

VII Selected Topics

- ✓ VII Selected Topics
 - 27 Multithreaded Algorithms
 - 28 Matrix Operations
 - 29 Linear Programming
 - 30 Polynomials and the FFT
 - 31 Number-Theoretic Algorithms
 - 32 String Matching
 - 33 Computational Geometry
 - 34 NP-Completeness
 - 35 Approximation Algorithms

30 Polynomials, Convolution and the FFT

$$A(x) = a_0x^0 + a_1x^1 + \dots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_jx^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_jx^j$$

$$C(x) = A(x) + B(x)?$$

$$C(x) = A(x) * B(x) = A(x) \cdot B(x) = A(x)B(x)$$

- Straightforward method of **adding** two polynomials of degree n takes $\Theta(n)$ time
- Straightforward method of **multiplying** them takes $\Theta(n^2)$ time
- Fast Fourier Transform (FFT, 快速傅里叶变换算法), can reduce the time to multiply polynomials to $\Theta(n \lg n)$
- Why? How?

Fourier Transform (FT) and FFT

- FT: the most common use is in signal processing

- ◆ $s(t)$: *time domain* [信号的时域表示]

- ◆ $S(\omega)$: *frequency domain* [频域]

$$s(t) = \sin(\omega t) = \sin(60\pi t) = \sin(2\pi \cdot f \cdot t)$$

$$f = \frac{1}{T} = \frac{\omega}{2\pi} = \frac{60\pi}{2\pi} = 30$$

T : 周期, 转一圈所需的时间

f : 频率, 每秒转多少圈

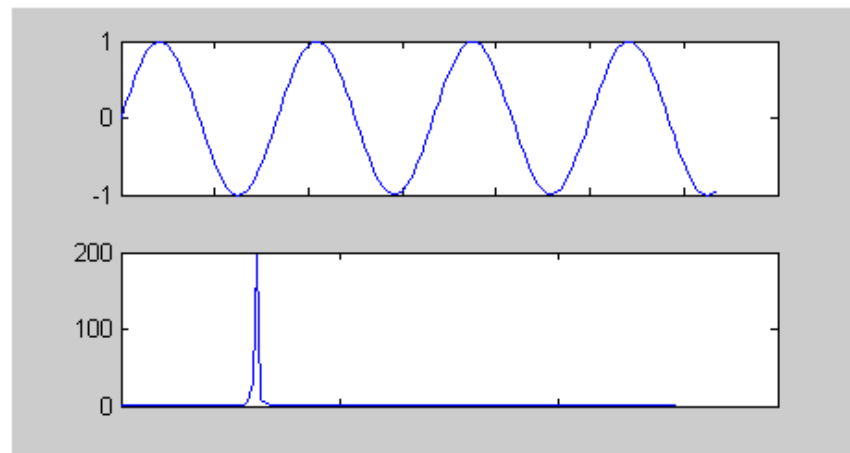
ω : 角频率, 每秒转多少 (弧) 度

$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

$$S(f) = \int e^{-i2\pi f \cdot t} s(t) dt$$

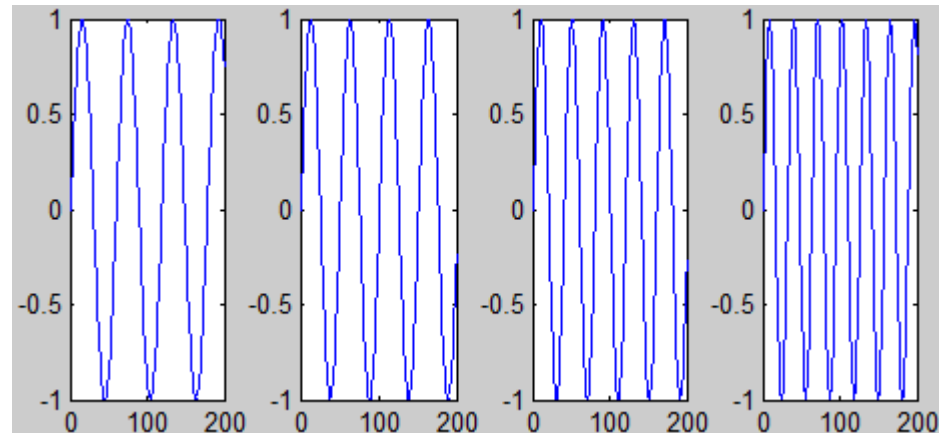
$$s(t) = \sin(60\pi t) + \sin(30\pi t)?$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)?$$

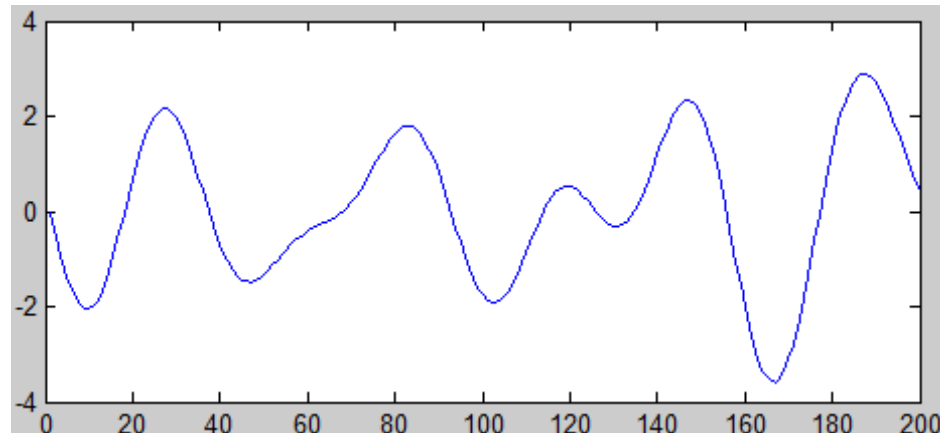


Fourier Transform (FT) and FFT

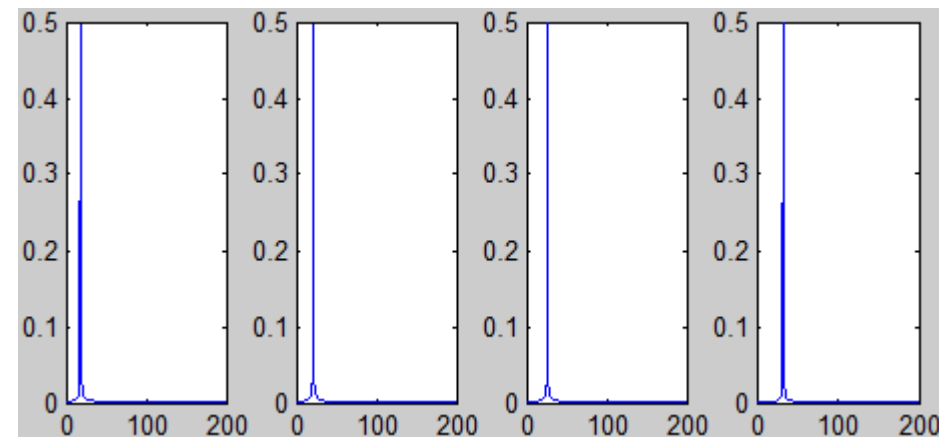
$$s_i(t) = \sin(w_i t)$$



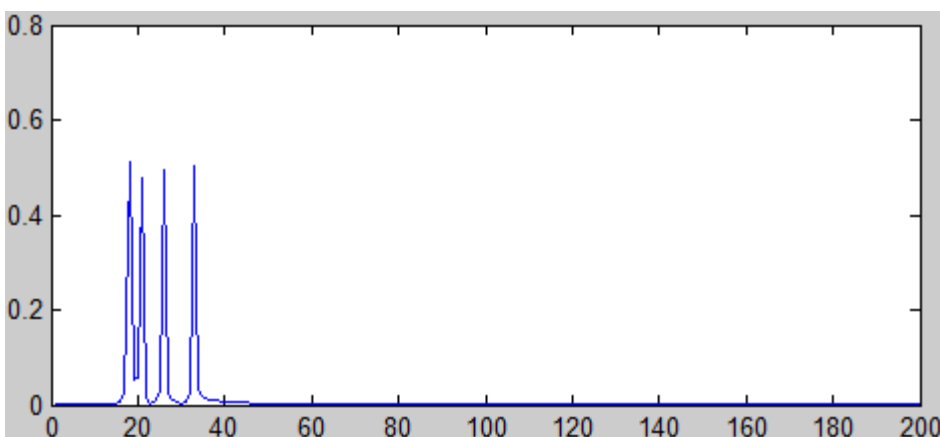
$$s(t) = \sum a_i s_i(t)$$



$$S_i(\omega) = F(s_i(t))$$



$$S(\omega) = F(s(t))$$



Fourier Transform (FT) and FFT

- FT: the most common use is in signal processing

- ◆ $s(t)$: time domain [信号的时域表示]

- ◆ $S(\omega)$: frequency domain [频域]

$$s(t) = \sin(\omega t) = \sin(60\pi t) = \sin(2\pi \cdot f \cdot t)$$

$$f = \frac{1}{T} = \frac{\omega}{2\pi} = \frac{60\pi}{2\pi} = 30$$

T : 周期, 转一圈所需的时间

f : 频率, 每秒转多少圈

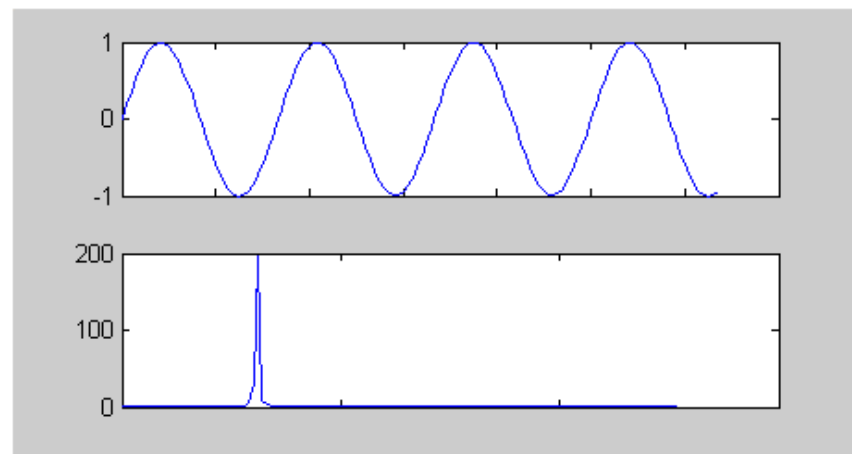
ω : 角频率, 每秒转多少 (弧) 度

$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

$$S(f) = \int e^{-i2\pi f \cdot t} s(t) dt$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t)?$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)?$$



计算科学中最重要的32个算法

Fourier Transform

$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)?$$



让·巴普蒂斯特·约瑟夫·傅里叶（Baron Jean Baptiste Joseph Fourier, 1768-1830），男爵，法国数学家、物理学家，1768年3月21日生于欧塞尔，1830年5月16日卒于巴黎。

1817年当选为科学院院士。主要贡献是在研究《热的传播》和《热的分析理论》时创立了一套数学理论，对19世纪的数学和物理学的发展都产生了深远影响。

- FT的基本思想首先由傅里叶提出，所以以其名字来命名以示纪念。傅里叶变换是一种特殊的积分变换。它能够将满足一定条件的某个函数表示成正弦基函数的线性组合或者积分。在不同的研究领域，傅里叶变换具有多种不同的变体形式，如连续傅里叶变换和离散傅里叶变换。
- 数学上看，用简单表示来对复杂函数的深入研究。正弦函数在物理上是被充分研究而相对简单的函数类，这一想法跟化学上的原子论想法何其相似！奇妙的是，现代数学发现傅里叶变换具有非常好的性质，使得它如此的好用和有用，让人不得不感叹造物的神奇。
- 哲学上看，"分析主义"和"还原主义"，就是要通过对事物内部适当的分析达到增进对其本质理解的目的。比如近代原子论试图把世界上所有物质的本源分析为原子，而原子不过数百种而已，相对物质世界的无限丰富，这种分析和分类无疑为认识事物的各种性质提供了很好的手段。

30 Polynomials, Convolution and the FFT

- FT: the most common use is in signal processing

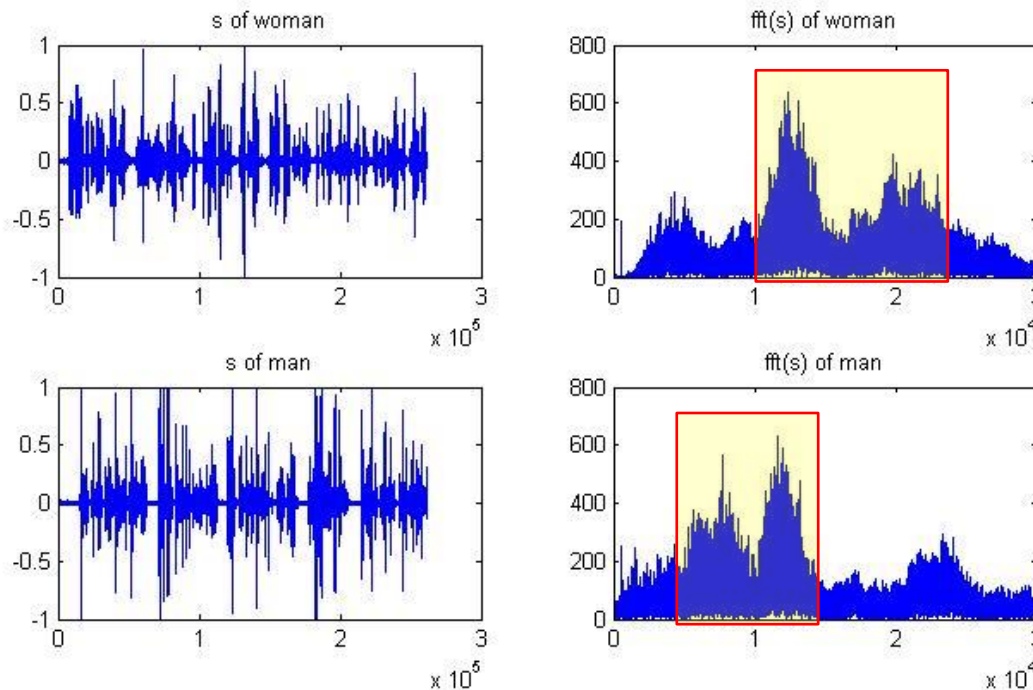
- ◆ $s(t)$: *time domain* [信号的时域表示]

- ◆ $S(w)$: *frequency domain* [频域]

$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

- FFT: Fast Fourier Transform

$$S(f) = \int e^{-i2\pi f t} s(t) dt$$



Polynomials

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad x \in F, \quad (R, C)$$

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

- polynomial *coefficients* : $a_0, a_1, \dots, a_{n-1}, \in F$.
- polynomial *degree* (维数) : $A(x)$ is said to have degree k if its highest nonzero coefficient is a_k .
- polynomial *degree-bound* (维界) : any integer strictly greater than the degree.
- Therefore, the degree of a polynomial of degree-bound n may be any integer between 0 and $n-1$, inclusive.

Polynomials addition [多项式相加]

- $A(x), B(x)$, degree-bound n
- $C(x) = A(x) + B(x)$, also degree-bound n

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$\text{then } C(x) = \sum_{j=0}^{n-1} c_j x^j$$

where $c_j = a_j + b_j$ for $j = 0, 1, \dots, n - 1$.

