FFT Supplementary Introduction

YAO ZHAO

Outline

- How FFT speed up DFT
- Applications of FFT

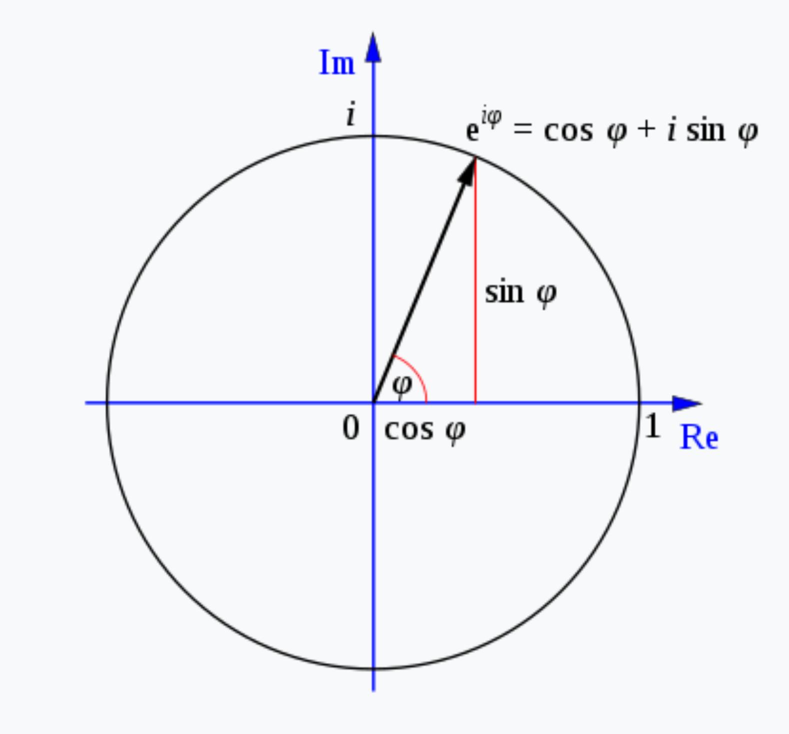
How FFT speed up DFT

The DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn} \quad \text{where } e^{\frac{2\pi i}{N}} \; is\ a\ primitive\ N^{th}\ root\ of\ 1$$

$$= \sum_{n=0}^{N-1} x_n * \boxed{\left(\cos\left(\frac{2\pi kn}{N}\right) - i * \sin\left(\frac{2\pi kn}{N}\right)\right)}$$

where the second expression follows from the first one by Euler's formula.



Roots of Unity

An n^{th} root of unity is a complex number x such that $x^n = 1$

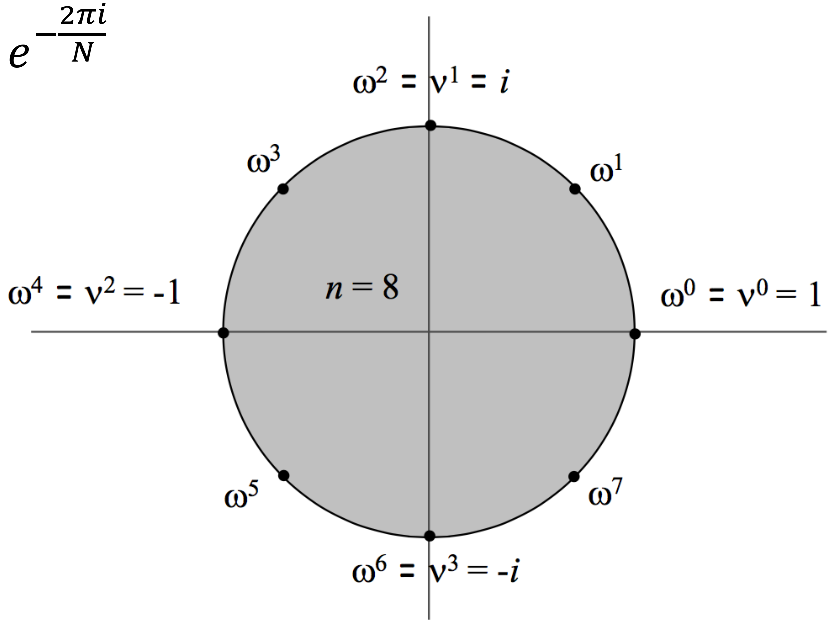
The n^{th} roots of unity are: $\omega^0, \ \omega^1, \ \ \omega^2, \ \ \ldots\ldots \omega^{\mathrm{N}-1}$ where $\omega = e^{-\frac{2\pi i}{N}}$

$$\{\omega^0, \ \omega^1, \ \ \omega^2, \ \ \ldots\ldots \omega^{\mathrm{N}-1}\} = \{e^{-\frac{2\pi i}{N}*0}, e^{-\frac{2\pi i}{N}*1}, \cdots, e^{-\frac{2\pi i}{N}*(N-1)}\}$$

$$\boxed{\omega^{\frac{N}{2}} = e^{-\frac{2\pi i}{N}*\frac{N}{2}} = e^{-\pi i} = -1}$$

$$\boxed{(\omega^{\frac{N}{2}+k})^2 = (\omega^k)^2}$$

Please remember this 2 formulas



DFT Calculation Example

$x_0 = 0$
$x_1 = 1$
$x_2 = 0$
$x_3 = -1$

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn}$$

$$X_0 = x_0 * e^{-\frac{2\pi i}{4}*0*0} + x_1 * e^{-\frac{2\pi i}{4}*0*1} + x_2 * e^{-\frac{2\pi i}{4}*0*2} + x_3 * e^{-\frac{2\pi i}{4}*0*3}$$

