Example



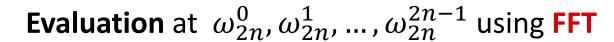
$$p(x) = 3x^3 - 15x^2 + 18x + 0,$$
 $a = [0,18, -15,3]$

Faster Polynomial Multiplication?



Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree < n):

p, q of degree n-1, n coefficients



 $2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)\right)$ and $\left(\omega_{2n}^k, q\left(\omega_{2n}^k\right)\right)$

Point-wise multiplication

2n point-value pairs $\left(\omega_{2n}^k,p\left(\omega_{2n}^k\right)q\left(\omega_{2n}^k\right)\right)$

Interpolation

p(x)q(x) of degree 2n-2, 2n-1 coefficients

Interpolation



Convert point-value representation into coefficient representation

Input:
$$(x_0, y_0), ..., (x_{n-1}, y_{n-1})$$
 with $x_i \neq x_j$ for $i \neq j$

Output:

Degree-(n-1) polynomial with coefficients a_0, \ldots, a_{n-1} such that

$$p(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \dots + a_{n-1} x_0^{n-1} = y_0$$

$$p(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \dots + a_{n-1} x_1^{n-1} = y_1$$

$$\vdots$$

$$p(x_{n-1}) = a_0 + a_1 x_{n-1} + a_2 x_{n-1}^2 + \dots + a_{n-1} x_{n-1}^{n-1} = y_{n-1}$$

 \rightarrow linear system of equations for a_0, \dots, a_{n-1}

Interpolation



Matrix Notation:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

• System of equations solvable iff $x_i \neq x_j$ for all $i \neq j$

Special Case $x_i = \omega_n^i$:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Interpolation



• Linear system:

$$W \cdot \boldsymbol{a} = \boldsymbol{y} \implies \boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$$
 $W_{i,j} = \omega_n^{ij}, \qquad \boldsymbol{a} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}, \qquad \boldsymbol{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}$

Claim:

$$W_{ij}^{-1} = \frac{\omega_n^{-ij}}{n}$$

Proof: Need to show that $W^{-1}W = I_n$

DFT Matrix Inverse



$$W^{-1}W = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-i}}{n} & \cdots & \frac{\omega_n^{-(n-1)i}}{n} \\ \vdots & \vdots & \ddots & \vdots \\ & \cdots & \omega_n^{(n-1)j} & \cdots \end{pmatrix} \cdot \begin{pmatrix} \cdots & 1 & \cdots \\ \cdots & \omega_n^{j} & \cdots \\ \vdots & \vdots & \ddots \\ \cdots & \omega_n^{(n-1)j} & \cdots \end{pmatrix}$$

DFT Matrix Inverse



$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Need to show
$$(W^{-1}W)_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Case i = j:

DFT Matrix Inverse



$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Case $i \neq j$:

Inverse DFT



$$W^{-1} = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \dots & \frac{\omega_n^{-(n-1)k}}{n} \\ & \vdots & & & \\ & & \dots & & \end{pmatrix}$$

• We get $a = W^{-1} \cdot y$ and therefore

$$a_k = \left(\frac{1}{n} \frac{\omega_n^{-k}}{n} \dots \frac{\omega_n^{-(n-1)k}}{n}\right) \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$= \frac{1}{n} \cdot \sum_{i=0}^{n-1} \omega_n^{-kj} \cdot y_j$$

DFT and Inverse DFT



Inverse DFT:

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j$$

• Define polynomial $q(x) = y_0 + y_1 x + \dots + y_{n-1} x^{n-1}$:

$$a_k = \frac{1}{n} \cdot q(\omega_n^{-k})$$

DFT:

• Polynomial $p(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$:

$$y_k = p(\omega_n^k)$$

DFT and Inverse DFT



$$q(x) = y_0 + y_1 x + \dots + y_{n-1} x^{n-1}, \qquad a_k = \frac{1}{n} \cdot q(\omega_n^{-k})$$
:

• Therefore:

$$(a_0, a_1, \dots, a_{n-1})$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^{-0}), q(\omega_n^{-1}), q(\omega_n^{-2}), \dots, q(\omega_n^{-(n-1)}) \right)$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^0), q(\omega_n^{n-1}), q(\omega_n^{n-2}), \dots, q(\omega_n^1) \right)$$