For example, if the polynomials $A(x) = 6x^3 + 7x^2 - 10x + 9$

$$\text{and} \quad B(x) = -2x^3 + 4x - 5,$$

$$\text{then} \quad C(x) = 4x^3 + 7x^2 - 6x + 4.$$

- Running time: $T(n) = \Theta(n)$

Polynomials multiplication [多项式相乘]

- $A(x), B(x)$, degree-bound n
- $C(x) = A(x) * B(x)$, degree-bound $2n-1$

$$= \sum_{j=0}^{2n-2} c_j x^j$$

For example, multiply $A(x) = 6x^3 + 7x^2 - 10x + 9$
and $B(x) = -2x^3 + 4x - 5$

$$\begin{array}{r} 6x^3 + 7x^2 - 10x + 9 \\ - 2x^3 \qquad \qquad + 4x - 5 \\ \hline - 30x^3 - 35x^2 + 50x - 45 \\ 24x^4 + 28x^3 - 40x^2 + 36x \\ - 12x^6 - 14x^5 + 20x^4 - 18x^3 \\ \hline - 12x^6 - 14x^5 + 44x^4 - 20x^3 - 75x^2 + 86x - 45 \end{array}$$

- Running time: $T(n) = \Theta(n^2)$

Polynomials multiplication [多项式相乘]

- $A(x), B(x)$, degree-bound n $A(x) = \sum_{j=0}^{n-1} a_j x^j$ and $B(x) = \sum_{j=0}^{n-1} b_j x^j$
- $C(x) = A(x) * B(x)$, degree-bound $2n-1$

Another way to express the product $C(x)$ is

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \quad (30.1) \quad , \text{ where } \quad c_j = \sum_{k=0}^j a_k b_{j-k} \quad (30.2)$$

卷积

$$a \otimes b = (a_0, a_1, \dots, a_{n-1}) \otimes (b_0, b_1, \dots, b_{n-1}) = (c_0, c_1, \dots, c_{2n-2})$$

where, $c_j = a_0 b_j + a_1 b_{j-1} + \dots + a_k b_{j-k} + \dots + a_j b_0$

Note that $\text{degree}(C) = \text{degree}(A) + \text{degree}(B)$, implying
 $\text{degree-bound}(C) = \text{degree-bound}(A) + \text{degree-bound}(B) - 1$
 $\leq \text{degree-bound}(A) + \text{degree-bound}(B)$.

We also say: $\text{degree-bound}(C) = \text{degree-bound}(A) + \text{degree-bound}(B)$

Polynomials multiplication [多项式相乘]

- $A(x), B(x)$, degree-bound n $A(x) = \sum_{j=0}^{n-1} a_j x^j$ and $B(x) = \sum_{j=0}^{n-1} b_j x^j$
- $C(x) = A(x)B(x)$, degree-bound $2n-1$

Another way to express the product $C(x)$ is

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \quad (30.1) \quad , \text{ where } \quad c_j = \sum_{k=0}^j a_k b_{j-k} \quad (30.2)$$

卷积

$$a \otimes b = (a_0, a_1, \dots, a_{n-1}) \otimes (b_0, b_1, \dots, b_{n-1}) = (c_0, c_1, \dots, c_{2n-2})$$

where, $c_j = a_0 b_j + a_1 b_{j-1} + \dots + a_k b_{j-k} + \dots + a_j b_0$

- Running time: $T(n) = \Theta(n^2)$, since each coefficient in A must be multiplied by each coefficient in B.
- Can we accelerate the computation? **Yes!**

Chapter outline

- 30.1, presents two ways to represent polynomials:
 - ◆ coefficient representation
 - ◆ point-value representation
- Adding polynomials
- The straightforward methods for multiplying polynomials
 - ◆ $\Theta(n^2)$, polynomials are represented in coefficient form
 - ◆ $\Theta(n)$, when they are represented in point-value form
- Multiply polynomials using the coefficient representation in only $\Theta(n \lg n)$ time? FFT and its inverse
- 30.2, DFT and FFT
- 30.3, how to implement the FFT quickly

30.1 Representation of polynomials

- **equivalence** : The coefficient and point-value representations of polynomials are in a sense **equivalent**.
- That is, a polynomial in point-value form has a unique counterpart in coefficient form.

30.1.1 Coefficient representation

- A *coefficient representation* of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$
 $a = (a_0, a_1, \dots, a_{n-1})^T$, column vectors
- Convenient for certain operations. For example,
 - ◆ *evaluating* $A(x)$ at a given point x_0 , takes time $\Theta(n)$ using *Horner's rule*:
$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \dots + x_0(a_{n-2} + x_0(a_{n-1}))))$$
 - ◆ *adding* two polynomials represented $a = (a_0, a_1, \dots, a_{n-1})^T$ and $b = (b_0, b_1, \dots, b_{n-1})^T$, takes $\Theta(n)$ time: we just produce the coefficient vector $c = (c_0, c_1, \dots, c_{n-1})$, where $c_j = a_j + b_j$ for $j = 0, 1, \dots, n - 1$.

30.1.1 Coefficient representation

- multiplication of $A(x)$ and $B(x)$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \quad a = (a_1, a_2, \dots, a_{n-1})^T; \quad B(x) = \sum_{j=0}^{n-1} b_j x^j, \quad b = (b_1, b_2, \dots, b_{n-1})^T$$

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \quad (30.1)$$

$$\text{where } c_j = \sum_{k=0}^j a_k b_{j-k} \quad (30.2)$$

- takes time $\Theta(n^2)$, why?
- more difficult than adding two polynomials.
- vector c , given by equation (30.2), is also called the *convolution*(卷积) of a and b , denoted by $c = a \odot b$ ($a \otimes b$).
- Multiplying polynomials and computing convolutions are fundamental computational problems, and important.

30.1.2 Point-value representation

- A *point-value representation* of a polynomial

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

$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs

such that all of the x_k are distinct and

$$y_k = A(x_k), k = 0, 1, \dots, n - 1 \quad (30.3)$$

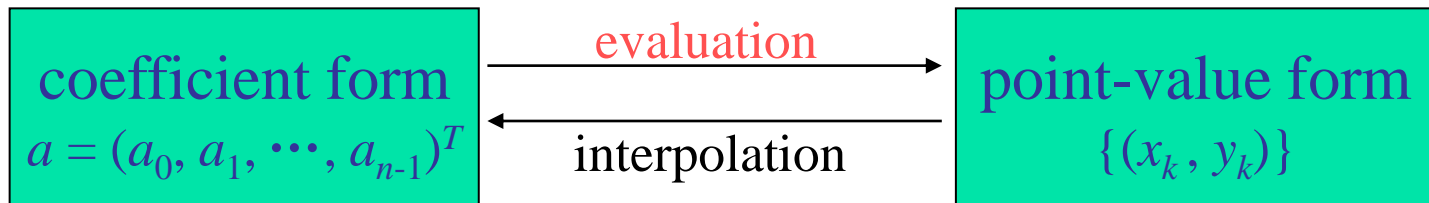
- any set of n distinct points x_0, x_1, \dots, x_{n-1} can be used as a basis for a polynomial representation, which is not unique. [多项式的点值表示不唯一]

30.1.2 Point-value representation

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

$a = (a_0, a_1, \dots, a_{n-1})^T$, column vectors

$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs



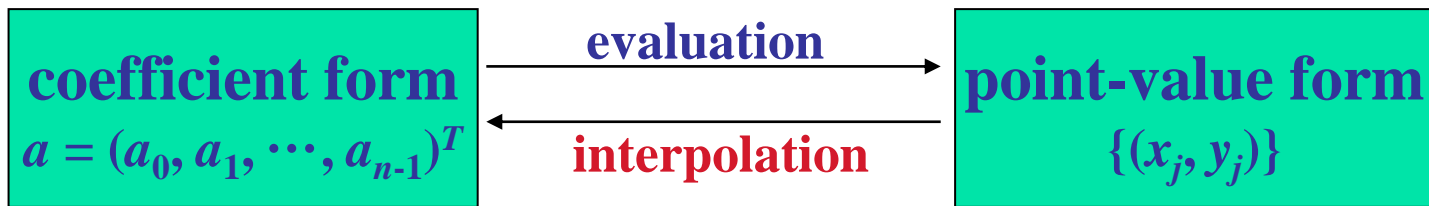
- Evaluation**
 - select n distinct points x_0, x_1, \dots, x_{n-1} ,
 - evaluate $A(x_k)$ for $k = 0, 1, \dots, n - 1$.
 - the n -point evaluation takes time $\Theta(n^2)$. (Horner method)
 - if choose the x_k cleverly, the computation only needs $\Theta(n \lg n)$.

30.1.2 Point-value representation

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

$a = (a_0, a_1, \dots, a_{n-1})^T$, column vectors

$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs



- Interpolation** [插值], The inverse of evaluation
- Theorem 30.1 (**Uniqueness** of an interpolating polynomial)
- For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \dots, n - 1$.

30.1.2 Point-value representation

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



- **Theorem 30.1** For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a **unique** polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \dots, n-1$.

Proof Equation (30.3) “ $y_k = A(x_k), k = 0, 1, \dots, n-1$ ” is equivalent to

$$\begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ & & \dots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \dots, x_{n-1}) \cdot a \quad (30.4)$$

V is Vander-monde matrix, has determinant $\prod_{0 \leq j < k \leq n-1} (x_k - x_j)$, therefore, V^{-1} exists. Thus, $a = V(x_0, x_1, \dots, x_{n-1})^{-1} \cdot y$. (chap28, LU decomposition, $O(n^3)$)

30.1.2 Point-value representation

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



- \square **Theorem 30.1** For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a **unique** polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \dots, n - 1$.
- Lagrange's formula:** A faster algorithm for n -point interpolation

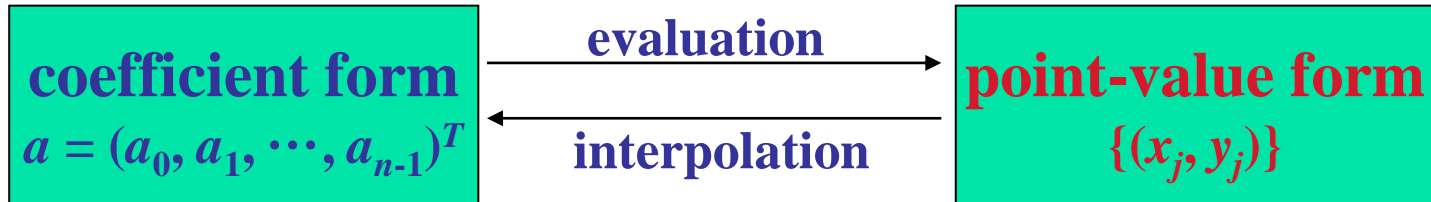
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

$$y = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1}.$$

Computing a of A using Lagrange's formula takes time $\Theta(n^2)$.

30.1.2 Point-value representation

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



- quite convenient for many operations on polynomials,
 - addition**, if $C(x)=A(x)+B(x)$, then $C(x_k)=A(x_k)+B(x_k)$ for any point x_k .
 - More precisely, if we have a point-value representation for $A : \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$, and for $B : \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$, then
$$C : \{(x_0, y_0+z_0), (x_1, y_1+z_1), \dots, (x_{n-1}, y_{n-1}+z_{n-1})\}$$
 - takes time $\Theta(n)$.

30.1.2 Point-value representation

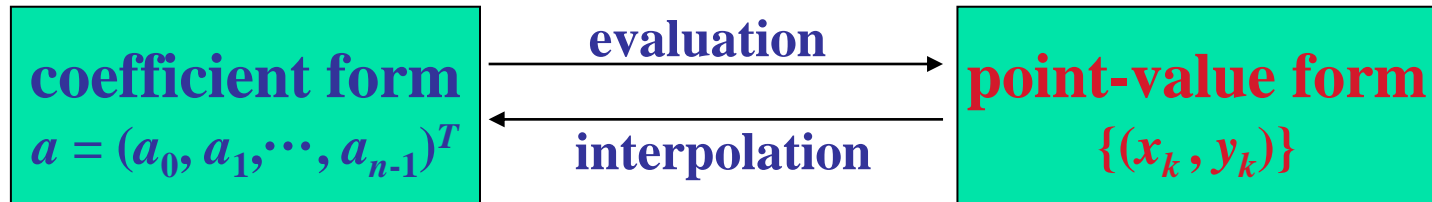
- polynomial representation $A(x) = \sum_{j=0}^{n-1} a_j x^j$



- quite convenient for many operations on polynomials,
 - ♦ **multiplying**, if $C(x)=A(x)B(x)$, then $C(x_k)=A(x_k)B(x_k)$ for any point x_k , **however**
 - ♦ $\text{degree-bound}(C)=\text{degree-bound}(A)+\text{degree-bound}(B)$
 - ♦ multiplying A and B gives n point-value pairs for C ,
 - ♦ but, **$\text{degree-bound}(C) = 2n$**
 - ♦ need $2n$ point-value pairs for C
 - ♦ must "extended" point-value for A and B consisting of $2n$ point-value pairs each.

30.1.2 Point-value representation

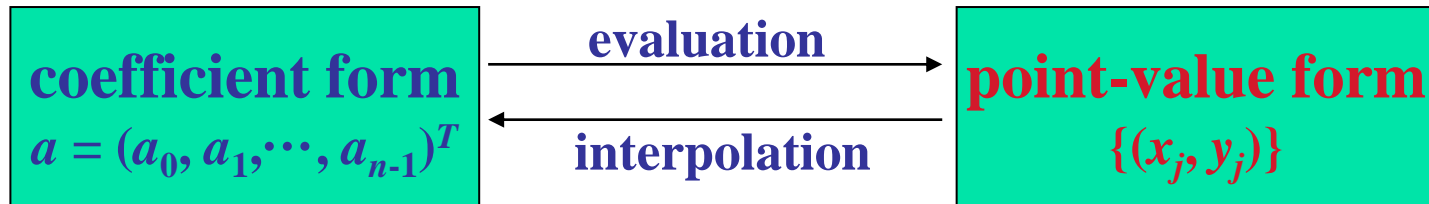
- polynomial representation $y = A(x) = \sum_{j=0}^{n-1} a_j x^j$



- quite convenient for many operations on polynomials,
 - ♦ *multiplying*, $C(x) = A(x)B(x) \Rightarrow C(x_k) = A(x_k)B(x_k)$ for any x_k
 - ♦ **"extended"** point-value for A and B
 - ♦ Given an extended point-value representation for A and B ,
 $A : \{(x_0, y_0), (x_1, y_1), \dots, (x_{2n-1}, y_{2n-1})\}$, and
 $B : \{(x_0, z_0), (x_1, z_1), \dots, (x_{2n-1}, z_{2n-1})\}$, then
 $C : \{(x_0, y_0 z_0), (x_1, y_1 z_1), \dots, (x_{2n-1}, y_{2n-1} z_{2n-1})\}$
 - ♦ takes time $\Theta(n)$, much less than the time required to multiply polynomials in coefficient form.