$$X_1 = x_0 * e^{-\frac{2\pi i}{4}*1*0} + x_1 * e^{-\frac{2\pi i}{4}*1*1} + x_2 * e^{-\frac{2\pi i}{4}*1*2} + x_3 * e^{-\frac{2\pi i}{4}*1*3}$$

$$X_2 = x_0 * e^{-\frac{2\pi i}{4}*2*0} + x_1 * e^{-\frac{2\pi i}{4}*2*1} + x_2 * e^{-\frac{2\pi i}{4}*2*2} + x_3 * e^{-\frac{2\pi i}{4}*2*3}$$

$$X_3 = x_0 * e^{-\frac{2\pi i}{4}*3*0} + x_1 * e^{-\frac{2\pi i}{4}*3*1} + x_2 * e^{-\frac{2\pi i}{4}*3*2} + x_3 * e^{-\frac{2\pi i}{4}*3*3}$$

$O(n^2)$

$X_0 = 0 \quad X_1 = -2i \quad X_2 = 0 \quad X_3 = 2i$

FFT accelerates DFT

$x_0 = 0$
$x_2 = 0$

$$\Rightarrow X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn}\ (N=2) \Rightarrow \begin{aligned} E_0 &= x_0{*}e^{-\frac{2\pi i}{2}*0*0} + x_2{*}e^{-\frac{2\pi i}{2}*0*1} \\ E_1 &= x_0{*}e^{-\frac{2\pi i}{2}*1*0} + x_2{*}e^{-\frac{2\pi i}{2}*1*1} \end{aligned}$$

$x_1 = 1$
$x_3 = -1$

$$\Rightarrow X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn}(N=2) \Rightarrow \begin{aligned} D_0 &= x_1{*}e^{-\frac{2\pi i}{2}*0*0} + x_3{*}e^{-\frac{2\pi i}{2}*0*1} \\ D_1 &= x_1{*}e^{-\frac{2\pi i}{2}*1*0} + x_3{*}e^{-\frac{2\pi i}{2}*1*1} \end{aligned}$$

$$\begin{aligned} X_1 &= x_0{*}e^{-\frac{2\pi i}{4}*1*0} + x_1{*}e^{-\frac{2\pi i}{4}*1*1} + x_2{*}e^{-\frac{2\pi i}{4}*1*2} + x_3{*}e^{-\frac{2\pi i}{4}*1*3}(N=4) \\ &= x_0{*}e^{-\frac{2\pi i}{4}*1*0} + x_2{*}e^{-\frac{2\pi i}{4}*1*2} + (x_1{*}e^{-\frac{2\pi i}{4}*1*1} + x_3{*}e^{-\frac{2\pi i}{4}*1*3}) \\ &= x_0{*}e^{-\frac{2\pi i}{4}*1*0} + x_2{*}e^{-\frac{2\pi i}{4}*1*2} + e^{-\frac{2\pi i}{4}*1}\ (x_1{*}e^{-\frac{2\pi i}{4}*1*0} + x_3{*}e^{-\frac{2\pi i}{4}*1*2}) \\ &= x_0{*}e^{-\frac{2\pi i}{2}*1*\frac{0}{2}} + x_2{*}e^{-\frac{2\pi i}{2}*1*\frac{2}{2}} + e^{-\frac{2\pi i}{4}*1}\ (x_1{*}e^{-\frac{2\pi i}{2}*1*\frac{0}{2}} + x_3{*}e^{-\frac{2\pi i}{2}*1*\frac{2}{2}}) \\ &= E_1 + e^{-\frac{2\pi i}{4}*1}D_1 \end{aligned}$$

In the same way: $X_0 = E_0 + e^{-\frac{2\pi i}{4}*0}D_0$

FFT accelerates DFT

$$X_2 = x_0{}^*e^{-\frac{2\pi i}{4}*2*0} + x_1*e^{-\frac{2\pi i}{4}*2*1} + x_2*e^{-\frac{2\pi i}{4}*2*2} + x_3*e^{-\frac{2\pi i}{4}*2*3} (N = 4)$$

$$= x_0{}^*e^{-\frac{2\pi i}{2}*(0+2)*\frac{0}{2}} + x_2*e^{-\frac{2\pi i}{2}*(0+2)*\frac{2}{2}} + e^{-\frac{2\pi i}{4}*2} (x_1*e^{-\frac{2\pi i}{2}*(0+2)*\frac{0}{2}} + x_3*e^{-\frac{2\pi i}{2}*(0+2)*\frac{2}{2}})$$

$$= e^{-\frac{2\pi i}{2}*2} * x_0{}^*e^{-\frac{2\pi i}{2}*0*0} + e^{-\frac{2\pi i}{2}*2} * x_2*e^{-\frac{2\pi i}{2}*0*\frac{2}{2}} + e^{-\frac{2\pi i}{4}*2} * e^{-\frac{2\pi i}{4}*0} (e^{-\frac{2\pi i}{2}*2} * x_1*e^{-\frac{2\pi i}{2}*0*\frac{0}{2}} + e^{-\frac{2\pi i}{2}*2} * x_3*e^{-\frac{2\pi i}{2}*0*\frac{2}{2}})$$

$$= x_0{}^*e^{-\frac{2\pi i}{2}*0*0} + x_2*e^{-\frac{2\pi i}{2}*0*1} - e^{-\frac{2\pi i}{4}*0} (x_1{}^*e^{-\frac{2\pi i}{2}*0*0} + x_3*e^{-\frac{2\pi i}{2}*0*1})$$

$$= E_0 - e^{-\frac{2\pi i}{4}*0} D_0$$

In the same way:

$$X_3 = E_1 - e^{-\frac{2\pi i}{4}*1} D_1$$

1

-1

Euler: $e^{\pi i} + 1 = 0$

FFT Pseudo-code

input： n, a_0, a_1,…, a_{n-1} （n = 2^k (k = 0,1,2…) you can check n & (n - 1) == 0）
output： y_0, y_1,…, y_{n-1}

