The Fast Fourier Transform and Polynomial Multiplication

Version of September 9, 2016

Polynomial Multiplication

If $\quad A(x) = \sum_{i=0}^{n-1} a_i x^i, \quad B(x) = \sum_{i=0}^{n-1} b_i x^i \quad$ then

$$C(x) = A(x)B(x) \quad \Leftrightarrow \quad C(x) = \sum_{i=0}^{2n-1} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^{i} a_i b_{i-j}$$

Polynomial Multiplication

If $$A(x) = \sum_{i=0}^{n-1} a_i x^i, \quad B(x) = \sum_{i=0}^{n-1} b_i x^i$$ then

$$C(x) = A(x)B(x) \quad \Leftrightarrow \quad C(x) = \sum_{i=0}^{2n-1} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^{i} a_i b_{i-j}$$

- sequence $< c_i >$ is *convolution* of $< a_i >$ and $< b_i >$
- polynomial multiplication equivalent to calculating convolutions
- Straightforward multiplication alg is $\Theta(n^2)$
- Divide and conquer Karatsuba mult is $O(n^{\log_2 3})$

Polynomial Multiplication

If $$A(x) = \sum_{i=0}^{n-1} a_i x^i, \quad B(x) = \sum_{i=0}^{n-1} b_i x^i$$ then

$$C(x) = A(x)B(x) \quad \Leftrightarrow \quad C(x) = \sum_{i=0}^{2n-1} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

- sequence $< c_i >$ is *convolution* of $< a_i >$ and $< b_i >$
- polynomial multiplication equivalent to calculating convolutions
- Straightforward multiplication alg is $\Theta(n^2)$
- Divide and conquer Karatsuba mult is $O(n^{\log_2 3})$

This Lecture

- $O(n \log n)$ divide and conquer algorithm
- Uses Fast Fourier Transform (FFT)
- FFT calculates the Discrete Fourier Transform (DFT)

Polynomial Evaluation & Interpolation

Coefficient Representation

- $A(x) = \sum_{i=0}^{n-1} a_i x^i$

- Evaluation of $A(x)$ for fixed x: $O(n)$ time
- Evaluation at n fixed values, $x_0, x_1, \ldots, x_{n-1}$: $O(n^2)$ time

Polynomial Evaluation & Interpolation

Coefficient Representation

- $A(x) = \sum_{i=0}^{n-1} a_i x^i$

- Evaluation of $A(x)$ for fixed x: $O(n)$ time
- Evaluation at n fixed values, $x_0, x_1, \ldots, x_{n-1}$: $O(n^2)$ time

Point Representation

- Polynomial of degree n uniquely represented by $n + 1$ values
  - e.g., 2 points determine a line; 3 points, a parabola

- Reconstructing coefficients of $A(x)$ from $n + 1$ values
  $A(x_1), A(x_2), \ldots, A(x_n)$
  requires $O(n^2)$ time using Lagrangian Interpolation

Review of Lagrangian Interpolation

To construct degree-n polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j}$$

Review of Lagrangian Interpolation

To construct degree-n polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad I_i(x_j) = \begin{cases} 0 & \text{if } j \ne i \\ 1 & \text{if } j = i \end{cases}$$

Review of Lagrangian Interpolation

To construct degree-n polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $$I_i(x) = \prod_{0 \le j \le n,\ j \ne i} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad I_i(x_j) = \begin{cases} 0 & \text{if } j \ne i \\ 1 & \text{if } j = i \end{cases}$$

Set $P(x) = \sum_i A(x_i) I_i(x)$. $\Rightarrow$ $P(x_i) = A(x_i) I_i(x_i) = A(x_i)$.

Review of Lagrangian Interpolation

To construct degree-n polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad I_i(x_j) = \begin{cases} 0 & \text{if } j \ne i \\ 1 & \text{if } j = i \end{cases}$$

Set $P(x) = \sum_i A(x_i) I_i(x)$. $\quad \Rightarrow \quad P(x_i) = A(x_i) I_i(x_i) = A(x_i)$.

Since $P(x)$ has degree n (why?), $P(x)$ is the *unique* polynomial satisfying $P(x_i) = A(x_i)$ for all i, so $A(x) = P(x)$.

