CS 580
ALGORITHM DESIGN AND ANALYSIS

D&C 3:
Convolution and FFT

Vassilis Zikas

RECAP

- So far:
  - Mergesort
  - Counting inversions
  - Closer pair of points
  - Integer and Matrix multiplication
- Next:
  - FFT (5.6 in KT)

CONVOLUTION

- Input:
  - Two vectors $a = (a_0, a_1, \dots, a_{n-1})$ and $b = (b_0, b_1, \dots, b_{n-1})$
- Output:
  - The convolution $\mathrm{c} = a * b$
  - For $k = 0, \dots, 2(n-1)$:

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

CONVOLUTION

- What???
- $a * b = (a_0 b_0, a_0 b_1 + a_1 b_0, a_0 b_2 + a_1 b_1 + a_2 b_0, \ldots, a_{n-1} b_{n-1})$
- Ok, this makes marginally more sense, but who cares??
- Polynomials!

POLYNOMIALS

- $a(x) = 1 + 2x + 3x^2$
- $b(x) = 2 + x + 4x^2$
- $c(x) = a(x) * b(x) = 2 + 5x + 12x^2 + 11x^3 + 12x^4$
- Representing $a(x)$ as the vector (1,2,3,0,0) and $b(x)$ as the vector (2,1,4,0,0) then $c(x)$ is exactly (2,5,12,11,12) where

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

POLYNOMIALS

- Ok, so the question really is "how can we multiply polynomials quickly"?
- Again, who cares?
- The algorithm we will develop today, the Fast Fourier Transform (FFT), has been described as "*the most important numerical algorithm of our lifetime*"!
  - Wide range of applications: signal processing, speech recognition, image processing, optics, quantum physics
    - Even multiplying integers!

ONE STEP AT A TIME

- Different way to represent polynomials
- **Fundamental theorem of algebra** [Gauss, PhD thesis] : A degree n polynomial with complex coefficients has n complex roots.
- Corollary: A degree $n - 1$ polynomial is uniquely specified by its values at n distinct points.
  - Proof: Say $A(x)$ and $B(x) \neq A(x)$ has the same value in n points. Then the polynomial $\mathrm{C(x)} = A(x) - B(x)$ would have degree $n - 1$ but n roots!

ONE STEP AT A TIME

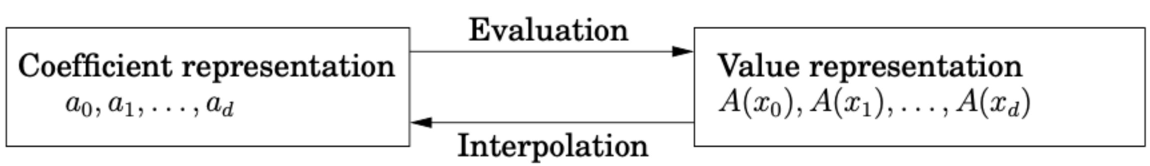
- The corollary gives us a new way to represent polynomials
- Old representation:
  - $A(x)$: $(a_0, a_1, \ldots, a_{n-1})$
- New representation:
  - $(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{n-1}, A(x_{n-1}))$