30.1.2 Point-value representation

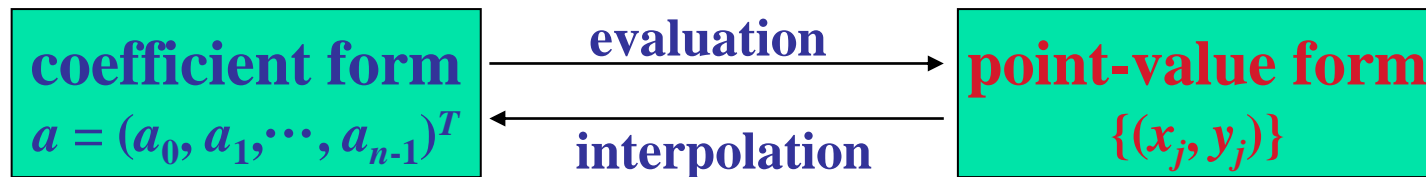
- polynomial representation $y = A(x) = \sum_{j=0}^{n-1} a_j x^j$



- How to evaluate a polynomial given in point-value form at a new point?
〔给定多项式的点值表示形式，如何求多项式在新点处的值？〕
- There is apparently no approach that is simpler than converting the polynomial to coefficient form first, and then evaluating it at the new point.
〔最简单的方法，先将多项式转换为系数表示形式，再求多项式在新点处的值？〕

30.1.3 Fast multiplication of polynomials

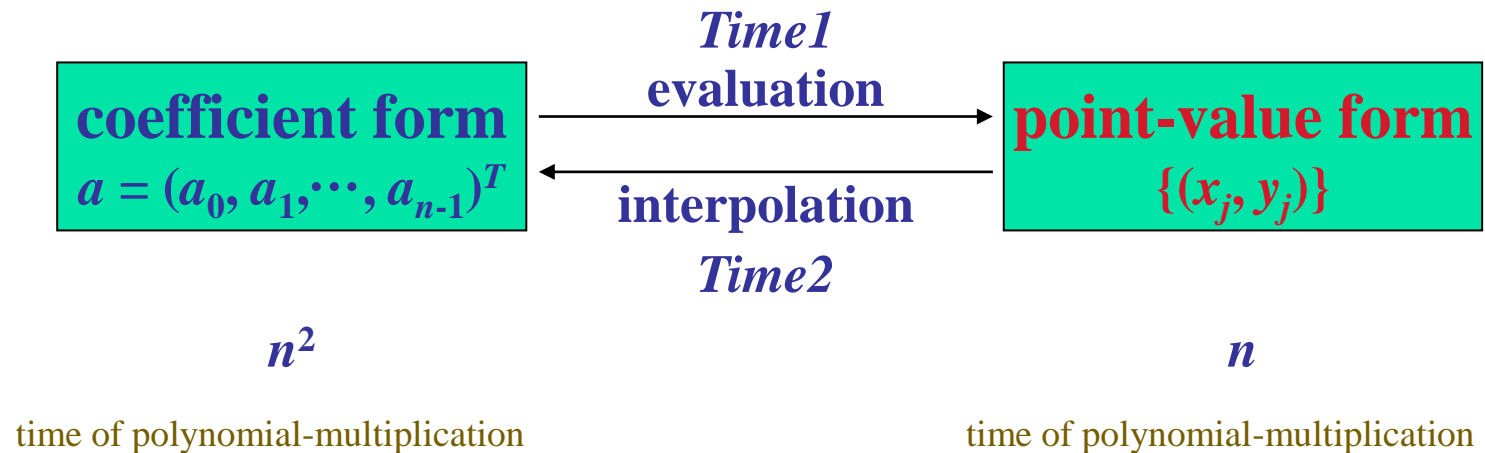
- polynomial representation $A(x) = \sum_{j=0}^{n-1} a_j x^j$



- Multiplying polynomial $C(x)=A(x)B(x)$*
 - point-value form, $\Theta(n)$
 - coefficient form, $\Theta(n^2)$
- Can we use the linear-time multiplication method for polynomials in point-value form to expedite(加速) polynomial multiplication in coefficient form? [对于多项式乘法, 能否依据点值形式的线性乘法来改进系数形式的乘法?]
- The answer hinges on our ability to convert a polynomial quickly from coefficient form to point-value and vice-versa. [该问题依赖于我们将多项式从系数表示快速转换为点值表示的能力, 或相反。]

30.1.3 Fast multiplication of polynomials

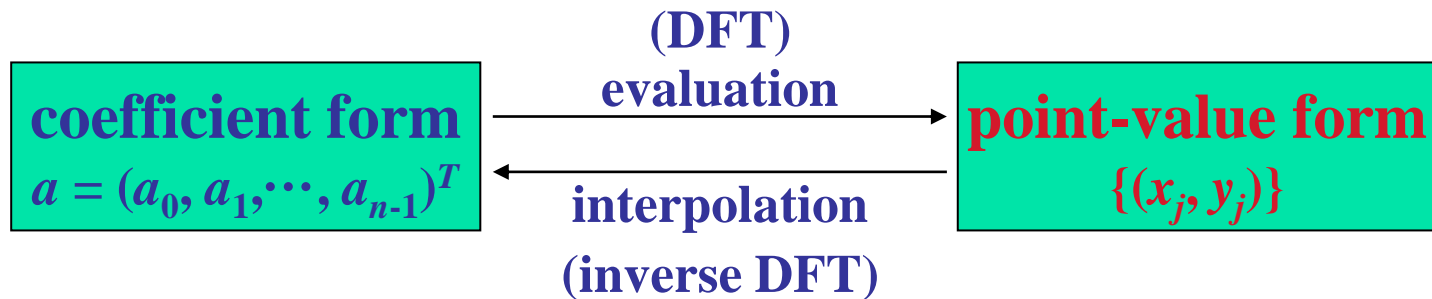
- polynomial multiplication



If $Time1 + Time2 + n < n^2$

30.1.3 Fast multiplication of polynomials

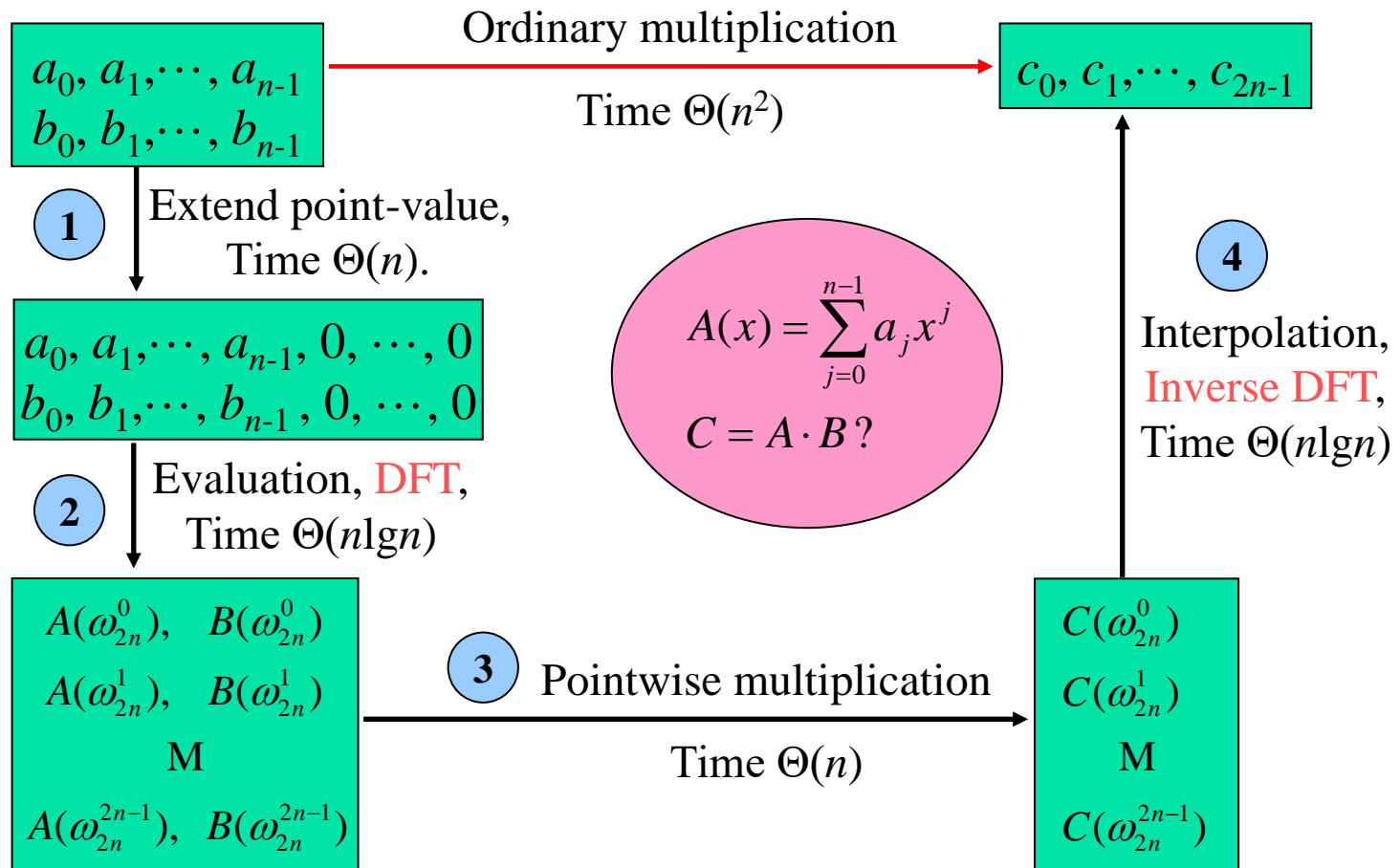
- polynomial representation $A(x) = \sum_{j=0}^{n-1} a_j x^j$



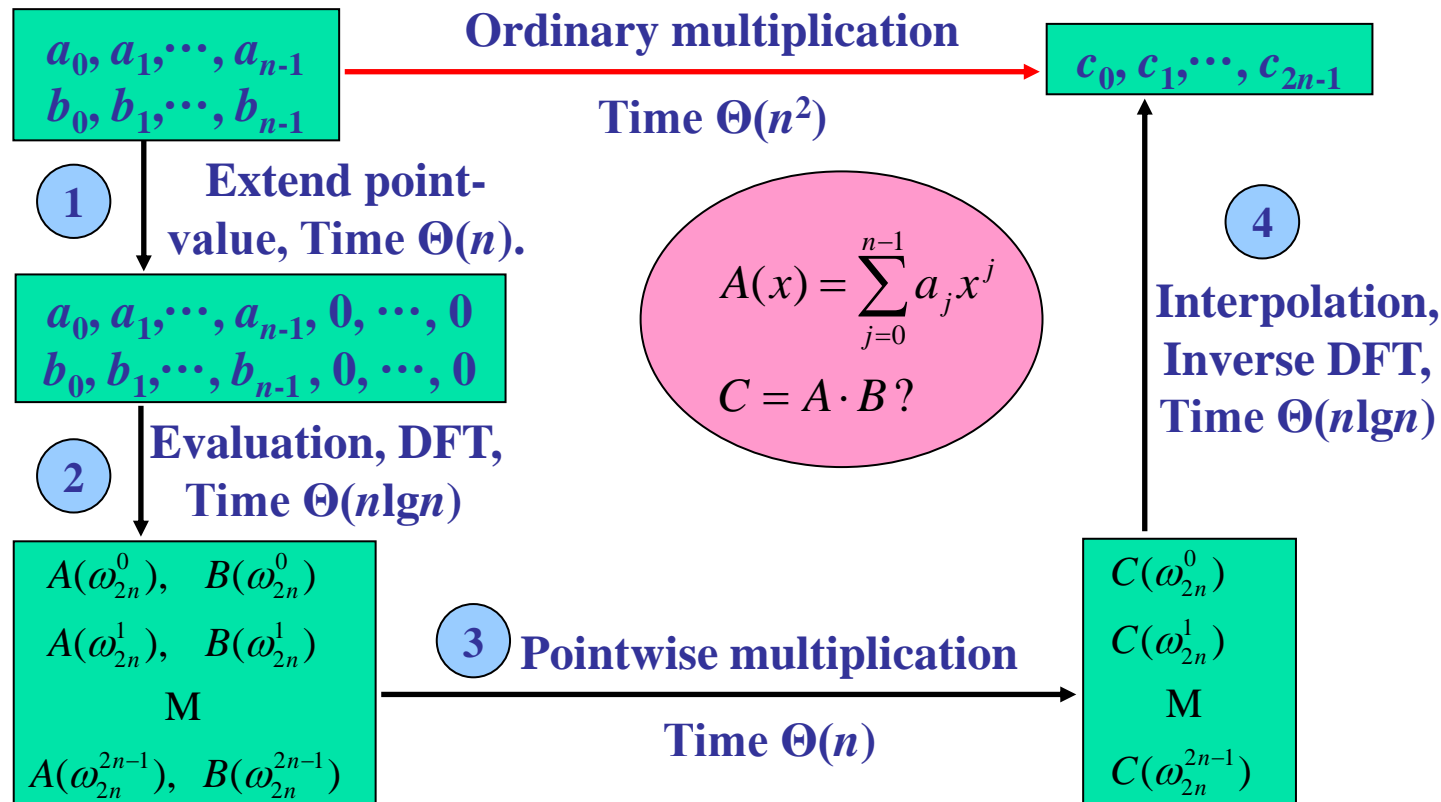
- We can use any points we want as evaluation points,
- but by choosing “complex roots of unity” as the evaluation points, we can convert between representations in only $\Theta(n \lg n)$ time. [选单位复数根作为求值点]
- 30.2, FFT performs the DFT and inverse DFT operations in $\Theta(n \lg n)$ time.

30.1.3 Fast multiplication of polynomials

A graphical outline of an efficient polynomial-multiplication process



30.1.3 Fast multiplication of polynomials



- **Theorem 30.2** The product of two polynomials of degree-bound n can be computed in time $\Theta(n \lg n)$, with both the input and output representations in coefficient form.