Review of Lagrangian Interpolation

To construct degree-n polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad I_i(x_j) = \begin{cases} 0 & \text{if } j \ne i \\ 1 & \text{if } j = i \end{cases}$$

Set $P(x) = \sum_i A(x_i) I_i(x)$. $\quad \Rightarrow \quad P(x_i) = A(x_i) I_i(x_i) = A(x_i)$.

Since $P(x)$ has degree n (why?), $P(x)$ is the *unique* polynomial satisfying $P(x_i) = A(x_i)$ for all i, so $A(x) = P(x)$.

Note: $P(x)$ can be constructed in $O(n^2)$ time.

A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

$$\boxed{\begin{array}{l} A(x) : \; a_0, \ldots, a_{n-1} \\ B(x) : \; b_0, \ldots, b_{n-1} \end{array}}$$

A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

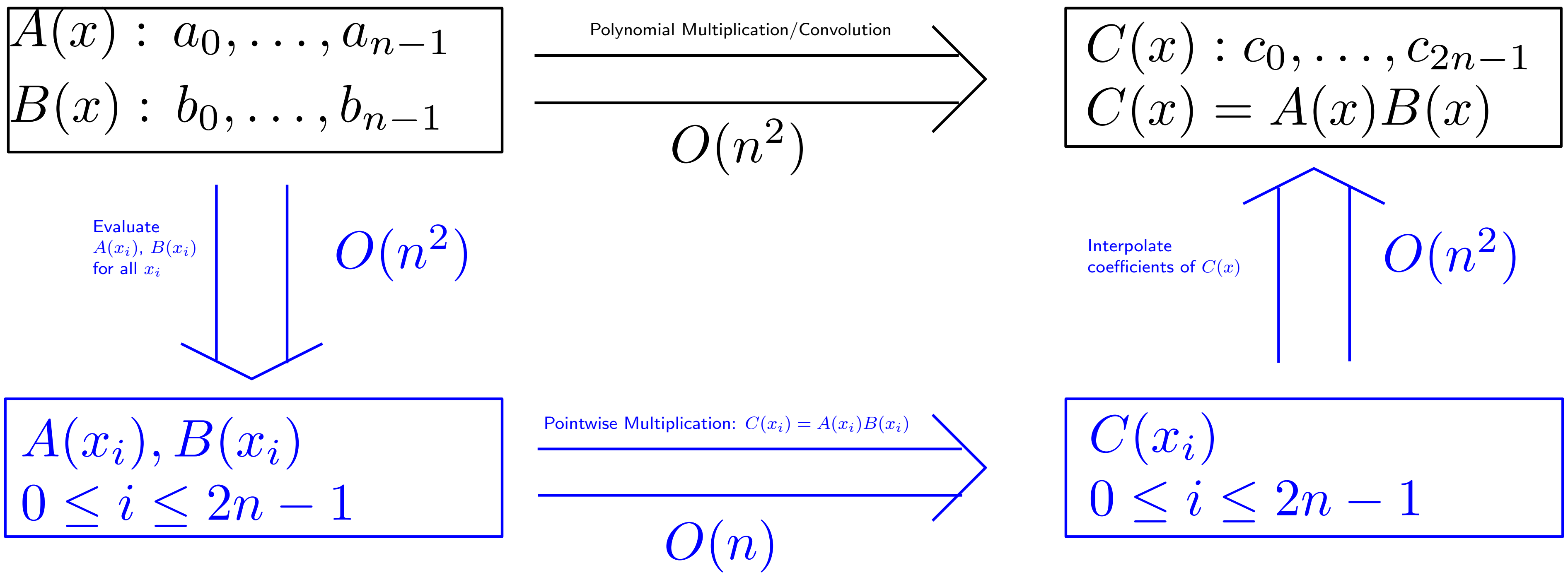
$A(x):\ a_0, \ldots, a_{n-1}$
$B(x):\ b_0, \ldots, b_{n-1}$

