Design and Analysis of Algorithms

CSE 5311

Lecture 5 Divide and Conquer:

Fast Fourier Transform

Junzhou Huang, Ph.D.

Department of Computer Science and Engineering

Reviewing: Master Theorem

The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n),$$

where constants $a \geq 1$, $b > 1$, and f is asymptotically positive function

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

2. $f(n) = O(n^{\log_b a})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a}\, lgn)$

3. $f(n) = O(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

How to theoretically prove it?

Fast Fourier Transform

- **Applications**
  - Optics, acoustics, quantum physics, telecommunications, control systems
  - Signal processing, speech recognition, data compression, image processing
  - Machine learning, data mining, computer vision, big data analytics
  - DVD, JPEG, MP3, MRI, CAT scan
- **Charles van Loan:**
  - The FFT is one of the truly great computational developments of this [20th] century.
  - It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.

Fast Fourier Transform

- **History**
  - Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.
  - Runge-König (1924). Laid theoretical groundwork.
  - Danielson-Lanczos (1942). Efficient algorithm.
  - Cooley-Tukey (1965). Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.
  - Importance not fully realized until advent of digital computers.

Representation of Polynomials

A polynomial in the variable x over an algebraic field F is representation of a function A(x) as a formal sum

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

•Coefficient representation

$$a = (a_0, a_1, \ldots a_{n-1})$$

•Point-value representation $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$

	Coefficient representation	Point-value representation
Adding	$\Theta(n)$	$\Theta(n)$
Multiplication	$\Theta(n^2)$	$\Theta(n)$

Polynomials: Coefficient Representation

- **Polynomial. [coefficient representation]**

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

- **Add: O(n) arithmetic operations**

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

- **Evaluate: O(n) using Horner's method.**

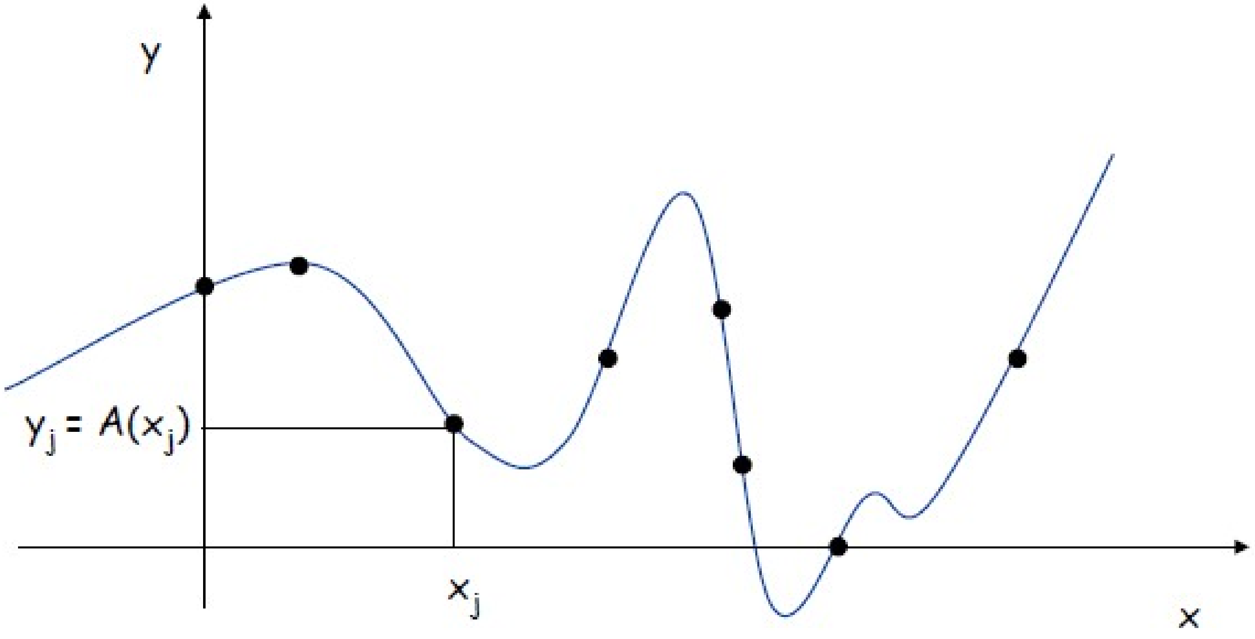
$$A(x) = a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))\cdots))$$

- **Multiply (convolve): O(n^2) using brute force.**

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

Polynomials: Point-Value Representation

- **Fundamental theorem of algebra. [Gauss, PhD thesis]: A degree *n* polynomial with complex coefficients has *n* complex roots.**
- **Corollary. A degree n-1 polynomial A(x) is uniquely specified by its evaluation at n distinct values of x.**



Polynomials: Point-Value Representation

- **Polynomial. [Point-value representation]**

$$A(x):\ (\mathrm{x}_0, y_0),\ \ldots,(\mathrm{x}_{\text{n-1}}, y_{n-1})$$
$$B(x):\ (\mathrm{x}_0,\ z_0),\ \ldots,(\mathrm{x}_{\text{n-1}}, z_{n-1})$$

- **Add: O(n) arithmetic operations**

$$A(x)+B(x):\ \ (\mathrm{x}_0, y_0+z_0),\ldots,(\mathrm{x}_{\text{n-1}}, y_{n-1}+z_{n-1})$$

- **Multiple: O(n), extend A(x) and B(x) to 2n-1 points**

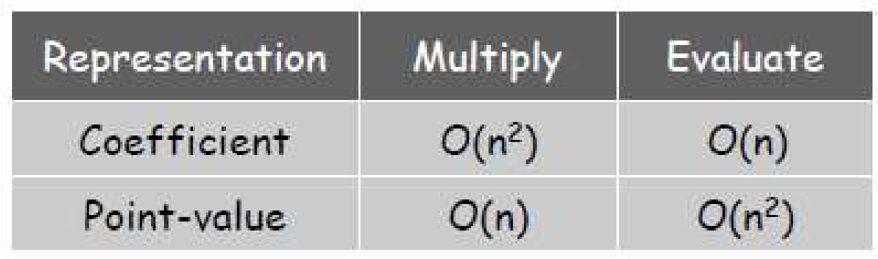
$$A(x)\times B(x):\ \ (\mathrm{x}_0, y_0\times z_0),\ldots,(\mathrm{x}_{\text{2n-1}}, y_{2n-1}\times z_{2n-1})$$

- **Evaluate: O(n^2) using Lagrange's formula**

$$A(x)=\sum_{k=0}^{n-1} y_k \frac{\prod_{j\neq k}(x-x_j)}{\prod_{j\neq k}(x_k-x_j)}$$

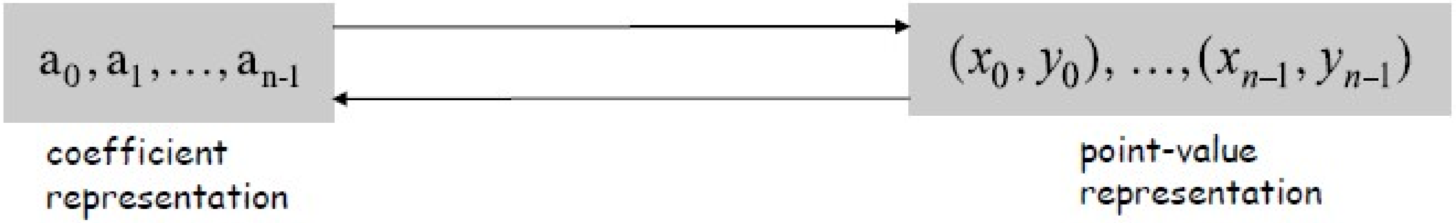
Converting Between Two Polynomial Representations

- Tradeoff between fast evaluation or fast multiplication. We want both!



