

CSE 421: Introduction to Algorithms



Fast Fourier Transform
Paul Beame

Integer Multiplication

■ Given:

- Two n -bit integers X and Y

- $X = a_0 + a_1 2 + a_2 2^2 + \dots + a_{n-2} 2^{n-2} + a_{n-1} 2^{n-1}$

- $Y = b_0 + b_1 2 + b_2 2^2 + \dots + b_{n-2} 2^{n-2} + b_{n-1} 2^{n-1}$

■ Compute:

- $2n-1$ -bit integer XY

- $XY = a_0 b_0 + (a_0 b_1 + a_1 b_0) 2 + (a_0 b_2 + a_1 b_1 + a_2 b_0) 2^2$
 $+ \dots + (a_{n-2} b_{n-1} + a_{n-1} b_{n-2}) 2^{2n-3} + a_{n-1} b_{n-1} 2^{2n-2}$

- **Last time:** Karatsuba's Algorithm beats naïve algorithm, using $O(n^\alpha)$ where $\alpha = \log_2 3 = 1.59\dots$

Polynomial Multiplication

■ Given:

- Degree $n-1$ polynomials **P** and **Q**

- $P = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$

- $Q = b_0 + b_1 x + b_2 x^2 + \dots + b_{n-2} x^{n-2} + b_{n-1} x^{n-1}$

■ Compute:

- Degree $2n-2$ Polynomial **P Q**

- $P Q = a_0 b_0 + (a_0 b_1 + a_1 b_0) x + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2$
 $+ \dots + (a_{n-2} b_{n-1} + a_{n-1} b_{n-2}) x^{2n-3} + a_{n-1} b_{n-1} x^{2n-2}$

■ Obvious Algorithm, just like Integer Mult.:

- Compute all $a_i b_j$ and collect terms
- $\Theta(n^2)$ time

Divide and Conquer

- Assume $n=2k$
 - $P = P_0 + P_1 x^k$ where P_0 and P_1 are degree $k-1$ polys
 - Similarly $Q = Q_0 + Q_1 x^k$
- $P Q = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k)$
 $= P_0 Q_0 + (P_1 Q_0 + P_0 Q_1) x^k + P_1 Q_1 x^{2k}$
- **Naïve:** 4 sub-problems of size $k=n/2$ plus linear combining $T(n)=4 \cdot T(n/2)+cn$ Solution $T(n) = \Theta(n^2)$
- **Karatsuba's :** 3 instead 4: $A \leftarrow P_0 Q_0$ $B \leftarrow P_1 Q_1$
 $C \leftarrow (P_0 + P_1)(Q_0 + Q_1)$ and then $C - A - B = P_1 Q_0 + P_0 Q_1$
so $T(n) = 3 T(n/2) + cn$ and $T(n) = O(n^\alpha)$ where $\alpha = \log_2 3 = 1.59\dots$