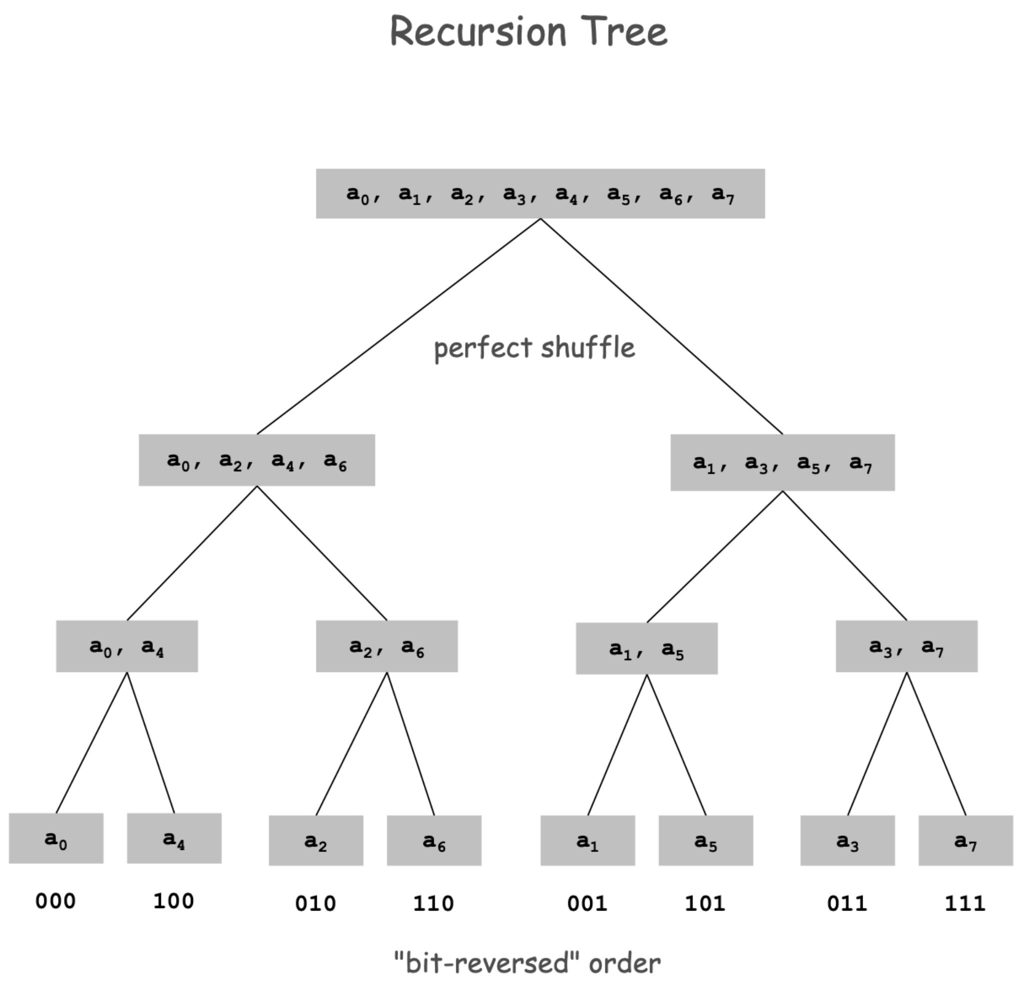
```
FFT(n, a_0, a_1,…, a_{n-1}) {
    if (n == 1) return a_0
        (e_0, e_1,…, e_{n/2-1})<=FFT(n/2, a_0, a_2,…, a_{n-2})
        (d_0, d_1,…, d_{n/2-1})<=FFT(n/2, a_1, a_3,…, a_{n-1})
        for k = 0 to n/2 - 1 {
            ω^k = e^{-2πik/n}  //When you write your code, you can use:ω^k=cos(2πk/n) − isin(2πk/n)

            y_k = e_k + ω^k*d_k
            y_{k+n/2} = e_k − ω^k*d_k
    }
   return (y_0, y_1,…, y_{n-1})
}
```

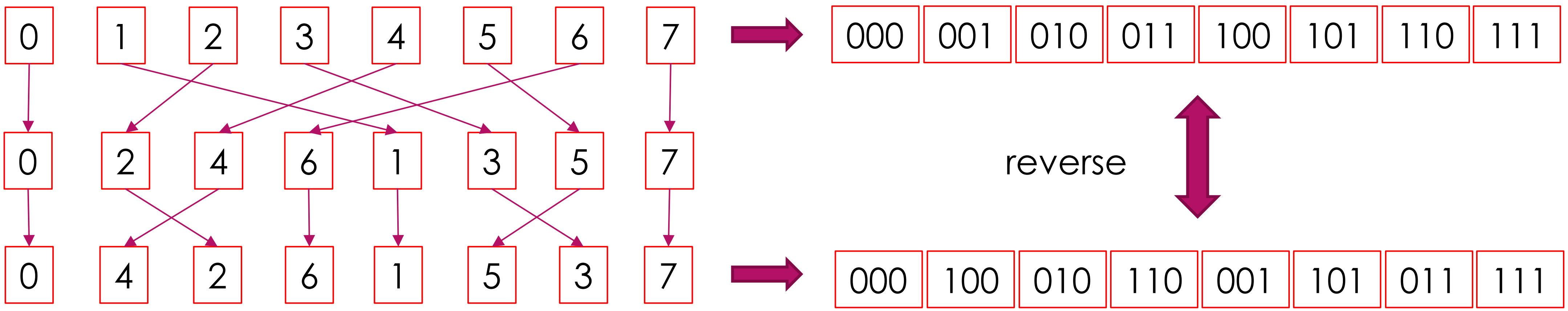
Recursive program：Stack Overflow

Observe that each time FFT takes all even index elements as a group and all odd index elements as the other group. Finally, you should find that in the bottom of the recursive, the order of the calculating which index is **the bit-reversed order** vs the original order.