Representation	Multiply	Evaluate
Coefficient	$O(n^2)$	$O(n)$
Point-value	$O(n)$	$O(n^2)$

- Goal. Make all ops fast by efficiently converting between two representations.



$a_0, a_1, \ldots, a_{n-1}$

coefficient representation

$(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

point-value representation

Converting Between Two Polynomial Representations

- **Coefficient to point-value:** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at *n* distinct points $x_0, \ldots , x_{n-1}$.

Brute Force!

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$O(n^2)$ for matrix-vector multiply

$O(n^3)$ for Gaussian elimination

Vandermonde matrix is invertible iff x_i distinct

- **Point-value to coefficient:** Given n distinct points $x_0, \ldots , x_{n-1}$ and values $y_0, \ldots , y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ that has given values at given points.

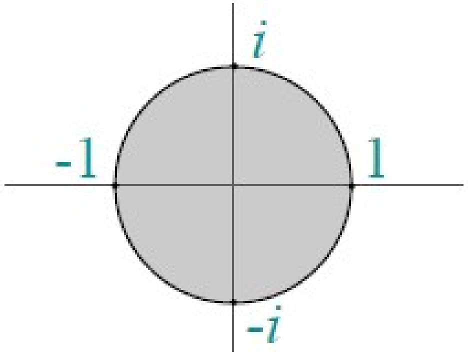
Coefficient to Point-Value Representation: Intuition

- **Coefficient to point-value:** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at *n* distinct points $x_0, \ldots, x_{n-1}$.

- **Divide. Break polynomial up into even and odd powers.**

$$
\begin{aligned}
A(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7\\
A_{even}(x) &= a_0 + a_2x + a_4x^2 + a_6x^3.\\
A_{odd}(x) &= a_1 + a_3x + a_5x^2 + a_7x^3.\\
A(\ x) &= A_{even}(x^2) + x\ A_{odd}(x^2).\\
A(-x) &= A_{even}(x^2) - x\ A_{odd}(x^2).
\end{aligned}
$$

Why? Useful Trick



- **Intuition. Choose four points to be ±1, ±i.**

$$
\begin{aligned}
A(\ 1) &= A_{even}(\ 1) + 1\ A_{odd}(\ 1).\\
A(-1) &= A_{even}(\ 1) - 1\ A_{odd}(\ 1).\\
A(\ i) &= A_{even}(-1) + i\ A_{odd}(-1).\\
A(-i) &= A_{even}(-1) - i\ A_{odd}(-1).
\end{aligned}
$$

Can evaluate polynomial of degree ≤ n at 4 points by evaluating two polynomials of degree ≤ n/2 at 2 points.

Useful Trick

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + . . . + a_{n-1}x^{n-1}$$

$$A_{even}(x) = a_0 + a_2x + a_4x^2 + . . . + a_{n-2}x^{(n-2)/2}$$

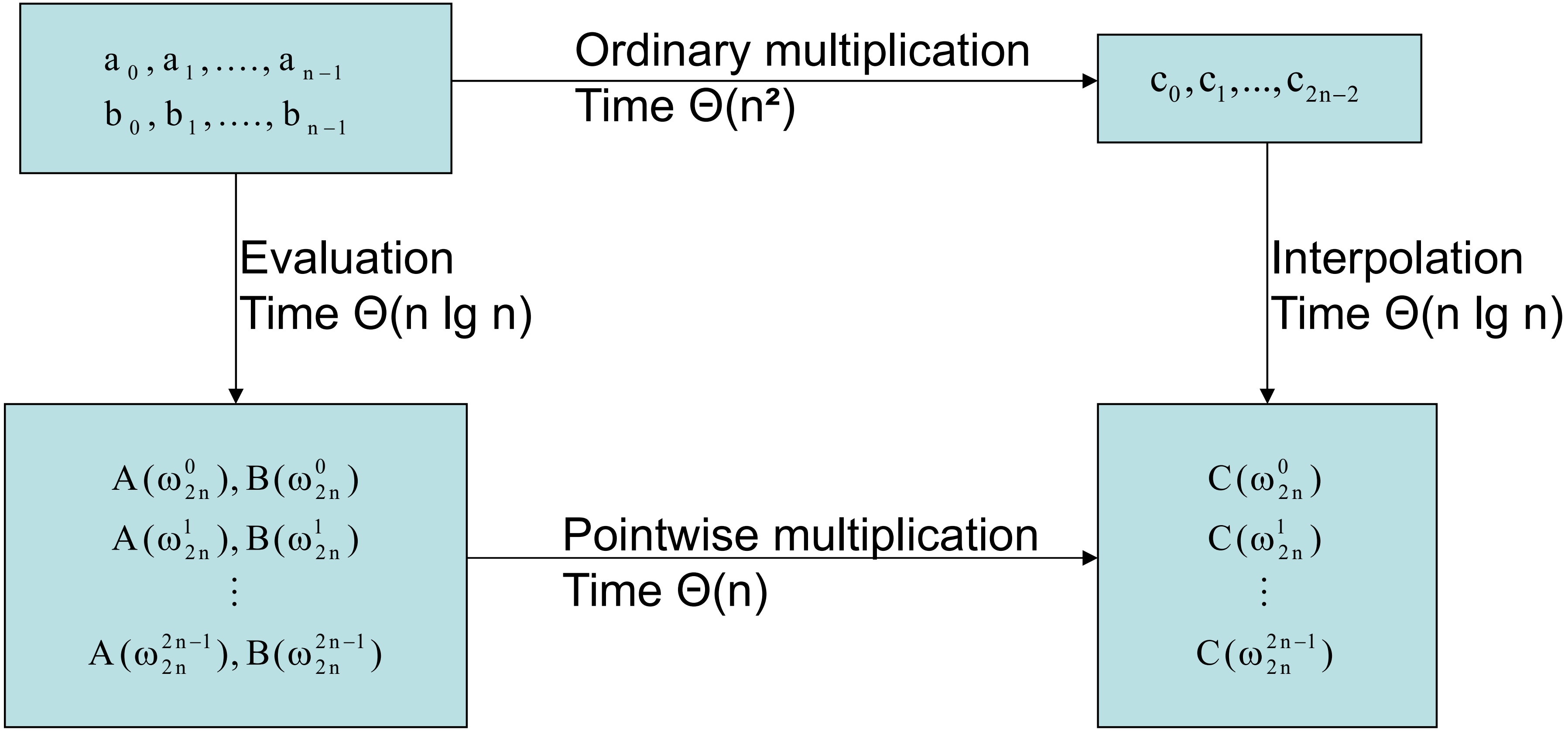
$$A_{odd}(x) = a_1 + a_3x + a_5x^2 + \ldots + a_{n-1}x^{(n-2)/2}$$

Show: $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$

Fast Multiplication

Question. Can we use the linear-time multiplication method for polynomials in point-value form to expedite polynomial multiplication in coefficient form?

Answer. Yes, but we are to be able to convert quickly from one form to another.