30.1-2

Another way to evaluate a polynomial $A(x)$ of degree-bound n at a given point x_0

Solution 30.1-2

30.1-2

Another way
to evaluate a
polynomial
 $A(x)$ of
degree-bound
 n at a given
point x_0

$$a_0 + a_1x + \cdots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j =$$

$$A(x) = q(x)(x - x_0) + r$$

$$= (q_0 + q_1x + \cdots + q_{n-2}x^{n-2})(x - x_0) + r$$

$$= (r - q_0x_0) + (q_0 - q_1x_0)x + \cdots + (q_{n-3} - q_{n-2}x_0)x^{n-2} + q_{n-2}x^{n-1}$$

$$\left\{ \begin{array}{l} r - q_0x_0 = a_0 \\ q_0 - q_1x_0 = a_1 \\ \dots \\ q_{n-3} - q_{n-2}x_0 = a_{n-2} \\ q_{n-2} = a_{n-1} \end{array} \right. \Rightarrow \left\{ \begin{array}{l} r = a_0 + q_0x_0 \\ q_0 = a_1 + q_1x_0 \\ \dots \\ q_{n-3} = a_{n-2} + q_{n-2}x_0 \\ q_{n-2} = a_{n-1} \end{array} \right. \begin{array}{l} \uparrow \\ \Theta(1) \\ \Theta(1) \\ \Theta(1) \\ \Theta(1) \end{array}$$

$\Theta(n)$

Horner's rule: $A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))))$

30.2 The DFT and FFT

- complex roots of unity and their properties
〔单位复数根及其性质〕
- DFT (Discrete Fourier Transform, 离散傅里叶变换)
- how the FFT (Fast Fourier Transform, 快速傅里叶变换) computes the DFT and its inverse in just $\Theta(n \lg n)$ time

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

$$C = A \cdot B?$$

$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

30.2.1 Complex roots of unity

- $\omega^n = 1$, complex number ω is a complex n th root of unity
〔 ω 称为1的 n 次复根，或单位 n 次复根〕
- $\omega_n = e^{2\pi i/n}$: principal n th root of unity 〔单位 n 次复根的基元〕
- $\omega_n^k = (\omega_n)^k = e^{2\pi i k/n}$, $k = 0, 1, \dots, n-1$

exactly n complex n th roots of unity

〔单位1有 n 个 n 次复根〕

- $e^{iu} = \cos(u) + i \sin(u)$:

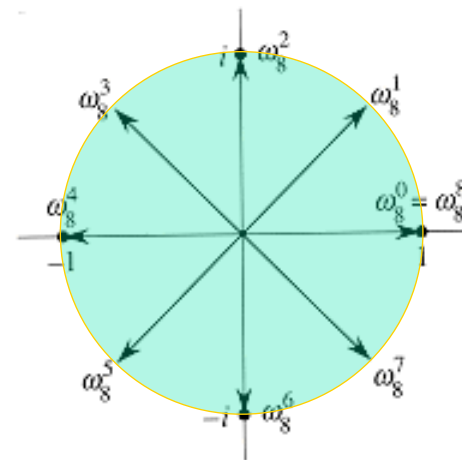
the exponential of a complex number

$$\omega_n^k = e^{2\pi i k/n} = \cos(2\pi k / n) + i \sin(2\pi k / n),$$

$$k = 0, 1, \dots, n-1$$

- All of the other complex n th roots of unity are **powers** of ω_n .
The n complex n th roots of unity $\{\omega_n^k, k = 0, \dots, n-1\}$ form a group.

〔 n 个单位 n 次复根组成一个群（伽罗瓦，法，1811~1832）〕



30.2.1 Complex roots of unity

- some properties

1) $\omega_n^n = \omega_n^0 = 1$

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

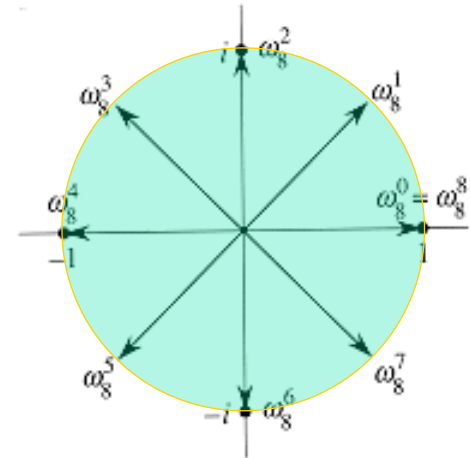
2) $\omega_n^j \omega_n^k = \omega_n^{j+k} = \omega_n^{(j+k) \bmod n}$

(if $j+k = mn+r$, then

$$\omega_n^{j+k} = \omega_n^{mn+r} = \omega_n^{mn} \omega_n^r = (\omega_n^n)^m \omega_n^r = \omega_n^r = \omega_n^{(j+k) \bmod n})$$

3) $\omega_n^{-1} = \omega_n^{n-1}$

(proof by using $\omega_n^k = e^{2\pi i k/n}$, $k = 0, 1, \dots, n-1$)



30.2.1 Complex roots of unity

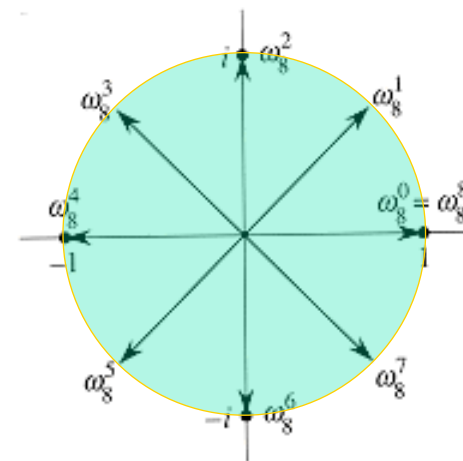
□ Lemma 30.3 (Cancellation lemma, 可约 (相消) 引理)

For any integers $n \geq 0$, $k \geq 0$, and $d > 0$,

$$\omega_{dn}^{dk} = \omega_n^k. \quad (30.7)$$

Proof $\omega_{dn}^{dk} = (e^{2\pi i / dn})^{dk} = (e^{2\pi i / n})^k = \omega_n^k.$

$$(\omega_8^2 = \omega_4^1, \quad \omega_8^4 = \omega_4^2, \quad \omega_8^6 = \omega_4^3, \quad \omega_8^8 = \omega_4^4)$$



□ Corollary (推论) 30.4 For any even integer $n > 0$,

$$\omega_n^{n/2} = \omega_2 = -1.$$

$$\omega_n^{n/2} = \omega_{(n/2) \cdot 2}^{n/2} = \omega_2 = e^{2\pi i / 2} = \cos \pi + i \sin \pi = -1.$$

30.2.1 Complex roots of unity

□ Lemma 30.5 (Halving lemma, 等分引理)

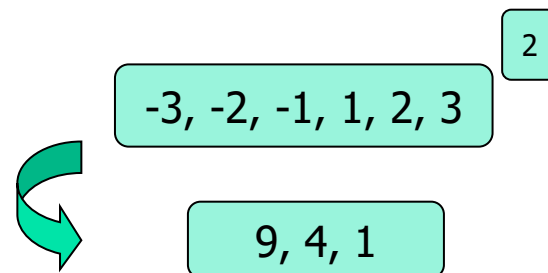
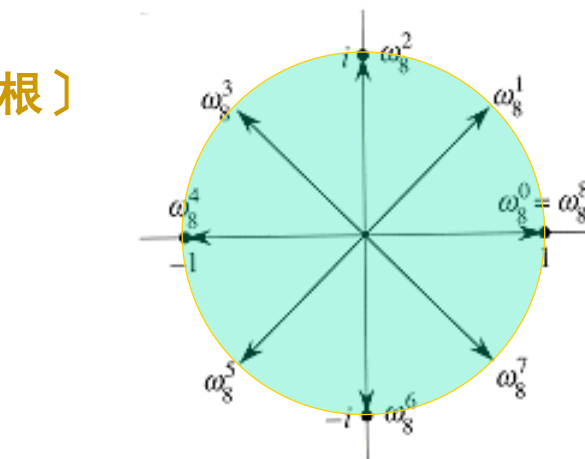
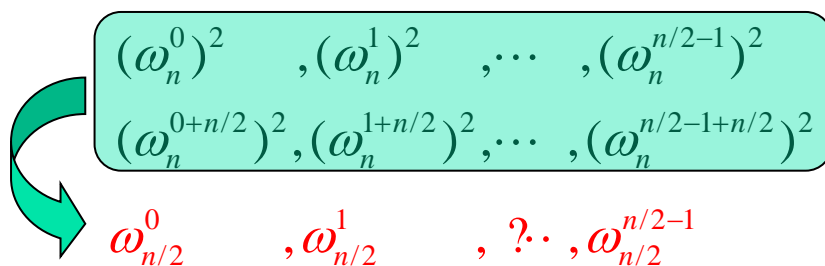
If $n > 0$ is even, then the **squares** of the n complex n th roots of unity are the $n/2$ complex $(n/2)$ th roots of unity.

〔 n 个单位 n 次复方根的平方是 $n/2$ 个 $n/2$ 次复方根〕

Proof idea

$$(\omega_n^k)^2 = \omega_{2(n/2)}^{2k} = \omega_{n/2}^k, \quad k = 0, 1, \dots, \frac{n}{2} - 1$$

$$(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k} \omega_n^n = \omega_n^{2k} = (\omega_n^k)^2 = \omega_{n/2}^k, \quad k = 0, 1, \dots, \frac{n}{2} - 1$$



30.2.1 Complex roots of unity

□ Lemma 30.6 (Summation lemma, 求和引理)

For any integer $n \geq 1$ and nonnegative integer k not divisible by n , ($k \neq m \cdot n$), we have

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

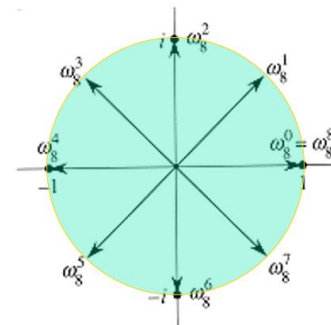
Proof

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0, \quad k \neq mn.$$

(证明思想：利用实数的等幂级数求和性质)

30.2.2 The DFT

$$\begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ & & \dots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \dots, x_{n-1}) \cdot a \quad (30.4)$$



- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ at $x = \omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$
- without loss of generality, assume that $n=2^m$, if not, let $a_{n+k}=0$
- Discrete Fourier Transform (DFT, 离散傅里叶变换)

let A is given in coefficient form: $a=(a_0, a_1, \dots, a_{n-1})^T$,

let $x_k = \omega_n^k$, define y_k , for $k = 0, 1, \dots, n-1$, by

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \dots & \omega_n^{n-1} \\ & & \dots & & \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \dots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix}$$

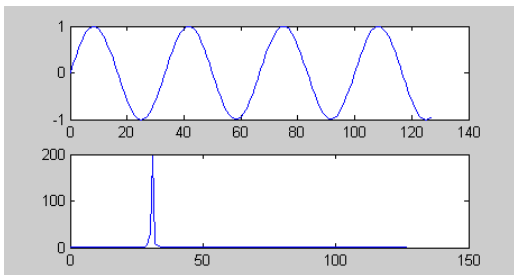
also write $y = \text{DFT}_n(a)$.

Take time $\Theta(n^2)$ to compute straightforward? (不好!)

Application of DFT

- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree-bound n at $x = \omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$.
- Discrete Fourier Transform (DFT, 离散傅里叶变换)
let A is given in coefficient form: $a = (a_0, a_1, \dots, a_{n-1})^T$,
let $x_k = \omega_n^k$, define y_k , for $k = 0, 1, \dots, n-1$, by

$$y_k = A(x_k) = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \cdots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$$



$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

- Signal $s(t)$: discrete, $a_i = s(t_i)$
- Spectrum $S(\omega)$: $S(\omega) = \text{DFT}_n(s)$

30.2.3 The FFT

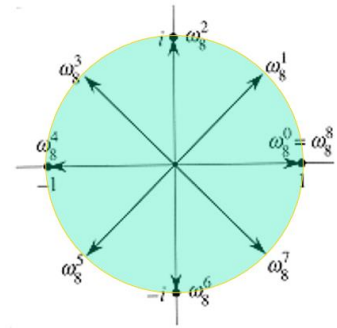
$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}$$

- **Fast Fourier Transform (FFT , 快速傅里叶变换)**
 - ◆ takes advantage of the special properties of the complex roots of unity
 - ◆ we can compute $\text{DFT}_n(a)$ in time $\Theta(n \lg n)$
 - ◆ employs a **divide-and-conquer strategy**,
 - even-index, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$
 - odd-index, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$
 - ◆ $A[0]$ contains all the even-index coefficients of A (the binary representation of the index ends in 0)
 - ◆ $A[1]$ contains all the odd-index coefficients (the binary representation of the index ends in 1). It follows that

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \quad (30.9)$$

30.2.3 The FFT

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



- **Fast Fourier Transform (FFT, 快速傅里叶变换)**

- ◆ employs a divide-and-conquer strategy,

even-index, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$

odd-index, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \quad (30.9)$$

$$A(\omega_n^k) = A^{[0]}((\omega_n^k)^2) + \omega_n^k A^{[1]}((\omega_n^k)^2)$$

- the problem of evaluating $A(x)$ at $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$ reduces to

- 1) evaluating the $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points

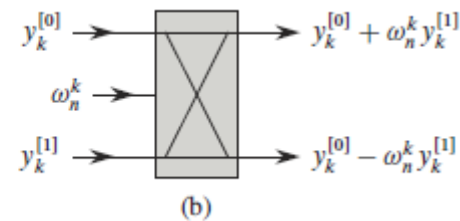
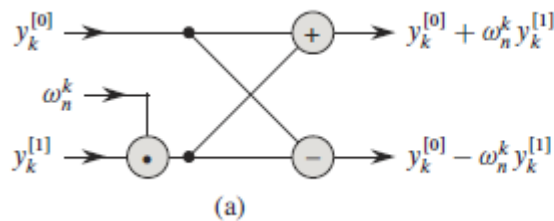
$(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n/2-1})^2$
 $(\omega_n^{0+n/2})^2, (\omega_n^{1+n/2})^2, \dots, (\omega_n^{n/2-1+n/2})^2$

(32.10)

and then $\omega_{n/2}^0, \omega_{n/2}^1, \dots, \omega_{n/2}^{n/2-1}$

- 2) combining the results according to equation (30.9).

30.2.3 The FFT



- **Fast Fourier Transform (FFT , 快速傅里叶变换)**

- ◆ **employs a divide-and-conquer strategy,**

even-index, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$

odd-index, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2) \quad (30.9)$$

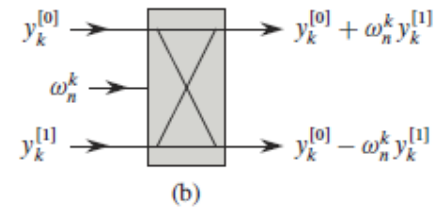
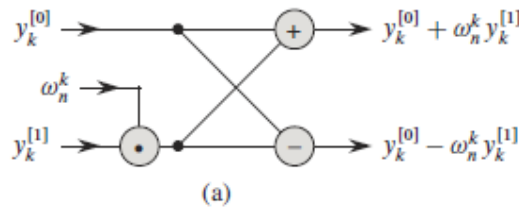
$$\begin{aligned} A(x_r) &= A^{[0]}(x_r^2) + x_r A^{[1]}(x_r^2) \\ &= A(\omega_n^r) = A^{[0]}((\omega_n^r)^2) + (\omega_n^r) \cdot A^{[1]}((\omega_n^r)^2) \\ &= A^{[0]}(\omega_{n/2}^r) + (\omega_n^r) \cdot A^{[1]}(\omega_{n/2}^r), \quad r = 0, \dots, n/2-1, n/2, \dots, n-1 \end{aligned}$$

let $k = 0, \dots, n/2-1$, then

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$\begin{aligned} A(\omega_n^{n/2+k}) &= A^{[0]}(\omega_{n/2}^{n/2+k}) + (\omega_n^{n/2+k}) \cdot A^{[1]}(\omega_{n/2}^{n/2+k}) \\ &= A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k) \end{aligned}$$