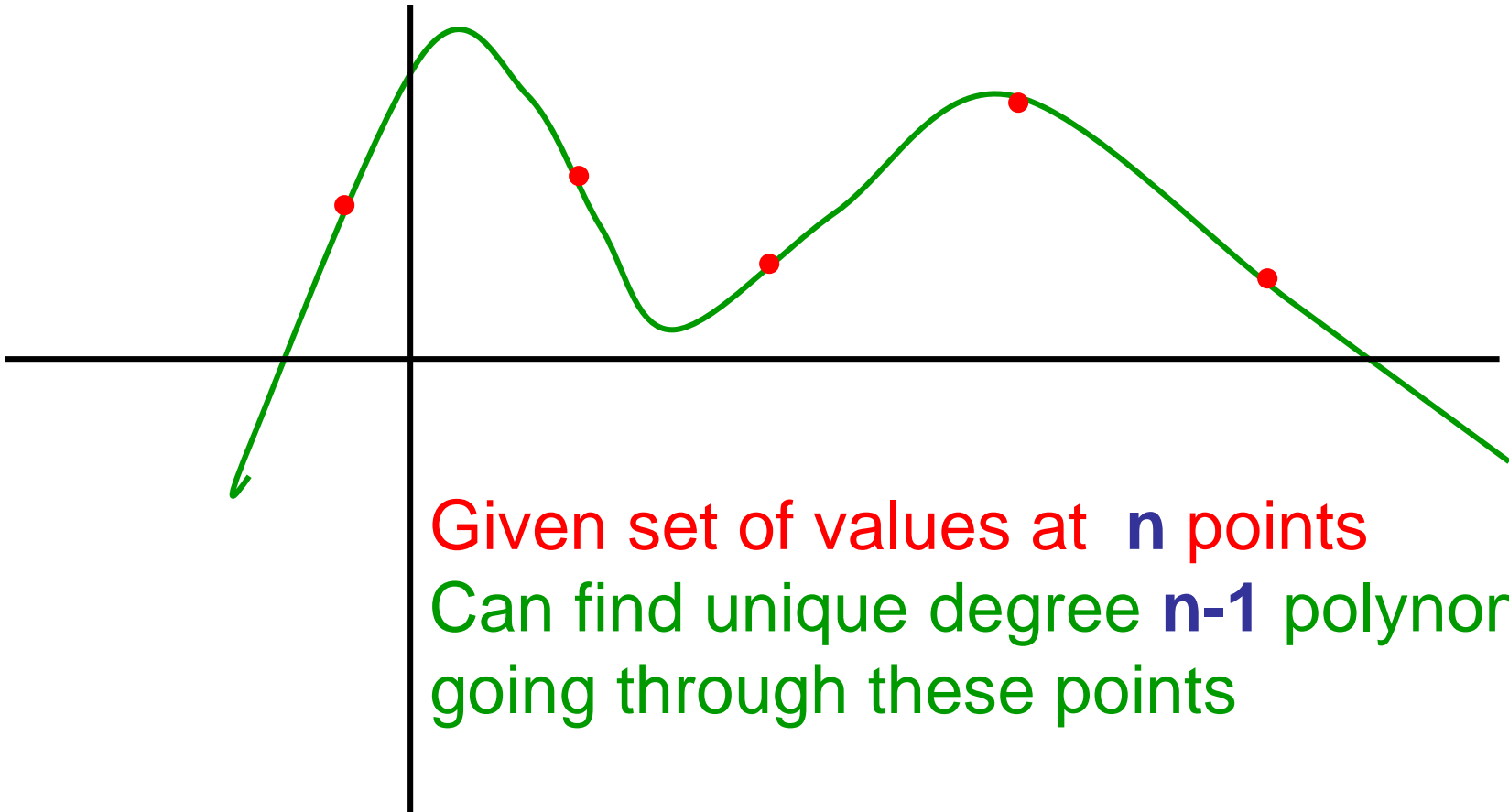
Integer and Polynomial Multiplication

- Naïve: $\Theta(n^2)$
- Karatsuba: $\Theta(n^{1.59\dots})$
- Best known: $\Theta(n \log n)$
 - "Fast Fourier Transform"
 - FFT widely used for signal processing
 - Used in practice in symbolic manipulation systems like Maple
 - MUCH easier for Polynomial Multiplication than for integer multiplication because of ugly details with carries, etc.
 - Schonhage-Strassen (1971) gives $\Theta(n \log n \log \log n)$
 - Furer (2007) gives $\Theta(n \log n 2^{\log^* n})$
 - Harvey, van der Hoeven (2019) finally got $\Theta(n \log n)$

Hints towards FFT: Interpolation

2 points determine a unique line (degree 1)

3 points determine a unique parabola (degree 2)



Multiplying Polynomials by Evaluation & Interpolation

- Any degree $n-1$ polynomial $R(y)$ is determined by $R(y_0), \dots, R(y_{n-1})$ for any n distinct y_0, \dots, y_{n-1}
- To compute PQ (assume degree at most $n/2-1$)
 - Evaluate $P(y_0), \dots, P(y_{n-1})$
 - Evaluate $Q(y_0), \dots, Q(y_{n-1})$
 - Multiply values $P(y_i)Q(y_i)$ for $i=0, \dots, n-1$
 - Interpolate to recover PQ