$$\xrightarrow[O(n^2)]{\text{Polynomial Multiplication/Convolution}}$$

$C(x):c_0, \ldots, c_{2n-1}$
$C(x) = A(x)B(x)$

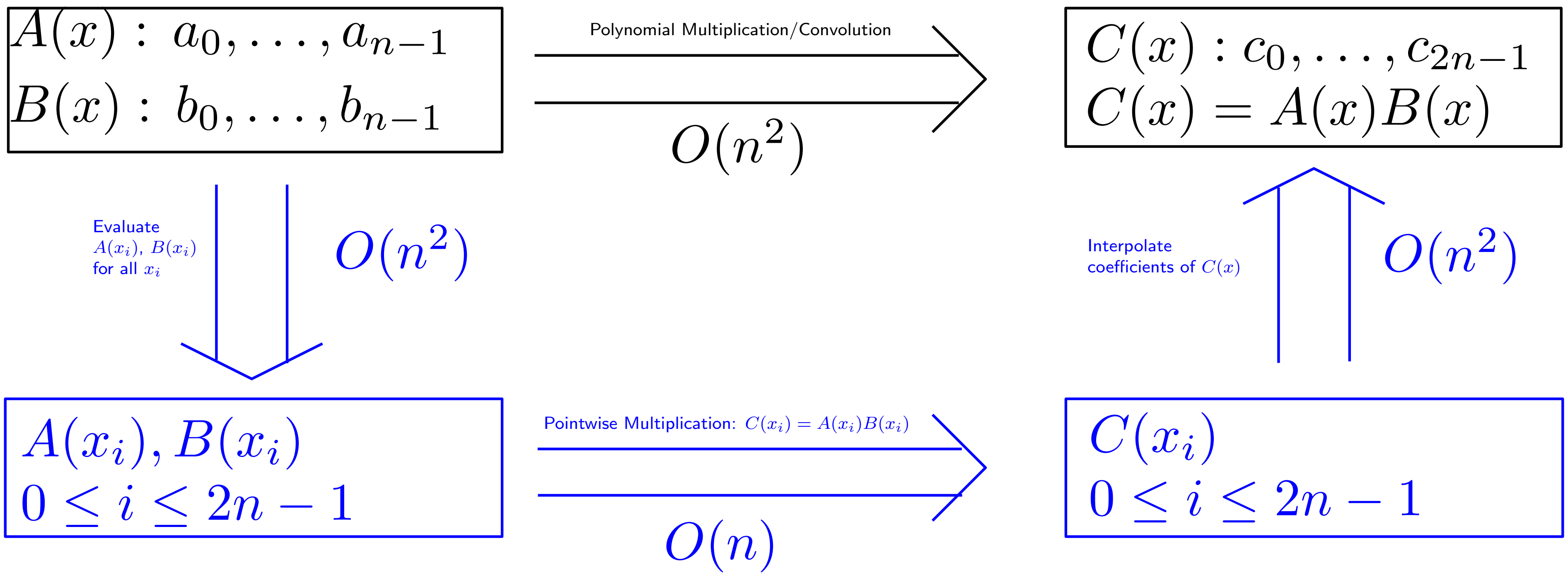
A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values



