Polynomial multiplication:

Consider 2 polynomials $A(x)$ & $B(x)$ where:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$$

$$\& \; B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_d x^d$$

We want to compute their product polynomial:

$$C(x) = A(x)B(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{2d} x^{2d}$$

where for $1 \le k \le 2d$

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0$$

Here's an example:

$$A(x) = 1 + 2x + 3x^2 \;\;\&\;\; B(x) = 2 - x + 4x^2$$

then $C(x) = A(x)B(x) = 2 + 3x + 8x^2 + 5x^3 + 12x^4$

We'll take as input the two vectors of coefficients:

$$a = (a_0, a_1, \ldots, a_d) \;\;\&\;\; b = (b_0, b_1, \ldots, b_d)$$

and our goal is to compute the vector:

$$c = (c_0, c_1, \ldots, c_{2d}).$$

The vector c is called the convolution of a & b which is denoted as: $c = a * b$.

Naive approach: $O(k)$ time for $c_k \Rightarrow O(d^2)$ total time

Better approach: Using FFT = fast Fourier transform, we'll see it can be done in $O(d \log d)$ time. It's a beautiful divide & conquer algorithm.

Some applications:

– Pattern matching:

Given binary strings $s = s_0 s_1 \dots s_{n-1}$ (string)

& $p = p_0 p_1 \dots p_{m-1}$ (pattern)

where $n \geq m$

Goal: Find all occurrences of p in s.

In other words find all positions k where:

$$s_k s_{k+1} \dots s_{k+m-1} = p_0 p_1 \dots p_{m-1}$$

Example: $s = aaabbabbabbaa$ $\quad p = abbab$

Idea: map $a \to -1$ & $b \to +1$

then in our example: $s = (-1,-1,-1,1,1,-1,1,1,-1,1,1,-1,-1)$

& $p = (-1,1,1,-1,1)$

Consider 2 strings p & q of length m,

Say $p = (-1,1,1,-1,1)$ & $q = (-1,-1,1,-1,1)$

then $p \cdot q = 1 - 1 + 1 + 1 + 1 = 3$

dot product

we get $+1$ if they agree in that coordinate

& -1 if they differ

In general, if $p=q$ then $p\cdot q=m$
and if $p\neq q$ then $p\cdot q<m$

Thus, for a specific k, to check if:

$$s_k s_{k+1} \cdots s_{k+m-1} = p_0 p_1 \cdots p_{m-1}$$

we can take $p=(p_0, \dots, p_{m-1})$ & $q=(s_k, \dots, s_{k+m-1})$

& check if $p\cdot q=m$?

this is equivalent to: $s_k p_0 + s_{k+1}p_1 + \cdots + s_{k+m-1}p_{m-1} = m$?

An obvious thing to try is setting $s=(s_0, s_1, \dots, s_{n-1})$
& $p=(p_0, p_1, \dots, p_{m-1})$

& look at $c=s*p$.

Let's look at c_{k+m-1} to see if it matches

We have: $c_{k+m-1} = s_{k+m-1}p_0 + s_{k+m-2}p_1 + \cdots + s_k p_{m-1}$

this is not exactly right — need to <u>reverse</u> p.

So let $p'=(p_{m-1}, p_{m-2}, \dots, p_1, p_0)$

Look at $c=s*p'$.

then $c_{k+m-1} = s_{k+m-1}p_{m-1} + s_{k+m-2}p_{m-2} + \cdots + s_k p_0$

So if $c_{k+m-1}=m$ then $s_k s_{k+1} \dots s_{k+m-1} = p_0 p_1 \dots p_{m-1}$

Thus, we check c to find all entries which $=m$.

Linear filtering: Replace a datapoint by a linear combination of neighboring points.

Used for: reducing noise, adding effects, etc.

Examples:

Mean filter: (also called box smoothing or moving average) replace a point by the average of itself & the $2M$ neighboring points (where M is a parameter)

So for data $Y = (Y_1, \ldots, Y_n)$ we replace it by: $\hat{Y}_j = \frac{1}{2M+1} \sum_{i=-M}^{M} Y_{j+i}$

This can be computed by taking the convolution of $Y = (Y_1, \ldots, Y_n)$ with $f = \frac{1}{2M+1} \underbrace{(1, 1, \ldots, 1)}_{\substack{2M+1 \\ \text{coordinates}}} = \left(\frac{1}{2M+1}, \ldots, \frac{1}{2M+1}\right)$

Gaussian filter: reduce outliers by replacing datapoints by "Gaussian average" of neighboring points.

So for $Y = (Y_1, \ldots, Y_n)$

replace it by: $\hat{Y}_j = \frac{1}{Z} \sum_{i=-M}^{M} e^{-i^2} Y_{j+i}$

where $Z = \sum_{i=-M}^{M} e^{-i^2}$ is just a normalizing factor.