EASY MULTIPLICATION

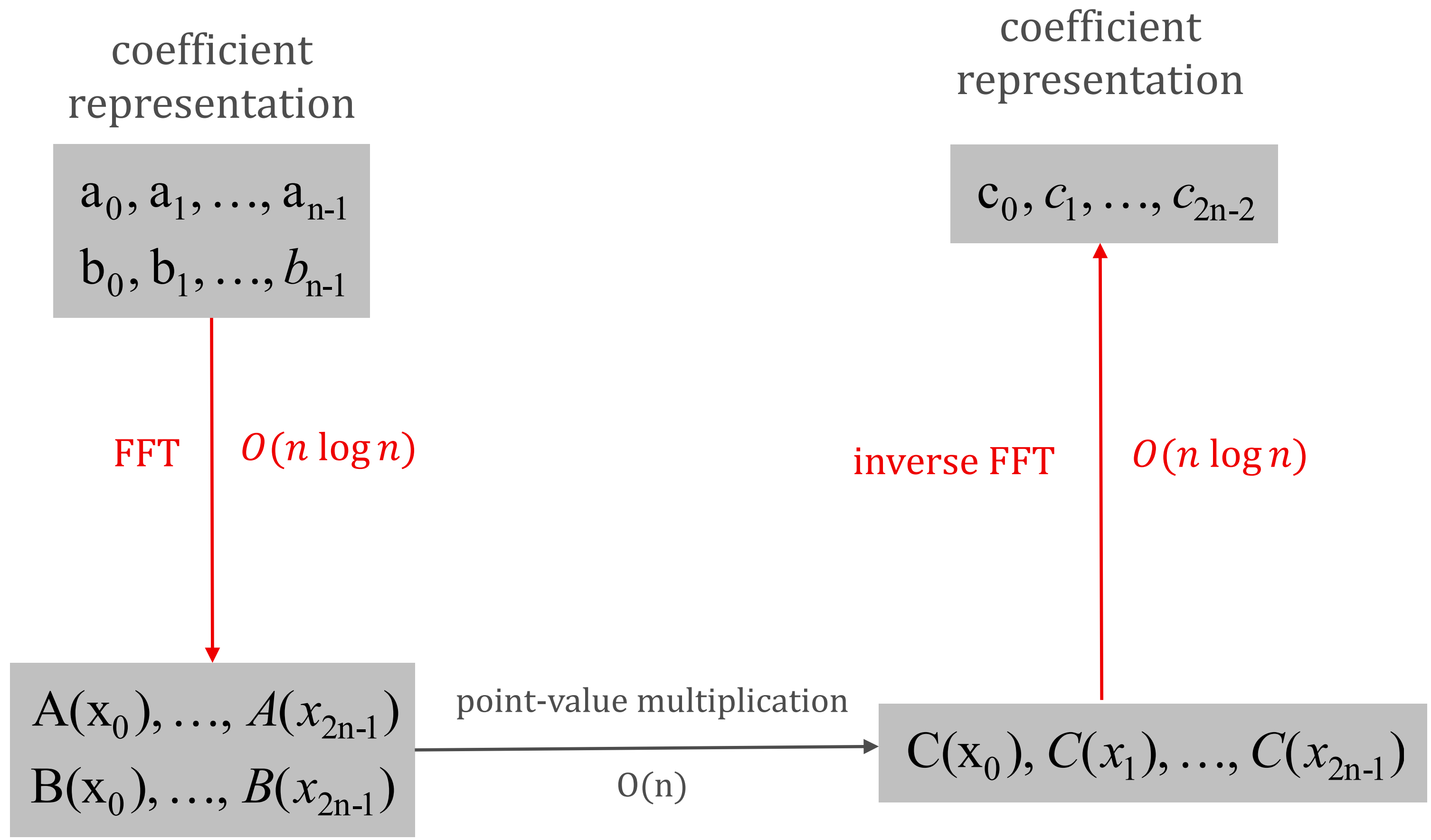
- Given two polynomials in the value representation, we can multiply them pretty fast!
  - $A(x)$: $A(x_0), \ldots, A(x_{n-1})$
  - $B(x)$: $B(x_0), \ldots, B(x_{n-1})$
  - $C(x)$: $A(x_0)B(x_0), \ldots$
- Just make sure you have enough points for $C(x)$ to be unique!
- That is, evaluate $A(x)$ and $B(x)$ at $2n$ points, the maximum degree for $C(x)$!

TRADEOFFS

- We need to go from the coefficient representation to the value representation quickly, i.e. **evaluate** quickly
- And then we need to go back to the coefficient representation, i.e. **interpolate** quickly



TODAY


coefficient representation
$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$
FFT
$O(n \log n)$
$A(x_0), \ldots, A(x_{2n-1})$
$B(x_0), \ldots, B(x_{2n-1})$
point-value multiplication
O(n)
$C(x_0), C(x_1), \ldots, C(x_{2n-1})$
inverse FFT
$O(n \log n)$
coefficient representation
$c_0, c_1, \ldots, c_{2n-2}$

EVALUATION

- Evaluation given coefficients for $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$
- $$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_k & x_k^2 & \dots & x_k^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_{n-1} \end{bmatrix}$$
- We can multiply matrices fast-ish
- Any of those ideas useful here?
- How should we “divide”?