$a_0, a_1, \ldots., a_{n-1}$

$b_0, b_1, \ldots., b_{n-1}$

Ordinary multiplication

Time Θ(n²)

$c_0, c_1, \ldots, c_{2n-2}$

Evaluation

Time Θ(n lg n)

Interpolation

Time Θ(n lg n)

$A(\omega_{2n}^{0}), B(\omega_{2n}^{0})$

$A(\omega_{2n}^{1}), B(\omega_{2n}^{1})$

$\vdots$

$A(\omega_{2n}^{2n-1}), B(\omega_{2n}^{2n-1})$

Pointwise multiplication

Time Θ(n)

$C(\omega_{2n}^{0})$

$C(\omega_{2n}^{1})$

$\vdots$

$C(\omega_{2n}^{2n-1})$

Complex Roots of Unity

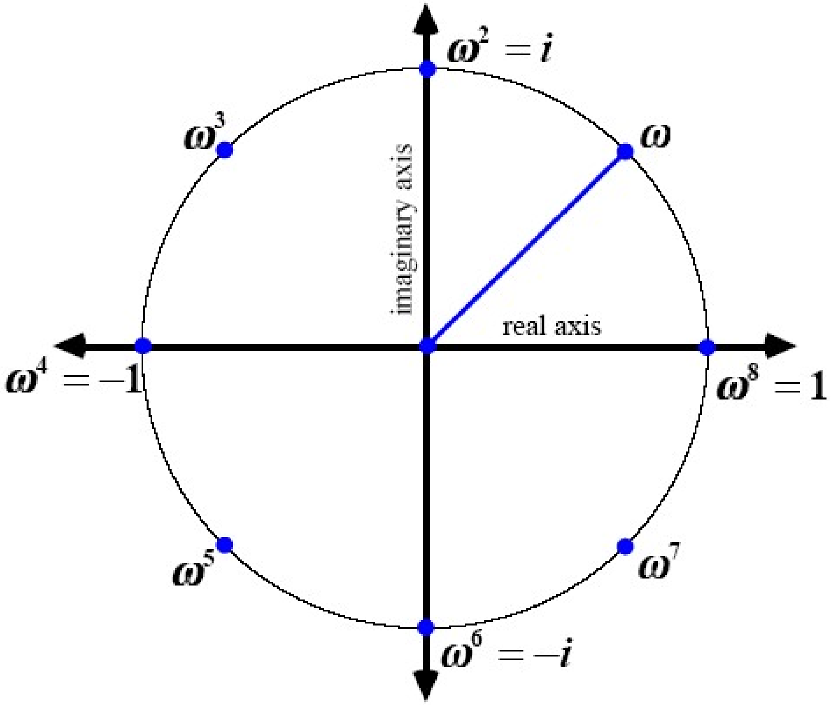
$$Z^n - 1 = 0$$

There are exactly n complex roots of unity. They form a cyclic multiplication group:

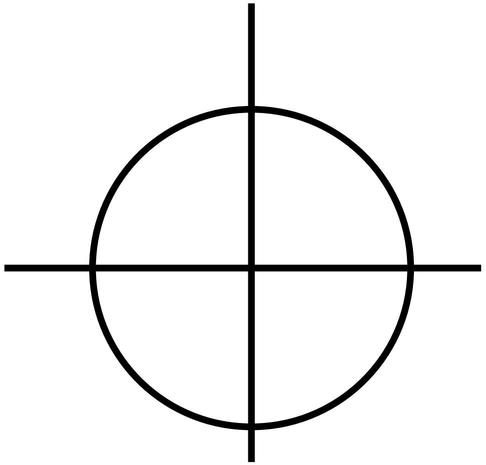
$$\omega_k = e^{2\pi ik/n}$$

The value $\omega_1 = e^{2\pi i/n}$ is called **the primitive root of unity**; all of the other complex roots are powers of it.

$\omega = e^{2\pi i/8}$ is the 8th root of unity



Complex Analysis



- Polar coordinates: $re^{\theta i}$
- $e^{\theta i} = \cos\theta + i \sin\theta$
- a is an nth root of unity if $a^n = 1$
- Square roots of unity: +1, -1
- Fourth roots of unity: +1, -1, i, -i
  - Eighth roots of unity: +1, -1, i, -i, $\beta + i\beta$, $\beta - i\beta$, $-\beta + i\beta$, $-\beta - i\beta$ where β = sqrt(2)

$e^{2\pi ki/n}$

- $e^{2\pi i} = 1$
- $e^{\pi i} = -1$
- n^{th} roots of unity: $e^{2\pi ki/n}$ for $k = 0 \ldots n-1$
- Notation: $\omega_{k,n} = e^{2\pi ki/n}$

- Interesting fact:

$$1 + \omega_{k,n} + \omega^2_{k,n} + \omega^3_{k,n} + \ldots + \omega^{n-1}_{k,n} = 0 \qquad \text{for } k \mathrel{!}= 0$$

Discrete Fourier Transform (DFT)

Coefficient to point-value: Let F(x) be the polynomial with degree-bound n (power of 2), where $F(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_0$

Key idea: choose $x_k = \omega^k$ where ω is principal nth root of unity.

Let $y_k = F(\omega^k)$. Then

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

The vector $y=(y_0,y_1,...y_{n-1})$ is called the *Discrete Fourier Transform* of vector *a*. The matrix is denoted by $F_n(\omega)$.

How to find F_n^{-1}?

Proposition. Let ω be a primitive *l*-th root of unity over a field L. Then

$$\sum_{k=0}^{l-1} \omega^k = \begin{Bmatrix} 0 & \mathit{if}\ l > 1 \\ 1 & \mathit{otherwise} \end{Bmatrix}$$

Proof. The $l=1$ case is immediate since ω=1.

Since ω is a primitive *l*-th root, each ω^k, $k\neq0$ is a distinct *l*-th root of unity.

$$Z^l - 1 = (Z-\omega_l^0)(Z-\omega_l)(Z-\omega_l^2)\ldots(Z-\omega_l^{l-1}) =$$

$$= Z^l - (\sum_{k=0}^{l-1}\omega_l^k)Z^{l-1} + \ldots + (-1)^l \prod_{k=0}^{l-1}\omega_l^k$$

Comparing the coefficients of Z^{l-1} on the left and right hand sides of this equation proves the proposition.

Inverse Matrix to F_n

Proposition. Let ω be an n-th root of unity. Then, $F_n(\omega)\cdot F_n(\omega^{-1}) = nE_n$

Proof. The ij^{th} element of $F_n(\omega)F_n(\omega^{-1})$ is

$$\sum_{k=0}^{n-1} \omega^{ik}\omega^{-ik} = \sum_{k=0}^{n-1} \omega^{k(i-j)} = \begin{Bmatrix} 0, \text{ if } i \neq j \\ n, \text{ otherwise} \end{Bmatrix}$$

The *i=j* case is obvious. If *i≠j* then ω^{i-j} will be a primitive root of unity of order *l*, where *l*|*n*. Applying the previous proposition completes the proof.

So, $F_n^{-1}(\omega) = \dfrac{1}{n}F_n(\omega^{-1})$

Evaluating	$\mathbf{y} = F_n(\omega)\,\mathbf{a}$
Interpolation	$\mathbf{a} = \dfrac{1}{n}F_n(\omega^{-1})\,\mathbf{y}$

Fast Fourier Transform

Goal. Evaluate a degree n-1 polynomial $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_{n-1}x^{n-1}$ at its nth roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break polynomial up into even and odd powers.

$$\begin{aligned}
A_{even}(x) &= a_0 + a_2x + a_4x^2 + \ldots + a_{n/2-2}\, x^{(n-1)/2} \\
A_{odd}(x) &= a_1 + a_3x + a_5x^2 + \ldots + a_{n/2-1}\, x^{(n-1)/2}. \\
A(x) &= A_{even}(x^2) + x\, A_{odd}(x^2).
\end{aligned}$$

