FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Divide and Conquer (2)

Convolution and Fast Fourier Transform

(Arlisa Yuliawati)

Introduction

- **Fast Fourier Transform (FFT)** is one of many algorithms that implements the Divide and Conquer paradigm.
  - It is commonly used in signal processing, i.e. to convert between time domain and frequency domain.
  - It is also can be viewed as a fast way to multiply and evaluate polynomials.
    - Suppose we have two polynomials of degree n. Addition of these two polynomials takes $\Theta(n)$ but the multiplication of them takes $\Theta(n^2)$
    - Fast Fourier Transform (FFT) can reduce the time to multiply two polynomials to $\Theta(n \lg n)$.

Polynomials

A polynomial in the variable x over an algebraic field F represents a function $A(x)$ as a formal sum:

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

- a_i is the coefficient with $i = 0,1, \dots, n-1$
- degree of the polynomial is the largest power of x whose coefficient is not equals to zero.

Any integer strictly greater than the degree of a polynomial is a ***degree-bound*** of that polynomial. Therefore, the degree of a polynomial of degree-bound n may be any integer between 0 and $n-1$, inclusive.

Example: $x^3 + 2x^2 - x + 1$

- The degree of this polynomial is 3 and this is a polynomial of degree-bound 4

Polynomials

Suppose we have two polynomials of degree-bound n:

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

- The **sum** of these polynomials is also a polynomial of **degree-bound n** $C(x)$ such that $C(x) = A(x) + B(x) = \sum_{i=0}^{n-1} c_i x^i$ where $c_i = a_i + b_i$.

- Example: $A(x) = x^2 + 2x - 1$ and $B(x) = x^2 - x + 1$ are polynomials of degree-bound 3.
  - The sum result: $C(x) = 2x^2 + x$ (a polynomial of degree-bound 3)

Polynomials

- The **product** of $A(x)$ and $B(x)$ is a polynomial of **degree bound $\boldsymbol{2n-1}$** $D(x)$ such that $D(x) = A(x)B(x)$.

- Example: $A(x) = x^2 + 2x - 1$ and $B(x) = x^2 - x + 1$
- The product result: $D(x) = x^4 + x^3 - 2x^2 + 3x - 1$ (a polynomial of degree-bound 5)
  - To obtain $D(x)$, we can use the standard multiplication by multiplying each term in $A(x)$ by each term in $B(x)$ and then combining terms with equal powers.
  - The other way, $D(x)$ can also be expressed as $D(x) = A(x)B(x) = \Sigma_{i=0}^{2n-2} d_i x^i$ where $\boldsymbol{d_i = \Sigma_{k=0}^{i} a_k b_{i-k}}$.

This formula represents the **convolution** of input vectors a and b, denoted $d = a \otimes b$

Polynomials Representation

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Coefficient Representation

- Given a polynomial $A(x) = \sum_{i=0}^{n-1} a_i x^i$. Its **coefficient representation** is a vector of coefficients $a = (a_0, a_1, \ldots, a_{n-1})$.

- Example:
  - $A(x) = 6x^3 + 7x^2 - 10x + 9$ can be represented as a vector $a = (9, -10, 7, 6)$
  - $B(x) = -2x^3 + 4x - 5$ can be represented as a vector $b = (5, 4, 0, -2)$

Polynomials Representation

Coefficient Representation (2)

- Evaluating a polynomial $A(x) = \Sigma_{i=0}^{n-1} a_i x^i$ at a point x_o takes $\Theta(n)$ time using ***Horner's Rule***:

$$A(x_0) = a_0 + a_1(x_0)^2 + a_2(x_0)^3 + \cdots + a_{n-1}(x_0)^{n-1}$$
$$= a_0 + x_0\left(a1 + x_0\left(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))\right)\right)$$

- Adding two polynomials also takes $\Theta(n)$ time but multiplying them takes $\mathbf{\Theta}\left(\boldsymbol{n}^{\mathbf{2}}\right)$ time by using standard multiplication or convolution.
  - Since multiplying polynomials and computing convolutions are fundamental computational problems of considerable practical importance, this chapter concentrates on efficient algorithms for them

Polynomials Representation

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Coefficient Representation (3)

- Multiplication Example:

$a = (9, -10, 7, 6)$

$b = (-5, 4, 0, -2)$

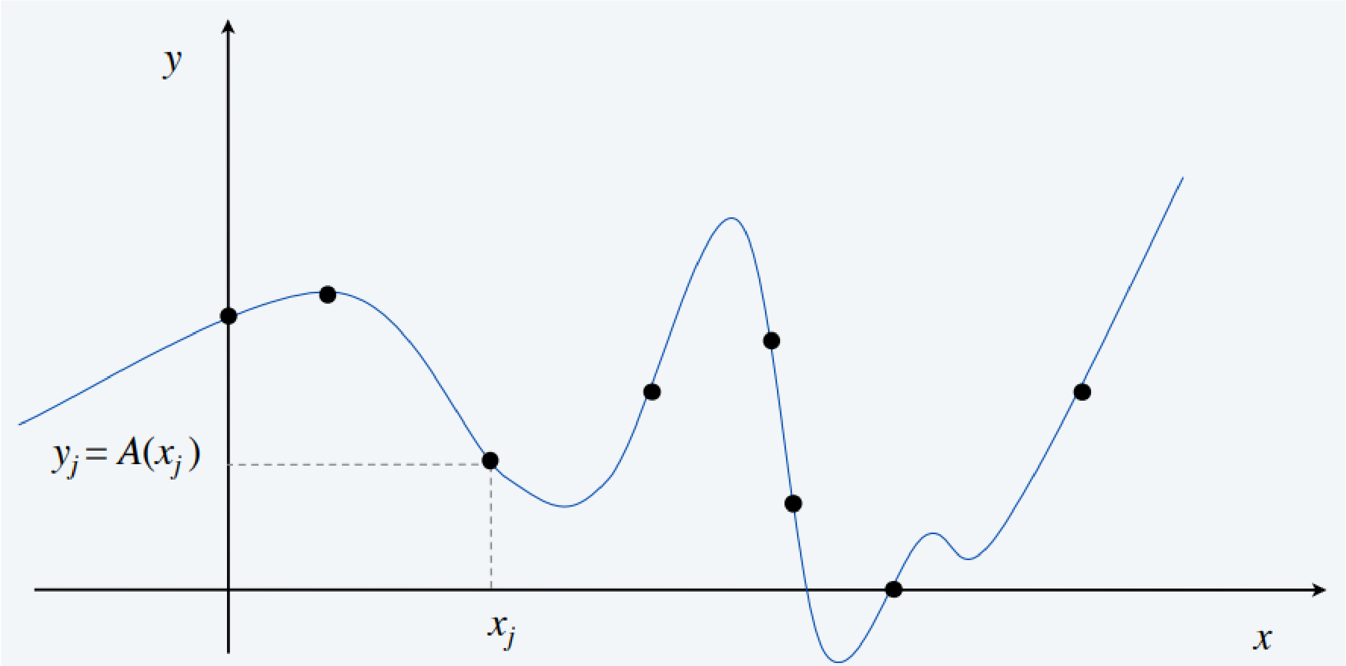
$$
\begin{array}{rrrrrrrr}
 & & & 6x^3 & + 7x^2 & - 10x & + 9 \\
 & & & - 2x^3 & & + 4x & - 5 \\
\hline
 & & & - 30x^3 & - 35x^2 & + 50x & - 45 \\
 & & 24x^4 & + 28x^3 & - 40x^2 & + 36x & \\
- 12x^6 & - 14x^5 & + 20x^4 & - 18x^3 & & & \\
\hline
- 12x^6 & - 14x^5 & + 44x^4 & - 20x^3 & - 75x^2 & + 86x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

convolution $a \otimes b = (-45, 86, -75, -20, 44, -14, -12)$

Polynomials Representation

Point-value Representation

- Given a polynomial $A(x) = \Sigma_{i=0}^{n-1} a_i x^i$. Its **point-value representation** is a set of n point-value pairs $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ such that all the x_k are distinct and $y_k = A(k)$ for $k = 0,1, \dots, n-1$. It has many point-value representations.



Polynomials Representation

Point-value Representation (2)

- Adding two polynomials takes $\Theta(n)$.
- If $C(x) = A(x) + B(x)$, then $C(x_k) = A(x_k) + B(x_k)$

- Suppose $A(x)$ is represented as $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$

and for $B(x)$ is $\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$.

The point-value representation for $C(x)$ is

$$\{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \ldots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

Polynomials Representation

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Point-value Representation (3)

- Multiplying two polynomials also takes $\Theta(n)$.
    - It is much less than the operation in coefficient representation.
- If $C(x) = A(x)B(x)$, then $C(x_k) = A(x_k)B(x_k)$ for any point x_k.

- The multiplication use the **extended point-value representations** consisting of $2n$ point-value pairs each
    - Suppose $A(x)$ and $B(x)$ are expressed in $\{(x_0, y_0), (x_1, y_1), \dots, (x_{2n-1}, y_{2n-1})\}$ and $\{(x_0, y'_0), (x_1, y'_1), \dots, (x_{2n-1}, y'_{2n-1})\}$ respectively.
    - The point-value representation for $C(x)$ is: $\{(x_0, y_0 y'_0), (x_1, y_1 y'_1), \dots, (x_{2n-1}, y_{2n-1}\, y'_{2n-1})\}$

Polynomials Representation

FAKULTAS ILMU KOMPUTER

Converting coefficient to point-value representation

- Computing a point-value representation for a polynomial given in coefficient form is in principle straightforward, since all we have to do is select n distinct points $x_0, x_1, \ldots, x_{n-1}$ and then evaluate $A(x_k)$ for $k = 0, 1, \ldots, n-1$. With Horner's method, evaluating a polynomial at n points takes time $\Theta(n^2)$. We shall see later that if we choose the points x_k cleverly, we can accelerate this computation to run in time $\Theta(n \lg n)$.

Polynomials Representation

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Converting coefficient to point-value representation (2)

Coefficient $\Rightarrow$ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

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

Polynomials Representation

Converting point-value to coefficient representation

- Determining the coefficient-representation of a polynomial from a point-value representation (inverse of evaluation) is called **interpolation.**
- **Uniqueness of an interpolating polynomial theorem:**

  For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of n point-value pairs such that 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 $k = 0, 1, \ldots, n-1$.