That means that if you **switch the elements first**, FFT can be operated from the bottom to top **using loop**, without recursion, to improve efficiency.



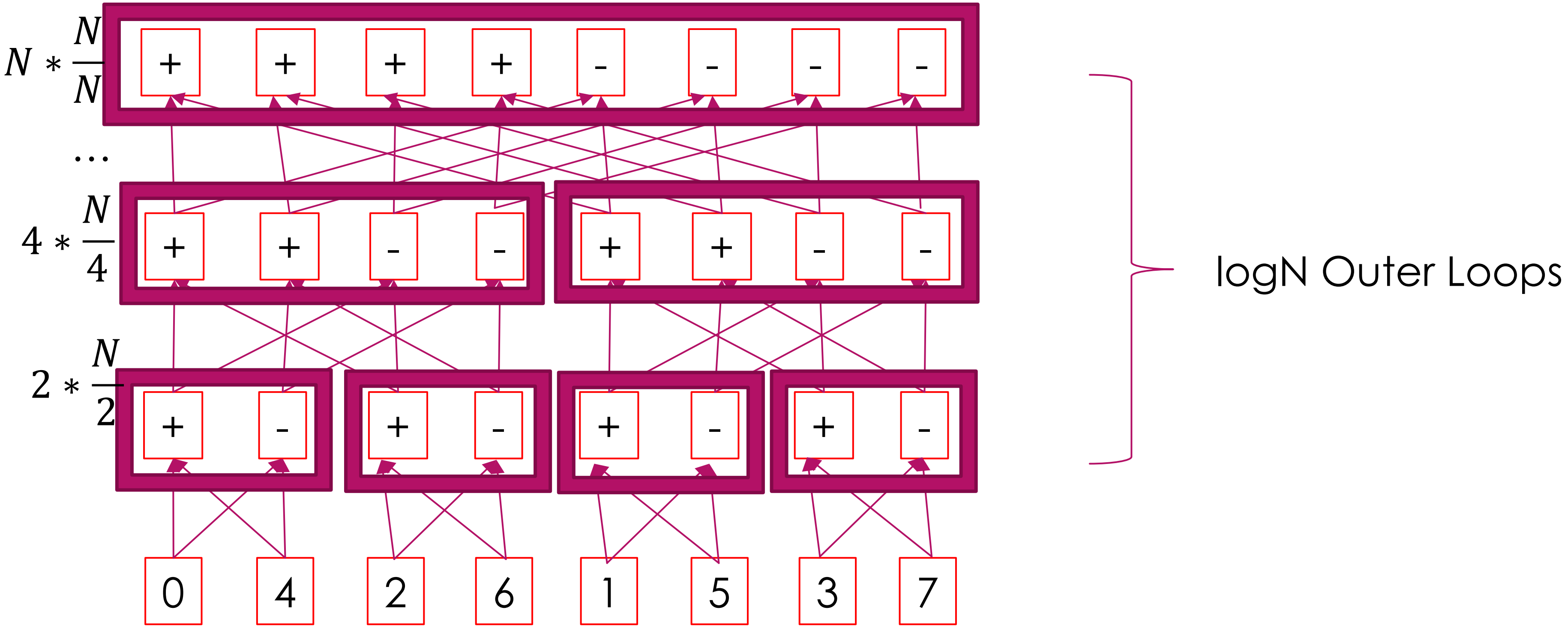
How to implements FFT with loop: Bit-reversed order