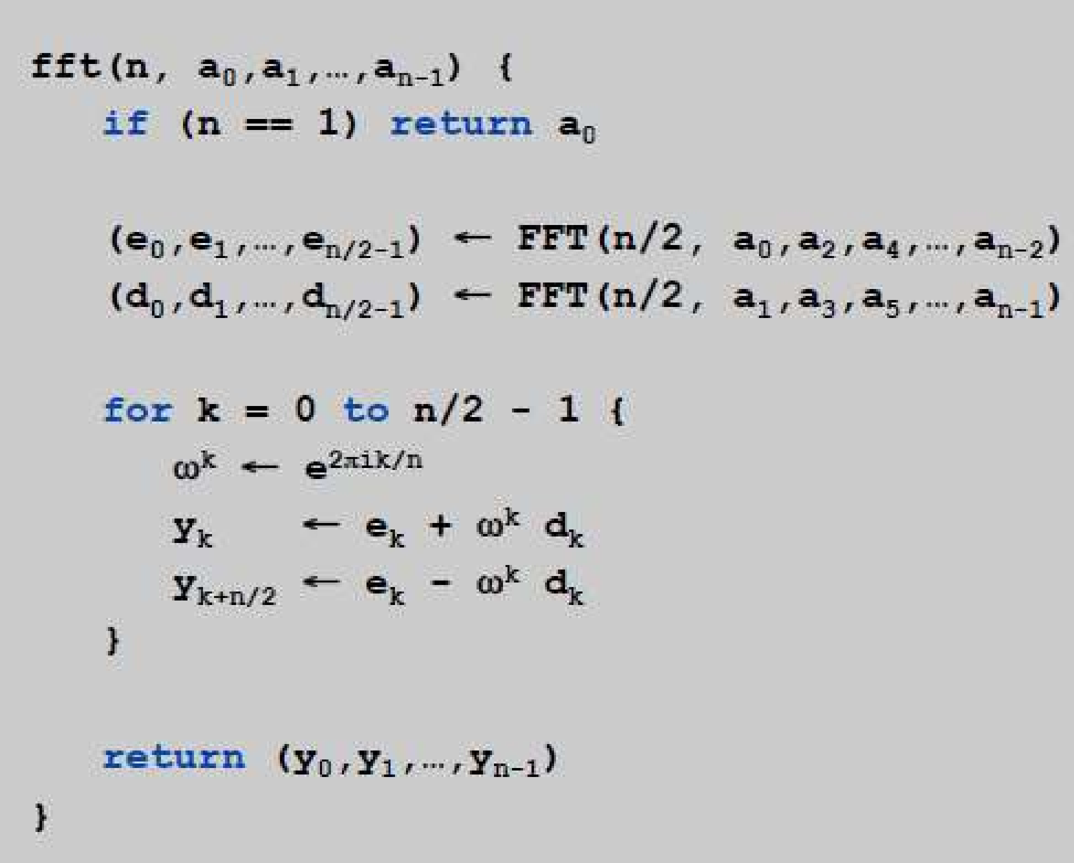
Conquer. Evaluate degree $A_{even}(x)$ and $A_{odd}(x)$ at the (n/2)-th roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

Combine.

$$\begin{aligned}
A(\omega^k) &= A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2 \\
A(\omega^{k+n/2}) &= A_{even}(\nu^k) \underset{\substack{\uparrow \\ \omega^{k+n/2} = -\omega^k}}{-} \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2
\end{aligned}$$

$$\nu^k = (\omega^k)^2 = (\omega^{k+n/2})^2$$

Recursive FFT



```
fft(n, $a_0,a_1,\dots,a_{n-1}$) {
    if (n == 1) return $a_0$

    $(e_0,e_1,\dots,e_{n/2-1})$ ← FFT(n/2, $a_0,a_2,a_4,\dots,a_{n-2}$)
    $(d_0,d_1,\dots,d_{n/2-1})$ ← FFT(n/2, $a_1,a_3,a_5,\dots,a_{n-1}$)

    for k = 0 to n/2 - 1 {
        $\omega^k$ ← $e^{2\pi ik/n}$
        $y_k$       ← $e_k + \omega^k d_k$
        $y_{k+n/2}$ ← $e_k - \omega^k d_k$
    }

    return $(y_0,y_1,\dots,y_{n-1})$
}
```

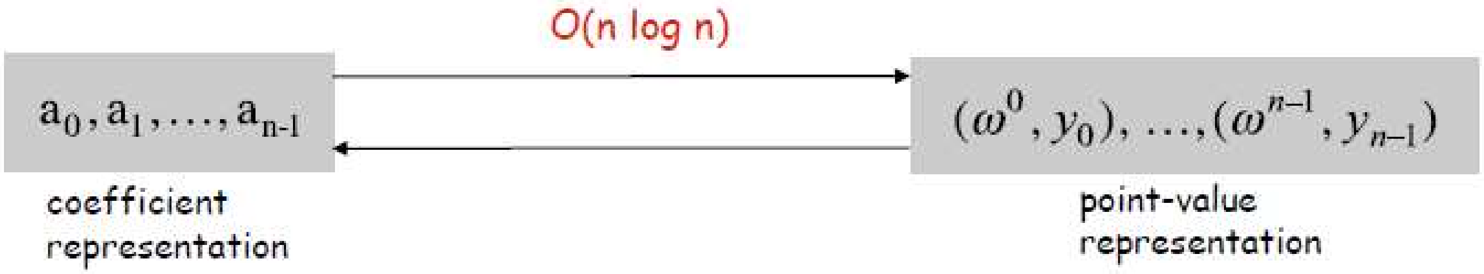
```
n ← length[a]
if n = 1
    then return a
$\omega_n \leftarrow e^{2\pi i/n}$
$\omega \leftarrow 1$
$a^{[0]} \leftarrow (a_0, a_2, \dots, a_{n-2})$
$a^{[1]} \leftarrow (a_1, a_3, \dots, a_{n-1})$
$y^{[0]} \leftarrow$ Recursive-FFT($a^{[0]}$)
$y^{[1]} \leftarrow$ Recursive-FFT($a^{[1]}$)
for k ← 0 to n/2-1
        do $y_k \leftarrow y_k^{[0]} + \omega y_k^{[1]}$
           $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega y_k^{[1]}$
           $\omega \leftarrow \omega \omega_n$
return y
```

Time of the Recursive-FFT

To determine the running time of procedure Recursive-FFT, we note, that exclusive of the recursive calls, each invocation takes time Θ(n), where n is the length of the input vector.The recurrence for the running time is therefore

$$\mathbf{T(n) = 2T(n/2) + \Theta(n) = ?}$$

$$\Theta(n \log n)$$



O(n log n)

$a_0, a_1, \ldots, a_{n-1}$

coefficient representation

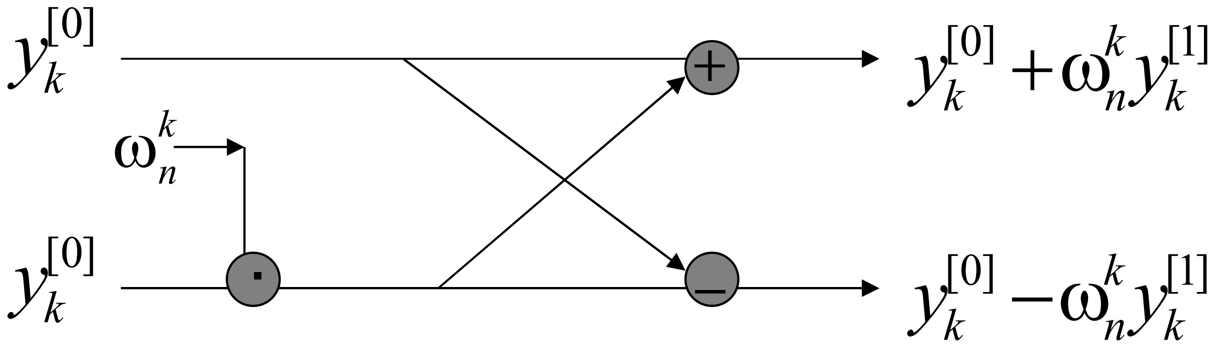
$(\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$