A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

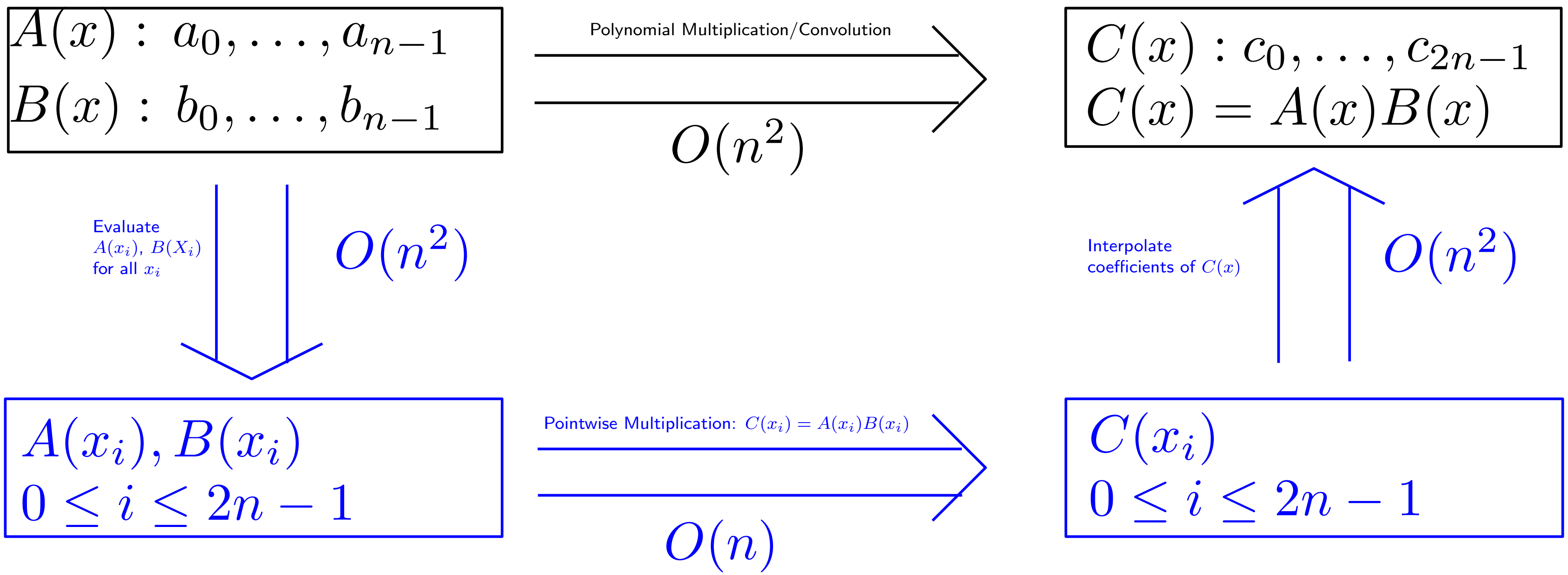


- An alternative convolution method is to
  (1) evaluate all values $A(x_i)$, $B(x_i)$
  (2) pointwise multiply to find $C(x_i) = A(x_i)B(x_i)$
  (3) interpolate to find $C(x)$

- Since Evaluation and Interpolation require $O(n^2)$, new method still requires $O(n^2)$.

A New Approach To Polynomial Multiplication

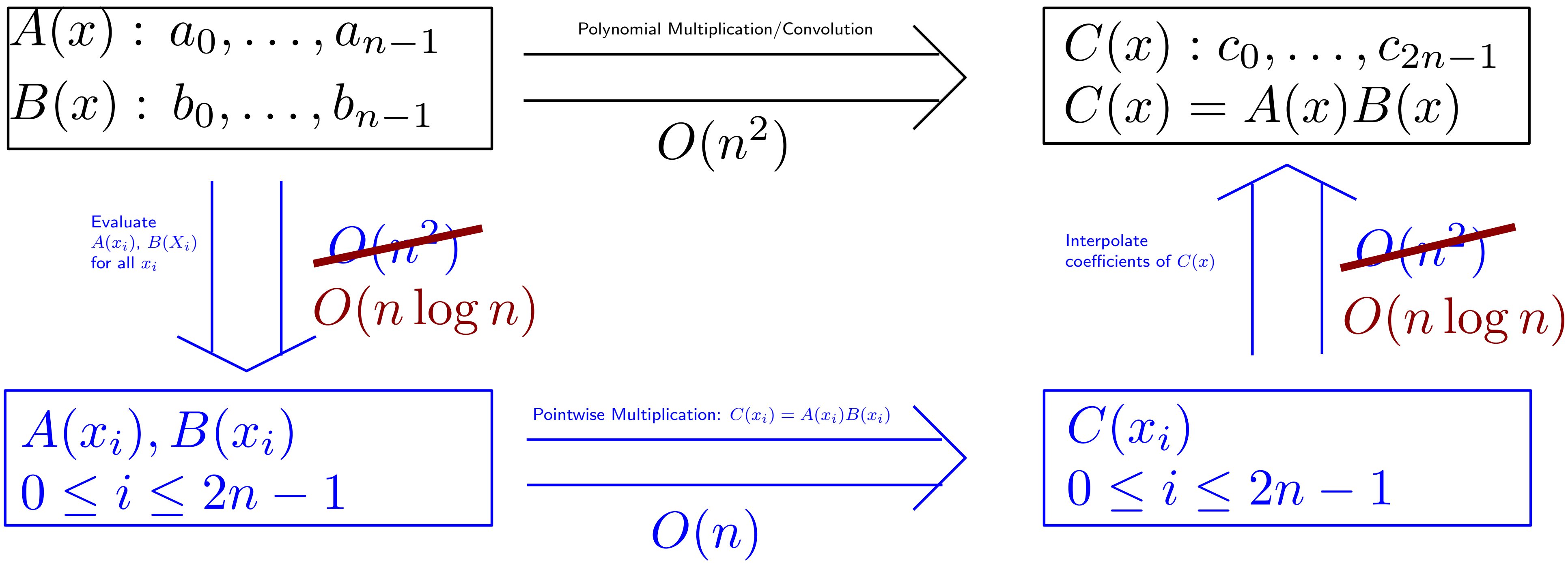
$x_0, x_1, \ldots, x_{2n-1}$ are given values



- Assumption was that the x_i were arbitrary values
- If x_i are specified to be the (complex) roots of unity, FFT can evaluate and interpolate the values in $O(n \log n)$ time.