- An n-point interpolation based on

  Lagrange's Formula (takes $\Theta(n^2)$) :

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

Polynomials Representation

Converting point-value to coefficient representation (2)

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

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

Vandermonde matrix is invertible iff x_i distinct

The vector a can be obtained from:

$$a = V(x_0, x_1, \ldots, x_{n-1})^{-1} y$$

where V is the Vandermonde matrix

Polynomials Representation

Converting point-value to coefficient representation (3)

- Example: For a polynomial of degree-bound 3, we have:

$$A(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

- Suppose $A(x)$ is represented as $\{(0,1),(1,4),(2,9)\}$, we could identify the corresponding polynomial by using Lagrange's Formula:

$$\begin{aligned} A(x) &= \frac{(x-1)(x-2)}{(0-1)(0-2)} + 4\frac{(x-0)(x-2)}{(1-0)(1-2)} + 9\frac{(x-0)(x-1)}{(2-0)(2-1)} \\ &= \frac{(x-1)(x-2)}{2} - 4x(x-2) + \frac{9}{2}x(x-1) \\ &= \frac{(x^2-3x+2)}{2} - (4x^2-8x) + \frac{9}{2}x^2 - \frac{9}{2}x \\ &= \boldsymbol{x^2+2x+1} \end{aligned}$$

Fast Multiplication of Polynomials

- Coefficient vs point-value representation:

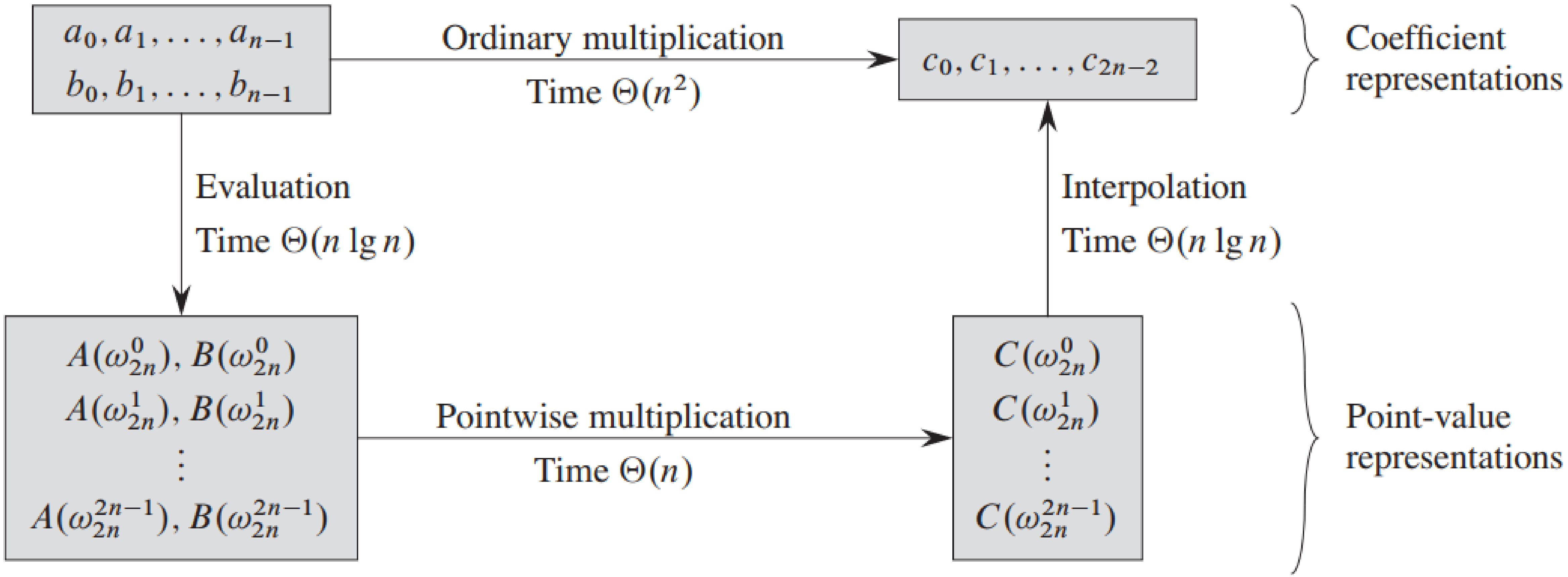
representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

- Can we use the linear time multiplication for polynomials in point-value form to accelerate polynomial multiplication in coefficient form?
  - It depends on whether we can convert a polynomial **quickly** from coefficient form to point-value form (evaluate) and vice versa (interpolate)

Fast Multiplication of Polynomials

- By choosing **complex roots of unity** as the evaluation points, we can produce a point-value representation by taking the **Discrete Fourier Transform (DFT)** of a coefficient vector.
  - Converting between representations takes $\Theta(n \lg n)$ by using DFT (evaluation) and inverse DFT (interpolation).
- The product of two polynomials of degree-bound n is a polynomial of degree-bound $2n$.
  - Before evaluating the input polynomial $A(x)$ and $B(x)$, the degree-bound is doubled to $2n$ by adding n high order coefficient of 0.
  - Thus, we use the **complex 2n$^{\text{th}}$ roots of unity,** denoted by $\boldsymbol{\omega_{2n}}$ terms.