EVALUATION

- What about even and odd powers?
- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$
- $A_e(x) = a_0 + a_2 x^2 + a_4 x^4$
- $A_o(x) = a_1 x + a_3 x^3$
- $A(x) = A_e(x) + A_o(x)$

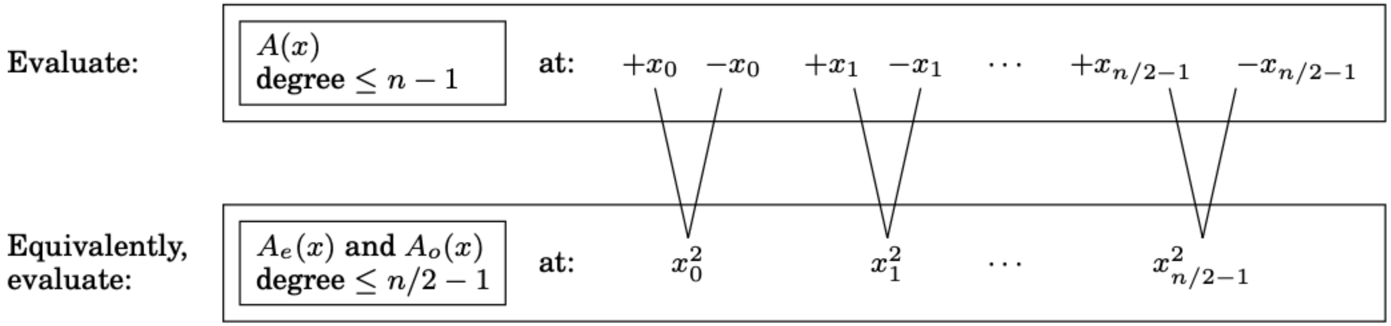
EVALUATION

- What about even and odd powers?
- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$
- $A_e(x) = a_0 + a_2 x \ + a_4 x^2$
- $A_o(x) = a_1 + a_3 x$
- $A(x) = A_e(x^2) + x \cdot A_o(x^2)$

EVALUATION

- What about even and odd powers?
- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$
- $A_e(x) = a_0 + a_2 x \ + a_4 x^2$
- $A_o(x) = a_1 + a_3 x$
- $A(x) = A_e(x^2) + x \cdot A_o(x^2)$
- Bonus: $A(-x) = A_e(x^2) - x \cdot A_o(x^2)$
- Realization: we get to pick the evaluation points $x_0, x_1, \dots, x_{n-1}$
- Pick them so that they look like $\pm\, x_0, \pm x_1, \dots, \pm x_{\frac{n}{2}-1}$

EVALUATION



- $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
  - $T(n) \in O(nlogn)$
- Made a mistake!
- How can we recurse???
- The plus-minus trick only works at the top
  - How can x_1^2 be equal to $-x_0^2$??

EVALUATION

- Complex numbers!
- Which complex numbers?
- At the bottom of a recursion we have some number
  - Say 1
- One level above, this number is expressed as the square of two numbers
  - Easy $+1$ and -1
- One level above, each of these numbers is expressed as the square of two numbers
  - Easy again: (1) $+1$ we've done. (2) $-1 = i^2 = (-i)^2$
- And so on, until the top , were we have all our points $x_0, x_1, \dots x_{n-1}$

ROOTS OF UNITY

- The nth roots of unit
  - The solutions to $z^n = 1$
- These are $\omega^0, \omega^1, \dots, \omega^{n-1}$ where

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

- Proof:

$$\left(\omega^k\right)^n = \left(e^{k\frac{2\pi i}{n}}\right)^n = \left(e^{\pi i}\right)^{2k} = (-1)^{2k} = 1$$

ROOTS OF UNITY

- Fact 1: Squaring produces the $\frac{1}{2}$ n-th roots of unity:
  - The $\frac{1}{2}n$-th roots of unity are $v^0, v^1, \ldots, v^{\frac{n}{2}-1}$ where $v = e^{\frac{4\pi i}{n}}$
  - Proof:
    - $\omega^2 = v$ and $(\omega^2)^k = v^k$
- Fact 2: The n-th roots of unity are plus-minus paired
  - $\omega^{\frac{n}{2}+j} = -\omega^j$

EVALUATION

- The two facts imply that starting with the n-th roots of unity at the top of the recursion, k levels down the subproblem will need to evaluate at the $\mathrm{n}/2^{k}$-th roots of unity
  - These will be plus-minus paired, so we can again recurse!

