

Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS251: Design and Analysis of Algorithms (4-0-0-4-4)

**Multiplication of Large Integers and
Strassen's Matrix Multiplication**

Dr. Shylaja S S

Multiplication of large Integers

Some applications, notably modern cryptology, require manipulation of integers that are over 100 decimal digits long. Since such integers are too long to fit in a single word of a modern computer, they require special treatment. This practical need supports investigations of algorithms for efficient manipulation of large integers. In this section, we outline an interesting algorithm for multiplying such numbers. Obviously, if we use the classic pen-and-pencil algorithm for multiplying two n -digit integers, each of the n digits of the first number is multiplied by each of the n digits of the second number for the total of n^2 digit multiplications. (If one of the numbers has fewer digits than the other, we can pad a shorter number with leading zeros to equal their lengths.) Though it might appear that it would be impossible to design an algorithm with fewer than n^2 digit multiplications, it turns out not to be the case. The miracle of divide-and-conquer comes to the rescue to accomplish this feat.

To demonstrate the basic idea of the algorithm, let us start with a case of two - digit integers, say, 23 and 14. These numbers can be represented as follows:

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0 \text{ and } 14 = 1 \cdot 10^1 + 4 \cdot 10^0$$

Now let us multiply them:

$$\begin{aligned} 23 * 14 &= (2 \cdot 10^1 + 3 \cdot 10^0) * (1 \cdot 10^1 + 4 \cdot 10^0) \\ &= (2 * 1)10^2 + (2 * 4 + 3 * 1)10^1 + (3 * 4)10^0. \end{aligned}$$

The last formula yields the correct answer of 322, of course, but it uses the same four digit multiplications as the pen-and-pencil algorithm. Fortunately, we can compute the middle term with just one digit multiplication by taking advantage of the products $2 * 1$ and $3 * 4$ that need to be computed anyway:

$$2 * 4 + 3 * 1 = (2 + 3) * (1 + 4) - 2 * 1 - 3 * 4.$$

Of course, there is nothing special about the numbers we just multiplied. For any pair of two-digit integers $a = a_1a_0$ and $b = b_1b_0$, their product c can be computed by the formula

$$c = a * b = c_2 10^2 + c_1 10^1 + c_0,$$

where

$c_2 = a_1 * b_1$ is the product of their first digits,

$c_0 = a_0 * b_0$ is the product of their second digits,

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ is the product of the sum of the a 's digits and the sum of the b 's digits minus the sum of c_2 and c_0 .

Now we apply this trick to multiplying two n -digit integers a and b where n is a positive even number. Let us divide both numbers in the middle-after all, we promised to take advantage of the divide-and-conquer technique. We denote the first half of the a 's digits by a_1 and the second half by a_0 ; for b , the notations are b_1 and b_0 , respectively. In these notations, $a = a_1a_0$ implies that $a = a_110^{n/2} + a_0$, and $b = b_1b_0$ implies that $b = b_110^{n/2} + b_0$. Therefore, taking advantage of the same trick we used for two-digit numbers, we get

$$\begin{aligned} c &= a * b = (a_110^{n/2} + a_0) * (b_110^{n/2} + b_0) \\ &= (a_1 * b_1)10^n + (a_1 * b_0 + a_0 * b_1)10^{n/2} + (a_0 * b_0) \\ &= c_210^n + c_110^{n/2} + c_0 \end{aligned}$$

where

$c_2 = a_1 * b_1$ is the product of their first halves

$c_0 = a_0 * b_0$ is the product of their second halves

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ is the product of the sum of the a 's halves and the sum of the b 's halves minus the sum of c_2 and c_0

If $n/2$ is even, we can apply the same method for computing the products c_2 , c_0 , and c_1 . Thus, if n is a power of 2, we have a recursive algorithm for computing the product of two n -digit integers. In its pure form, the recursion is stopped when n becomes one. It can also be stopped when we deem n small enough to multiply the numbers of that size directly.