A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values



- Assumption was that the x_i were arbitrary values
- If x_i are specified to be the (complex) roots of unity, FFT can evaluate and interpolate the values in $O(n \log n)$ time.

- Resulting in an $O(n \log n)$ convolution algorithm!

Review of Complex Numbers & Roots of Unity

- $\mathbf{i} = \sqrt{-1}$
- $e^{\mathbf{i}x} = cosx + \mathbf{i}\sin x$
- $w_n = e^{2\pi\mathbf{i}/n}$ is the n'th principle root of unity; $w_n^n = 1$
- The n roots of unity are $w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1}$

  w_n^i denotes $(w_n)^i$
- These n roots of unity are the n points, $e^{2j\pi\mathbf{i}/n}$, $j = 0, 1, \ldots, n-1$, equally spaced around the circle

Review of Complex Numbers & Roots of Unity

- $\mathbf{i} = \sqrt{-1}$
- $e^{\mathbf{i}x} = cosx + \mathbf{i}\sin x$
- $w_n = e^{2\pi\mathbf{i}/n}$ is the n'th principle root of unity; $w_n^n = 1$
- The n roots of unity are $w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1}$

  w_n^i denotes $(w_n)^i$
- These n roots of unity are the n points, $e^{2j\pi\mathbf{i}/n}$, $j = 0, 1, \ldots, n-1$, equally spaced around the circle

If n is even

- If $i < n/2$ then

$$\left(w_n^i\right)^2 = w_n^{2i} = w_{n/2}^i \quad \text{and} \quad \left(w_n^{i+n/2}\right)^2 = w_n^n w_n^{2i} = w_{n/2}^i$$

- $w_n^{n/2} = -1$

FFT Definition

FFT Definition

- Input: Coefficients $a_0, a_1, \ldots, a_{n-1}$
- Output: Values $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$.
  Output is called the *Discrete Fourier Transform ($DFT_n(A)$) of sequence $< a_i >$*
- Will denote algorithm by $FFT_n(A)$

Assumption

- n is a power of 2;
- if not, add zero coefficients, increasing size to power of 2

FFT Definition

FFT Definition

- Input: Coefficients $a_0, a_1, \ldots, a_{n-1}$
- Output: Values $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$.
  Output is called the *Discrete Fourier Transform ($DFT_n(A)$) of sequence $< a_i >$*
- Will denote algorithm by $FFT_n(A)$

Assumption

- n is a power of 2;
- if not, add zero coefficients, increasing size to power of 2

Termination Condition

- If $n = 1$ then sequence has only one value a_0 and $FFT_n(A)$ just returns the value a_0.

FFT Main Body

Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

FFT Main Body

Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts: $A(x) = A_0(x^2) + xA_1(x^2)$

$A_0(x) = a_0 + a_2x + a_4x^2 + \cdots$ $A_1(x) = a_1 + a_3x + a_5x^2 + \cdots$

FFT Main Body

Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts: $A(x) = A_0(x^2) + xA_1(x^2)$

$$A_0(x) = a_0 + a_2x + a_4x^2 + \cdots \qquad A_1(x) = a_1 + a_3x + a_5x^2 + \cdots$$

If $i < \frac{n}{2}$ then

$$\begin{aligned} A(w_n^i) &= A_0\left(\left(w_n^i\right)^2\right) + w_n^i A_1\left(\left(w_n^i\right)^2\right) \\ &= A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right) \end{aligned}$$

FFT Main Body

Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts: $A(x) = A_0(x^2) + xA_1(x^2)$

$A_0(x) = a_0 + a_2x + a_4x^2 + \cdots \qquad A_1(x) = a_1 + a_3x + a_5x^2 + \cdots$