Interpolation

- Given values of degree $n-1$ polynomial R at n distinct points y_0, \dots, y_{n-1}

- $R(y_0), \dots, R(y_{n-1})$

- Compute coefficients c_0, \dots, c_{n-1} such that

- $R(x) = c_0 + c_1x + c_2x^2 + \dots + c_{n-1}x^{n-1}$

- System of linear equations in c_0, \dots, c_{n-1}

$$c_0 + c_1y_0 + c_2y_0^2 + \dots + c_{n-1}y_0^{n-1} = R(y_0)$$

known

$$c_0 + c_1y_1 + c_2y_1^2 + \dots + c_{n-1}y_1^{n-1} = R(y_1)$$

...

unknown

$$c_0 + c_1y_{n-1} + c_2y_{n-1}^2 + \dots + c_{n-1}y_{n-1}^{n-1} = R(y_{n-1})$$

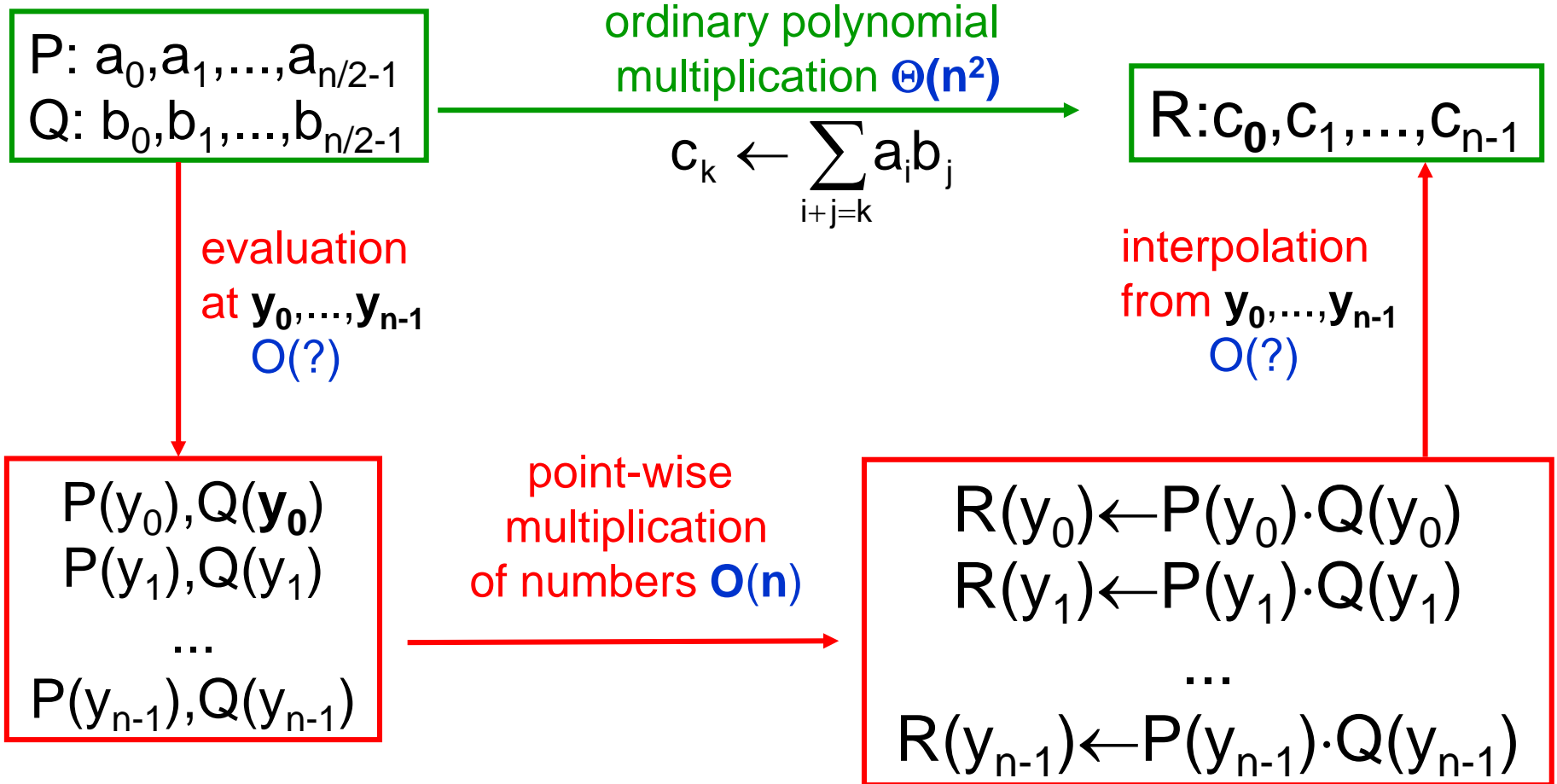
Interpolation: n equations in n unknowns