How many digit multiplications does this algorithm make? Since multiplication of n -digit numbers requires three multiplications of $n/2$ -digit numbers, the recurrence for the number of multiplications $M(n)$ will be

$$M(n) = 3M(n/2) \text{ for } n > 1, M(1) = 1.$$

Solving it by backward substitutions for $n = 2^k$ yields

$$\begin{aligned} M(2^k) &= 3M(2^{k-1}) = 3[3M(2^{k-2})] = 3^2M(2^{k-2}) \\ &= \dots = 3^i M(2^{k-i}) = \dots = 3^k M(2^{k-k}) = 3^k \end{aligned}$$

$$\text{Since } k = \log_2 n: M(n) = 3^{\log_2 n} = n^{\log_2 3} = n^{1.585}$$

Strassen's Matrix Multiplication

Now that we have seen that the divide-and-conquer approach can reduce the number of one-digit multiplications in multiplying two integers, we should not be surprised that a similar feat can be accomplished for multiplying

matrices. Such an algorithm was published by V. Strassen in 1969. The principal insight of the algorithm lies in the discovery that we can find the product C of two 2-by-2 matrices A and B with just seven multiplications as opposed to the eight required by the brute-force algorithm. This is accomplished by using the following formulas:

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

where,

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$

$$m_2 = (a_{10} + a_{11}) * b_{00}$$

$$m_3 = a_{00} * (b_{01} - b_{11})$$

$$m_4 = a_{11} * (b_{10} - b_{00})$$

$$m_5 = (a_{00} + a_{01}) * b_{11}$$

$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$$

$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$$

Thus, to multiply two 2-by-2 matrices, Strassen's algorithm makes seven multiplications and 18 additions/subtractions, whereas the brute-force algorithm requires eight multiplications and four additions. These numbers should not lead us to multiplying 2-by-2 matrices by Strassen's algorithm. Its importance stems from its asymptotic superiority as matrix order n goes to infinity.

Let A and B be two n-by-n matrices where n is a power of two. (If n is not a power of two, matrices can be padded with rows and columns of zeros.) We can divide A, B, and their product C into four n/2-by-n/2 submatrices each as follows:

$$\left[\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array} \right] = \left[\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array} \right] * \left[\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array} \right]$$

It is not difficult to verify that one can treat these submatrices as numbers to get the correct product. For example, C_{00} can be computed either as

$A_{00} * B_{00} + A_{01} * B_{10}$ or as $M_1 + M_4 - M_5 + M_7$ where M_1, M_4, M_5 , and M_7 are found by Strassen's formulas, with the numbers replaced by the corresponding submatrices. If the seven products of $n/2$ -by- $n/2$ matrices are computed recursively by the same method, we have Strassen's algorithm for matrix multiplication.

Let us evaluate the asymptotic efficiency of this algorithm. If $M(n)$ is the number of multiplications made by Strassen's algorithm in multiplying two n -by- n matrices (where n is a power of 2), we get the following recurrence relation for it:

$$M(n) = 7M(n/2) \text{ for } n > 1, M(1) = 1.$$

Since $n = 2^k$,

$$\begin{aligned} M(2^k) &= 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2M(2^{k-2}) = \dots \\ &= 7^i M(2^{k-i}) \dots = 7^k M(2^{k-k}) = 7^k \end{aligned}$$

Since $k = \log_2 n$,

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$$

which is smaller than n^3 required by the brute-force algorithm.

Since this saving in the number of multiplications was achieved at the expense of making extra additions, we must check the number of additions $A(n)$ made by Strassen's algorithm. To multiply two matrices of order $n > 1$, the algorithm needs to multiply seven matrices of order $n/2$ and make 18 additions of matrices of size $n/2$; when $n = 1$, no additions are made since two numbers are simply multiplied.

These observations yield the following recurrence relation:

$$A(n) = 7A(n/2) + 18(n/2)^2 \text{ for } n > 1, A(1) = 0$$

According to the Master Theorem, $A(n) \in \Theta(n^{\log_2 7})$. In other words, the number of additions has the same order of growth as the number of multiplications.

This puts Strassen's algorithm in $\Theta(n^{\log_2 7})$, which is a better efficiency class than $\Theta(n^3)$ of the brute-force method.