If $i < \frac{n}{2}$ then

$$\begin{aligned} A(w_n^i) &= A_0\left(\left(w_n^i\right)^2\right) + w_n^i A_1\left(\left(w_n^i\right)^2\right) \\ &= A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right) \end{aligned}$$

$$\begin{aligned} A(w_n^{i+n/2}) &= A_0\left(\left(w_n^{i+n/2}\right)^2\right) - w_n^i A_1\left(\left(w_n^{i+n/2}\right)^2\right) \\ &= A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right) \end{aligned}$$

FFT Algorithm

Just saw that when n even

(always true since n a power of 2)

$$
\begin{aligned}
A(w_n^i) &= A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right) \\
A(w_n^{i+n/2}) &= A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right)
\end{aligned}
$$

FFT Algorithm

Just saw that when n even

(always true since n a power of 2)

$$
\begin{aligned}
A(w_n^i) &= A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right) \\
A(w_n^{i+n/2}) &= A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right)
\end{aligned}
$$

Note that $A_0(x), A_1(x)$ have degree $\frac{n}{2} - 1$ so can recursively call FFT to evaluate them on the $n/2$ roots of unity.

FFT Algorithm

Just saw that when n even

(always true since n a power of 2)

$$A(w_n^i) = A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right)$$

$$A(w_n^{i+n/2}) = A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right)$$

Note that $A_0(x), A_1(x)$ have degree $\frac{n}{2} - 1$ so can recursively call FFT to evaluate them on the $n/2$ roots of unity.

$\underline{FFT_n(A)}$

If $n = 1$

  return a_0

otherwise

  Evaluate $FFT_{n/2}(A_0)$, $FFT_{n/2}(A_1)$

  Calculate $A(w_n^i)$ for all i using those precalculated values

FFT Algorithm

Just saw that when n even

(always true since n a power of 2)

$$A(w_n^i) = A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right)$$

$$A(w_n^{i+n/2}) = A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right)$$

Note that $A_0(x), A_1(x)$ have degree $\frac{n}{2} - 1$ so can recursively call FFT to evaluate them on the $n/2$ roots of unity.

$\underline{FFT_n(A)}$ $\quad T(n)$

If $n = 1$

return a_0

otherwise

Evaluate $FFT_{n/2}(A_0)$, $FFT_{n/2}(A_1)$ $\quad 2T(n/2)$

Calculate $A(w_n^i)$ for all i using those precalculated values $\quad O(n)$

Running Time: $T(n) = 2T(n/2) + O(n)$

FFT Algorithm

Just saw that when n even

(always true since n a power of 2)

$$A(w_n^i) = A_0\left(w_{n/2}^i\right) + w_n^i A_1\left(w_{n/2}^i\right)$$

$$A(w_n^{i+n/2}) = A_0\left(w_{n/2}^i\right) - w_n^i A_1\left(w_{n/2}^i\right)$$

Note that $A_0(x), A_1(x)$ have degree $\frac{n}{2} - 1$ so can recursively call FFT to evaluate them on the $n/2$ roots of unity.

$\underline{FFT_n(A)}$ $\quad T(n)$

If $n = 1$

  return a_0

otherwise

  Evaluate $FFT_{n/2}(A_0)$, $FFT_{n/2}(A_1)$ $\quad 2T(n/2)$

  Calculate $A(w_n^i)$ for all i using those precalculated values $\quad O(n)$

Running Time: $T(n) = 2T(n/2) + O(n)$

$$\Rightarrow \; T(n) = O(n \log n)$$

FFT Based Interpolation

- Just saw that $FFT_n(A)$ evaluates $A(x)$ at n roots of unity in $O(n \log n)$ time
- Now need to see how to efficiently interpolate A's coefficients given $DFT_n(A) = \{A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1})\}$
- Amazingly, (proof soon)

  Lemma: If $d_j = A(w_n^j)$ set $D(x) = \sum_j d_j x^j$. Then, for all $i < n$,

  $$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

- This says that given $DFT_n(A)$ we can calculate coefficients of A in $O(n \log n)$ time by another call to FFT and dividing answers by n.