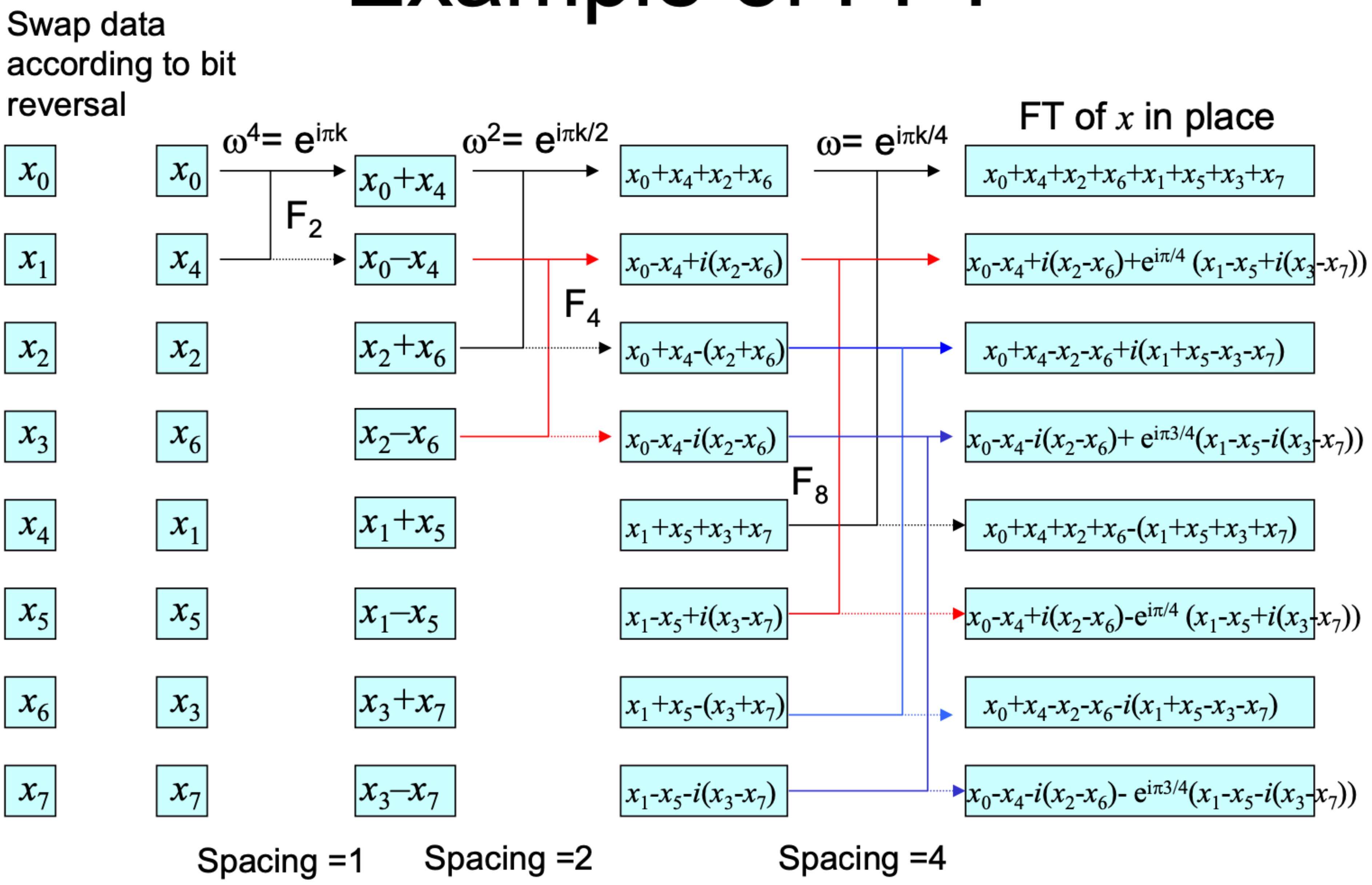
Observe:

	The lowest bit is 1	2 =1*2=1<<1	3=2+1,2 is even	4 =2*2=2<<1
000	001	010	011	100
	The highest bit is 1	Reverse: a[2]=a[1]>>1	a[2] and then set the highest bit 1	Reverse: a[4]=a[2]>>1
000	100	010	110	001

Loop: from the bottom to top



Example of FFT



Applications of FFT

- Signal processing
- Convolution

Signal processing

Time domain to frequency domain

Suppose a sinusoidal signal is sampled, its frequency is 10Hz, the sampling frequency Fs is 40Hz, the number of sampling points is N = 4, and then FFT is performed on it

As shown in the figure on the right, the values of the four points sampled are:
0 1 0 -1

The physical significance of the results of the FFT

$x_0 = 0$
$x_1 = 1$
$x_2 = 0$
$x_3 = -1$

$\xrightarrow{\text{FFT}}$

$X_0 = 0$
$X_1 = -2i$
$X_2 = 0$
$X_3 = 2i$

Each index of X represents a frequency : F(n)=n*Fs/N (Fs represents the sampling frequency, in this case, the 40 hz; N is the number of sampling points, which is 4 in this case.)

If X_i = a+bi, which modulus $A_i = \sqrt{a^2 + b^2}$. For the signal of i =0, it is the **dc signal**, which amplitude is $\boldsymbol{A_0/N}$, and the amplitude of other points is $\mathbf{2 * \boldsymbol{A_i/N}}$.

With the symmetry of FFT result, usually **we only use the first half (N / 2) results**, which are the results less than half the sampling frequency, so we only analyze X_0 and X_1.

$\boldsymbol{X_0 = 0}$, means **that the amplitude of frequency 0 is 0**.

$\boldsymbol{X_1 = -2i}$, so **F1 = 1*40Hz/4 = 10 Hz**, A1 = $\sqrt{0^2 + (-2)^2} = 2$, **amplitude = 2*2/4 = 1**.

According to the analysis of the DFT results, we know that the sampled signal is a sinusoidal wave with a frequency of 10hz and an amplitude of 1.