30.2.3 The FFT



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

$$\begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \dots & \omega_n^{n-1} \\ & & & \dots & \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & L & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix}$$

even: $A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{n/2-1}$

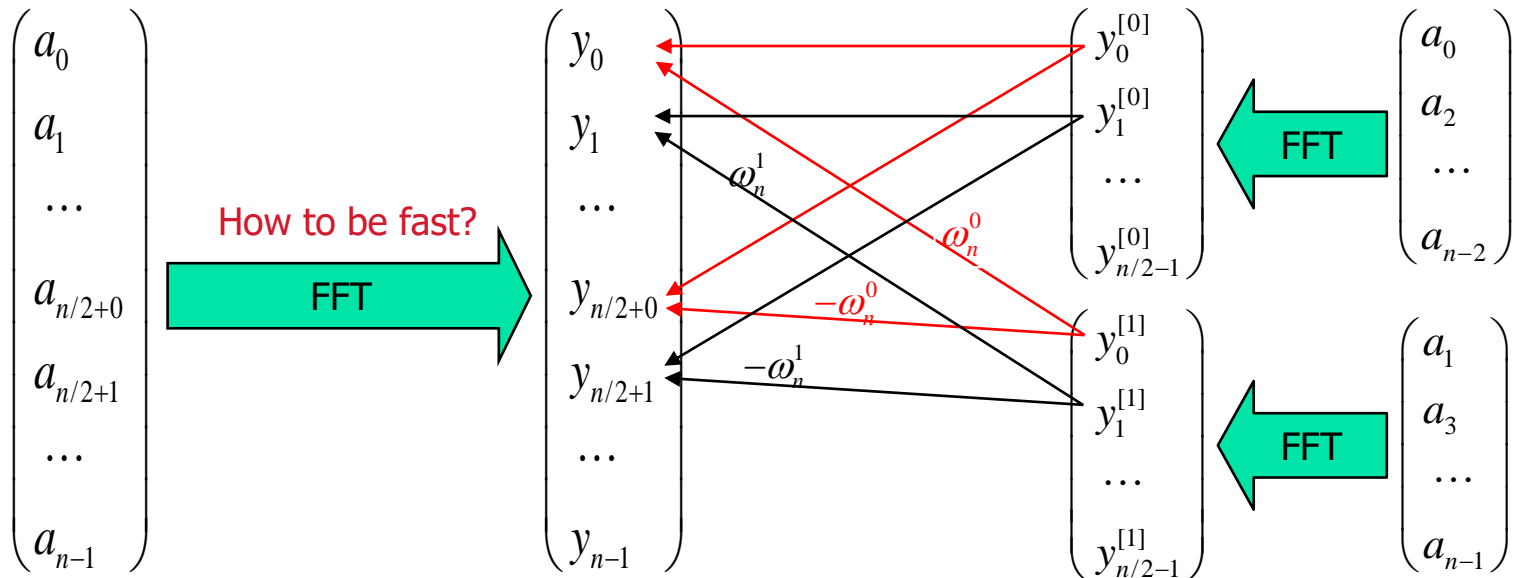
odd: $A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{n/2-1}$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$$

let $k = 0, \dots, n/2 - 1$, then

$$\begin{aligned} y_k &= A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k) \\ &= y_k^{[0]} + (\omega_n^k) \cdot y_k^{[1]} \end{aligned}$$

$$\begin{aligned} y_{k+n/2} &= A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^{n/2+k}) + (\omega_n^{n/2+k}) \cdot A^{[1]}(\omega_{n/2}^{n/2+k}) \\ &= y_k^{[0]} - (\omega_n^k) \cdot y_k^{[1]} \end{aligned}$$



30.2.3 The FFT

even, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

RECURSIVE-FFT(a)

```

1   $n = a.length$            //  $n$  is a power of 2
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$            //  $y$  is assumed to be a column vector
    
```

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


$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$k = 0, \dots, n/2 - 1$$



$$\begin{array}{c}
 (\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n/2-1})^2 \\
 (\omega_n^{0+n/2})^2, (\omega_n^{1+n/2})^2, \dots, (\omega_n^{n/2-1+n/2})^2
 \end{array}
 \rightarrow
 \omega_{n/2}^0, \omega_{n/2}^1, \dots, \omega_{n/2}^{n/2-1}$$

30.2.3 The FFT

even, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

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


$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

RECURSIVE-FFT(a)

```

1   $n = a.length$                 //  $n$  is a power of 2
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$                         //  $y$  is assumed to be a column vector
    
```



$$\begin{array}{c}
 (\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n/2-1})^2 \\
 (\omega_n^{0+n/2})^2, (\omega_n^{1+n/2})^2, \dots, (\omega_n^{n/2-1+n/2})^2
 \end{array}$$

$$\omega_{n/2}^0, \omega_{n/2}^1, \dots, \omega_{n/2}^{n/2-1}$$

Line 2-3

$$y_0 = a_0 \omega_1^0 = a_0 \cdot 1 = a_0$$

30.2.3 The FFT

even, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

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


$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

RECURSIVE-FFT(a)

```

1   $n = a.length$            //  $n$  is a power of 2
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$            //  $y$  is assumed to be a column vector
```



$$(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n/2-1})^2$$

$$(\omega_n^{0+n/2})^2, (\omega_n^{1+n/2})^2, \dots, (\omega_n^{n/2-1+n/2})^2$$

$$\omega_{n/2}^0, \omega_{n/2}^1, \dots, \omega_{n/2}^{n/2-1}$$

Line 8-9

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k}),$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k}).$$

Line 11

$$\begin{aligned}
 y_k &= y_k^{[0]} + \omega_n^k y_k^{[1]} \\
 &= A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) = A(\omega_n^k)
 \end{aligned}$$

30.2.3 The FFT

even, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

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

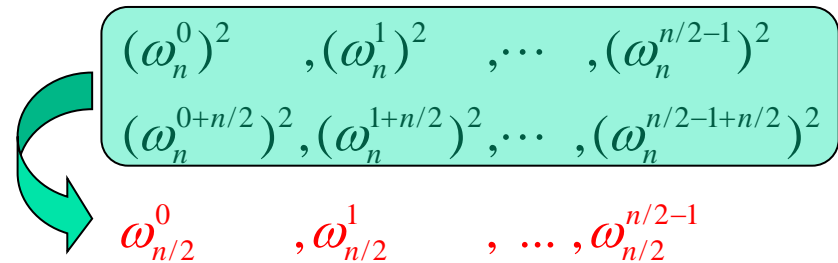
$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

RECURSIVE-FFT(a)

```

1   $n = a.length$ 
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+n/2} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$ 
```



Line 8-9

$$y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k}),$$

$$y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k}).$$

Line 12

$$\begin{aligned}
 y_{k+n/2} &= y_k^{[0]} - \omega_n^k y_k^{[1]} \\
 &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\
 &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k+n}) \\
 &= A(\omega_n^{k+n/2})
 \end{aligned}$$

30.2.3 The FFT

even, $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$

RECURSIVE-FFT(a)

```
1   $n = a.length$ 
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$ 
```

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

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$k = 0, \dots, n/2 - 1$$

in line 11-12, each $y_k^{[1]}$ is multiplied by ω_n^k , the product is both *added to* and *subtracted from* $y_k^{[0]}$. each ω_n^k is used in both its positive and negative forms, ω_n^k is called *twiddle factors*.

旋转因子

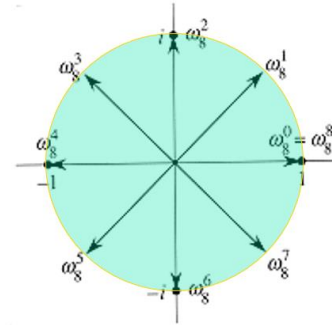
Running Time?

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$$

30.2.4 Interpolation at the complex roots of unity

DFT

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \dots & \omega_n^{n-1} \\ & & & \dots & \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \dots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = V_n a$$



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

$$x_k = \omega_n^k$$

inverse DFT

$$a = \text{DFT}_n^{-1}(y) = V_n^{-1} y$$

□ **Theorem 30.7**

For $j, k = 0, 1, \dots, n-1$, the (j, k) entry of V_n^{-1} is ω_n^{-kj} / n .

30.2.4 Interpolation at the complex roots of unity

□ Lemma 30.6 (Summation lemma, 求和引理)

For any integer $n \geq 1$ and nonnegative integer k not divisible by n , ($k \neq m \cdot n$), we have

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

□ Theorem 30.7

For $j, k = 0, 1, \dots, n-1$, the (j, k) entry of V_n^{-1} is ω_n^{-kj} / n .

Proof We show that $V_n^{-1} V_n = I_n$

$$[V_n^{-1} V_n]_{j j'} = \sum_{k=0}^{n-1} (\omega_n^{-kj} / n) (\omega_n^{kj'}) = \sum_{k=0}^{n-1} \omega_n^{k(j'-j)} / n = \begin{cases} 1 & \text{if } j' = j \\ 0 & \text{if } j' \neq j \end{cases}$$

since $-(n-1) < j' - j < n-1$, apparently $j' - j \neq mn$, if $m \neq 0$

30.2.4 Interpolation at the complex roots of unity

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

DFT

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad (k = 0, 1, \dots, n-1) \quad (30.8)$$

inverse DFT

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}, \quad (j = 0, 1, \dots, n-1) \quad (30.11)$$

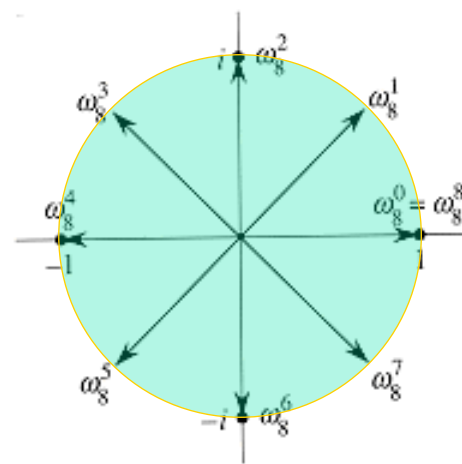
- The **inverse DFT** (逆DFT) can be computed in $\Theta(n \lg n)$ time, for (30.8), by **replacing** ω_n by ω_n^{-1} , and divide each element of the result by n .

DFT

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \cdots & & \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \cdots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = V_n a$$

inverse DFT

$$a = \text{DFT}_n^{-1}(y) = V_n^{-1} y$$



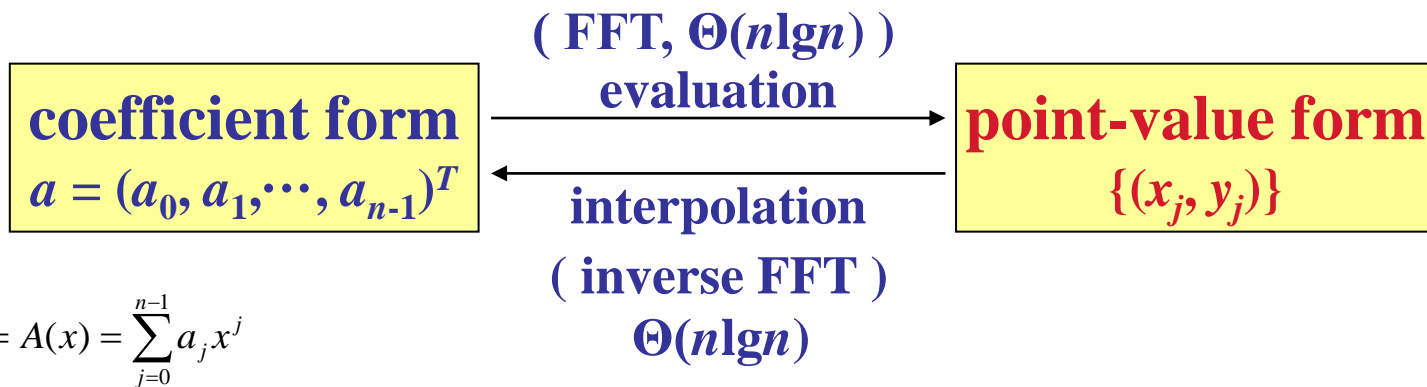
30.2.4 Interpolation at the complex roots of unity

DFT

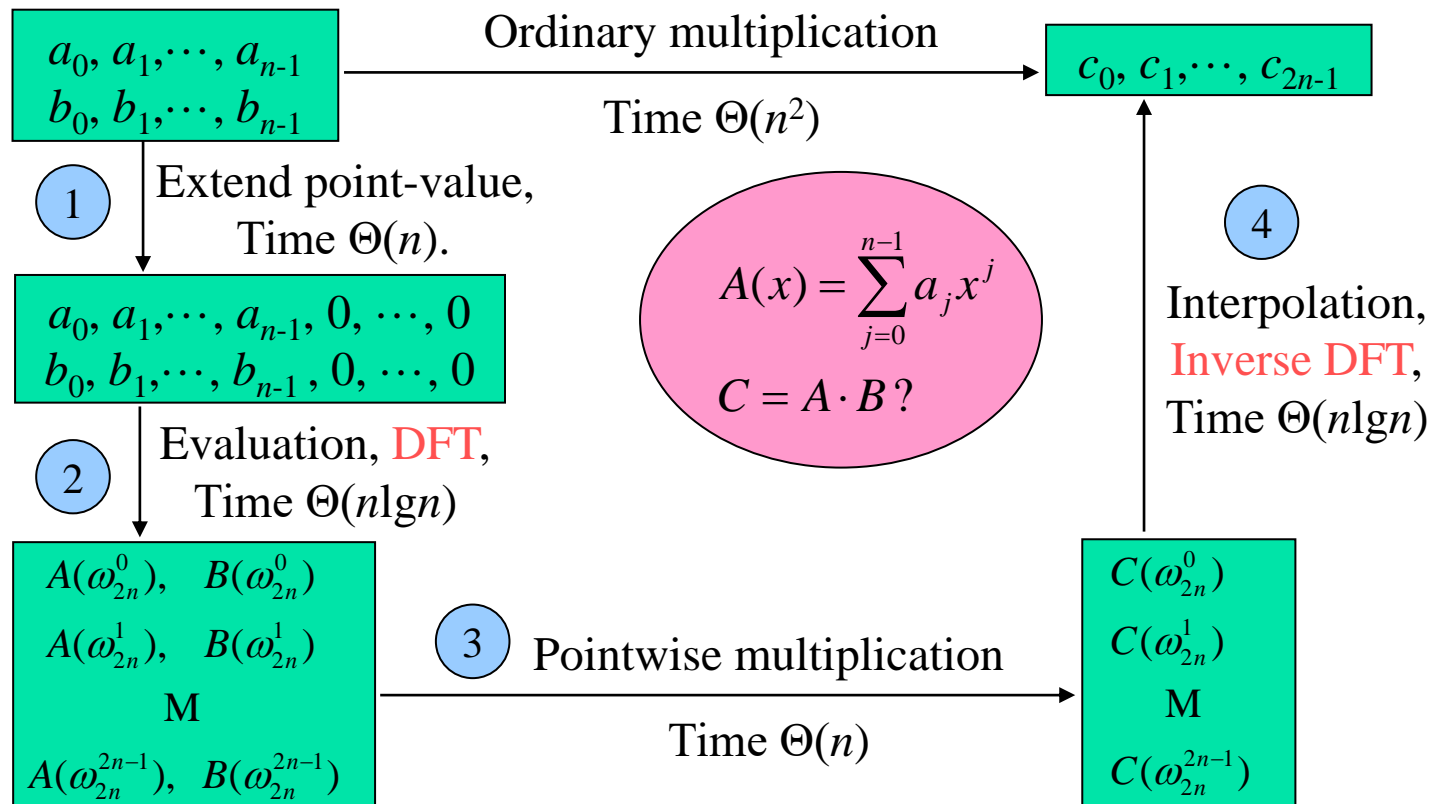
$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad (k = 0, 1, \dots, n-1) \quad (30.8)$$

inverse DFT

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}, \quad (j = 0, 1, \dots, n-1) \quad (30.11)$$



30.2.4 Interpolation at the complex roots of unity



□ Theorem 30.8 (Convolution theorem, 卷积定理)

For any two vectors a and b of length n , where n is a power of 2,

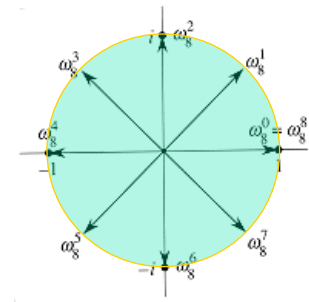
$$a \otimes b = \text{DFT}_{2n}^{-1}(\text{DFT}_{2n}(a) \cdot \text{DFT}_{2n}(b))$$

where the vectors a and b are padded with 0's to length $2n$
and \cdot denotes the component-wise product of two $2n$ -element vectors.