point-value representation

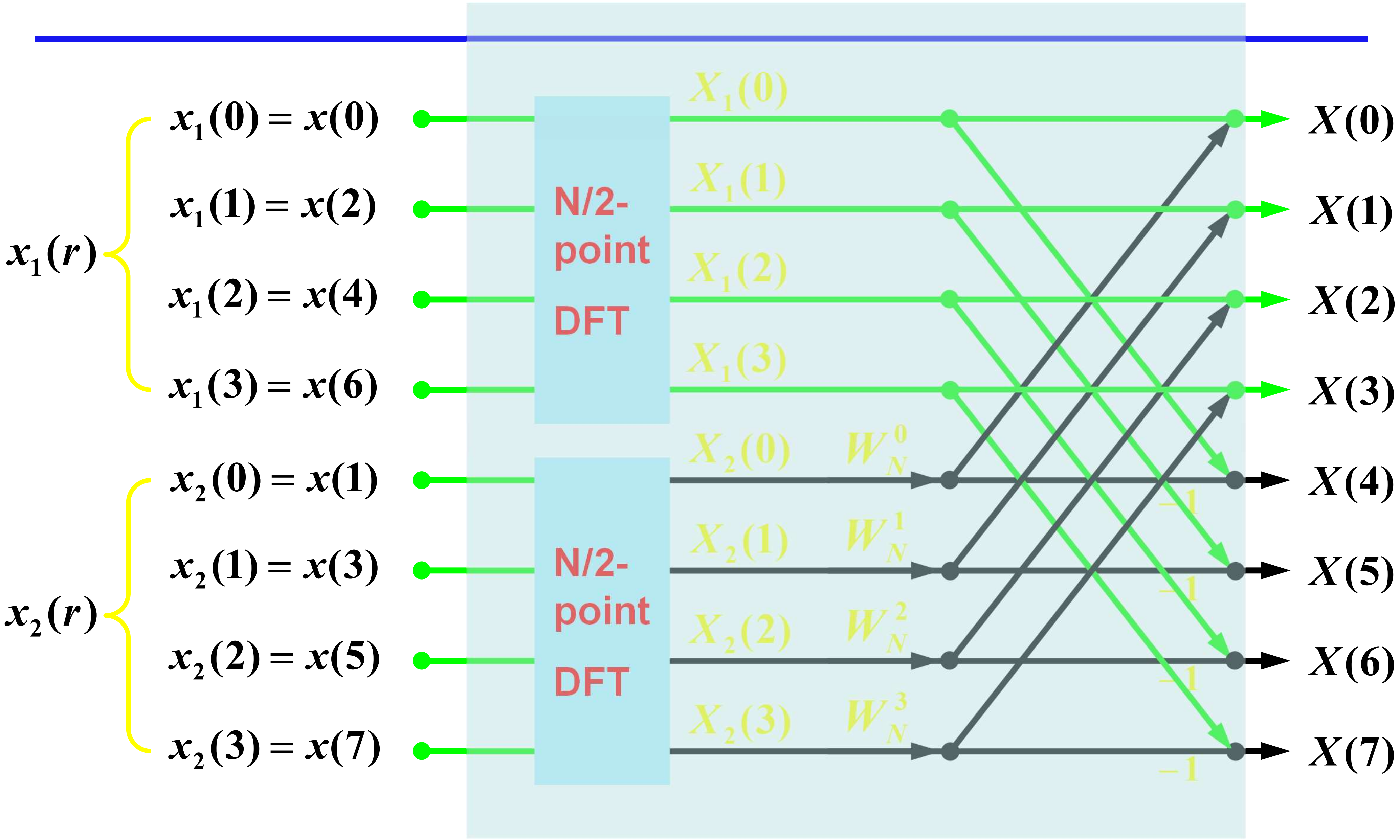
More Effective Implementations

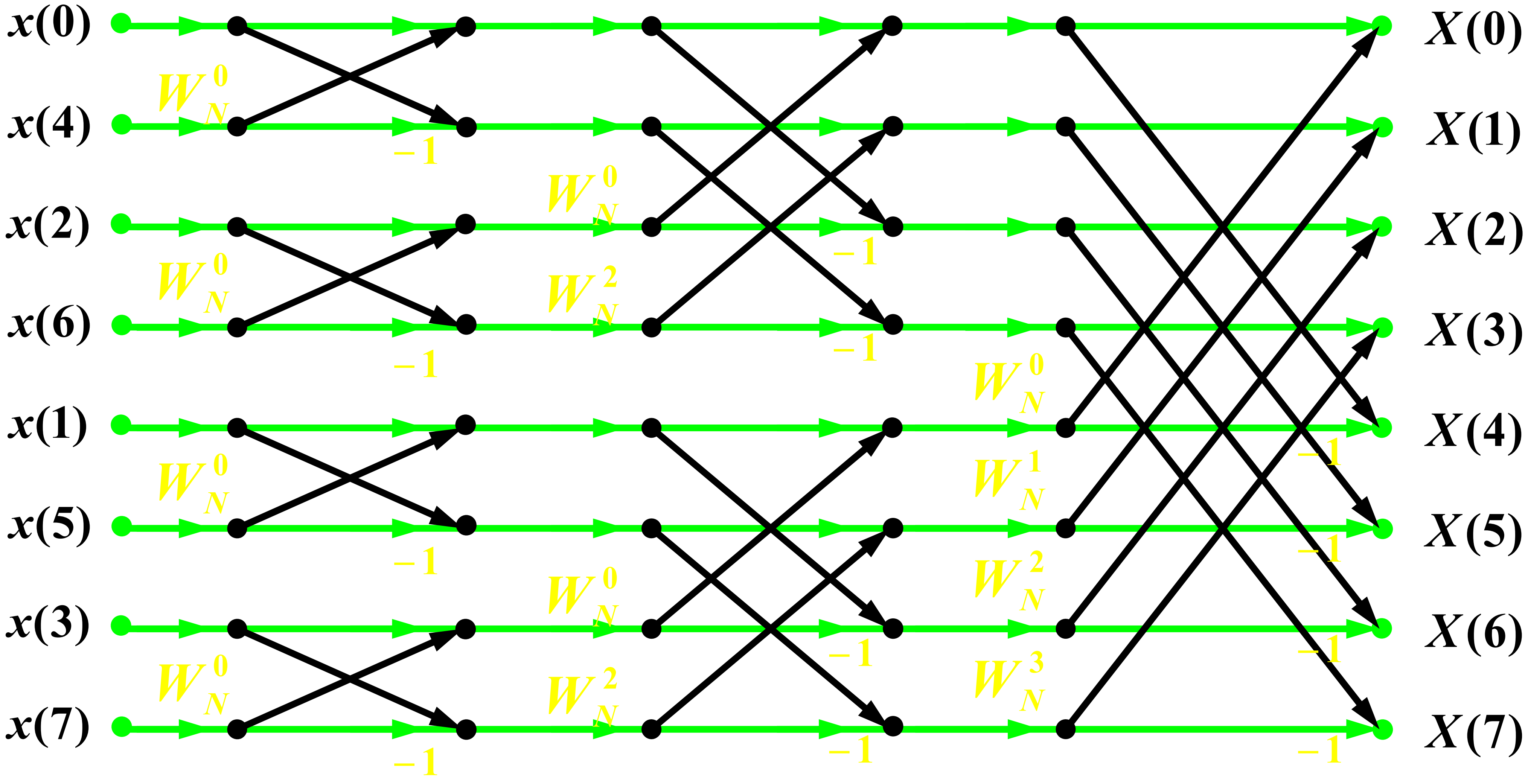
The **for** loop involves computing the value $\omega_n^k y_k^{[1]}$ twice.We can change the loop(the butterfly operation):

```
for  k ← 0 to n/2-1
    do t ← $\omega y_k^{[1]}$
        $y_k \leftarrow y_k^{[0]} + t$
        $y_{k+(n/2)} \leftarrow y_k^{[0]} - t$
        $\omega \leftarrow \omega\omega_n$
```

