Today:
- Polynomial multiplication
- Fast Fourier Transform (FFT)
- Cross-correlation

Poly mult:
\[ \text{Input: } A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{d-1} x^{d-1} \]
\[ \text{Input: } B(x) = b_0 + b_1 x + \cdots + b_{d-1} x^{d-1} \]

Output: coefficients of
\[ C(x) = c_0 + c_1 x + \cdots + c_{2d-2} x^{2d-2} \]

Define \( N = 2d-1 \)

\( \text{will treat } \frac{A,B,C}{\text{as having degree } \leq N} \)
\( \text{(can pad } A,B \text{ w/ 0 coeff.)} \)
Relationship b/w poly and int multiplication

Given int $\alpha, \beta$, want $Y = \alpha \times \beta$

$\alpha = \alpha_{n-1} \alpha_{n-2} \ldots \alpha_0$ ( $\alpha_i$ are digits 0-9)

$\beta = \beta_{n-1} \ldots \beta_0$

$A(x) = \alpha_0 + \alpha_1x + \ldots + \alpha_{n-1}x^{n-1}$ ( $\alpha = A(10)$)

$B(x) = \beta_0 + \ldots + \beta_{n-1}x^{n-1}$ ( $\beta = B(10)$)

Want $Y = \alpha \beta = (A \cdot B)(10)$
Algorithm 1: "Straightforward" also.

$$C(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{n-1} x^{n-1}$$

- $c_0 = a_0 \cdot b_0$
- $c_1 = a_0 \cdot b_0 + a_1 \cdot b_0$

$$c_k = \sum_{j=0}^{k} a_j \cdot b_{k-j}$$

- Loop over $k = 0 \to n-1$
  - Compute $c_k$ w/ a loop from $j = 0 \to k$

$\Rightarrow$ Total time is $O(N^2)$

Also: computing each of $c_0 \ldots c_N$ requires $\geq \frac{N}{2} \cdot \frac{N}{2} = \frac{N^2}{4}$ flops

$\Rightarrow$ total # flops $\geq \frac{N}{2} \cdot \frac{N}{4} = \frac{N^2}{8}$ flops

$\Rightarrow \Omega(N^2)$ flops $\Rightarrow \Theta(N^2)$ flops
\[ A(x) = a_0 + a_1 x + a_2 x^2 + \cdots \]
\[ B(x) = b_0 + b_1 x + b_2 x^2 + \cdots \]

\[ A(x) = A_2(x) + x^{\frac{n}{2}} \cdot A_n(x) \]
\[ B(x) = B_2(x) + x^{\frac{n}{2}} \cdot B_n(x) \]

Karatsuba trick \[ \Rightarrow T(N) \leq 3T\left(\frac{N}{2}\right) + \Theta(N) \]
\[ = \Theta\left( N^{\log_2 3} \right) \]
Polynomial Interpolation

A degree \( N \) polynomial is fully determined by its evaluations on \( N \) distinct points.

Rather than obtain the coefficients of \( C \) directly by

\[
C(x) = A(x) \cdot B(x)
\]

we will determine multipliers \( A \) and \( B \), we will determine

\[
C(x) = \frac{C(x_N)}{C(x_N-x)} - C(x_{N-1})\text{ for distict } x_i; C(x_0), C(x_N).
\]

will compute evaluation of \( A \) and \( B \) on \( N \) distinct

points each, multiply them, then "interpolate"

to get back coefficients of \( C \) from evals of \( C \).
Why does interpolation work?

\[
V c = y \implies c = V^{-1} y
\]

Such \( V \) is called "Vandermonde" matrix.

