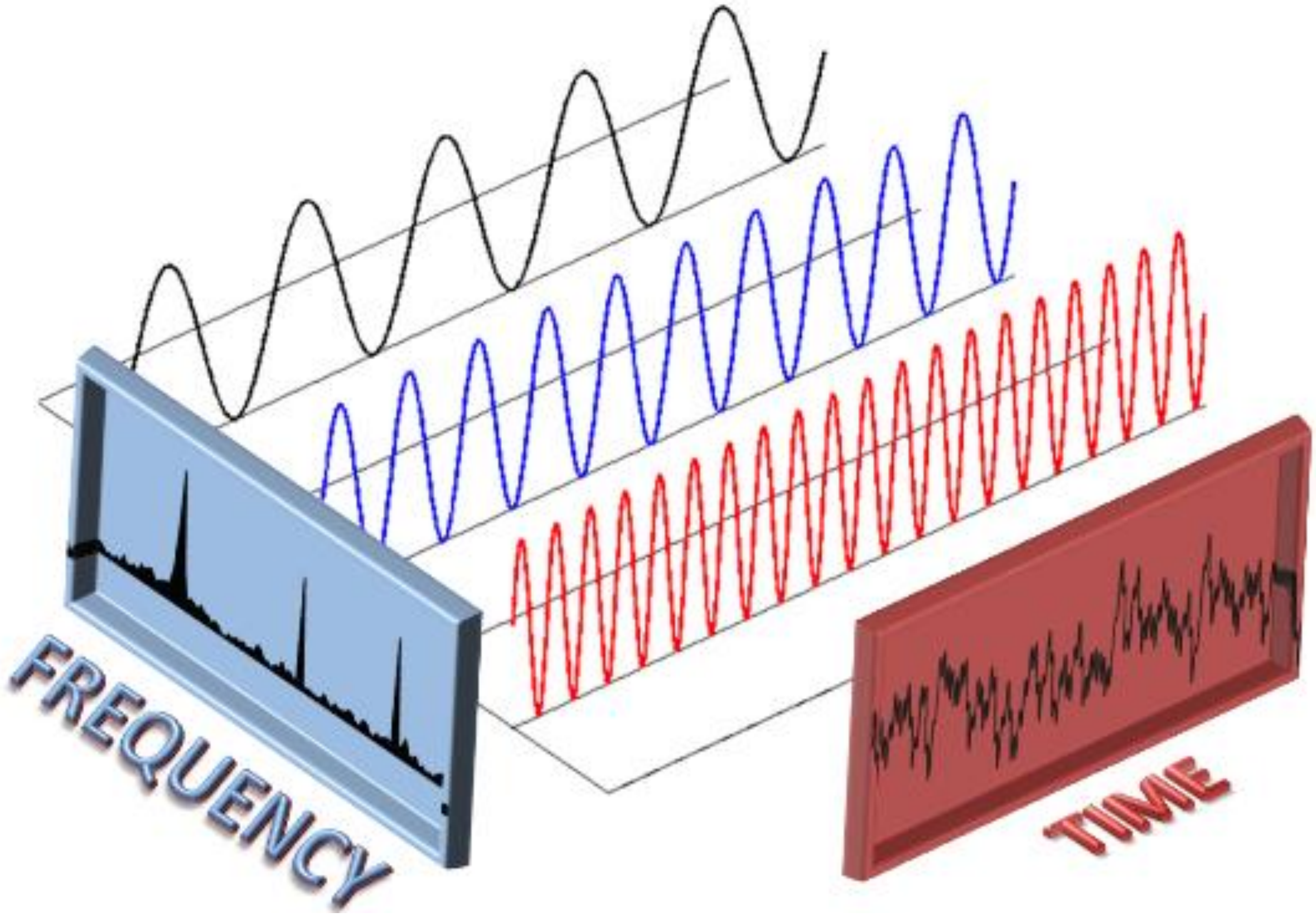
Fast Fourier Transform

Jordi Cortadella and Jordi Petit
Department of Computer Science

Why Fourier Transform?



Fourier series

- Periodic function $f(t)$ of period 1:

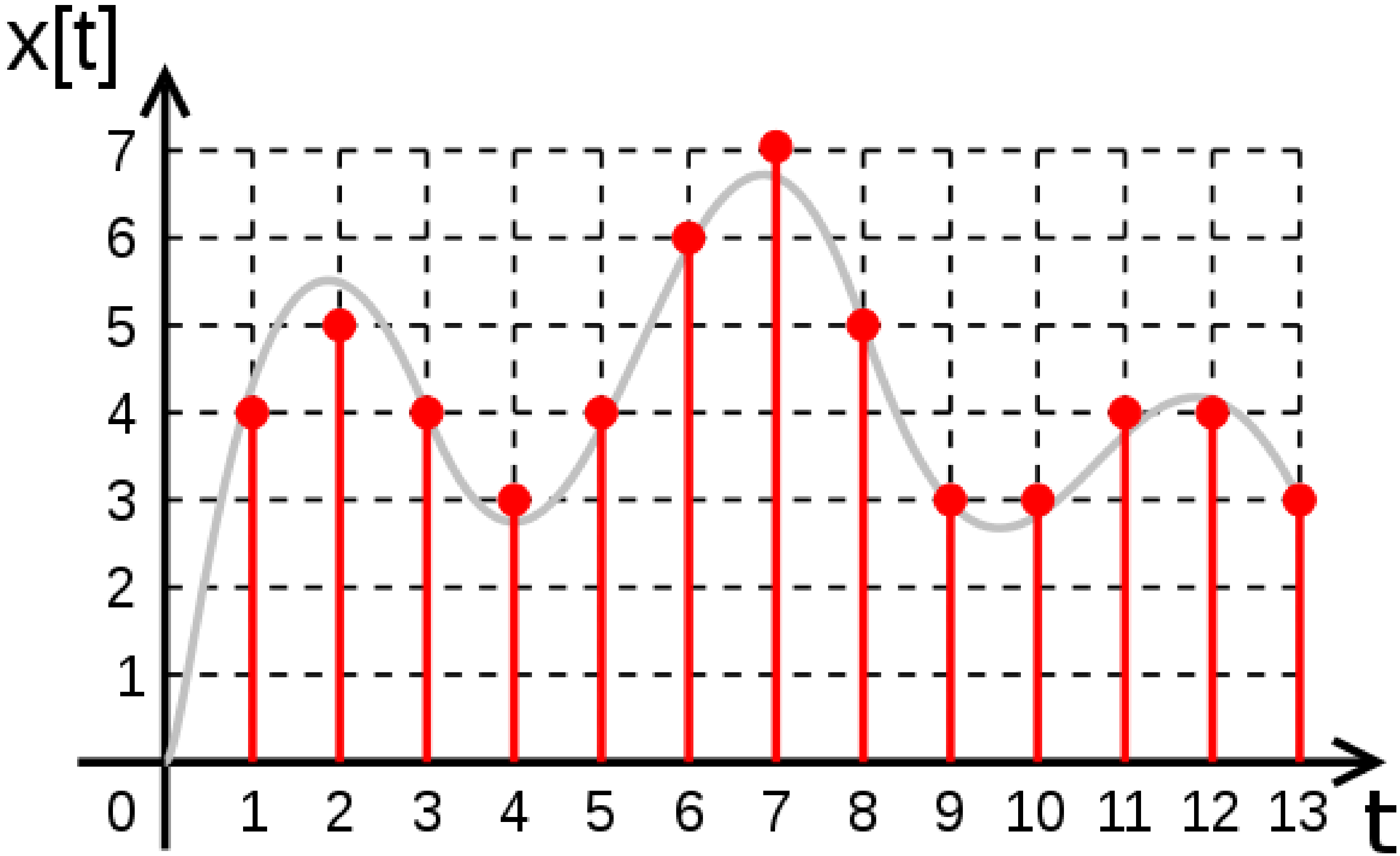
$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(2\pi n t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n t)$$

- Fourier coefficients:

$$a_n = 2\int_0^T f(t)\cos(2\pi n t)\,dt, \qquad b_n = 2\int_0^T f(x)\sin(2\pi n t)\,dt$$

- Fourier series is fundamental for signal analysis (to move from time domain to frequency domain, and vice versa)