A New Approach To Polynomial Multiplication

$A(x) : a_0, \ldots, a_{n-1}$
$B(x) : b_0, \ldots, b_{n-1}$

Polynomial Multiplication/Convolution

$O(n^2)$

$C(x) : c_0, \ldots, c_{2n-1}$
$C(x) = A(x)B(x)$

Use FFT to valuate
$A(w_n^i)$, $B(w_n^i)$
for all $i < 2n - 1$

Interpolate
coefficients of $C(x)$
using FFT

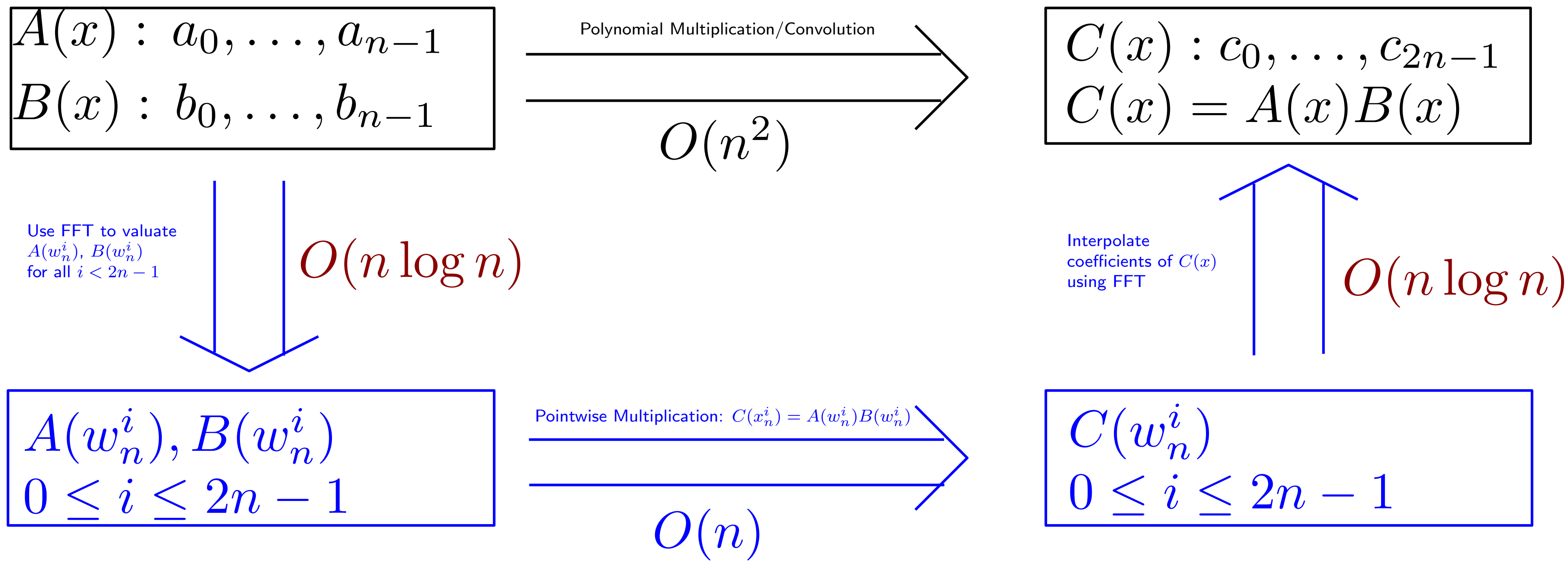
$A(w_n^i), B(w_n^i)$
$0 \leq i \leq 2n - 1$

Pointwise Multiplication: $C(x_n^i) = A(w_n^i)B(w_n^i)$

$O(n)$

$C(w_n^i)$
$0 \leq i \leq 2n - 1$

A New Approach To Polynomial Multiplication



- Following the blue arrows gives an $O(n \log n)$ polynomial multiplication algorithm!

Last Piece

Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

Last Piece

Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

First note $$\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \begin{cases} n & \text{if } t = 0, n \\ 0 & \text{if } t \neq 0, n \end{cases}$$

Last Piece

Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

First note $$\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \begin{cases} n & \text{if } t = 0, n \\ 0 & \text{if } t \neq 0, n \end{cases}$$

If $t = 0, n$, $w_n^t = 1$ and obvious. If $t \neq 0, 1$, $\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \frac{w_n^{n\,t} - 1}{w_n^t - 1} = 0$

Last Piece

Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

First note
$$\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \begin{cases} n & \text{if } t = 0, n \\ 0 & \text{if } t \neq 0, n \end{cases}$$