FFT

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

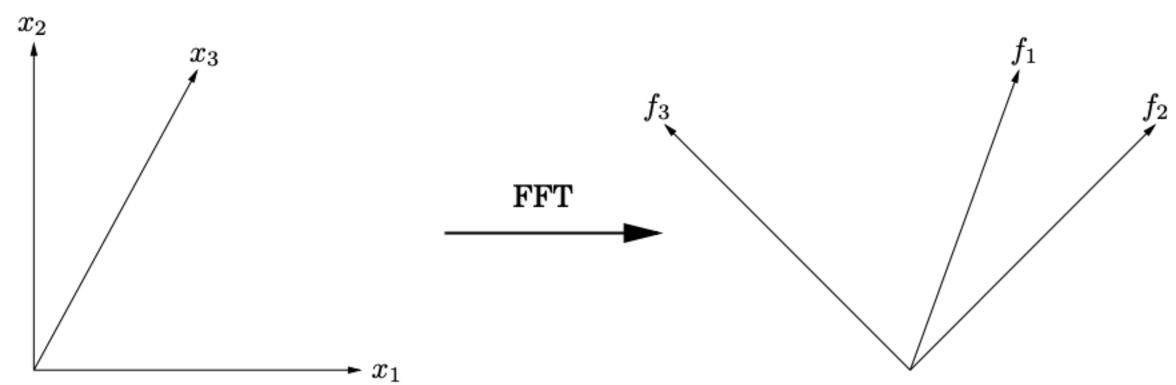
FFT

- FFT($\omega, a_0, \dots, a_n$):
  - If $\omega = 1$: return a_0
  - Express $A(x)$ as $A_e(x^2) + \mathrm{xA_o}(\mathrm{x}^2)$
  - $\mathrm{FFT}(A_e, \omega^2) \leftarrow A_e$ at even powers of ω
  - $\mathrm{FFT}(A_o, \omega^2) \leftarrow A_o$ at even powers of ω
  - For $j = 0, \dots, n-1$:
    - $A(\omega^j) = A_e(\omega^{2j}) + \omega^j A_o(\omega^{2j})$
  - Return $A(\omega^0), \dots, A(\omega^{n-1})$

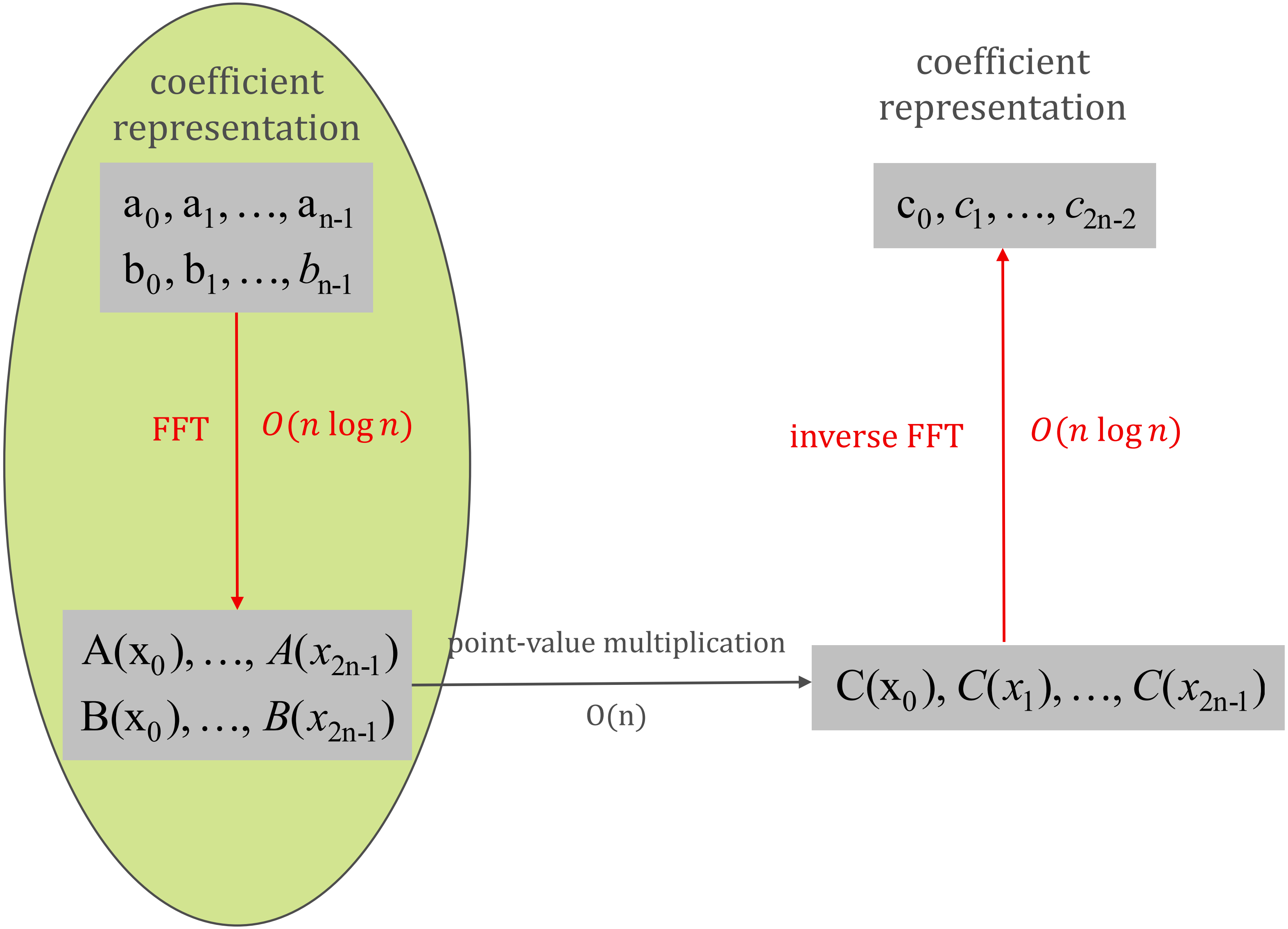
- Theorem: FFT evaluates an $n - 1$ degree polynomial at each of the n-th roots of unity in $O(nlogn)$ steps
- Runtime analysis:
  - $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