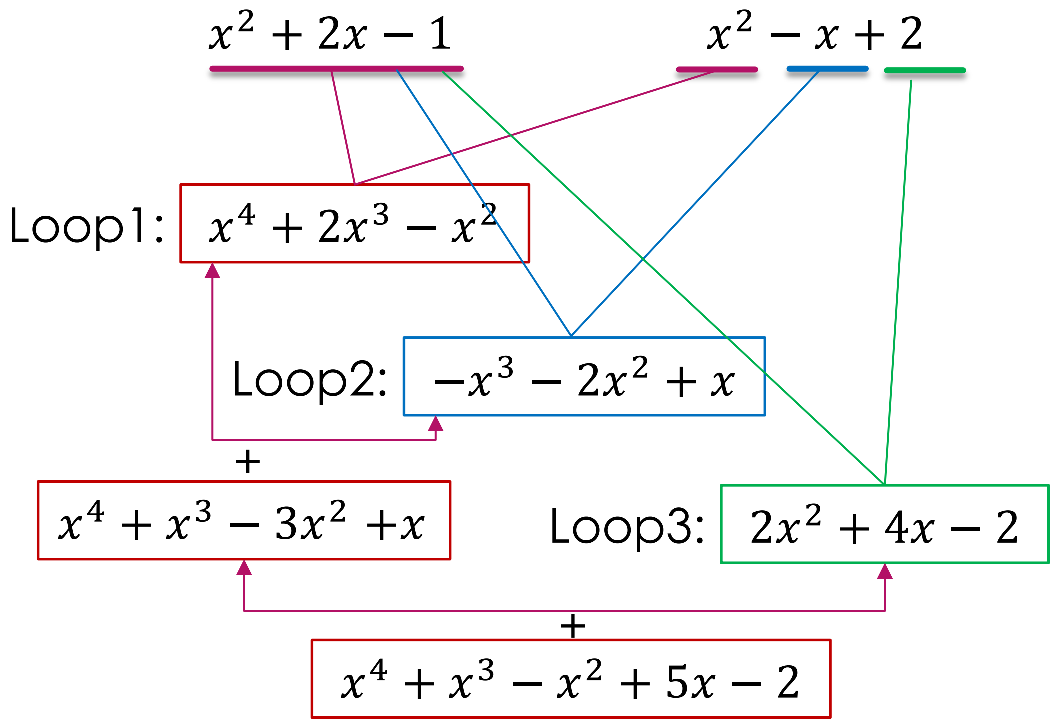
Convolution

Given an example of multiplying Polynomials:

$$A(x) = x^2 + 2x - 1$$
$$B(x) = x^2 - x + 2$$
$$C(x) = A(x)*B(x)$$

What are the coefficients of $C(x)$?

You might have used an algorithm like this before:



Multipoint evaluation of a polynomial

The multipoint evaluation problem is the task of evaluating f at n distinct points $u_0, \ldots, u_{n-1}$.

$A(x) = x^2 + 2x - 1$

$B(x) = x^2 - x + 2$

x	$A(x)$	$B(x)$
0	-1	2
1	2	2
2	7	4
3	14	8
4	23	14

Polynomial interpolation

Polynomial interpolation is, in a way, the converse task of multipoint evaluation: Given a set of n tuples $(u_0, v_0), \ldots, (u_{n-1}, v_{n-1})$, where u_i , v_i belong to a field F and u_is are distinct, find the unique polynomial f of degree less than n such that f(u_i) = v_i for all $0 \le i < n$.

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

x	$A(x)$	$B(x)$	$C(x)$
0	-1	2	-1*2=2
1	2	2	2*2=4
2	7	4	7*4=28
3	14	8	14*8=112
4	23	14	23*14=322

(0,−2),(1,4),(2,28),(3,112),(4,322) are points on $C(x)$

Finding the coefficients of $C(x)$ can be converted to the problem of finding the coefficient of the following equation:

$$\left.\begin{aligned}
-2 &= c_4 0^4 + c_3 0^3 + c_2\, 0^2 + c_1 0 + c_0 \\
4 &= c_4 1^4 + c_3 1^3 + c_2\, 1^2 + c_1 1 + c_0 \\
28 &= c_4 2^4 + c_3 2^3 + c_2\, 2^2 + c_1 2 + c_0 \\
112 &= c_4 3^4 + c_3 3^3 + c_2\, 3^2 + c_1 3 + c_0 \\
322 &= c_4 4^4 + c_3 4^3 + c_2\, 4^2 + c_1 4 + c_0
\end{aligned}\right\} \Longleftarrow$$

$\boxed{C(x) = c_4 x^4 + c_3 x^3 + c_2\, x^2 + c_1 x + c_0}$

$\boxed{(0,-2),(1,4),(2,28),(3,112),(4,322) \text{ are points on } C(x)}$

Brute-force Algorithm: Gaussian elimination($\boldsymbol{O(n^3)}$).

Let's look at a special set of values for x

Review the DFT formula:

$x_0 = 0$
$x_1 = 1$
$x_2 = 0$
$x_3 = -1$

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn}$$

$$X_0 = x_0 * e^{-\frac{2\pi i}{4}*0*0} + x_1 * e^{-\frac{2\pi i}{4}*0*1} + x_2 * e^{-\frac{2\pi i}{4}*0*2} + x_3 * e^{-\frac{2\pi i}{4}*0*3}$$

$$X_1 = x_0 * e^{-\frac{2\pi i}{4}*1*0} + x_1 * e^{-\frac{2\pi i}{4}*1*1} + x_2 * e^{-\frac{2\pi i}{4}*1*2} + x_3 * e^{-\frac{2\pi i}{4}*1*3}$$

$$X_2 = x_0 * e^{-\frac{2\pi i}{4}*2*0} + x_1 * e^{-\frac{2\pi i}{4}*2*1} + x_2 * e^{-\frac{2\pi i}{4}*2*2} + x_3 * e^{-\frac{2\pi i}{4}*2*3}$$

$$X_3 = x_0 * e^{-\frac{2\pi i}{4}*3*0} + x_1 * e^{-\frac{2\pi i}{4}*3*1} + x_2 * e^{-\frac{2\pi i}{4}*3*2} + x_3 * e^{-\frac{2\pi i}{4}*3*3}$$