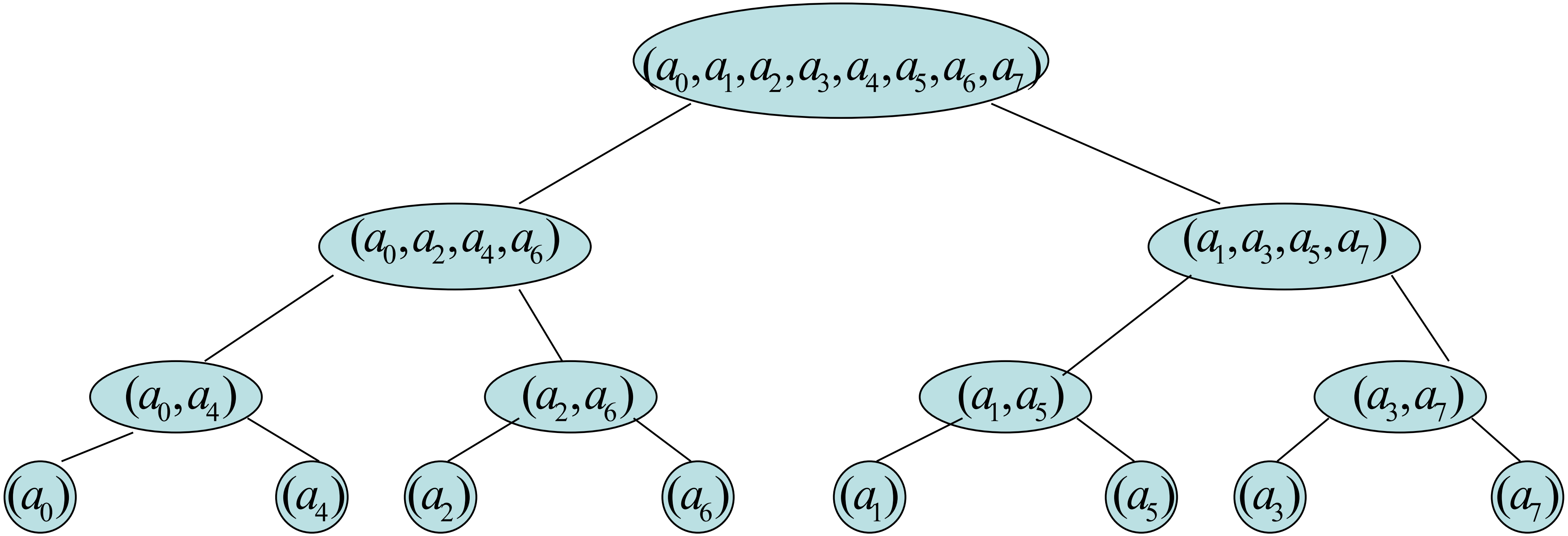

$y_k^{[0]}$
ω_n^k
$y_k^{[0]}$
·
+
–
$y_k^{[0]}+\omega_n^k y_k^{[1]}$
$y_k^{[0]}-\omega_n^k y_k^{[1]}$

There are 1 complex multiplication and 2 complex additions


$x_1(r)$
$x_1(0)=x(0)$
$x_1(1)=x(2)$
$x_1(2)=x(4)$
$x_1(3)=x(6)$
$x_2(r)$
$x_2(0)=x(1)$
$x_2(1)=x(3)$
$x_2(2)=x(5)$
$x_2(3)=x(7)$
N/2-point DFT
N/2-point DFT
$X_1(0)$
$X_1(1)$
$X_1(2)$
$X_1(3)$
$X_2(0)$
$X_2(1)$
$X_2(2)$
$X_2(3)$
W_N^0
W_N^1
W_N^2
W_N^3
-1
-1
-1
-1
$X(0)$
$X(1)$
$X(2)$
$X(3)$
$X(4)$
$X(5)$
$X(6)$
$X(7)$
N-point DFT


$x(0)$
$x(4)$
$x(2)$
$x(6)$
$x(1)$
$x(5)$
$x(3)$
$x(7)$
W_N^0
W_N^0
W_N^0
W_N^0
-1
-1
-1
-1
W_N^0
W_N^2
W_N^0
W_N^2
-1
-1
-1
-1
W_N^0
W_N^1
W_N^2
W_N^3
-1
-1
-1
-1
$X(0)$
$X(1)$
$X(2)$
$X(3)$
$X(4)$
$X(5)$
$X(6)$
$X(7)$