ALTERNATE VIEW

Figure 2.8 The FFT takes points in the standard coordinate system, whose axes are shown here as x_1, x_2, x_3, and rotates them into the Fourier basis, whose axes are the columns of $M_n(\omega)$, shown here as f_1, f_2, f_3. For instance, points in direction x_1 get mapped into direction f_1.



FFT



coefficient representation
$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$
FFT
$O(n \log n)$
$A(x_0), \ldots, A(x_{2n-1})$
$B(x_0), \ldots, B(x_{2n-1})$
point-value multiplication
O(n)
$C(x_0), C(x_1), \ldots, C(x_{2n-1})$
inverse FFT
$O(n \log n)$
coefficient representation
$c_0, c_1, \ldots, c_{2n-2}$

INVERSE FFT: POLYNOMIAL INTERPOLATION

- How do we go from $C(x_0), \ldots, C(x_{2n-1})$ to $c_0, c_1, \ldots, c_{2n-1}$?
- Evaluation:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

INVERSE FFT: POLYNOMIAL INTERPOLATION

- Interpolation:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

INVERSE FFT: POLYNOMIAL INTERPOLATION

- Claim: Inverse Fourier matrix is given by:

$$G_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

- To compute inverse FFT, use the same algorithm for $\omega^{-1} = e^{-\frac{2\pi i}{n}}$ and divide by n