If $t = 0, n$, $w_n^t = 1$ and obvious. If $t \neq 0, 1$, $\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \frac{{w_n^t}^n - 1}{w_n^t - 1} = 0$

$$\begin{aligned} D(w_n^i) &= \sum_{j=0}^{n-1} d_j \left(w_n^i\right)^j \\ &= \sum_{j=0}^{n-1} \left(\sum_{k=0}^{n-1} a_k \left(w_n^j\right)^k \right) \left(w_n^i\right)^j \\ &= \sum_{k=0}^{n-1} a_k \left(\sum_{j=0}^{n-1} \left(w_n^{k+i}\right)^j \right) \end{aligned}$$

Last Piece

Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

First note $$\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \begin{cases} n & \text{if } t = 0, n \\ 0 & \text{if } t \neq 0, n \end{cases}$$

If $t = 0, n$, $w_n^t = 1$ and obvious. If $t \neq 0, 1$, $\sum_{j=0}^{n-1} \left(w_n^t\right)^j = \frac{w_n^{nt} - 1}{w_n^t - 1} = 0$

$$\begin{aligned} D(w_n^i) &= \sum_{j=0}^{n-1} d_j \left(w_n^i\right)^j \\ &= \sum_{j=0}^{n-1} \left(\sum_{k=0}^{n-1} a_k \left(w_n^j\right)^k \right) \left(w_n^i\right)^j \\ &= \sum_{k=0}^{n-1} a_k \left(\sum_{j=0}^{n-1} \left(w_n^{k+i}\right)^j \right) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases} \end{aligned}$$

Odds and Ends

- Note that FFT presented here as divide-and-conquer algorithm
- Can also be written iteratively and also implemented in a dedicated circuit (butterfly pattern)
- Can design $O(n \log n)$ algorithms that work on powers of other numbers, e.g., $n = 3^k$ or $n = 5^k$.
- Some other transforms using other orthogonal function bases can be implemented quickly, similar to the FFT, e.g., Hadamard Transforms
- Gauss actually developed something similar to FFT in 1805. Rediscovered many times afterwards
- Was "forgotten" until Cooley and Tukey published a paper describing it in 1965.
  After that, became one of the most used algorithms in the world!

A Matrix View of DFTs

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n^1 & w_n^2 & \cdots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \cdots & w_n^{2(n-1)} \\ \vdots & \vdots & \dots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} A(w_n^0) \\ A(w_n^1) \\ A(w_n^2) \\ \vdots \\ A(w_n^{n-1}) \end{pmatrix}$$

Let V be the Vandermonde matrix on the left and use A to denote the vector of the a_i.
The DFT can then be seen as calculating $VA = DFT(A)$.

A little bit of work (similar to lemma on slides) shows that the (j,k)-th entry of the inverse matrix of V has value w_n^{-kj}/n.
Using the fact that $w_n^{-k} = w_n^{n-k}$ we see that

The (j,k)-th entry of the inverse matrix of V has value w_n^{-kj}/n and $w_n^{-k} = w_n^{n-k}$ so

$$V = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n^1 & w_n^2 & \cdots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \cdots & w_n^{2(n-1)} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \quad \text{and} \quad V^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \cdots & w_n^{(n-1)(n-1)} \\ 1 & w_n^{n-2} & w_n^{2(n-2)} & \cdots & w_n^{(n-1)(n-2)} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ 1 & w_n^1 & w_n^2 & \cdots & w_n^{n-1} \end{pmatrix}$$

Since $$V \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} A(w_n^0) \\ A(w_n^1) \\ A(w_n^2) \\ \vdots \\ A(w_n^{n-1}) \end{pmatrix},$$ we have $$V^{-1} \begin{pmatrix} A(w_n^0) \\ A(w_n^1) \\ A(w_n^2) \\ \vdots \\ A(w_n^{n-1}) \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

This allows implementing the inverse DFT by mutiplying by V^{-1}.

But, from the above, multiplying by V^{-1} is the same as multiplying by V, flipping some of the results and then dividng by n.

This can be done in $O(n \log n)$ by running the FFT algorithm and then doing $O(n)$ more work.

Definition: The *Vandermonde matrix* on n values $x_1, x_2, \ldots, x_n$ is the matrix below.

If $x_i \neq x_j$ for all i, j, the Vandermonde matrix is invertable.

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{n-1} \\ \vdots & \vdots & \ldots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}$$