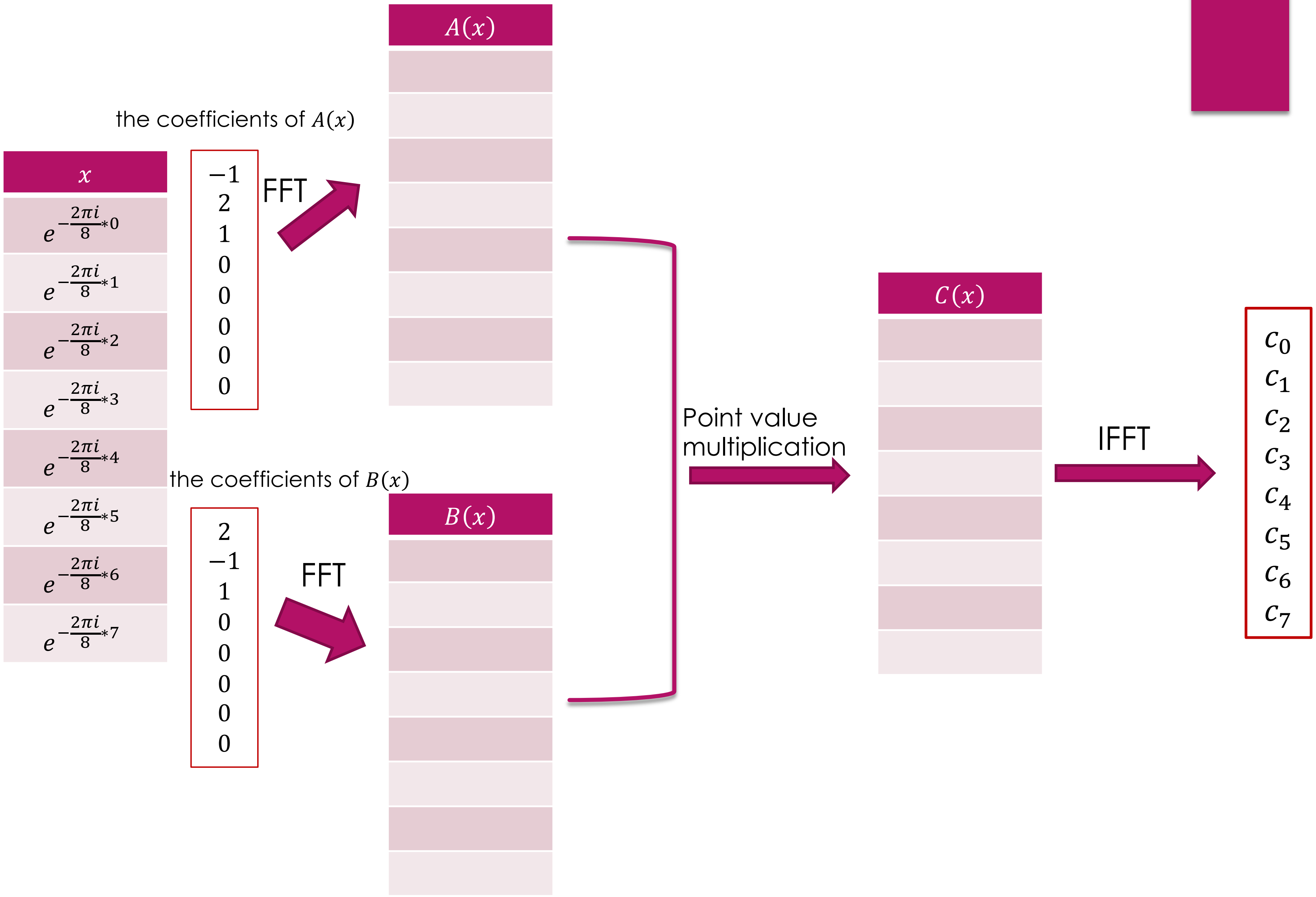
Think of $\boldsymbol{x_0, \quad x_1, \quad \ldots\ldots x_{n-1} as\ coefficients, \ e^{-\frac{2\pi i}{n}*0}, \ e^{-\frac{2\pi i}{n}*1}, \ \ldots\ldots e^{-\frac{2\pi i}{n}*(n-1)}}$ **are the** $\boldsymbol{n\ variables}$**.**
The formula for the DFT is actually a polynomial with a special values of $\boldsymbol{x}$**.**