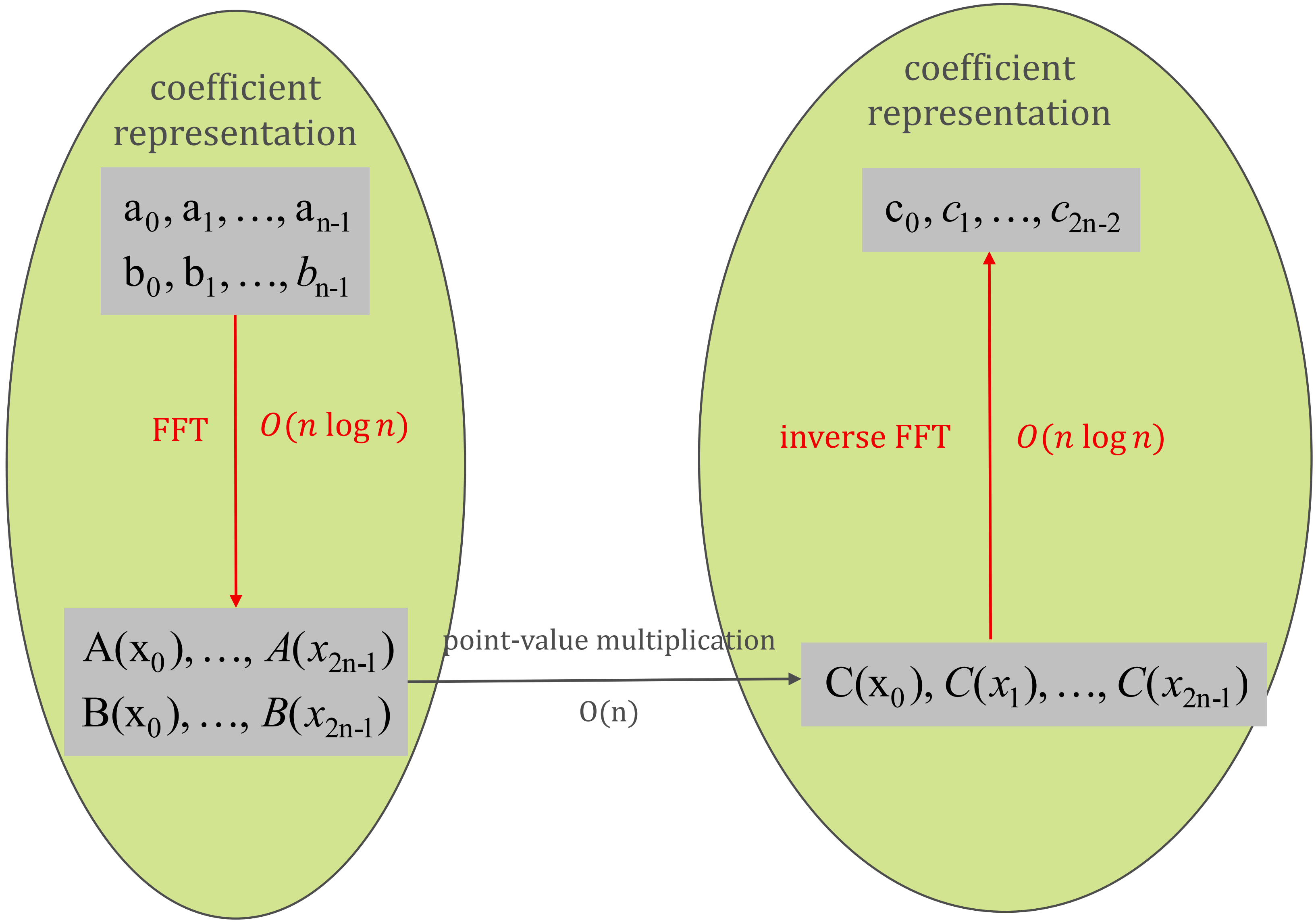
INVERSE FFT

- Let's prove that these two matrices are inverses of each other
- $(F_n G_n)_{k,k'} = \frac{1}{n}\sum_{j=0}^{n-1} \omega^{kj} \cdot \omega^{-jk'}$
- $= \frac{1}{n}\sum_{j=0}^{n-1} \omega^{j(k-k')}$
- If $k = k'$, then $(F_n G_n)_{k,k'} = 1$

INVERSE FFT

- $(F_n G_n)_{k,k'} = \frac{1}{n}\sum_{j=0}^{n-1} \omega^{j(k-k')}$
- $\omega^{(k-k')}$ is a root of
  $x^n - 1 = (x-1)(1 + x + x^2 + \cdots + x^{n-1})$
- If $k \neq k'$, $\omega^{(k-k')} \neq 1$
- Therefore
- $1 + \omega^{(k-k')} + \omega^{2(k-k')} + \cdots + \omega^{(n-1)(k-k')} = 0$
- Thus, $(F_n G_n)_{k,k'} = 0$ for $k \neq k'$
- And $(F_n G_n)_{k,k'} = 1$ for $k = k'$

FFT


coefficient representation
$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$
FFT
$O(n \log n)$
$A(x_0), \ldots, A(x_{2n-1})$
$B(x_0), \ldots, B(x_{2n-1})$
point-value multiplication
O(n)
$C(x_0), C(x_1), \ldots, C(x_{2n-1})$
inverse FFT
$O(n \log n)$
coefficient representation
$c_0, c_1, \ldots, c_{2n-2}$

SUMMARY

- Mergesort
- Counting inversions
- Closer pair of points
- Integer and Matrix multiplication
- Fast Fourier Transform

- Next: Dynamic Programming!