Fast Multiplication of Polynomials



Fast Multiplication of Polynomials

- Given the FFT, the following is $\Theta(n \lg n)$ time procedure for multiplying two polynomials $A(x)$ and $B(x)$ of degree-bound n. It is assumed that n is power of 2.
  - **Double degree-bound**: create the coefficient representation of $A(x)$ and $B(x)$ as degree-bound $2n$ polynomials. $(\Theta(n))$
  - **Evaluate**: Compute point-value representations for $A(x)$ and $B(x)$ by applying the FFT of order $2n$ on each polynomial. $(\Theta(n \lg n))$
    - Contain of the value of both polynomials at the $2n^{th}$ roots of unity.
  - **Pointwise multiply**: Compute point-value representation for $C(x) = A(x)B(x)$ by multiplying these value <u>pointwise</u>. $(\Theta(n))$
    - Contain of the value of $C(x)$ at the $2n^{th}$ roots of unity.
  - **Interpolate**: Create the coefficient representation of the polynomial $C(x)$ by applying the FFT on $2n$ point-value pairs to compute the inverse DFT. $(\Theta(n \lg n))$

Fast Multiplication of Polynomials

- Based on the use of FFT, we have the following theorem:

Theorem 30.2
We can multiply two polynomials of degree-bound n in time $\Theta(n \lg n)$, with both the input and output representations in coefficient form. ■

  - Proof: from the steps described in previous slides, the total time needed is $\Theta(n \lg n)$.

Complex Roots of Unity

A ***complex nth root of unity*** is a complex number ω such that

$\omega^n = 1$. (So, it is any complex number, when it multiplied by itself some number of times, yields 1)

There are exactly n complex nth roots of unity: $e^{2\pi ik/n}$ for $k = 0, 1, \ldots, n-1$. To interpret this formula, we use the definition of the exponential of a complex number:

$e^{iu} = \cos(u) + i\sin(u)$. Euler's Formula

e denotes the euler's number and i denotes the imaginary number,

while $u = 0, \frac{1}{n}\tau, \frac{2}{n}\tau, \ldots, \frac{n-1}{n}\tau$ where $\tau = 2\pi$

Complex Roots of Unity

- ω_n is the **principal n^{th} root of unity**, defined as $\omega_n = e^{\frac{2\pi i}{n}}$
- All other complex n^{th} roots of unity are powers of ω_n:

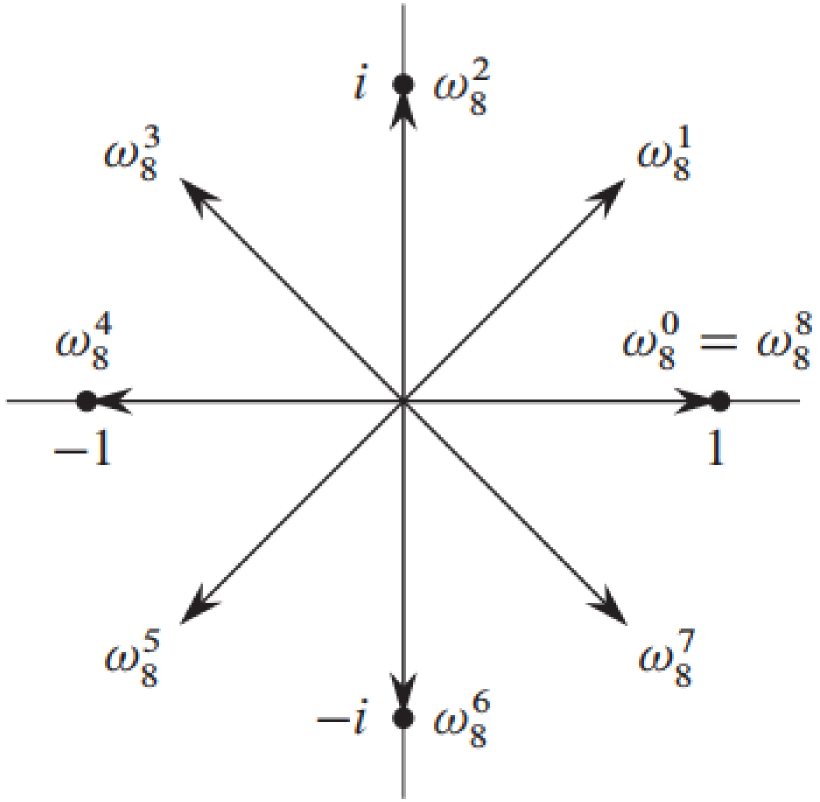
$$\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$$

- Proof for n complex n^{th} roots of unity:

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

By Euler's Formula, $e^{\pi i} = \cos\pi + i\sin\pi = -1$

Complex Roots of Unity



In this example, the values of 8 complex 8^{th} roots of unity are:

$$\omega_8^0 = e^{\frac{2\pi i 0}{8}} = \mathbf{1},$$
$$\omega_8^1 = e^{\frac{2\pi i}{8}} = e^{\frac{\pi}{4}i} = \cos\left(\frac{\pi}{4}\right) + i\sin\left(\frac{\pi}{4}\right) = \frac{1}{2}\sqrt{2} + \left(\frac{1}{2}\sqrt{2}\right)\boldsymbol{i},$$
$$\omega_8^2 = e^{\frac{4\pi i}{8}} = e^{\frac{\pi}{2}i} = \cos\left(\frac{\pi}{2}\right) + i\sin\left(\frac{\pi}{2}\right) = \boldsymbol{i},$$
$$\omega_8^3 = e^{\frac{6\pi i}{8}} = e^{\frac{3\pi}{4}i} = \cos\left(\frac{3\pi}{4}\right) + i\sin\left(\frac{3\pi}{4}\right) = -\frac{1}{2}\sqrt{2} + \left(\frac{1}{2}\sqrt{2}\right)\boldsymbol{i},$$
$$\omega_8^4 = -\mathbf{1},$$
$$\omega_8^5 = -\frac{1}{2}\sqrt{2} - \left(\frac{1}{2}\sqrt{2}\right)\boldsymbol{i},$$
$$\omega_8^6 = \cdots, \text{ and}$$
$$\omega_8^7 = \cdots.$$