Recursion Tree

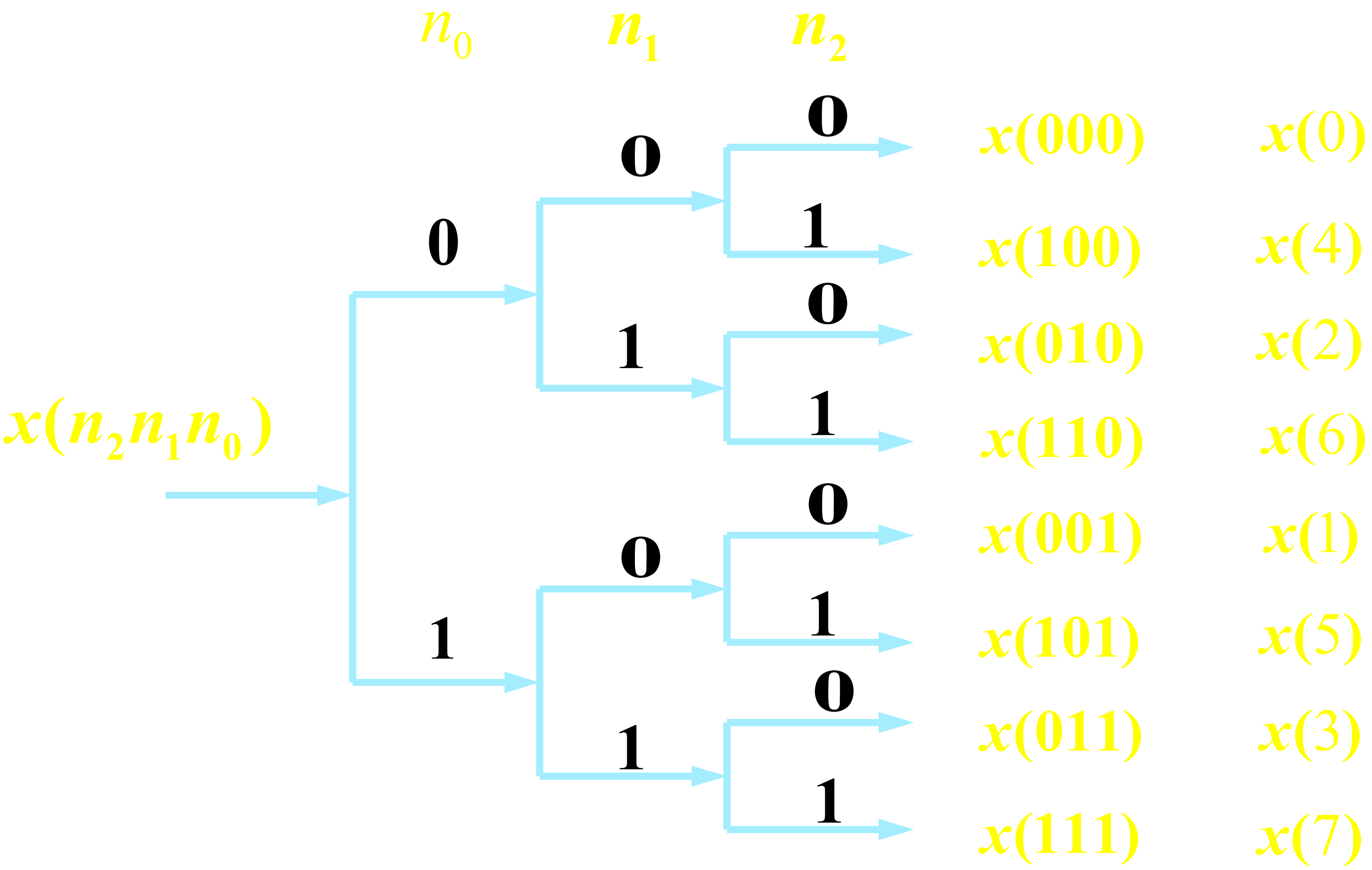


1) We take the elements in pairs, compute the DFT of each pair, using one butterfly operation, and replace the pair with its DFT

2) We take these n/2 DFT's in pairs and compute the DFT of the four vector elements

We take 2 (n/2)-element DFT's and combine them using n/2 butterfly operations into the final n-element DFT

Why Bit-reversed Order



Point-Value to Coefficient Representation: Inverse DFT

Goal. Given the values $y_0, \ldots, y_{n-1}$ of a degree n-1 polynomial at the n points $\omega^0, \omega^1, \ldots, \omega^{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ that has given values at given points.

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

↑ Inverse DFT

↑ Fourier matrix inverse $(F_n)^{-1}$

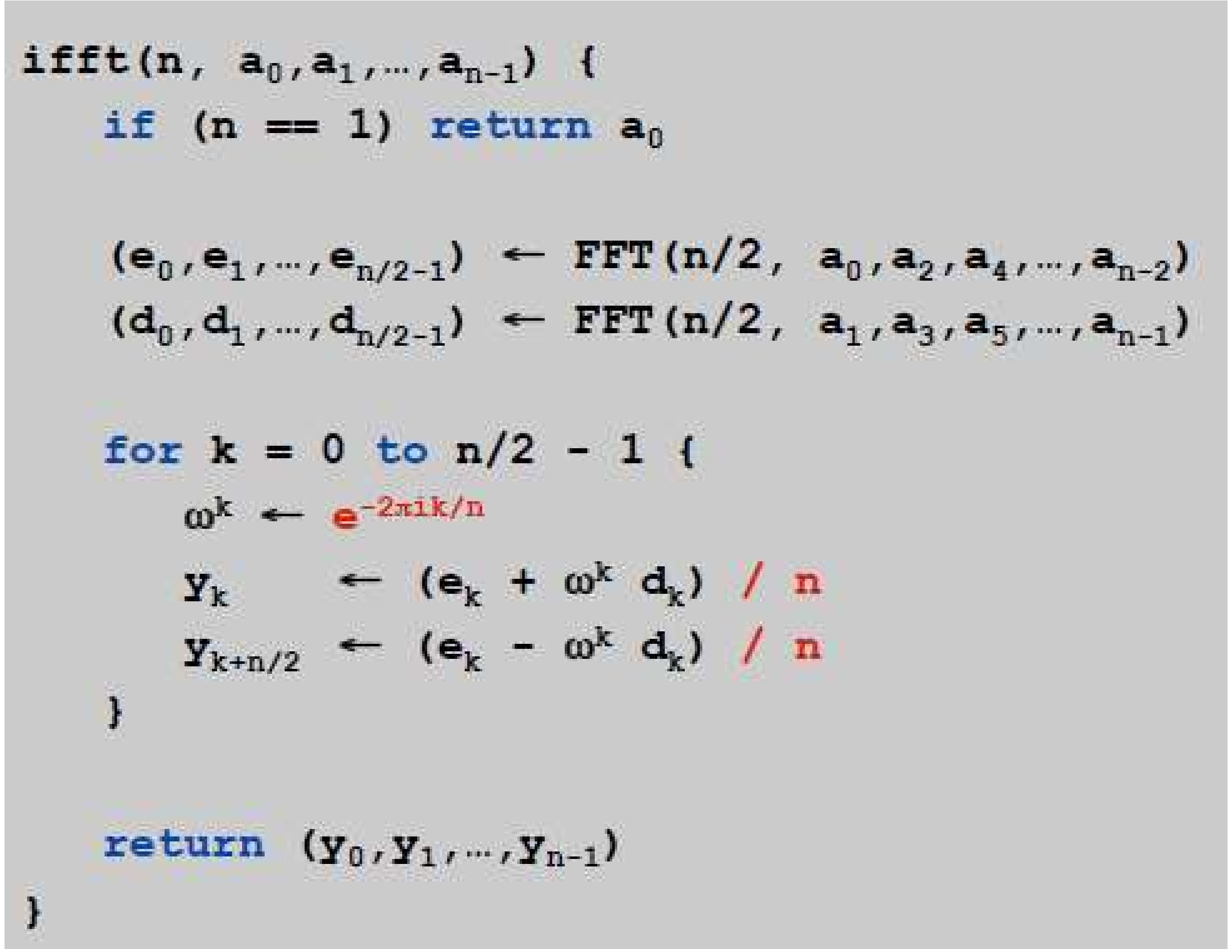
Inverse DFT

Inverse of Fourier matrix is given by following formula

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

To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i / n}$ as principal nth root of unity (and divide by n).

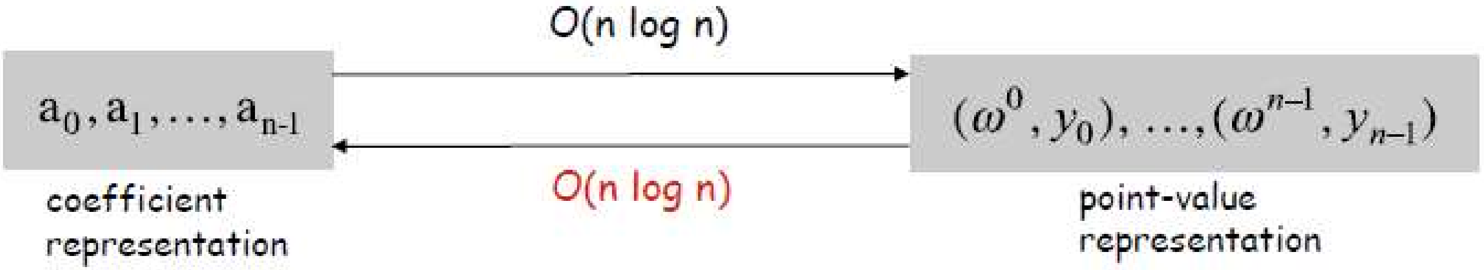
Inverse FFT



```
ifft(n, $a_0, a_1, \ldots, a_{n-1}$) {
   if (n == 1) return $a_0$

   ($e_0, e_1, \ldots, e_{n/2-1}$) ← FFT(n/2, $a_0, a_2, a_4, \ldots, a_{n-2}$)
   ($d_0, d_1, \ldots, d_{n/2-1}$) ← FFT(n/2, $a_1, a_3, a_5, \ldots, a_{n-1}$)

   for k = 0 to n/2 - 1 {
      $\omega^k$ ← $e^{-2\pi ik/n}$
      $y_k$       ← ($e_k$ + $\omega^k$ $d_k$) / n
      $y_{k+n/2}$ ← ($e_k$ - $\omega^k$ $d_k$) / n
   }

   return ($y_0, y_1, \ldots, y_{n-1}$)
}
```