Fact: \( \det(V) = \prod_{i<j} (x_i - x_j) \)
Types:
1. Discrete Fourier Transform (DFT) is a matrix
2. Fast Fourier Transform (FFT) is an algorithm

\[ W = \frac{e^{j2\pi k/N}}{N} \quad \text{complex number} \quad F_{ij} = (w^i)^j = w^{ij} \]

Complex recap
\[ z = a + j\cdot b = r \cdot e^{j\theta} \]
\[ r = \sqrt{a^2 + b^2} \]
\[ \theta = \tan^{-1} \left( \frac{b}{a} \right) \]

\[ e = \cos \theta + j \cdot \sin \theta \]
\( W \) (example, \( N = 8 \))

\[
w = e^{\frac{2\pi i}{N}} = e^{\frac{2\pi i}{8}} = e^{\frac{\pi i}{4}}
\]

will evaluate polynomial \( p(x) \) at points \( (w, w^2, \ldots, w^{N-1}) \)

\[
F = DFT ( \ast \text{Vandermonde matrix} )
\]

\[
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & w & w^2 & \cdots & w^{N-1} \\
1 & w^2 & w^4 & \cdots & w^{2(N-1)} \\
1 & w^3 & w^6 & \cdots & w^{3(N-1)} \\
1 & w^4 & w^8 & \cdots & w^{4(N-1)}
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
1 \\
w \\
w^2 \\
w^3 \\
w^4
\end{bmatrix}
\]
Fast Fourier Transform (FFT) (needs \(N\) is power of 2)

an algorithm for quickly computing \(P(0), P(\omega), \ldots, P(\omega^{N-1})\)

or some degree \(\leq N\) polynomial \(P\)

(\(\omega\) is a primitive \(N\)th root of unity)

\[
P(z) = p_0 + p_1 z + p_2 z^2 + \ldots + p_{N-1} z^{N-1}
\]

\[
= (p_0 + p_2 z^2 + p_4 z^4 + \ldots + p_{N-2} z^{N-2}) + z \cdot (p_1 + p_3 z^2 + p_5 z^4 + \ldots)
\]

the insight to eval \(\deg \leq N\) \(\omega^k\) on \(N\) roots of unity

\[
T(N) = 2T\left(\frac{N}{2}\right) + \Theta(N) = \Theta(N \log N)
\]
Poly mult algorithm

// given as input coeffs of A(x), B(x) coeff vector of A

1. use \text{FFT} to compute \( \hat{a} := F(a) \)

2. use \text{FFT} to compute \( \hat{b} := F(b) \)

3. for \( i = 0 \) to \( N-1 \): \( \hat{c}_i := \hat{a}_i \times \hat{b}_i \) (\( \hat{c} \) is eval of \( C \) on \( 1, i, \ldots, i^N \))

4. \( c := F^{-1}\hat{c} \) requires one more \( FFT \) to get \( F^{-1}\hat{c} \)

5. return \( c \) (coeff vector of \( C = A \times B \))

Total: \( O(N \log N) \) time assuming can mult/add complex #s in \( O(1) \) time.
Claim 1: \( F^{-1} = \frac{1}{N} F \)

Proof:

\[
(F \cdot \left( \frac{1}{N} F \right))_{ij} = \begin{cases} 1, & \text{if } i=j \\ 0, & \text{otherwise} \end{cases}
\]

Claim 2: \( \overline{M_X} = M_{\overline{X}} \)

\[
F^{-1} \overline{z} = \frac{1}{N} \cdot \overline{F \overline{z}}
\]

Another application of FFT
Cross-correlation

Inputs: \( x = (x_0, x_1, \ldots, x_{n-1}) \)

\( y = (y_0, y_1, \ldots, y_{n-1}) \)

(two vectors, \( n \geq m \))

want all shifted dot products of \( x \) with \( y \)

want: \( x_0 y_0 + x_1 y_1 + \cdots + x_{m-1} y_{n-1} \)

\( x_0 y_1 + x_0 y_2 + \cdots + x_{m-1} y_n \)

\( \vdots \)

\( \sum_{i=0}^{n-m+1} x_{m+i-1} y_{n-1} \)
Can we rely on f to do cross-