Other references may mention $\omega_8^1, \ldots, \omega_8^8$ as the 8 complex 8^{th} roots of unity. It is the same since $\omega_8^0 = \omega_8^8$

Figure 30.2 The values of $\omega_8^0, \omega_8^1, \ldots, \omega_8^7$ in the complex plane, where $\omega_8 = e^{2\pi i/8}$ is the principal 8th root of unity.

Complex Roots of Unity

- Cancellation Lemma

  For any integers $n \geq 0, k \geq 0, d > 0$, $\omega_{dn}^{dk} = \omega_n^k$

  Proof: $\omega_{dn}^{dk} = \left(e^{\frac{2\pi i d k}{dn}}\right) = \left(e^{2\pi \frac{i}{n}}\right)^k$

  Example: $\omega_8^2 = \omega_4^1$

- A Corollary

  For any even integer $n > 0$, $\omega_n^{\frac{n}{2}} = \omega_2 = -1$

  Proof: $\omega_n^{\frac{n}{2}} = \left(e^{\frac{2\pi i}{n}}\right)^{\frac{n}{2}} = \left(e^{2\pi i/2}\right) = \omega_2 = -1$

Complex Roots of Unity

- Halving Lemma

If $n > 0$ is even, the squares of the n complex n^{th} roots of unity are the $\frac{n}{2}$ complex $\left(\frac{n}{2}\right)^{th}$ roots of unity.

Proof By the cancellation lemma, we have $(\omega_n^k)^2 = \omega_{n/2}^k$, for any nonnegative integer k. Note that if we square all of the complex nth roots of unity, then we obtain each $(n/2)$th root of unity exactly twice, since

$$\begin{aligned}
(\omega_n^{k+n/2})^2 &= \omega_n^{2k+n} \\
&= \omega_n^{2k}\omega_n^n \\
&= \omega_n^{2k} \\
&= (\omega_n^k)^2 .
\end{aligned}$$

Example: $(\omega_8^2)^2 = \omega_4^2 = e^{\pi i} = -1$

Complex Roots of Unity

- Summation Lemma

For any even integer $n \geq 1$ and nonzero integer k not divisible by n,

$$\Sigma_{i=0}^{n-1}\left(\omega_n^k\right)^i = 0$$

Proof:

$$\begin{aligned}
\sum_{j=0}^{n-1}\left(\omega_n^k\right)^j &= \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} \\
&= \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} \\
&= \frac{(1)^k - 1}{\omega_n^k - 1} \\
&= 0 .
\end{aligned}$$

Because we require that k is not divisible by n, and because $\omega_n^k = 1$ only when k is divisible by n, we ensure that the denominator is not 0. ■

The DFT

- Back to the problem of evaluating polynomial $A(x) = \Sigma_{i=0}^{n-1} a_i x^i$ of degree-bound n at $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$.
  - The coefficient representation for A is $(a_0, a_1, \dots, a_{n-1})$
  - The degree-bound n refers to $2n$ since we perform double degree-bound

- Let $y_k = A(x_k) = A\left(\omega_n^k\right)$ for $k = 0,1,2, \dots, n-1$, then we have

$$A(x_k) = \Sigma_{i=0}^{n-1} a_i \left(\omega_n^k\right)^i$$

- The vector $y = (y_0, y_1, \dots, y_{n-1})$ is the **Discrete Fourier Transform (DFT)** of the coefficient vector $(a_0, a_1, \dots, a_{n-1})$.
  - It is also written as $y = DFT_n(a)$.

The DFT

- Example: Compute DFT for $p = (0,1,2,3)$.

The FFT

- By using a method known as the **Fast Fourier Transform (FFT)**, the computation of $y = DFT_n(a)$ takes $\Theta(n \lg n)$.
  - FFT takes the advantages of <u>the special properties of complex roots of unity</u>
  - We assume that n is an exact power of 2

- The divide and conquer strategy for FFT:
  - Two new polynomials degree-bound $\frac{n}{2}$ are generated: the even-indexed coefficient ($A^{[0]}(x)$) and odd-indexed coefficient ($A^{[1]}(x)$).
    - $A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{\frac{n}{2}-1}$
    - $A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{\frac{n}{2}-1}$
  - Then it follows that $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

The FFT

- By using FFT, the problem of evaluating A at $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$ reduces to:
  - Evaluating the degree-bound $\frac{n}{2}$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points $(\omega_n^0)^2, (\omega_n^1)^2, (\omega_n^2)^2, \dots, (\omega_n^{n-1})^2$.
    - It recursively evaluate the polynomials $A^{[0]}$ and $A^{[1]}$ of degree-bound $\frac{n}{2}$ at the $\frac{n}{2}$ complex $\left(\frac{n}{2}\right)^{th}$ roots of unity
    - Following the halving lemma, the n element DFT_n computation is divided into two $\frac{n}{2}$ element $DFT_{n/2}$ computation.
  - Combining the result into $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

The FFT

RECURSIVE-FFT(a)

1 $n = a.length$ // n is a power of 2
2 **if** $n == 1$
3 **return** a

Represents the basis of the recursion

4 $\omega_n = e^{2\pi i/n}$
5 $\omega = 1$

Make sure that ω is updated properly, together with line 13.

