5T(\frac{n}{3}) + \Theta(n) \Rightarrow T(n) = \Theta(n^{log_3 5}) = \Theta(n^{\frac{log 5}{log 3}}) \Rightarrow$$

$$T(n) \approx \Theta(n^{1.46})$$

This method is notable, because it encapsulates both long multiplication (Toom-1) and Karatsuba's method (Toom-2) but is ultimately not preferred to implementations using Fourier transforms.

Post-FFT era

The next breakthrough was made by Arnold Schönhage and Volker Straßen. To really appreciate the genius of this method, let's consider long multiplication again, but this time without performing any carrying:

$$
\begin{array}{rr}
 & 1\ \ 2\ \ 3 \\
\times & 4\ \ 5\ \ 6 \\
\hline
 & 6\ 12\ 18 \\
 & 5\ 10\ 15 \\
+ & 4\ \ 8\ 12 \\
\hline
 & 4\ 13\ 28\ 27\ 18
\end{array}
$$

It's as if we convoluted two vectors with elements the digits of each integer. Performing carries on the resulting vector does indeed return the product of 123 and 456, 56088.

Post-FFT era

Thus, a procedure to perform multiplication of two integers could be the following:

$multiply(x, y)$:
$\overrightarrow{x} \leftarrow vectorize(x)$
$\overrightarrow{y} \leftarrow vectorize(y)$
$\overrightarrow{prod} \leftarrow \overrightarrow{x} * \overrightarrow{y}$
$prod \leftarrow stitch(\overrightarrow{prod})$
$return\ prod$

But how do we perform a convolution optimally? The traditional approach is of $\mathcal{O}(n^2)$, which is now unacceptable. The **cyclic convolution theorem** comes to mind:

$$\mathbf{a} * \mathbf{b} = \mathcal{DF}^{-1}\{\mathcal{DF}\{\mathbf{a}\}\mathcal{DF}\{\mathbf{b}\}\}$$

Using FFT to compute the above is inobvious, as complex multiplication of this magnitude is a non-constant-time and very error-prone operation.

Schönhage-Straßen method

Luckily, the Discrete Fourier Transform is an abstract operation that can be performed in any algebraic ring, not just in the complex plane. Schönhage and Straßen use the

$$mod\ (2^N + 1),\ N = 2^{3k} + 1\ where\ k = \#groups$$

ring, which allows for intense optimization by shifting and also usage of the faster **negacyclic convolution**:

$$\mathbf{a} * \mathbf{b} = A^{-1}\mathcal{DF}^{-1}\{\mathcal{DF}\{A\mathbf{a}\}\mathcal{DF}\{A\mathbf{b}\}\}$$
$$A = \{a^j, 0 \leq j < n = |\mathbf{a}| = |\mathbf{b}|\}$$
$$A^{-1} = \{a^{-j}, 0 \leq j < n\}$$
where 'a' is a primitive root of unity of order 2n

through the Number Theoretic FFT (denoted from then on NTT).

Schönhage-Straßen method

Compiling the above, Schönhage and Straßen produced the following algorithm:

$schonhage\_strassen(x, y, N)$:
$if(x\ or\ y\ less\ than\ some\ overhead)\ then\ return\ xy$
$parts \leftarrow 2^k \ni 2^k | N$
$\overrightarrow{x} \leftarrow split(x, parts)$
$\overrightarrow{y} \leftarrow split(y, parts)$
$n \leftarrow n \in \mathbb{N} \geq \frac{2N}{2^k} + k \ni 2^k | n$
$A \leftarrow \frac{jn}{2^k},\ 0 \leq j < |\overrightarrow{x}| = |\overrightarrow{y}|$
$X \leftarrow NTT(A\overrightarrow{x})$
$Y \leftarrow NTT(A\overrightarrow{y})$
$C \leftarrow \emptyset$
$for\ X_j \in X\ and\ Y_j \in Y\ do\ C \leftarrow C \cup schonhage\_strassen(X_j, Y_j, n)$
$\overrightarrow{c} \leftarrow A^{-1} INTT(C)$
$adjust\_negative\_signs(\overrightarrow{c})$
$prod \leftarrow stitch(\overrightarrow{c}, 2^N + 1)$
$return\ prod$

All in all, this algorithm is used mainly for bit multiplication of huge numbers, and it is claimed to be of $\mathcal{O}(NlogNloglogN)$ *bitwise* operations. While not of $\mathcal{O}(nlogn)$, it is the last method that has practical usage today.

Transcending practicality

Further improvements aimed more at lowering the complexity than finding actually useful solutions. Inspired by the work of Schönhage and Straßen , Martin Fürer would power through using normal FFT with complex arithmetic to achieve a complexity of

$$nlog(n)K^{log^* n}$$

while De, Saha, Kurur and Saptharishi would discover a modular ring with N much smaller than the one used by Schönhage and Straßen to achieve a similar complexity to the above.

To appreciate how close to $\mathcal{O}(nlogn)$ the above is, consider that the iterated logarithm

$$log^*(n) = \begin{cases} 1 + log^*(logn), & n > 1 \\ 0, & n \leq 1 \end{cases}$$

tops to 5 for input the *number of atoms in the universe*.

Triumph by Harvey and van Der Hoeven

Fürer made the approach with complex FFT's the norm with his astounding complexity ceiling, and many would take turns lowering the undefined constant K. David Harvey, Joris van Der Hoeven and Grégoire Lecerf would bring that number down to 8, and then 4 under some conditions.

Finally, on 28th of November 2020, Harvey and van Der Hoeven would present to the world the much coveted $\mathcal{O}(nlogn)$ method, which uses FFTs over $\mathbb{C}$, that bypass complex multiplication, in *1729 dimensions*.

In their paper, they make it crystal clear that they made no attempt at optimizing the constants of this method, meaning that it becomes useful for n-bit integers where

$$n \geq n_0 \geq 2^{4096}$$

Obviously no system to date uses this method and no implementations exist. Its discovery is purely theoretical and it righteously belongs in the field of **Galactic Algorithms**.

The end(?)

With the publication of their aforementioned $\mathcal{O}(nlog(n)loglogn)$ method, Schönhage and Straßen conjectured a new lower bound for integer multiplication:

Multiplication of two N-digit integers requires $\Omega(NlogN)$ *steps.*

Although still unproven, it would mean that Harvey-van-Der-Hoeven is the optimal solution. It would surprise nobody if the above turned out to be true, however we have to bear in mind that Kolmogorov was also using his common sense when he announced his own conjecture, so long ago.

Perhaps, then, we might see an even faster method one day?

References

1. Covanov, S. (2015)
   "Putting Fürer Algorithm into Practice"
2. Harvey, D.; van Der Hoeven, J.; Lecerf, G. (2015)
   "Even faster integer multiplication"
3. Harvey, D.; van Der Hoeven, J. (2020)
   "Integer multiplication in time O(nlogn)
4. Yedugani, S.K.; I.S.U.
   "Toom-Cook"
5. Nemean (2021)
   "How Karatsuba's algorithm gave us new ways to multiply"
6. Wikipedia
   "Multiplication algorithm"