Background

begin rmk Throughout this article I will use the convention (common in digital signal processing literature) to call j the imaginary unit i.e. $j = e^{j\pi/2}$.

Also, I assume that you are familiar with complex numbers, in particular Euler's formula: $e^{j\theta} = \cos(\theta) + j\sin(\theta)$.

I will also use the convention that w_n denotes the n-th root of unity. That is,

$$w_n = e^{j\frac{2\pi}{n}}.$$

Note that $w_n^n = e^{j2\pi} = 1$, but not so for any other lower powers, so this is indeed the n-th root of unity. end rmk

Rough Idea

We can represent integers as polynomials:

$$P(z) = \sum_{i=0}^{n-1} p_i z^i$$

$$Q(z) = \sum_{i=0}^{n-1} q_i z^i$$

The integers are $P(B), Q(B)$ where B is the base (typically $B = 10$). The length of $P(B), Q(B)$ is n digits.

We want the product $R(z) = P(z)Q(z)$ specifically $R(B)$. Note that a polynomial is defined fully by its output on n points, so we can construct $R(z)$ via interpolation if we have it on $2n$ points.

Our method will proceed as follows:

- We will compute the polynomials values at the complex $2n$-th roots of unity, i.e. $w_{2n}^k = e^{\frac{2j\pi k}{2n}} \quad k \in \{0, 1, \ldots 2n-1\}$.
- Then we will pointwise multiply the polynomials P, Q to get the values of R at the roots of unity.
- Then we can figure out the coefficients of R from its values at the roots of unity.

It turns out that this makes the multiplication take time $O(n \log n)$. yay!!

Discrete Fourier transform:

The tool that makes this all possible is the Discrete Fourier Transform (DFT). The DFT is a really interesting change of basis. It is a way of interpretting a signal in the time domain in the frequency domain. That is, we break up a time signal into its different frequency components, a sum of complex exponentials.

It's just like the decomposition of white light into all the frequencies, except better, because we are doing it for vectors in $\mathbb{C}^n$.

The DFT takes in a vector $(x_0, x_1, \ldots, x_{n-1}) \in \mathbb{C}^n$ and outputs a vector $X \in \mathbb{C}^n$ with

$$X[k] = (w_k|x) = \sum_{i=0}^{n-1} x[i] w_{2n}^{ik}$$

Fast Fourier Transform

It turns out that there is a very fast algorithm to compute the DFT, much faster than $O(n^2)$. It is called the Fast Fourier Transform (FFT).

DFT can be computed in $O(n \log n)$ time (much better than the naive $O(n^2)$ algorithm). This is because of the following remarkable fact:

Let $x[i]$ be a signal of length $2n$.

Let

$$x_e[i] = x[2i] \quad i = 0, 1, \ldots, n-1.$$

Let

$$x_o[i] = x[2i+1] \quad i = 0, 1, \ldots, n-1.$$

Observe the following about the DFT of $x[i]$.

$$X[k] = \sum_{i=0}^{2n-1} x[i] e^{j \frac{2\pi}{2n} ik}.$$

$$X[k] = \sum_{i=0}^{n-1} x_e[i] e^{j \frac{2\pi}{2n} 2ik} + x_o[i] e^{j \frac{2\pi}{2n} (2i+1)k}.$$

$$X[k] = \sum_{i=0}^{n-1} x_e[i] e^{j \frac{2\pi}{n} ik} + e^{j \frac{\pi}{n} k} \sum_{i=0}^{n-1} x_o[i] e^{j \frac{2\pi}{n} ik}.$$

This is great, because we reduced the problem of finding the DFT of a length n signal to computing the DFT of 2 length $n/2$ signals (and then adding them with a weight on the odd DFT). There are more optimizations that go into the FFT that I haven't discussed here, but this is the gist of it. If the sequence is of length $n = 2^c$ then we can just keep recursing, hence the $O(n \log n)$ runtime. You can get $O(n \log n)$ running-time even for non power of 2 signal sizes, but it's more *complex* [;)]. Also you can do some fancy group theory stuff to make it faster according to wikipedia.

Using this for the Integer multiplication problem

Now I formally outline Strassen's fast integer multiplication algorithm

First compute the polynomials at the roots of unity We compute the fourier transform (with FFT) of the vector $(p_0, p_1, \ldots, p_{n-1}, 0, 0, \ldots, 0)$ of the coefficients of P followed by n zeros, and of the vector of coefficients for Q followed by n zeros. This gives us the values of $P(w_{2n}^k), Q(w_{2n}^k) \quad k \in \{0, 1, \ldots, 2n-1\}$ because

$$P(w_{2n}^k) = \sum_{i=0}^{n-1} p_i z^i|^{z=w_{2n}^k} = \sum_{i=0}^{n-1} p_i w_{2n}^{ki} = \sum_{i=0}^{n-1} p_i w_{2n}^{ki} + \sum_{i=n}^{2n-1} 0 w_{2n}^{ki}$$

Which is simply the DFT of the vector of coefficients of P padded on the right with n zeros. The argument is exactly the same for how we evaluate Q at the roots of unity.

Next we obtain the values of the product function at the roots of unity This is trivial, it takes $O(n)$ time. We just do

$$Z(w_{2n}^k) = P(w_{2n}^k) \cdot Q(w_{2n}^k) \quad k \in \{0, 1, \ldots, 2n-1\}.$$

Now we compute the coefficients of the product polynomial from its values at the roots of unity

To do this, we must compute the "conjugate fourier transform", which can also be computed with FFT. That is, we retrieve the k-th coefficient r_k of R by taking $\frac{1}{2n} T(\overline{w_{2n}^k})$

The End