6 $a^{[0]} = (a_0, a_2, \ldots, a_{n-2})$
7 $a^{[1]} = (a_1, a_3, \ldots, a_{n-1})$

The coefficient vectors for polynomials $A^{[0]}$ and $A^{[1]}$

8 $y^{[0]} =$ RECURSIVE-FFT$(a^{[0]})$
9 $y^{[1]} =$ RECURSIVE-FFT$(a^{[1]})$

Recursive $DFT_{\frac{n}{2}}$ computations

10 **for** $k = 0$ **to** $n/2 - 1$
11 $y_k = y_k^{[0]} + \omega\, y_k^{[1]}$
12 $y_{k+(n/2)} = y_k^{[0]} - \omega\, y_k^{[1]}$

Combine the result of the recursive $DFT_{n/2}$ calculations

13 $\omega = \omega\, \omega_n$
14 **return** y // y is assumed to be a column vector

The FFT

Lines 11–12 combine the results of the recursive $\mathrm{DFT}_{n/2}$ calculations. For $y_0, y_1, \ldots, y_{n/2-1}$, line 11 yields

$$\begin{aligned} y_k &= y_k^{[0]} + \omega_n^k y_k^{[1]} \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) \\ &= A(\omega_n^k) && \text{(by equation (30.9))} . \end{aligned}$$

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

For $y_{n/2}, y_{n/2+1}, \ldots, y_{n-1}$, letting $k = 0, 1, \ldots, n/2 - 1$, line 12 yields

$$\begin{aligned} y_{k+(n/2)} &= y_k^{[0]} - \omega_n^k y_k^{[1]} \\ &= y_k^{[0]} + \omega_n^{k+(n/2)} y_k^{[1]} && \text{(since } \omega_n^{k+(n/2)} = -\omega_n^k\text{)} \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+(n/2)} A^{[1]}(\omega_n^{2k+n}) && \text{(since } \omega_n^{2k+n} = \omega_n^{2k}\text{)} \\ &= A(\omega_n^{k+(n/2)}) && \text{(by equation (30.9))} . \end{aligned}$$

The FFT

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

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) . \end{aligned}$$

Thus, we can evaluate a polynomial of degree-bound n at the complex nth roots of unity in time $\Theta(n \lg n)$ using the fast Fourier transform.

The FFT

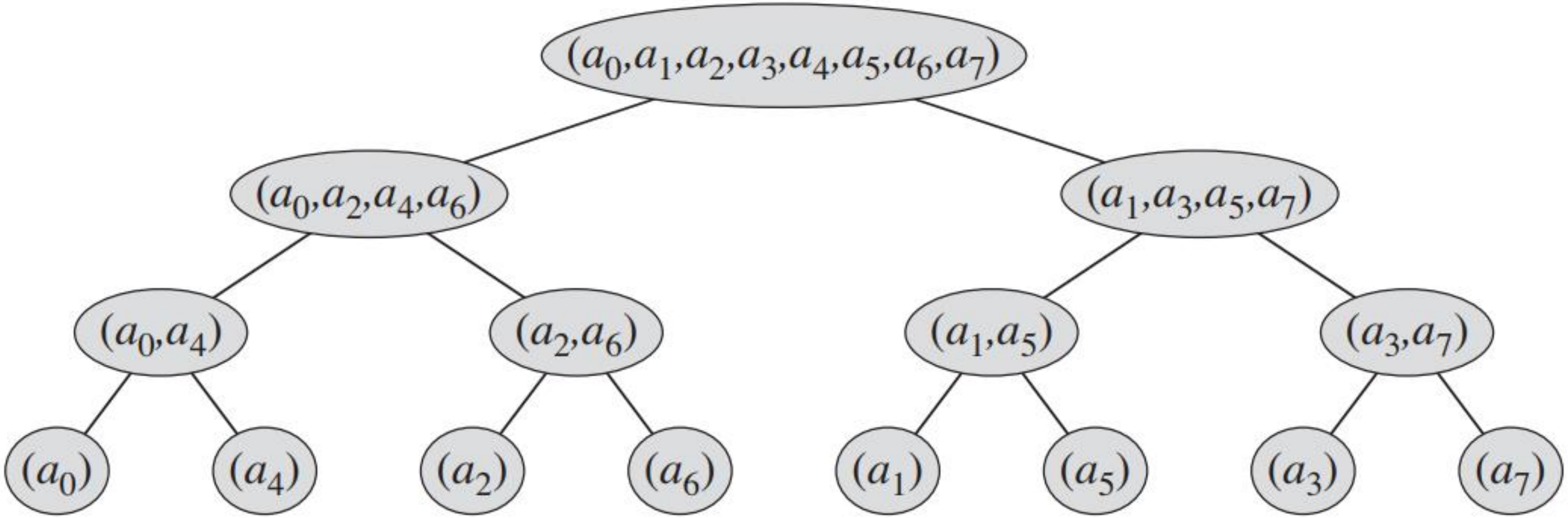


Figure 30.4 The tree of input vectors to the recursive calls of the RECURSIVE-FFT procedure. The initial invocation is for $n = 8$.