Recall:

$$DFT_{n}(\mathbf{y}) = (q(\omega_{n}^{0}), q(\omega_{n}^{1}), q(\omega_{n}^{2}), ..., q(\omega_{n}^{n-1}))$$
$$= n \cdot (a_{0}, a_{n-1}, a_{n-2}, ..., a_{2}, a_{1})$$

DFT and Inverse DFT



• We have $DFT_n(y) = n \cdot (a_0, a_{n-1}, a_{n-2}, ..., a_2, a_1)$:

$$a_i = \begin{cases} \frac{1}{n} \cdot (DFT_n(\mathbf{y}))_0 & \text{if } i = 0\\ \frac{1}{n} \cdot (DFT_n(\mathbf{y}))_{n-i} & \text{if } i \neq 0 \end{cases}$$

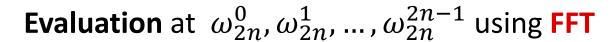
- DFT and inverse DFT can both be computed using FFT algorithm in $O(n \log n)$ time.
- 2 polynomials of degr. < n can be multiplied in time $O(n \log n)$.

Faster Polynomial Multiplication?



Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree < n):

p, q of degree n-1, n coefficients



 $2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p(\omega_{2n}^k)\right)$ and $\left(\omega_{2n}^k, q(\omega_{2n}^k)\right)$

Point-wise multiplication

2n point-value pairs $\left(\omega_{2n}^k,p\left(\omega_{2n}^k\right)q\left(\omega_{2n}^k\right)\right)$

Interpolation using FFT

p(x)q(x) of degree 2n-2, 2n-1 coefficients

Convolution



 More generally, the polynomial multiplication algorithm computes the convolution of two vectors:

$$egin{aligned} & \pmb{a} = (a_0, a_1, ..., a_{m-1}) \\ & \pmb{b} = (b_0, b_1, ..., b_{n-1}) \end{aligned} \ & \pmb{a} * \pmb{b} = (c_0, c_1, ..., c_{m+n-2}), \\ & \text{where } c_k = \sum_{\substack{(i,j): i+j=k \\ i < m, j < n}} a_i b_j \end{aligned}$$

• c_k is exactly the coefficient of x^k in the product polynomial of the polynomials defined by the coefficient vectors a and b

More Applications of Convolutions



Signal Processing Example:

- Assume $a = (a_0, ..., a_{n-1})$ represents a sequence of measurements over time
- Measurements might be noisy and have to be smoothed out
- Replace a_i by weighted average of nearby last m and next m measurements (e.g., Gaussian smoothing):

$$a'_{i} = \frac{1}{Z} \cdot \sum_{j=i-m}^{i+m} a_{j} e^{-(i-j)^{2}}$$

• New vector a' is the convolution of a and the weight vector

$$\frac{1}{Z}$$
 · $(e^{-m^2}, e^{-(m-1)^2}, \dots, e^{-1}, 1, e^{-1}, \dots, e^{-(m-1)^2}, e^{-m^2})$

Might need to take care of boundary points...

More Applications of Convolutions



Combining Histograms:

- Vectors \boldsymbol{a} and \boldsymbol{b} represent two histograms
- E.g., annual income of all men & annual income of all women
- Goal: Get new histogram c representing combined income of all possible pairs of men and women:

$$c = a * b$$

Also, the DFT (and thus the FFT alg.) has many other applications!

DFT in Signal Processing



Assume that y(0), y(1), y(2), ..., y(T-1) are measurements of a time-dependent signal.

Inverse DFT_N of (y(0), ..., y(T-1)) is a vector $(c_0, ..., c_{N-1})$ s.t.

$$y(t) = \sum_{k=0}^{N-1} c_k \cdot e^{\frac{2\pi i \cdot k}{N} \cdot t}$$

$$= \sum_{k=0}^{T-1} c_k \cdot \left(\cos\left(\frac{2\pi \cdot k}{N} \cdot t\right) + i\sin\left(\frac{2\pi \cdot k}{N} \cdot t\right)\right)$$

- Converts signal from time domain to frequency domain
- Signal can then be edited in the frequency domain
 - e.g., setting some $c_k=0$ filters out some frequencies