30.3 Efficient FFT implementations

- The practical applications of the DFT, such as signal processing, demand the **utmost speed**.
- Two efficient FFT implementations
 - ◆ iterative FFT algorithm
 - ◆ butterfly operation algorithm (parallel FFT circuit)

30.3.1 An iterative FFT implementation



$$A(x) = \sum_{j=0}^{n-1} a_j x^j = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$x = \omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1};$$

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

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj},$$

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

$$\text{even, } A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1};$$

$$\text{odd, } A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}.$$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

let $k = 0, \dots, n/2 - 1$, then

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

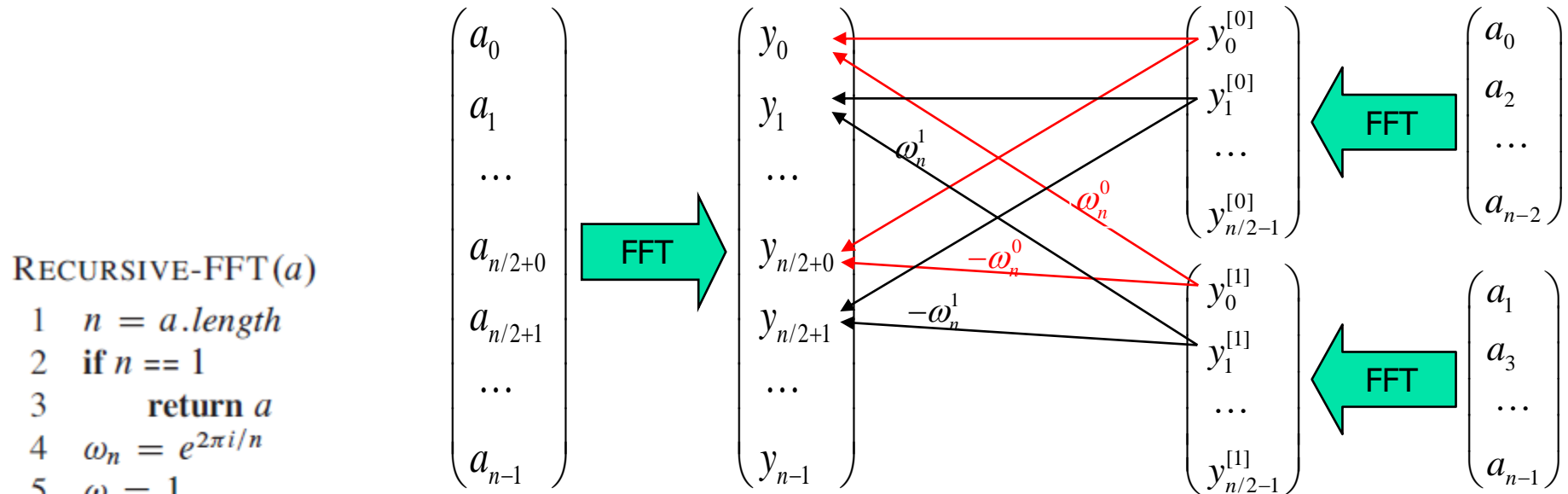
$$A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k)$$

$$y_k = y_k^0 + \omega_n^k y_k^1,$$

$$y_{k+n/2} = y_k^0 - \omega_n^k y_k^1$$

30.3.1 An iterative FFT implementation

lines 10-13, **RECURSIVE-FFT** involves computing the value $\omega_n^k y_k^{[1]}$ **twice**, change the loop to compute it **only once**.



RECURSIVE-FFT(a)

```

1   $n = a.length$ 
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$ 

```

butterfly operation (蝶型操作)
adding and subtracting t from $y_k^{[0]}$

```

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

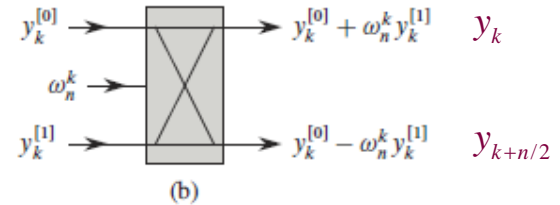
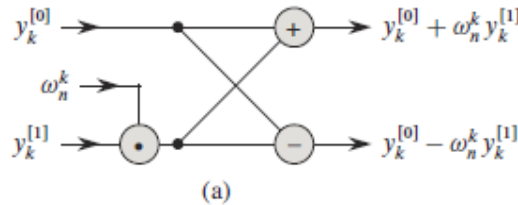
```

30.3.1 An iterative FFT implementation

RECURSIVE-FFT(a)

```

1   $n = a.length$ 
2  if  $n == 1$ 
3      return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11      $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12      $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13      $\omega = \omega \omega_n$ 
14 return  $y$ 
    
```



($k = 0, 1, \dots, n/2 - 1$)

y_k

$y_{k+n/2}$

butterfly operation

adding and subtracting t from $y_k^{[0]}$

for $k = 0$ to $n/2 - 1$

$t = \omega y_k^{[1]}$

$y_k = y_k^{[0]} + t$

$y_{k+(n/2)} = y_k^{[0]} - t$

$\omega = \omega \omega_n$

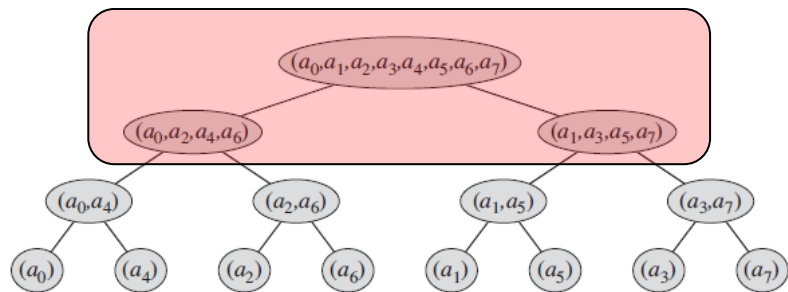
30.3.1 An iterative FFT implementation

- Make the FFT algorithm **iterative** rather than **recursive** in structure.

RECURSIVE-FFT(a)

```
1   $n = a.length$ 
2  if  $n == 1$ 
3    return  $a$ 
4   $\omega_n = e^{2\pi i/n}$ 
5   $\omega = 1$ 
6   $a^{[0]} = (a_0, a_2, \dots, a_{n-2})$ 
7   $a^{[1]} = (a_1, a_3, \dots, a_{n-1})$ 
8   $y^{[0]} = \text{RECURSIVE-FFT}(a^{[0]})$ 
9   $y^{[1]} = \text{RECURSIVE-FFT}(a^{[1]})$ 
10 for  $k = 0$  to  $n/2 - 1$ 
11    $y_k = y_k^{[0]} + \omega y_k^{[1]}$ 
12    $y_{k+(n/2)} = y_k^{[0]} - \omega y_k^{[1]}$ 
13    $\omega = \omega \omega_n$ 
14 return  $y$ 
```

Recursive calls of RECURSIVE-FFT
in a tree structure.



for $k = 0$ to $n/2 - 1$

```
   $t = \omega y_k^{[1]}$ 
   $y_k = y_k^{[0]} + t$ 
   $y_{k+(n/2)} = y_k^{[0]} - t$ 
   $\omega = \omega \omega_n$ 
```



30.3.1 An iterative FFT implementation

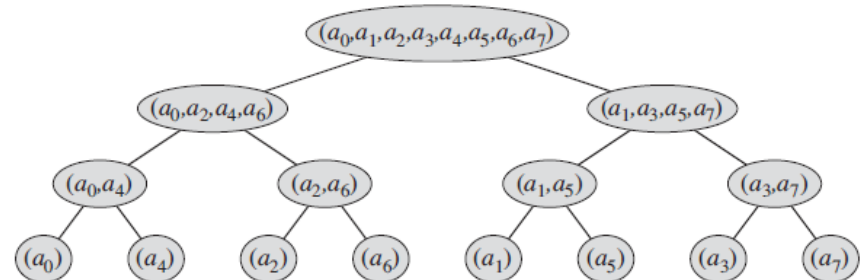
- **iterative** FFT algorithm
- arrange the elements of a into the order in which they appear in the leaves,

1) compute the DFT of each pair using one butterfly operation, replace the pair with its DFT, the vector then holds $(n/2)$ th 2-element DFT's;

2) take the $(n/2)$ th DFT's in pairs, compute the DFT of the four vector elements, 2 butterfly operations, the new vector holds $(n/4)$ th 4-element DFT's;

3) continue in this manner.

Recursive calls of **RECURSIVE-FFT** in a tree structure.



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

- 第一层循环：高度（从下往上）（如果是递归的话，相当于递归的深度）
- 第二层循环：求 m 个数的FFT(即每组有 m 个数)，共有 n/m 组数（即每组有 m 个数，共有 n 个数，跟源数据个数相等）
- 第三层循环：蝶形操作（每组有 m 个数，有 $m/2$ 个蝶形操作，一共 n/m 组数，所有蝶形操作 $(m/2)*(n/m)=n/2$ 个）

initial $A[0.. n-1]$

ITERATIVE-FFT (a)

1 **BIT-REVERSE-COPY** (a, A)

2 $n \leftarrow \text{length}[a]$ // n is 2^k

3 **for** $s \leftarrow 1$ **to** $\lg n$

4 $m \leftarrow 2^s$

5 $\omega_m \leftarrow e^{2\pi i/m}$

6 **for** $k \leftarrow 0$ **to** $n-1$ **by** m // n/m 组

7 $\omega \leftarrow 1$

8 **for** $j \leftarrow 0$ **to** $m/2 - 1$ // 每组 m 个数

9 $t \leftarrow \omega A[k+j+m/2]$

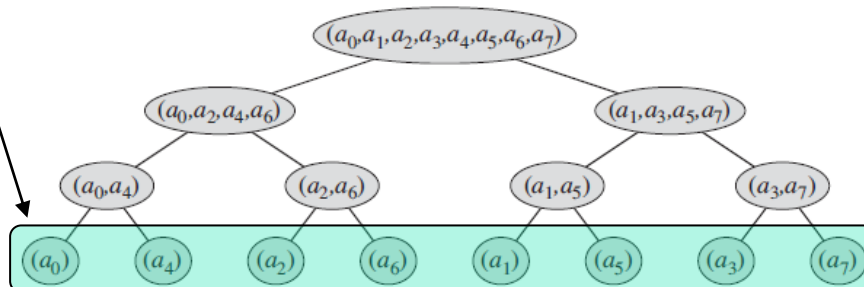
10 $u \leftarrow A[k+j]$

11 $A[k+j] \leftarrow u+t$

12 $A[k+j+m/2] \leftarrow u-t$

13 $\omega \leftarrow \omega \cdot \omega_m$

Recursive calls of RECURSIVE-FFT in a tree structure.



$s=1$

$m=2^s=2$

$k=0, 2, \dots, n-1$

$j=0$

$A[0] \ A[1]$

$A[2] \ A[3]$

...

$s=2$

$m=4$

$k=0, 4, \dots, n-1$

$j=0, 1$

$A[0] \ A[2]$

$A[1] \ A[3]$

...

$s=3$

$m=8$

$k=0, 8, \dots, n-1$

$j=0, 1, 2, 3$

$A[0] \ A[4]$

$A[1] \ A[5]$

...

30.3.1 An iterative FFT implementation

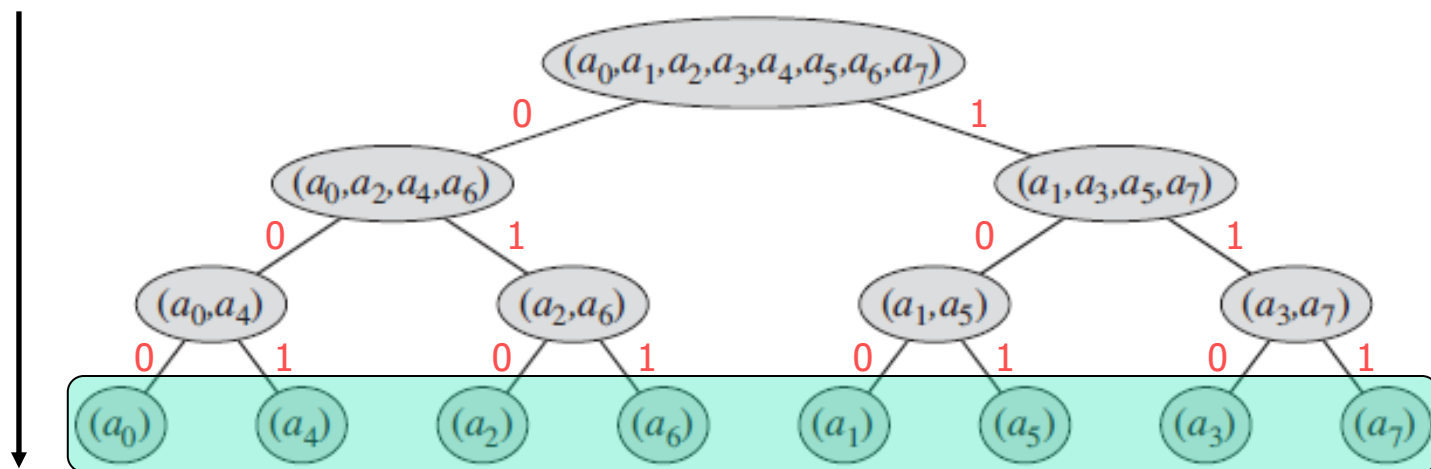
- Bit-reversal permutation (按位逆置换)

ITERATIVE-FFT (a)

1 BIT-REVERSE-COPY (a, A)

...