Interpolation at The Complex Roots of Unity

- Interpolate by writing DFT as a matrix equation $y = V_n a$ as follows and then looking at the form of the **matrix inverse**.

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

The (k, j) entry of V_n is ω_n^{kj}, for $j, k = 0, 1, \ldots, n-1$. The exponents of the entries of V_n form a multiplication table.

For the inverse operation, which we write as $a = \mathrm{DFT}_n^{-1}(y)$, we proceed by multiplying y by the matrix V_n^{-1}, the inverse of V_n.

Interpolation at The Complex Roots of Unity

Given the inverse matrix V_n^{-1}, we have that $\mathrm{DFT}_n^{-1}(y)$ is given by

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$$

- Recall how to evaluate a polynomial: $y_k = A(x_k) = \Sigma_{i=0}^{n-1} a_i \left(\omega_n^k\right)^i$. Both equations have similar form:
  - The role of a and y are switched
  - ω_n is replaced with ω_n^{-1}
  - Each element of the result is divided by n
- Thus, we can compute DFT_n^{-1} in $\Theta(n \lg n)$ as well.

The Convolution Theorem

- By using FFT and inverse FFT, transformation from coefficient representation to point-value representation (and the other direction) take $\Theta(n \lg n)$. In the context of polynomial multiplication, we have the following **Convolution Theorem**.

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

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

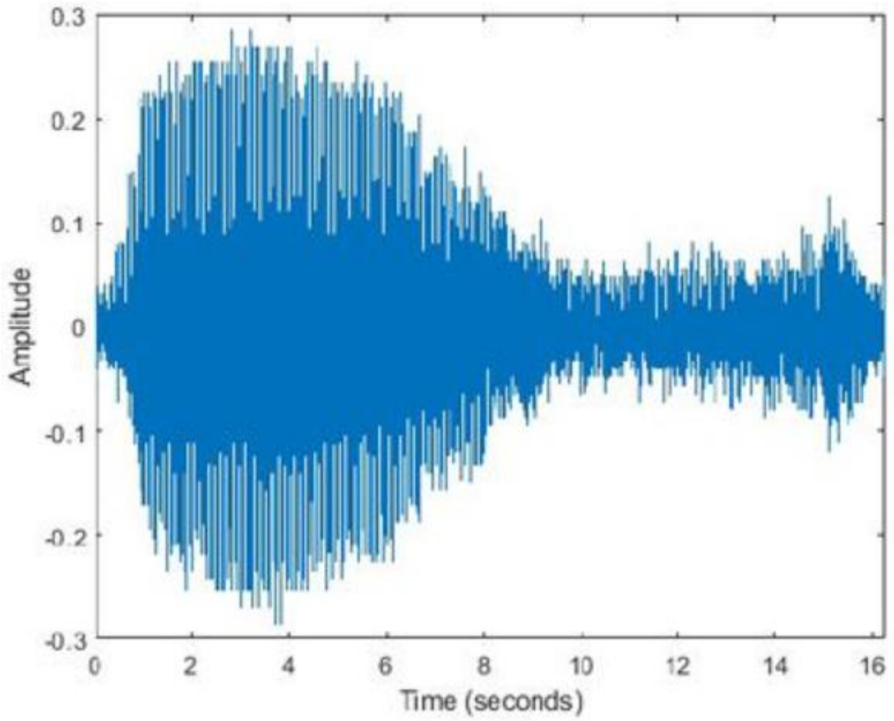
where the vectors a and b are padded with 0s to length $2n$ and $\cdot$ denotes the componentwise product of two $2n$-element vectors. ■