Inverse FFT

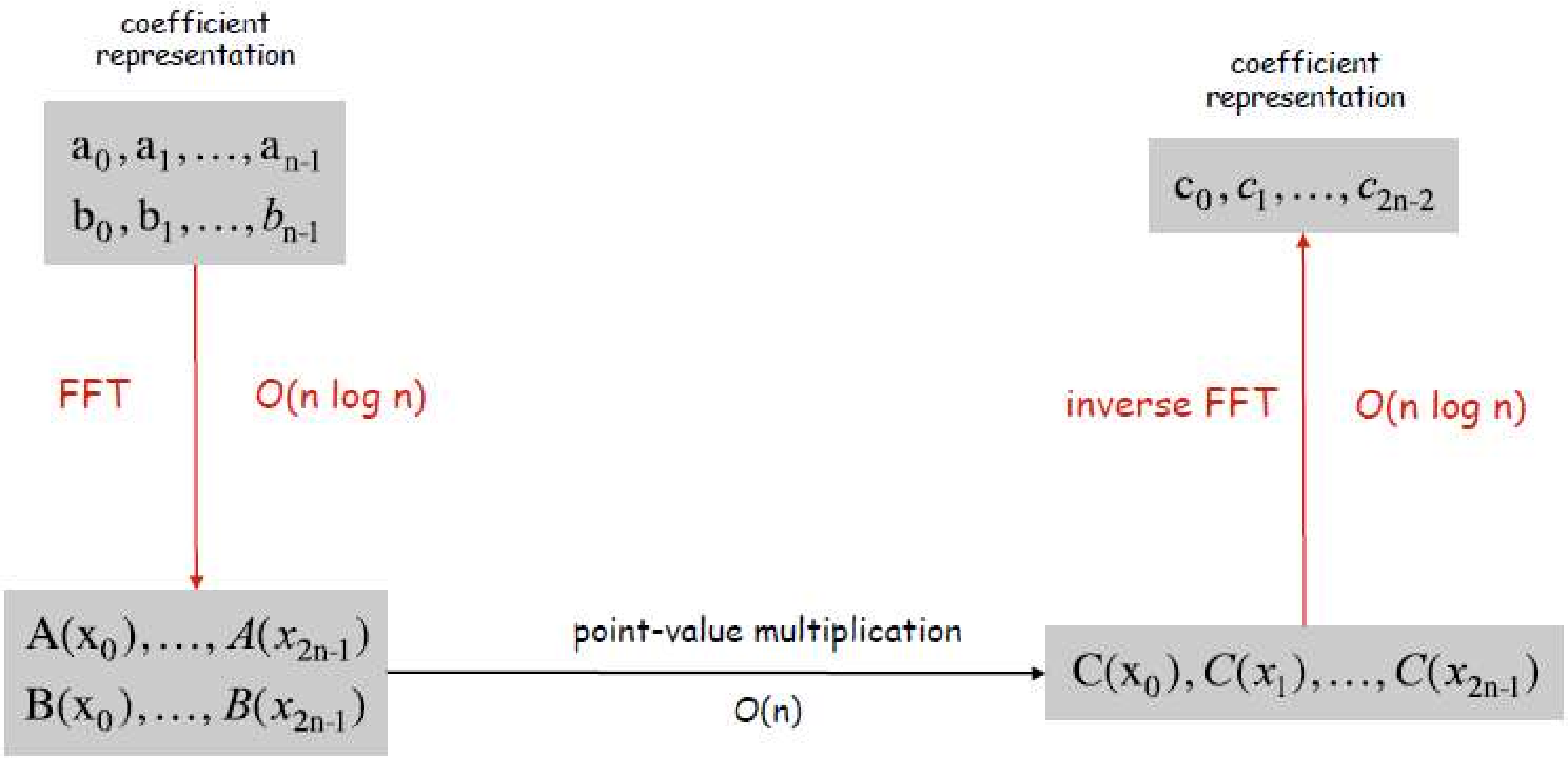
Theorem. Inverse FFT algorithm interpolates a degree n-1 polynomial given values at each of the nth roots of unity in O(n log n) steps.


O(n log n)
$a_0, a_1, \ldots, a_{n-1}$
$(\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$
O(n log n)
coefficient representation
point-value representation

Polynomial Multiplication

Theorem:

We can multiply two degree n-1 polynomials in O(n log n) steps.



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)

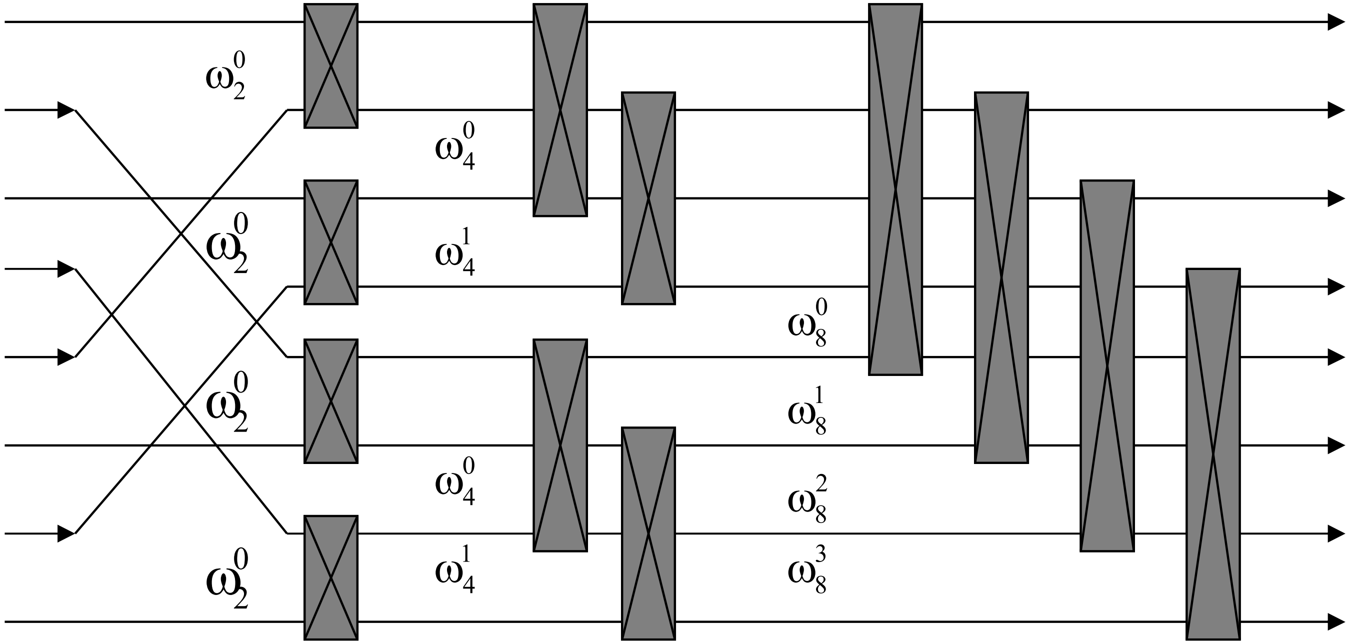
$C(x_0), C(x_1), \ldots, C(x_{2n-1})$

inverse FFT O(n log n)

coefficient representation

$c_0, c_1, \ldots, c_{2n-2}$

A Parallel FFT Circuit


ω_2^0
ω_2^0
ω_2^0
ω_2^0
ω_4^0
ω_4^1
ω_4^0
ω_4^1
ω_8^0
ω_8^1
ω_8^2
ω_8^3