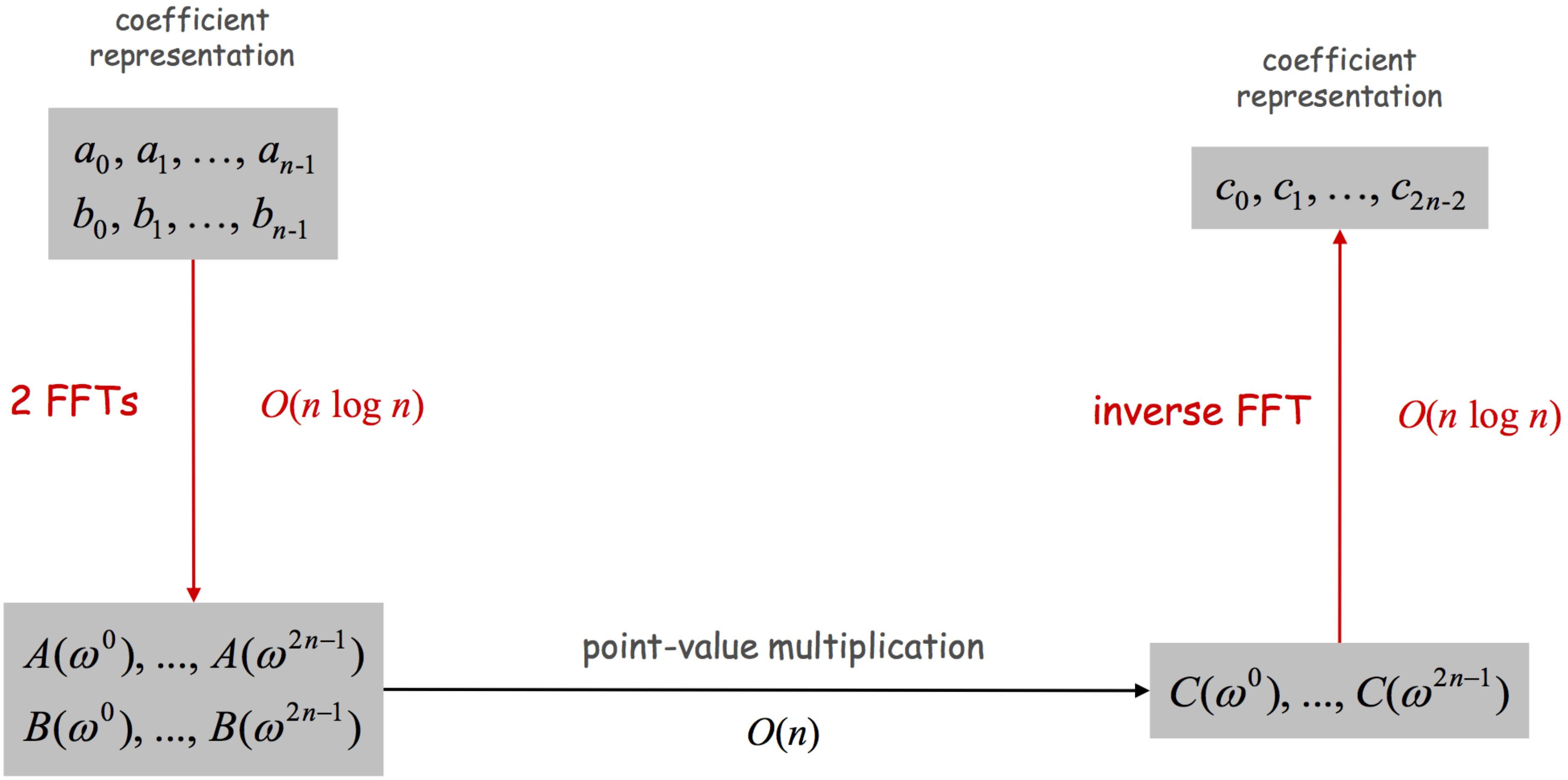
$$A(x) = x^2 + 2x - 1$$

$$B(x) = x^2 - x + 2$$

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

The degree of $C(x)$ is 4.
The elements number of FFT should be $n = 2^k$.
So we need 8 points.




coefficient representation
$a_0, a_1, \ldots, a_{n\text{-}1}$
$b_0, b_1, \ldots, b_{n\text{-}1}$
2 FFTs
$O(n \log n)$
coefficient representation
$c_0, c_1, \ldots, c_{2n\text{-}2}$
inverse FFT
$O(n \log n)$
$A(\omega^0), \ldots, A(\omega^{2n-1})$
$B(\omega^0), \ldots, B(\omega^{2n-1})$
point-value multiplication
$O(n)$
$C(\omega^0), \ldots, C(\omega^{2n-1})$

Application of Convolution: Pattern matching

Given the pattern string A of length M and the text string B of length N, find all the places where A appears in B.

If B[i],B[i+1]……B[i+m-1]equal A[0], A[1]……A[m-1]

$$dis_i(A,B) = \sum_{j=0}^{m-1} (B[i+j] - A[j])^2$$

If $dis_i(A,B) == 0$, A appears B[i]

Let A' = Reverse(A)

$$dis_i(A,B) = \sum_{j=0}^{m-1} (B[i+j] - A'[m-1-j])^2$$

$$= \sum_{j=0}^{m-1} (B[i+j])^2 + \sum_{j=0}^{m-1} (A'[m-1-j])^2 + \sum_{j=0}^{m-1} 2 * B[i+j] * A'[m-1-j]$$

Observe $i + j + \mathrm{m} - 1 - \mathrm{j} = \mathrm{i} + \mathrm{m} - 1$, $so\ we\ can\ calculate\ the\ convolution\ of\ B\ and\ A'$
$If\ C[k] == \sum_{j=0}^{m-1}[\,(B[i+j])^2 + (A'[m-1-j])^2]/2$ ($k = \mathrm{i} + \mathrm{m} - 1$), It means that the m characters of B starting from index i are the same as A.

Pattern Matching Example

B=AABCDEEAAA n=10

A=ABCDE m=5

Let $BSqrSum[i] = \sum_{j=0}^{m-1}(B[i+j])^2$

$BSqrSum[\ \ ]$ =[21919, 22455, 22991, 22860, 22596, 22197]

Let $ASqrSum = \sum_{j=0}^{m-1} A'[m-1-j])^2$

$ASqrSum$ =22455

Let $SqrSum[i] = \sum_{j=0}^{m-1}[\,(S[i+j])^2 + (T'[m-1-j])^2]/2$

$SqrSum[\ \ ]$=[22187, 22455, 22723, 22657, 22525, 22326]

$C[\ ]$=[4485, 8905, 13329, 17756, 22185, 22455, 22721, 22643, 22502, 22299, 17550, 12870, 8515, 4225, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Compare $SqrSum[\mathrm{i}]$ and $C[i+m-1]$, find $SqrSum[1] == C[1+5-1]$, from B[1] can match A.

If $B[i]$ can match A, $\sum_{j=0}^{m-1}(B[i+j])^2 = \sum_{j=0}^{m-1} A'[m-1-j])^2$,

$\sum_{j=0}^{m-1}[\,(B[i+j])^2 + (A'[m-1-j])^2]/2 = 2\sum_{j=0}^{m-1} A'[m-1-j])^2/2 = \sum_{j=0}^{m-1} A'[m-1-j])^2$

So we only need to calculate A$SqrSum$, then see *which* $C[k] ==$ A$SqrSum$, the start place i = *k+1-m.*

Recommend the video:

https://www.bilibili.com/video/av19141078/