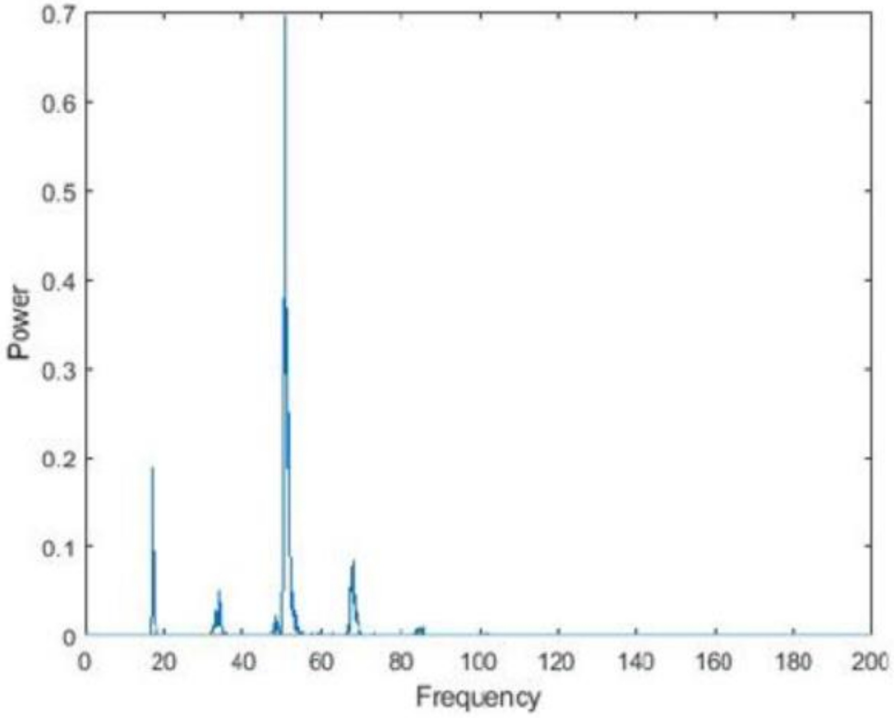
The Convolution Theorem

- Example: Find $(1+x)(1+x+x^2)$ using DFT

Applications

- In signal processing: to convert from time domain to frequency domain.

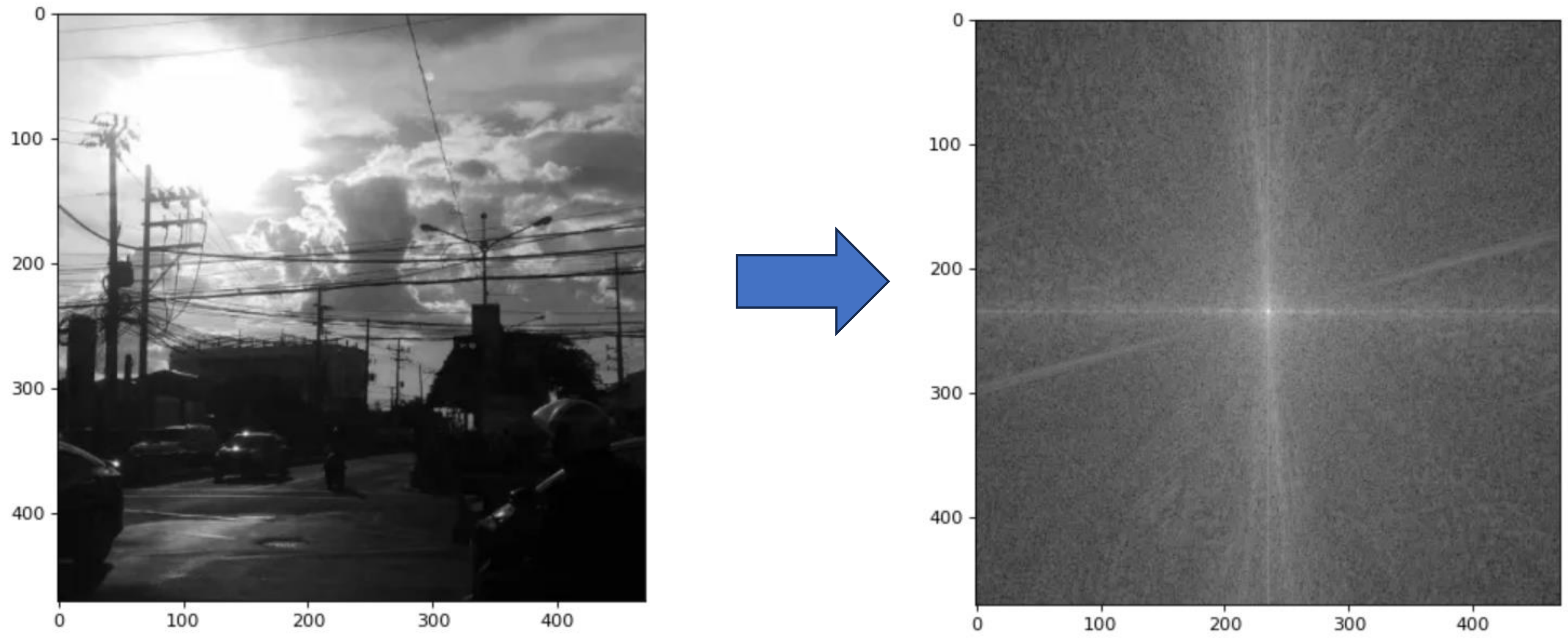




- More on [Fast Fourier Transform (FFT) - MATLAB & Simulink (mathworks.com)]

Applications

- In image processing: to convert from an image to frequency distribution map.



- More on [Image Processing with Python — Application of Fourier Transformation | by Tonichi Edeza | Towards Data Science]

Summary

- FFT speed up the standard polynomial multiplication from $\Theta(n^2)$ to $\Theta(n \lg n)$.
  - The point-wise multiplication contribute the most to linear time multiplication, but converting from point-wise to coefficient representation is not as simple as the other way around (Lagrange formula needs $\Theta(n^2)$)
  - By using DFT, the transformation from coefficient to point-wise representation (and vice versa) takes only $\Theta(n \lg n)$
- The convolution for polynomial multiplication also can be modified such that it takes $\Theta(n \lg n)$.
- FFT is useful for many applications, especially for ..

References

- Lecturer Slides by Bapak L. Yohanes Stefanus
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
- Jon Kleinberg and Eva Tardos. 2013. Algorithm Design. Pearson Education
- https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/mit6_046js15_lec03/