This can be can computed by taking the convolution of
$Y = (y_1, \dots, y_n)$ with

$$f = \frac{1}{Z}\left(e^{-M^2}, e^{-(M-1)^2}, \dots, e^{-1}, 1, e^{-1}, \dots, e^{-M^2}\right)$$

<u>Gaussian Blur</u>: 2-dimensional Gaussian smoothing applied to an image to blur it.

Many other examples, widely used.

Back to multiplying polynomials:

Two ways to represent a polynomial $A(x) = a_0 + a_1 x + \cdots + a_d x^d$

1) Coefficients: $a_0, a_1, \ldots, a_d$

or 2) Values: $A(x_0), A(x_1), \ldots, A(x_d)$

Lemma: A degree d polynomial is uniquely characterized by its values at any $d+1$ distinct points.

example: a line has $d=1$ & is defined by any two points on the line.

We assume the input/output is in coefficients representation, but the values representation is useful for multiplying polynomials:

Given $A(x_0), A(x_1), \ldots, A(x_{2d})$
& $B(x_0), B(x_1), \ldots, B(x_{2d})$

Then $C(x_i) = A(x_i) B(x_i)$ for $i = 0, 1, \ldots, 2d$
& this defines $C(x)$.

Need to convert between: coefficients $\Leftrightarrow$ values

FFT: does this conversion for carefully chosen set of points.

Consider polynomial $A(x) = a_0 + a_1 x + \cdots + a_n x^{n-1}$
where n is a power of 2.

We are given $a = (a_0, a_1, \ldots, a_{n-1})$
& we want to output $A(x_1), A(x_2), \ldots, A(x_{2n})$
for $2n$ points $\underbrace{x_1, \ldots, x_{2n}}$ that we choose.

How should we choose these points

Key idea: Suppose we have n points $x_1, \ldots, x_n$
& the other n are opposites,

So: $x_{n+1} = -x_1$, $x_{n+2} = -x_2, \ldots, x_{2n} = -x_n$

Look at $A(x_i)$ & $A(-x_i) = A(x_{n+i})$
- same for even terms $a_{2k} x^{2k}$
- opposite for odd terms $a_{2k+1} x^{2k+1}$

Thus split $A(x)$ into even & odd terms.

Let $a_{even} = (a_0, a_2, a_4, \ldots, a_{n-2})$

$$A_{even}(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{n-2} y^{\frac{n-2}{2}}$$

& $a_{odd} = (a_1, a_3, a_5, \ldots, a_{n-1})$

$$A_{odd}(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{n-1} y^{\frac{n-2}{2}}$$

Observe that $A(x) = A_{even}(x^2) + x\,A_{odd}(x^2)$

<u>Example:</u> $A(x) = 5 - 3x + 4x^3 - 2x^4 + 7x^5 + 6x^6 - x^7$

$a_{even} = (5, 0, -2, 6)$, $a_{odd} = (-3, 4, 7, -1)$

$A_{even}(y) = 5 - 2y^2 + 6y^3$, $A_{odd}(y) = -3 + 4y + 7y^2 - y^3$

$$A(x_i) = A_{even}(x_i^2) + x_i A_{odd}(x_i^2)$$
$$A(x_{n+i}) = A(-x_i) = A_{even}(x_i^2) - x_i A_{odd}(x_i^2)$$

So given $A_{even}(y_1), \dots, A_{even}(y_n)$
& $A_{odd}(y_1), \dots, A_{odd}(y_n)$

where $y_1 = x_1^2, \dots, y_n = x_n^2$

then in $O(n)$ time we get:

$$A(x_1), \dots, A(x_n), \underset{A(-x_1)}{\underset{\shortparallel}{A(x_{n+1})}}, \dots, \underset{A(-x_n)}{\underset{\shortparallel}{A(x_{2n})}}$$

Note $A_{even}(y)$ & $A_{odd}(y)$ are of degree $\frac{n-2}{2} = \frac{n}{2} - 1$
whereas $A(x)$ has degree $n-1$.

So to get $A(x)$ of deg. $n-1$ at $2n$ points
we need $A_{even}(y)$ & $A_{odd}(y)$ each of deg. $\frac{n}{2} - 1$
at n points

$\Rightarrow$ Divide & conquer!

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n).$$

But what about the next level of recursion?

we have $x_1^2, x_2^2, \ldots, x_n^2$

& we want that:

$$x_1^2 = -x_{\frac{n}{2}+1}^2$$

$$x_2^2 = -x_{\frac{n}{2}+2}^2$$

$$\vdots$$

$$x_{\frac{n}{2}}^2 = -x_n^2$$

But this is impossible because $x_1^2 \geq 0$

& $x_{\frac{n}{2}+1}^2 \geq 0$

Unless we use complex numbers.