1 Multiplication of integers

1.1 Addition and multiplication of integers

Given integers $a, b \in \mathbb{Z}$ we will want to compute $a + b$ and $a \times b$. For simplicity we will not bother with signs and consider positive integers.

$$\begin{array}{cccccc} & 1 & 1 & & & \\ 6 & 5 & 5 & 3 & 6 & \\ & & 7 & 9 & 2 & + \\ \hline 6 & 6 & 3 & 2 & 8 & \end{array}$$

We have to apply a 'single digit addition with carry' for every digit in the longest number. The time this algorithm takes to complete is a function in the *length* N of the input. For this algorithm this is the sum of the lengths of the two numbers (note: this is roughly $\log_{10} a + \log_{10} b$). Although the exact time in seconds required depends on the computer, the *complexity class* is uniquely defined (given the model of computation). Usually the model of computation will be either that of a *Multitape Turing Machine* or a *Random Access Machine*. An algorithm generally has the same time complexity in both models. We will not worry about such details and take a more intuitive approach to the machine.

Definition 1.1. For a function $f \in \mathcal{C} = \mathrm{Map}(\mathbb{R}_{\geq 0}, \mathbb{R}_{\geq 0})$ we define

$$O(f) = \left\{ g \in \mathcal{C} \,\middle|\, \limsup_{x \to \infty} \frac{g(x)}{f(x)} < \infty \right\}.$$

The complexity of addition is in $O(N)$, which is optimal. The algorithm depends on the base of the numbers, but the complexity is the same for all choices.

Multiplication is more tricky. An algorithm is the following: (See figure). With 10 additions compute the following in time $O(N)$. Then apply $O(N)$ additions for a total complexity of $O(N^2)$.
Two goals:

$$
\begin{array}{rcl}
1 \cdot 1337 & = & 1337 \\
2 \cdot 1337 & = & 2674 \\
3 \cdot 1337 & = & 4011 \\
4 \cdot 1337 & = & 5348 \\
 & \vdots & \\
9 \cdot 1337 & = & 12033
\end{array}
\qquad
\begin{array}{ccccccc}
 & & & 4 & 2 & 0 & \\
 & & 1 & 3 & 3 & 7 & \times \\
\hline
 & & 0 & 0 & 0 & 0 & \\
 & 2 & 6 & 7 & 4 & & \\
5 & 3 & 4 & 8 & & & + \\
\hline
5 & 6 & 1 & 5 & 4 & 0 &
\end{array}
$$

1. Reduce number of multiplications needed.
2. Find faster multiplication algorithm.

1.2 Complex number multiplication

Computing

$$(a + bi) \times (c + di) := (a \times c - b \times d) + (b \times c + a \times d)i$$

takes 4 multiplications and 2 additions. However, with

$$
\begin{aligned}
s &= c \times (a + b) \\
t &= a \times (d - c) \\
u &= b \times (c + d)
\end{aligned}
$$

we get

$$(a + bi) \times (c + di) = (s - u) + (s + t)i$$

which is 3 multiplications and 5 additions.

1.3 Karatsuba multiplication

We will compute $a \times b$ for integers a and b given in base B (often $B = 2$ or $B = 10$). Let $m = B^k$ for some small k for which $a, b \leq m^2$. Thus without computation we may write

$$a = a_1 m + a_0 \quad \text{and} \quad b = b_1 m + b_0.$$

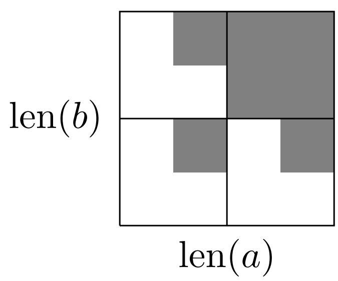
for $0 \le a_0, a_1, b_0, b_1 < m$. Think writing

$$65536 = 65 \cdot 10^3 + 536.$$

Then

$$\begin{aligned}
&(ma_1 + a_0) \times (mb_1 + b_0) \\
&= m^2(a_1 \times b_1) + m(a_1 \times b_0 + a_0 \times b_1) + a_0 \times b_0 \\
&= m^2 A + m[(a_1 + a_0) \times (b_1 + b_0) - A - B] + B
\end{aligned}$$