- Matrix form of the linear system

$$\begin{pmatrix} 1 & y_0 & y_0^2 & \dots & y_0^{n-1} \\ 1 & y_1 & y_1^2 & \dots & y_1^{n-1} \\ \dots & & & & \\ \dots & & & & \\ 1 & y_{n-1} & y_{n-1}^2 & \dots & y_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \cdot \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} R(y_0) \\ R(y_1) \\ \cdot \\ \cdot \\ R(y_{n-1}) \end{pmatrix}$$

- Fact:** Determinant of the matrix is $\prod_{i < j} (y_i - y_j)$ which is not **0** since points are distinct
 - System has a unique solution c_0, \dots, c_{n-1}

Hints towards FFT: Evaluation & Interpolation



Karatsuba's algorithm and evaluation and interpolation

- Karatsuba's algorithm can be thought of as a way of multiplying two degree **1** polynomials (which have **2** coefficients) using only **3** multiplications
 - $PQ = (P_0 + P_1z)(Q_0 + Q_1z)$
 $= P_0Q_0 + (P_1Q_0 + P_0Q_1)z + P_1Q_1z^2$
 - Evaluate at **0,1** plus compute P_1Q_1
 - $A = P(0)Q(0) = P_0Q_0$
 - $B = P_1Q_1$
 - $C = P(1)Q(1) = (P_0 + P_1)(Q_0 + Q_1)$
 - Alternative: replace **B** by the following: Evaluate at **-1**
 - $D = P(-1)Q(-1) = (P_0 - P_1)(Q_0 - Q_1)$
 - Interpolating, product is $A + (C - D)/2 z + [(C + D)/2 - A] z^2$



Evaluation at Special Points

- Evaluation of polynomial at **1** point takes **$O(n)$** time
 - So **$2n$** points (naively) takes **$O(n^2)$** —no savings
 - But the algorithm works no matter what the points are...
- So...choose points that are related to each other so that evaluation problems can share subproblems

The key idea: Evaluate at related points

$$\begin{aligned} \blacksquare \quad P(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \dots + a_{n-1}x^{n-1} \\ &= a_0 + a_2x^2 + a_4x^4 + \dots + a_{n-2}x^{n-2} \\ &\quad + a_1x + a_3x^3 + a_5x^5 + \dots + a_{n-1}x^{n-1} \\ &= P_{\text{even}}(x^2) + x P_{\text{odd}}(x^2) \end{aligned}$$

$$\begin{aligned} \blacksquare \quad P(-x) &= a_0 - a_1x + a_2x^2 - a_3x^3 + a_4x^4 - \dots - a_{n-1}x^{n-1} \\ &= a_0 + a_2x^2 + a_4x^4 + \dots + a_{n-2}x^{n-2} \\ &\quad - (a_1x + a_3x^3 + a_5x^5 + \dots + a_{n-1}x^{n-1}) \\ &= P_{\text{even}}(x^2) - x P_{\text{odd}}(x^2) \end{aligned}$$

where $P_{\text{even}}(z) = a_0 + a_2z + a_4z^2 + \dots + a_{n-2}z^{n/2-1}$

and $P_{\text{odd}}(z) = a_1 + a_3z + a_5z^2 + \dots + a_{n-1}z^{n/2-1}$

The key idea:

Evaluate at related points

- So... if we have half the points as negatives of the other half

- i.e., $y_{n/2} = -y_0, y_{n/2+1} = -y_1, \dots, y_{n-1} = -y_{n/2-1}$

then we can reduce the size n problem of evaluating degree $n-1$ polynomial P at n points to evaluating 2 degree $n/2 - 1$ polynomials P_{even} and P_{odd} at $n/2$ points $y_0^2, \dots, y_{n/2-1}^2$ and recombine answers with $O(1)$ extra work per point

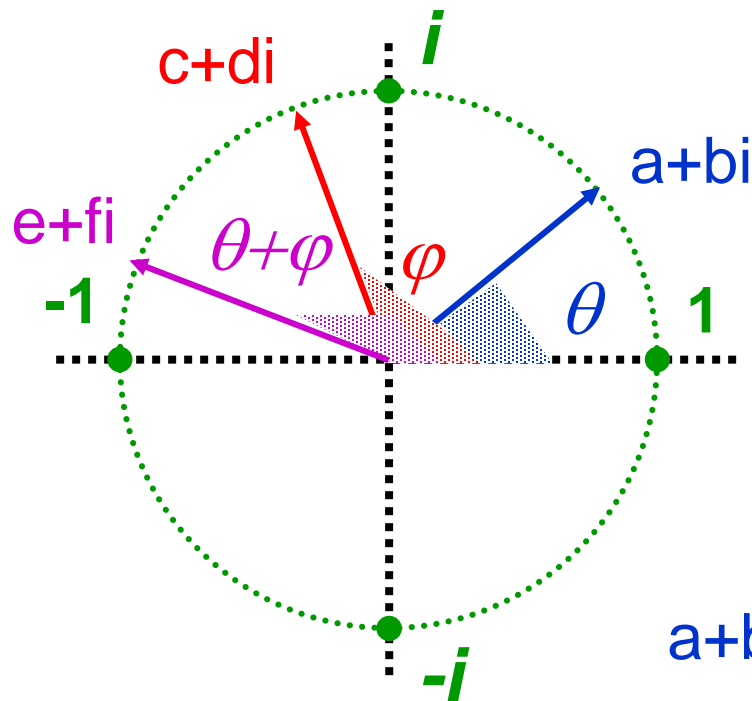
- But to use this idea recursively we need half of $y_0^2, \dots, y_{n/2-1}^2$ to be negatives of the other half

- If $y_{n/4}^2 = -y_0^2$, say, then $(y_{n/4}/y_0)^2 = -1$

- Motivates use of complex numbers as evaluation points

Complex Numbers

$$i^2 = -1$$



To multiply complex numbers:

1. add angles
2. multiply lengths
(all length 1 here)