Original order



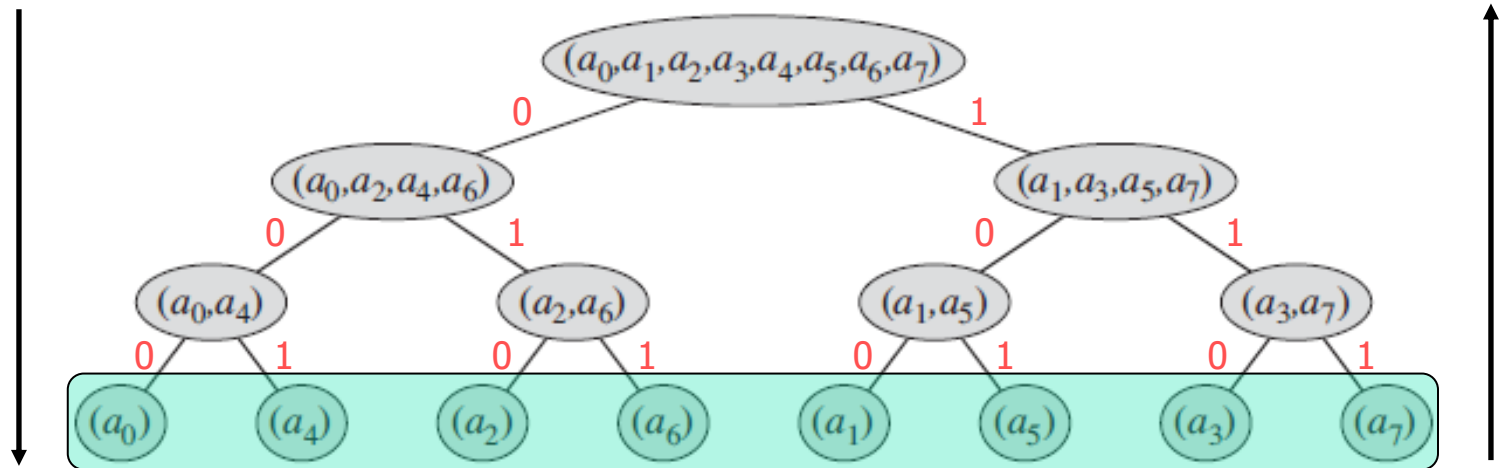
Bit-reversal

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
000	100	010	110	001	101	011	111
0	4	2	6	1	5	3	7

30.3.1 An iterative FFT implementation

- Bit-reversal permutation

Original order



Bit-reversal

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
000	100	010	110	001	101	011	111
0	4	2	6	1	5	3	7

BIT-REVERSE-COPY(a, A) $// \Theta(n \lg n)$
1 $n \leftarrow \text{length}[a]$
2 **for** $k \leftarrow 0$ **to** $n-1$ $// \Theta(n)$
3 **do** $A[\text{rev}(k)] \leftarrow a_k$ $// \Theta(\lg n)$

30.3.1 An iterative FFT implementation

- iterative FFT algorithm

ITERATIVE-FFT (a)

```
1  BIT-REVERSE-COPY ( $a, A$ )           //  $\Theta(n \lg n)$ 
2   $n \leftarrow \text{length}[a]$  //  $n$  is a power of 2.
3  for  $s \leftarrow 1$  to  $\lg n$              //  $\lg n$ 
4       $m \leftarrow 2^s$ 
5       $\omega_m \leftarrow e^{2\pi i/m}$ 
6      for  $k \leftarrow 0$  to  $n-1$  by  $m$      //  $n/m = n/2^s$ 
7           $\omega \leftarrow 1$ 
8          for  $j \leftarrow 0$  to  $m/2 - 1$    //  $m/2 = 2^s/2 = 2^{s-1}$ 
9               $t \leftarrow \omega A[k+j+m/2]$ 
10              $u \leftarrow A[k+j]$ 
11              $A[k+j] \leftarrow u+t$ 
12              $A[k+j+m/2] \leftarrow u-t$ 
13              $\omega \leftarrow \omega \cdot \omega_m$ 
```

$$\begin{aligned} L(n) &= \sum_{s=1}^{\lg n} \frac{n}{2^s} 2^{s-1} \\ &= \sum_{s=1}^{\lg n} \frac{n}{2} \\ &= \Theta(n \lg n) \end{aligned}$$

30.3.2 A parallel FFT circuit

FFT的并联图

- See book

30.3.3 A butterfly operation

$$\text{DFT, } \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \cdots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}, \quad \Theta(n^2)$$

FFT, $\Theta(n \lg n)$?

$$\text{FFT, } \begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{4 \cdot 1} & \omega_n^{4 \cdot 2} & \cdots & \omega_n^{4 \cdot (n-1)} \\ 1 & \omega_n^{2 \cdot 1} & \omega_n^{2 \cdot 2} & \cdots & \omega_n^{2 \cdot (n-1)} \\ 1 & \omega_n^{6 \cdot 1} & \omega_n^{6 \cdot 2} & \cdots & \omega_n^{6 \cdot (n-1)} \\ 1 & \omega_n^{1 \cdot 1} & \omega_n^{1 \cdot 2} & \cdots & \omega_n^{1 \cdot (n-1)} \\ 1 & \omega_n^{5 \cdot 1} & \omega_n^{5 \cdot 2} & \cdots & \omega_n^{5 \cdot (n-1)} \\ 1 & \omega_n^{3 \cdot 1} & \omega_n^{3 \cdot 2} & \cdots & \omega_n^{3 \cdot (n-1)} \\ 1 & \omega_n^{7 \cdot 1} & \omega_n^{7 \cdot 2} & \cdots & \omega_n^{7 \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = W_1 W_2 W_3 a, \quad \Theta(n \lg n)$$

稀疏矩阵?

30.3.3 A butterfly operation

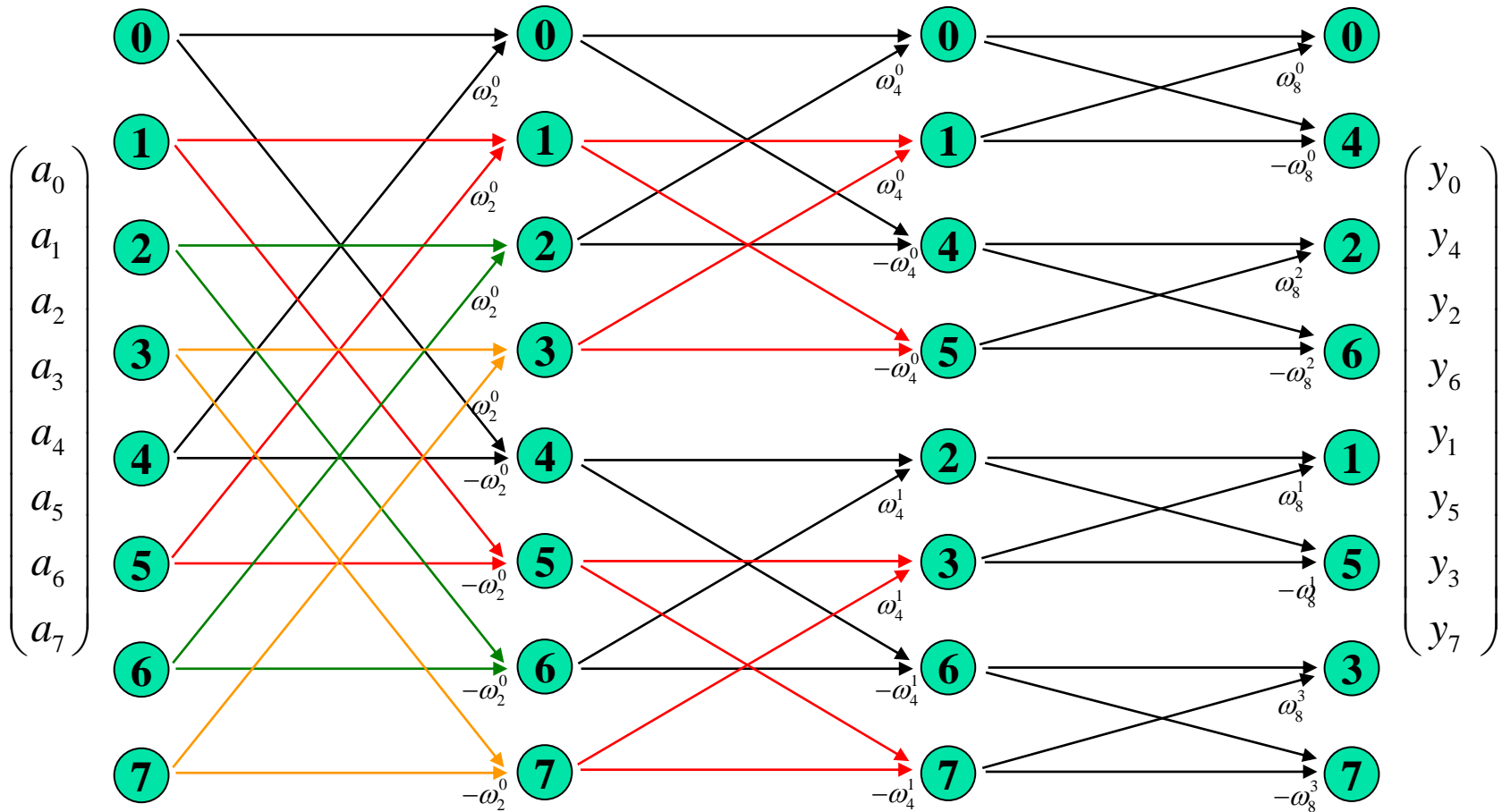
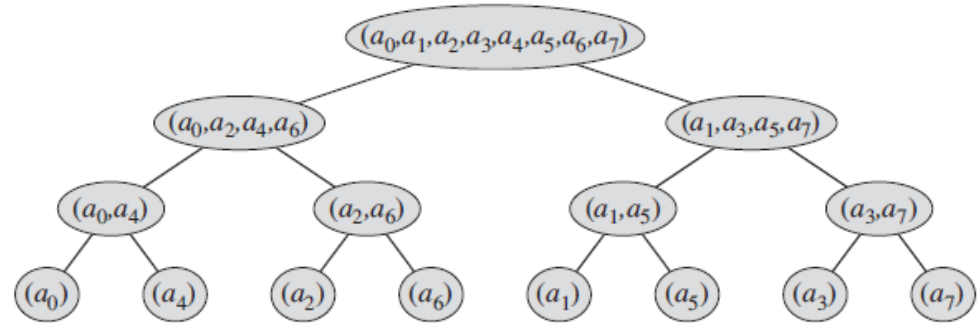
$$\omega = e^{2\pi i/8}$$

$$\begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & L & 1 \\ 1 & \omega_n^{4 \cdot 1} & \omega_n^{4 \cdot 2} & L & \omega_n^{4 \cdot (n-1)} \\ 1 & \omega_n^{2 \cdot 1} & \omega_n^{2 \cdot 2} & L & \omega_n^{2 \cdot (n-1)} \\ 1 & \omega_n^{6 \cdot 1} & \omega_n^{6 \cdot 2} & L & \omega_n^{6 \cdot (n-1)} \\ 1 & \omega_n^{1 \cdot 1} & \omega_n^{1 \cdot 2} & L & \omega_n^{1 \cdot (n-1)} \\ 1 & \omega_n^{5 \cdot 1} & \omega_n^{5 \cdot 2} & L & \omega_n^{5 \cdot (n-1)} \\ 1 & \omega_n^{3 \cdot 1} & \omega_n^{3 \cdot 2} & L & \omega_n^{3 \cdot (n-1)} \\ 1 & \omega_n^{7 \cdot 1} & \omega_n^{7 \cdot 2} & L & \omega_n^{7 \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

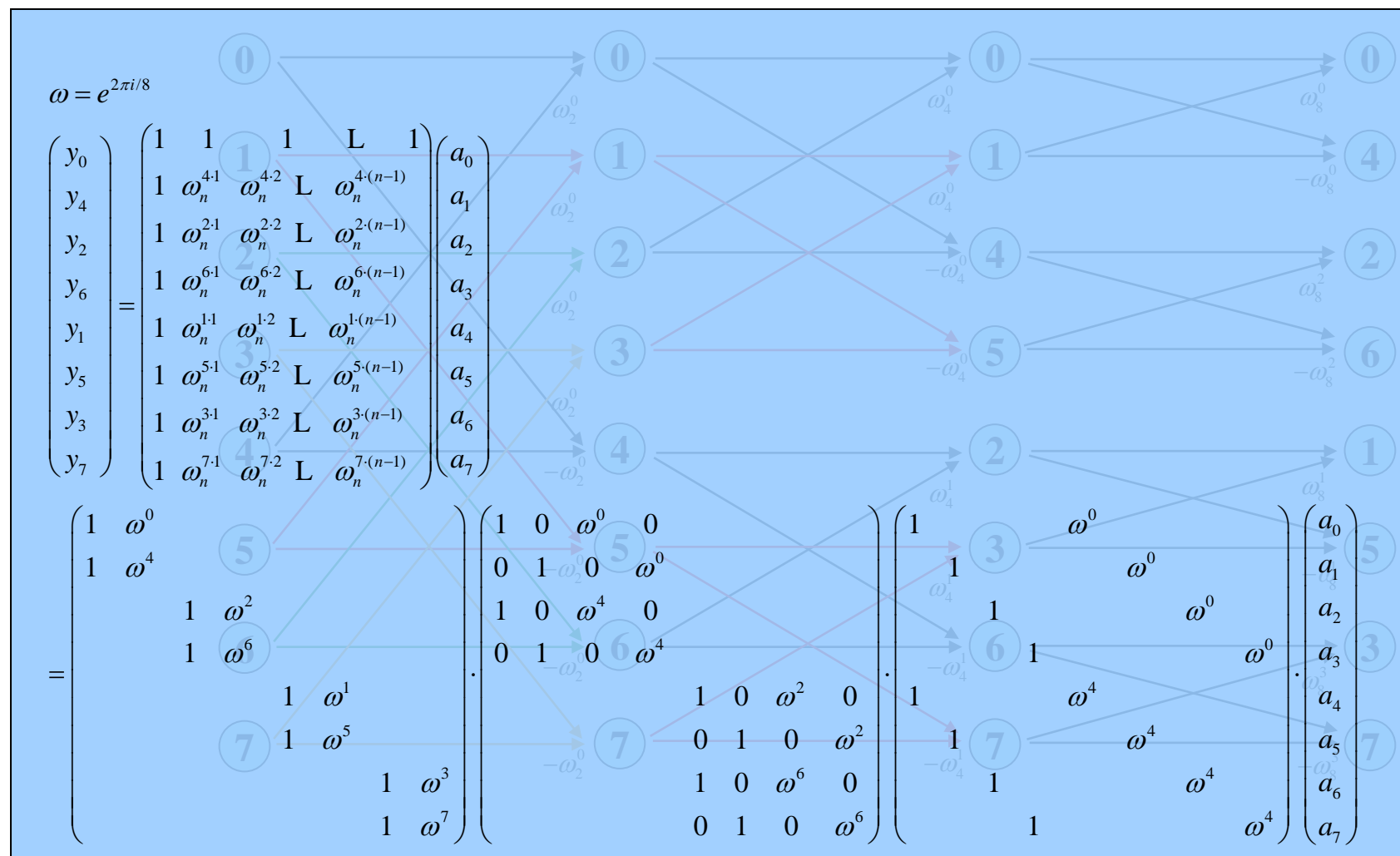
$$= \begin{pmatrix} 1 & \omega^0 & & & & & & \\ 1 & \omega^4 & & & & & & \\ & & 1 & \omega^2 & & & & \\ & & 1 & \omega^6 & & & & \\ & & & & 1 & \omega^1 & & \\ & & & & 1 & \omega^5 & & \\ & & & & & & 1 & \omega^3 \\ & & & & & & 1 & \omega^7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \omega^0 & 0 \\ 0 & 1 & 0 & \omega^0 \\ 1 & 0 & \omega^4 & 0 \\ 0 & 1 & 0 & \omega^4 \\ & & & & 1 & 0 & \omega^2 & 0 \\ & & & & 0 & 1 & 0 & \omega^2 \\ & & & & 1 & 0 & \omega^6 & 0 \\ & & & & 0 & 1 & 0 & \omega^6 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & \omega^0 & & & \\ & 1 & & & & \omega^0 & & \\ & & 1 & & & & \omega^0 & \\ & & & 1 & & & & \omega^0 \\ 1 & & & & & \omega^4 & & \\ & 1 & & & & & \omega^4 & \\ & & 1 & & & & & \omega^4 \\ & & & 1 & & & & \omega^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