Giving 3 multiplications and 6 additions. The multiplication is done on numbers half the length and can be done inductively. The resulting complexity is



$$\approx N^2 \cdot (\tfrac{3}{4})^{\log_2 N} = N^2 \cdot N^{\log_2 (3/4)} = N^{\log_2 3}.$$

1.4 Polynomial multiplication

Effectively, Karatsuba treated the integers as linear polynomials in $m = B^k$. We have a general strategy:

1. Write $a = A(m)$ and $b = B(m)$ as polynomials A and B for appropriate m.
2. Compute $A \times B$.
3. Evaluate $A \times B$ at m.

The first and last step are fast because the numbers are represented in base B, roughly $O(N)$. It suffices to find a fast algorithm for multiplying polynomials.

1.5 Fourier transform

Let R be a commutative ring with primitive n-th root of unity ζ and $n \in R^*$. (Think $R = \mathbb{C}$ or $R = \mathbb{Z}/w\mathbb{Z}$ for some w)

Definition 1.2. We define the *Fourier transform*

$$\begin{aligned} \mathcal{F}_\zeta : R^n &\to R^n \\ (a_i)_i &\mapsto \Big(\sum_{k=0}^{n-1} a_k \zeta^{jk}\Big)_j. \end{aligned}$$

Recall that

$$\sum_{k=0}^{n-1} \zeta^{jk} = \begin{cases} n & \text{if } j = 0 \\ 0 & \text{otherwise} \end{cases}.$$

It follows that

Lemma 1.3. *We have $\mathcal{F}_\zeta \circ \mathcal{F}_{\zeta^{-1}} = n$ and $\mathcal{F}_\zeta$ is invertible.* □

Remark 1.4. For $a, w \in \mathbb{Z}$ we may decide $a \in (\mathbb{Z}/w\mathbb{Z})^*$ and if so compute $b \in \mathbb{Z}$ such that $ab \equiv 1 \mod w$ in linear time. We use the extended Euclidean algorithm (Algebra 1) to solve the equation $ab + wc = \gcd(a, w)$ for $b, c \in \mathbb{Z}$.

Definition 1.5. For $a, b \in R^n$ we define their involution

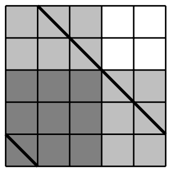
$$a * b = \Big(\sum_{i+j \equiv k \ (n)} a_i \cdot b_j\Big)_k$$

This operation resembles multiplication of polynomials.

Proposition 1.6. *Let $R[X]_d = \{f \in R[X] \mid \deg(f) < d\}$. For $d \leq n$ we have a natural inclusion $\mathcal{C} : R[X]_d \to R^n$. If $2d \leq n + 1$, then*

$$\mathcal{C}(f \cdot g) = \mathcal{C}(f) * \mathcal{C}(g).$$

Proof. As can be seen in the picture, the additional terms in the sum are all zero.



∎

Proposition 1.7. *For $a, b \in R^n$ we have*

$$\mathcal{F}(a * b) = \mathcal{F}(a) \cdot \mathcal{F}(b). \quad \square$$

Suppose now that we can efficiently compute $\mathcal{F}$.

Algorithm 1.8 (Schönhage–Strassen)**.** Let $a, b \in \mathbb{Z}_{>0}$ be represented in base B in ℓ digits.

1. (*) Choose $d, k \in \mathbb{Z}_{>0}$ such that $dk \geq \ell$ and write $m = B^k$.
2. Write

$$a = A(m) = \sum_{i=0}^{d-1} a_i m^i \quad \text{and similarly} \quad b = B(m)$$

with $0 \leq a_i, b_i < m$ and $A, B \in \mathbb{Z}[M]$.