$$e+fi = (a+bi)(c+di)$$

$$a+bi = \cos \theta + i \sin \theta = e^{i\theta}$$

$$c+di = \cos \varphi + i \sin \varphi = e^{i\varphi}$$

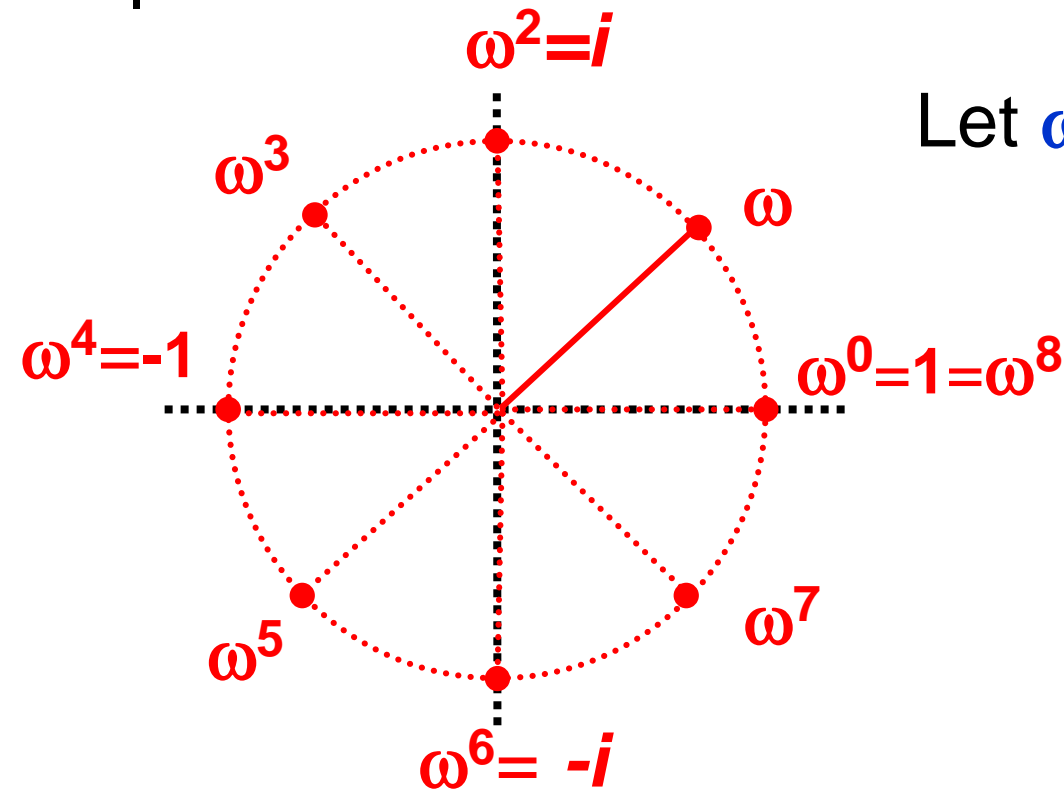
$$e+fi = \cos (\theta+\varphi) + i \sin (\theta+\varphi) = e^{i(\theta+\varphi)}$$

$$e^{2\pi i} = 1$$

$$e^{\pi i} = -1$$

Primitive n^{th} root of 1

$$\omega = \omega_n = e^{i 2\pi/n}$$



$$\text{Let } \omega = \omega_n = e^{i 2\pi/n}$$
$$= \cos(2\pi/n) + i \sin(2\pi/n)$$

$$j^2 = -1$$
$$e^{2\pi i} = 1$$



Facts about $\omega = e^{2\pi i/n}$ for even n

- $\omega = e^{2\pi i/n}$ for $i = \sqrt{-1}$
- $\omega^n = 1$
- $\omega^{n/2} = -1$
- $\omega^{n/2+j} = -\omega^j$ for all values of j
- $\omega^2 = e^{2\pi i/k}$ where $k=n/2$
- $\omega^j = \cos(2j\pi/n) + i \sin(2j\pi/n)$ so can compute with powers of ω
- ω^j is a root of $x^n - 1 = (x-1)(x^{n-1} + x^{n-2} + \dots + 1) = 0$
but for $j \neq 0$, $\omega^j \neq 1$ so $\omega^{j(n-1)} + \omega^{j(n-2)} + \dots + 1 = 0$

The key idea for n even

$$\begin{aligned} \blacksquare \quad P(\omega) &= a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 + a_4\omega^4 + \dots + a_{n-1}\omega^{n-1} \\ &= a_0 + a_2\omega^2 + a_4\omega^4 + \dots + a_{n-2}\omega^{n-2} \\ &\quad + a_1\omega + a_3\omega^3 + a_5\omega^5 + \dots + a_{n-1}\omega^{n-1} \\ &= P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2) \end{aligned}$$

$$\begin{aligned} \blacksquare \quad P(-\omega) &= a_0 - a_1\omega + a_2\omega^2 - a_3\omega^3 + a_4\omega^4 - \dots - a_{n-1}\omega^{n-1} \\ &= a_0 + a_2\omega^2 + a_4\omega^4 + \dots + a_{n-2}\omega^{n-2} \\ &\quad - (a_1\omega + a_3\omega^3 + a_5\omega^5 + \dots + a_{n-1}\omega^{n-1}) \\ &= P_{\text{even}}(\omega^2) - \omega P_{\text{odd}}(\omega^2) \end{aligned}$$

where $P_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

and $P_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

The recursive idea for n a power of 2

- Goal:

- Evaluate P at $1, \omega, \omega^2, \omega^3, \dots, \omega^{n-1}$

- Now

- P_{even} and P_{odd} have degree $n/2-1$ where
- $P(\omega^k) = P_{\text{even}}(\omega^{2k}) + \omega^k P_{\text{odd}}(\omega^{2k})$
- $P(-\omega^k) = P_{\text{even}}(\omega^{2k}) - \omega^k P_{\text{odd}}(\omega^{2k})$

- Recursive Algorithm

- Evaluate P_{even} at $1, \omega^2, \omega^4, \dots, \omega^{n-2}$
- Evaluate P_{odd} at $1, \omega^2, \omega^4, \dots, \omega^{n-2}$
- Combine to compute P at $1, \omega, \omega^2, \dots, \omega^{n/2-1}$
- Combine to compute P at $-1, -\omega, -\omega^2, \dots, -\omega^{n/2-1}$
(i.e. at $\omega^{n/2}, \omega^{n/2+1}, \omega^{n/2+2}, \dots, \omega^{n-1}$)

ω^2 is $e^{2\pi i/k}$ where $k=n/2$
so problems are of same
type but smaller size



Analysis and more

- Run-time
 - $T(n)=2\cdot T(n/2)+cn$ so $T(n)=O(n \log n)$
- So much for evaluation ... what about interpolation?
 - Given
 - $r_0=R(1), r_1=R(\omega), r_2=R(\omega^2), \dots, r_{n-1}=R(\omega^{n-1})$
 - Compute
 - c_0, c_1, \dots, c_{n-1} s.t. $R(x)=c_0+c_1x+\dots+c_{n-1}x^{n-1}$

Interpolation \approx Evaluation: strange but true

- Non-obvious fact:
 - If we define a new polynomial $\mathbf{S}(x) = r_0 + r_1x + r_2x^2 + \dots + r_{n-1}x^{n-1}$ where r_0, r_1, \dots, r_{n-1} are the evaluations of \mathbf{R} at $1, \omega, \dots, \omega^{n-1}$
 - Then $\mathbf{c}_j = \mathbf{S}(\omega^{-j})/n$ for $k=0, \dots, n-1$
 - Relies on the fact the interpolation (inverse) matrix has ij entry $\omega^{-(ij)}/n$ instead of ω^{ij}
- So...
 - evaluate \mathbf{S} at $1, \omega^{-1}, \omega^{-2}, \dots, \omega^{-(n-1)}$ then divide each answer by n to get the $\mathbf{c}_0, \dots, \mathbf{c}_{n-1}$
 - ω^{-1} behaves just like ω did so the same $\mathbf{O}(n \log n)$ evaluation algorithm applies !

Why this is called the discrete Fourier transform

■ Real Fourier series

- Given a real valued function f defined on $[0, 2\pi]$ the Fourier series for f is given by

$$f(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x) + \dots + a_m \cos(mx) + \dots$$

where

$$a_m = \frac{1}{2\pi} \int_0^{2\pi} f(x) \cos(mx) dx$$

- is the component of f of frequency m
- In signal processing and data compression one ignores all but the components with large a_m and there aren't many since

Why this is called the discrete Fourier transform

■ Complex Fourier series

- Given a function f defined on $[0, 2\pi]$ the complex Fourier series for f is given by

$$f(z) = b_0 + b_1 e^{iz} + b_2 e^{2iz} + \dots + b_m e^{miz} + \dots$$

where

$$b_m = \frac{1}{2\pi} \int_0^{2\pi} f(z) e^{-miz} dz$$

is the component of f of frequency m

- If we **discretize** this integral using values at n $2\pi/n$ apart equally spaced points between 0 and 2π we get

$$\bar{b}_m = \frac{1}{n} \sum_{k=0}^{n-1} f_k e^{-2kmi\pi/n} = \frac{1}{n} \sum_{k=0}^{n-1} f_k \omega^{-km} \text{ where } f_k = f(2k\pi/n)$$

just like interpolation!