30.3.3 A butterfly operation

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



30.3.3 A butterfly operation



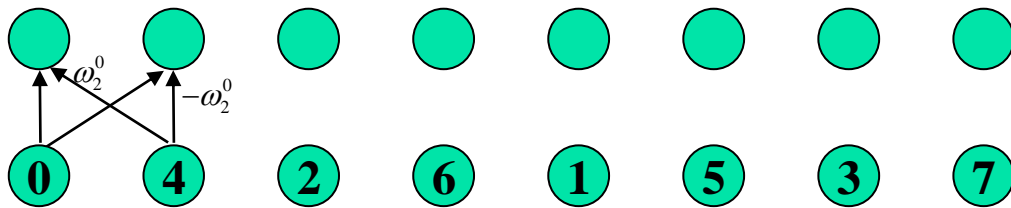
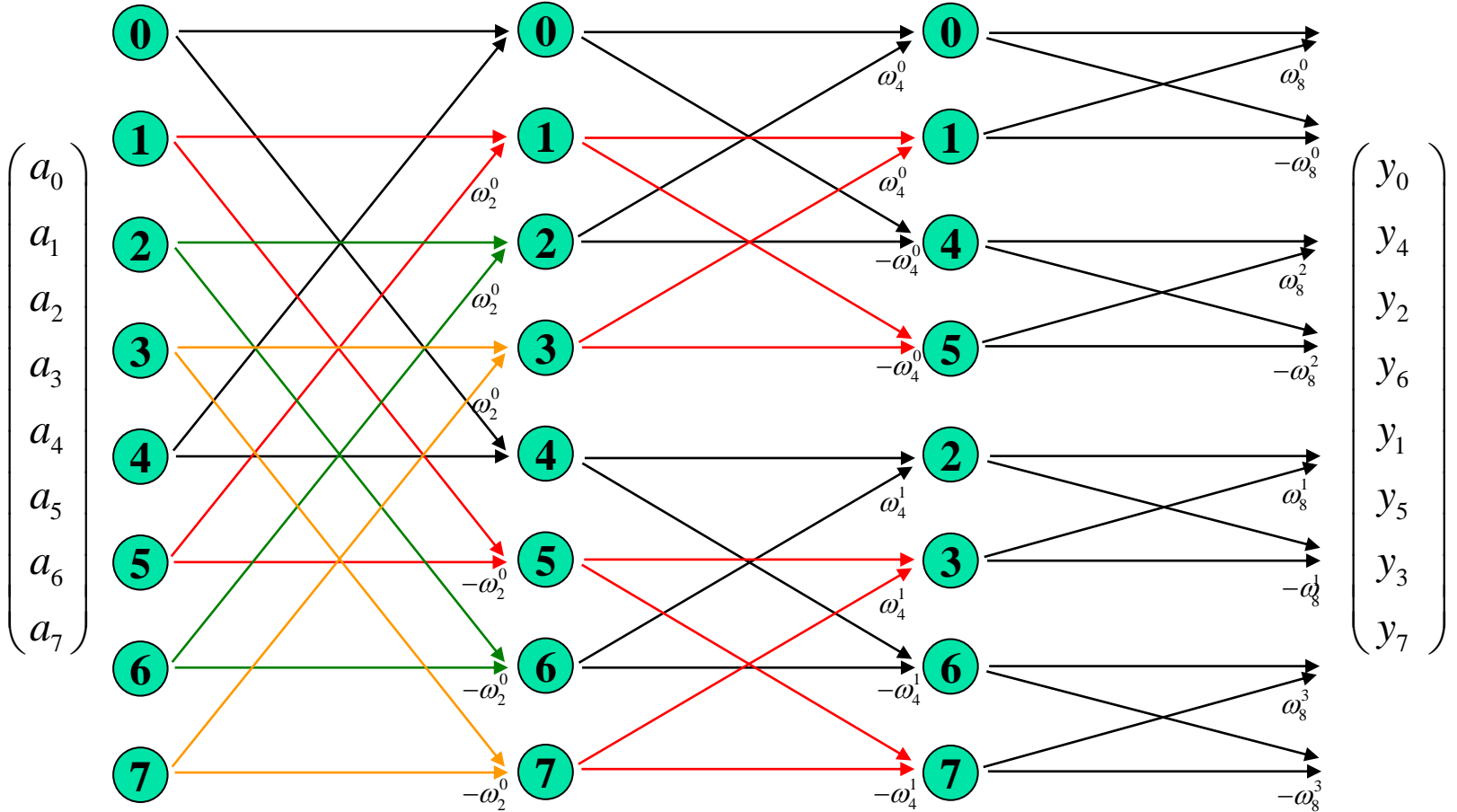
30.3.3 A butterfly operation

$$\omega = e^{2\pi i/8}$$

$$\begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & L & 1 \\ 1 & \omega_n^{4 \cdot 1} & \omega_n^{4 \cdot 2} & L & \omega_n^{4 \cdot (n-1)} \\ 1 & \omega_n^{2 \cdot 1} & \omega_n^{2 \cdot 2} & L & \omega_n^{2 \cdot (n-1)} \\ 1 & \omega_n^{6 \cdot 1} & \omega_n^{6 \cdot 2} & L & \omega_n^{6 \cdot (n-1)} \\ 1 & \omega_n^{1 \cdot 1} & \omega_n^{1 \cdot 2} & L & \omega_n^{1 \cdot (n-1)} \\ 1 & \omega_n^{5 \cdot 1} & \omega_n^{5 \cdot 2} & L & \omega_n^{5 \cdot (n-1)} \\ 1 & \omega_n^{3 \cdot 1} & \omega_n^{3 \cdot 2} & L & \omega_n^{3 \cdot (n-1)} \\ 1 & \omega_n^{7 \cdot 1} & \omega_n^{7 \cdot 2} & L & \omega_n^{7 \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \omega^0 & & & & & & \\ 1 & \omega^4 & & & & & & \\ & & 1 & \omega^2 & & & & \\ & & 1 & \omega^6 & & & & \\ & & & & 1 & \omega^1 & & \\ & & & & 1 & \omega^5 & & \\ & & & & & & 1 & \omega^3 \\ & & & & & & 1 & \omega^7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \omega^0 & 0 \\ 0 & 1 & 0 & \omega^0 \\ 1 & 0 & \omega^4 & 0 \\ 0 & 1 & 0 & \omega^4 \\ & & & & 1 & 0 & \omega^2 & 0 \\ & & & & 0 & 1 & 0 & \omega^2 \\ & & & & 1 & 0 & \omega^6 & 0 \\ & & & & 0 & 1 & 0 & \omega^6 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & \omega^0 & & & \\ & 1 & & & & \omega^0 & & \\ & & 1 & & & & \omega^0 & \\ & & & 1 & & & & \omega^0 \\ 1 & & & & \omega^4 & & & \\ & 1 & & & & \omega^4 & & \\ & & 1 & & & & \omega^4 & \\ & & & 1 & & & & \omega^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

30.3.4 A new butterfly operation?



30.3.4 A new butterfly operation?

(1)

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

$$\text{even, } A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$$

$$\text{odd, } A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$$

$$\text{DFT}(a_0, a_1, \dots, a_{n-1}) = A(\omega_n^k), k = 0, 1, \dots, n-1$$

$$\text{DFT}(a_0, a_2, \dots, a_{n-2}) = A^{[0]}(\omega_{n/2}^k), k = 0, 1, \dots, n/2-1$$

$$\text{DFT}(a_1, a_3, \dots, a_{n-1}) = A^{[1]}(\omega_{n/2}^k), k = 0, 1, \dots, n/2-1$$

(4)

for $k = 0$ to $n/2 - 1$

$$t = \omega_n y_k^{[1]}$$

$$y_k = y_k^{[0]} + t$$

$$y_{k+(n/2)} = y_k^{[0]} - t$$

$$\omega = \omega \omega_n$$

(2)

$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$$

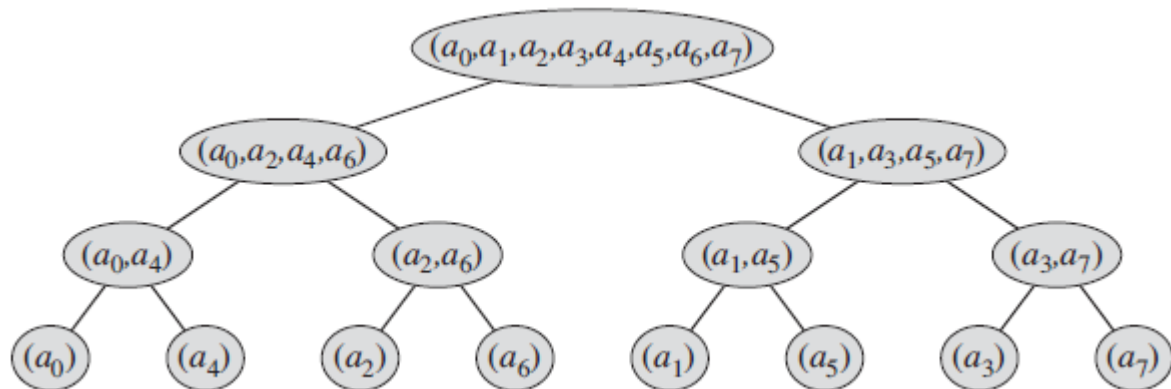
$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$k = 0, 1, \dots, n/2-1$$

(3)



30.3.4 A new butterfly operation?

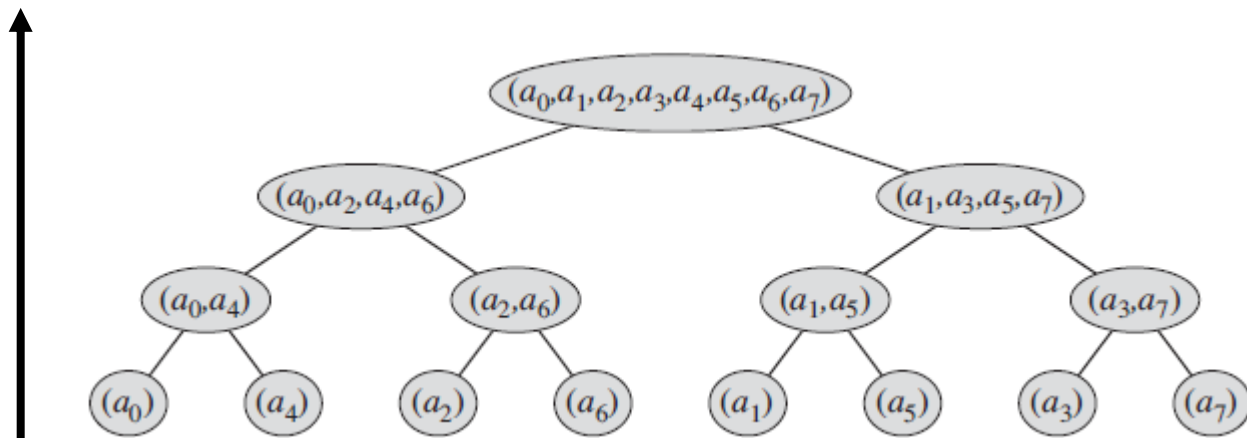
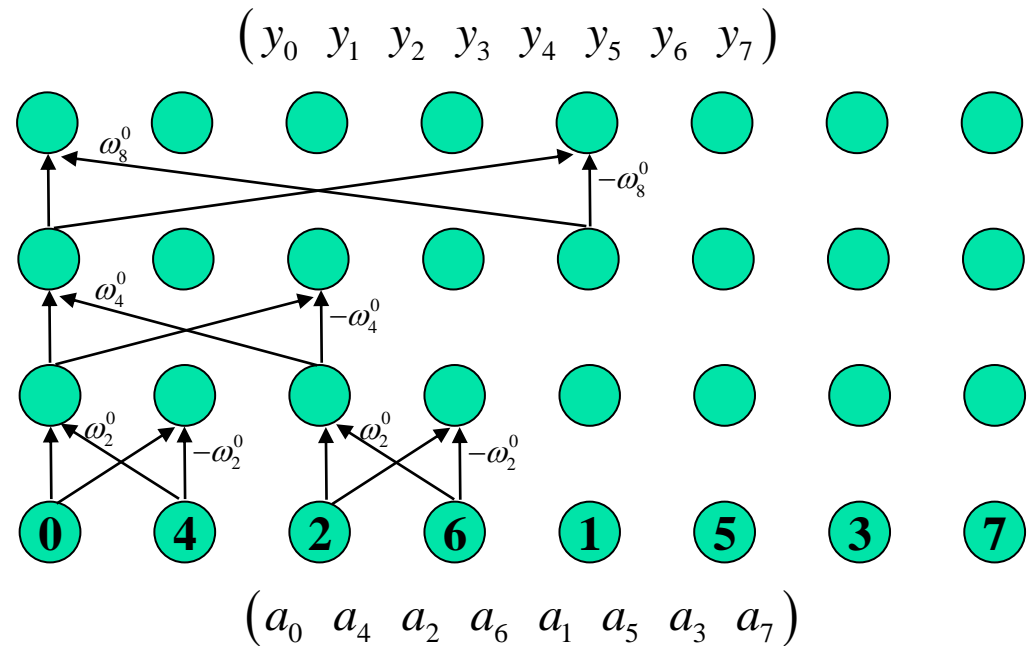
for $k = 0$ to $n/2 - 1$

$$t = \omega y_k^{[1]}$$

$$y_k = y_k^{[0]} + t$$

$$y_{k+(n/2)} = y_k^{[0]} - t$$

$$\omega = \omega \omega_n$$



Some Applications of FFT

- **Polynomials operation**
- **Signal processing (phonic, image, video, \cdots)**

Exercises

对下列输入向量 a ，给出用FFT求 输出向量 y 的蝶形操作过程和结果

$$a=(0,1)$$

$$a=(1,0)$$

$$a=(1,1,0,1)$$

$$a=(0,1,2,3)$$

30.3.1 An iterative FFT implementation

$$y(x) = A(x) = \sum_{j=0}^{n-1} a_j x^j = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

$$x = \omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1};$$

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

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj},$$

$$\begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \dots & \omega_n^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_n^{(n-1) \cdot 1} & \omega_n^{(n-1) \cdot 2} & \dots & \omega_n^{(n-1) \cdot (n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix}$$

even, $A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$

odd, $A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$

$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$$

$$\omega_n^k \quad (\omega_n^k)^2 = \omega_{n/2}^k$$

let $k = 0, \dots, n/2 - 1$, then

$$A(\omega_n^k) = A^{[0]}(\omega_{n/2}^k) + (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k),$$

$$A(\omega_n^{n/2+k}) = A^{[0]}(\omega_{n/2}^k) - (\omega_n^k) \cdot A^{[1]}(\omega_{n/2}^k)$$

$$y_k = y_k^0 + \omega_n^k y_k^1,$$

$$y_{k+n/2} = y_k^0 - \omega_n^k y_k^1$$