3. (*) Choose $w, n \in \mathbb{Z}_{>0}$ with $w \geq dm^2$ and $n \geq 2d - 1$, and $\zeta \in \mathbb{Z}/w\mathbb{Z}$ an n-th root of unity.
4. Compute $\hat{A} = \mathcal{F}_\zeta(A)$ and $\hat{B} = \mathcal{F}_\zeta(B)$ as elements of $(\mathbb{Z}/w\mathbb{Z})^n$.
5. Compute $\overline{C} = \mathcal{F}^{-1}(\hat{A} \cdot \hat{B})$.
6. Lift $\overline{C}$ to $C \in \mathbb{Z}[M]$ with coefficients in $\{0, \ldots, w-1\}$.
7. Evaluate $C(m)$.

The choice of w is such that the unique lift of $\overline{C}$ indeed equals $A \cdot B$. Namely, the coefficients of $A \cdot B$ are less than dm^2.

(*3) If we take $n \geq 2d-1$ prime or a power of two, and $w = 2^n + 1$ (Fermat number), then $2 \in \mathbb{Z}/w\mathbb{Z}$ is an n-th root of unity: $2^n \equiv -1$

so $\mathrm{ord}(2) \mid 2n$ and clearly $\mathrm{ord}(2) > n$. This algorithm has been improved several times simply by coming up with smaller constants w and n.

(*1) We want w to be small so that naive multiplication is sufficient modulo w. We take $k \in \Theta(\log \ell) = \Theta(\log N)$, so that $w \in O(\ell^3)$ and multiplication takes $O(\log\log \ell)$ time. It suffices to show that the Fourier transform can be done in $O(\ell \log \ell)$ multiplications, so that the resulting complexity is $O(N \log N \log\log N)$.

1.6 Cooley–Tukey algorithm

This algorithm was originally formulated by Gauss and later independently discovered by Cooley and Tukey.

Suppose that n is a power of two.

Definition 1.9. Consider functions $\mathcal{E}, \mathcal{O} : R^n \to R^n$ given by

$$\mathcal{E}(a) = \Big(\sum_{k=0}^{n/2-1} a_{2k} \zeta^{2ki} \Big)_i \quad \text{and} \quad \mathcal{O}(a) = \Big(\sum_{k=0}^{n/2-1} a_{2k+1} \zeta^{2ki} \Big)_i .$$

First note that

$$\mathcal{F}(a) = \mathcal{E}(a) + (\zeta^{2i})_i \cdot \mathcal{O}(a). \tag{1}$$

Secondly, we have

$$\mathcal{E}(a)_i = \mathcal{E}(a)_{i+n/2} \quad \text{and} \quad \mathcal{O}(a)_i = \mathcal{O}(a)_{i+n/2}. \tag{2}$$

since ζ^2 is an $(n/2)$-th root of unity. Thus it suffices to compute the first $n/2$ coefficients of $\mathcal{E}(a)$ and $\mathcal{O}(a)$. However, this is just a $(n/2)$-dimensional Fourier transform:

$$\mathcal{E}(a)_{0,1,\dots,n/2-1} = \mathcal{F}_{\zeta^2}(a_0, a_2, \dots, a_{n-2}) \tag{3}$$

$$\mathcal{O}(a)_{0,1,\dots,n/2-1} = \mathcal{F}_{\zeta^2}(a_1, a_3, \dots, a_{n-1}) \tag{4}$$

Thus the n-dimensional Fourier transform can be computed as two $(n/2)$-dimensional Fourier transformations, n additions and $2n$ multiplications in R. The entire algorithm requires $O(n \log n)$ multiplications. Hence the complexity is $O(n \log n \cdot \log^2 w)$.

1.7 Harvey–van der Hoeven

In 2021 Harvey and van der Hoeven published a $O(N \log N)$ algorithm, which can be proven to be optimal. It uses multi-dimensional Fourier transforms and approximate computation in $\mathbb{C}$.