Discrete-time signals



Polynomial representation

$$P(x) = x^3 - 2x^2 - 3x + 1$$

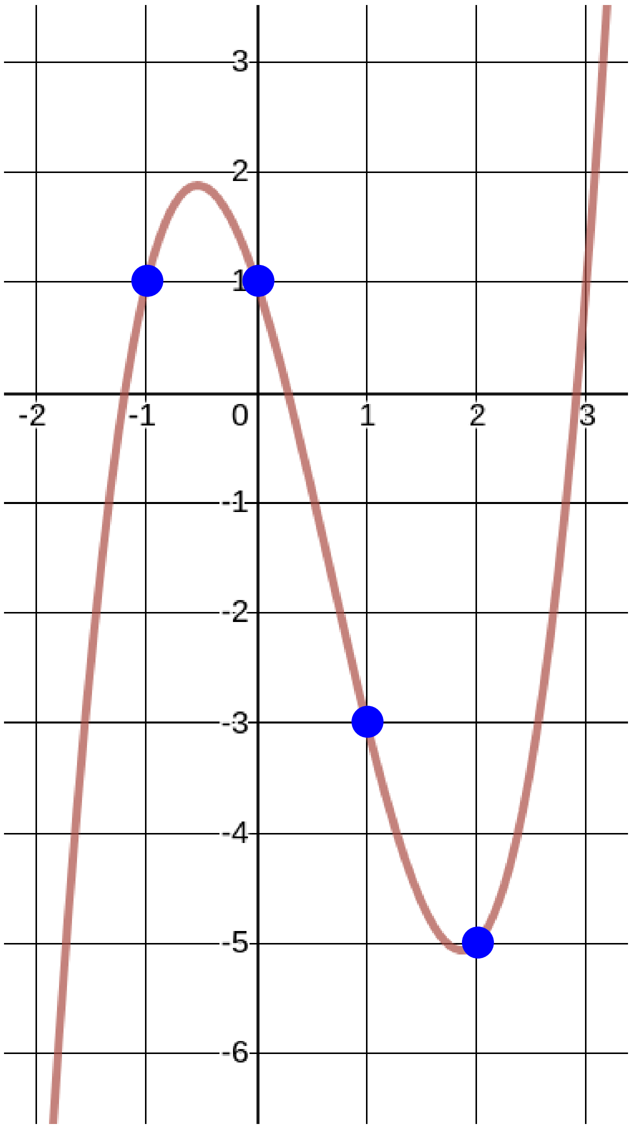
$$P(x) = (1, -2, -3, 1)$$

Coefficient representation

Evaluation ↓ ↑ **Interpolation**

Point-value representation

$$P(x) = \{(-1,1), (0,1), (1,-3), (2,-5)\}$$



Polynomials: coefficient representation

- A polynomial is represented as a vector of coefficients $(a_0, a_1, \ldots, a_{n-1})$:

$$A(x) = 2x^4 + x^2 - 4x + 3$$

$$A = (3, -4, 1, 0, 2)$$

- Addition: $\mathrm{O}(n)$

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

- Evaluation: $\mathrm{O}(n)$ using Horner's method

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

- Multiplication: $\mathrm{O}(n^2)$ using brute force

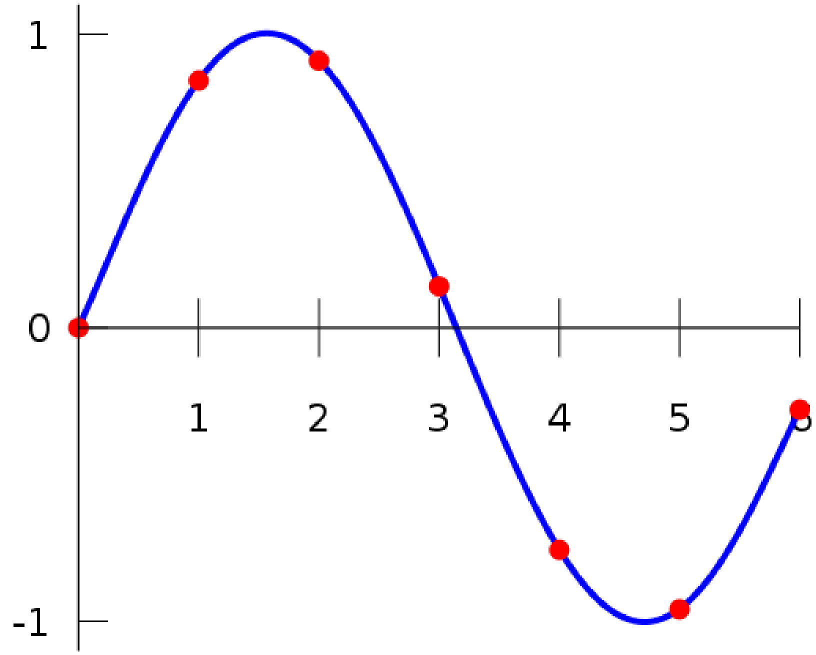
$$A(x) \cdot B(x) = \sum_{i=0}^{2n-2} c_i x^i, \quad \text{where} \;\; c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

© Dept. CS, UPC

Polynomials: point-value representation

- **Fundamental Theorem (Gauss)**: A degree-n polynomial with complex coefficients has exactly n complex roots.

- **Corollary**: A degree-n polynomial $A(x)$ is uniquely identified by its evaluation at $n+1$ distinct values of x.



Polynomials: point-value representation

- A polynomial is represented as a set of pairs (x_i, y_i):

$$A(x) = \{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$$
$$B(x) = \{(x_0, z_0), \ldots, (x_{n-1}, z_{n-1})\}$$

- Addition: $\mathrm{O}(n)$

$$A(x) + B(x) = \{(x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})\}$$

- Multiplication: $\mathrm{O}(n)$, but with $2n-1$ points

$$A(x) \cdot B(x) = \{(x_0, y_0 \cdot z_0), \ldots, (x_{n-1}, y_{n-1} \cdot z_{n-1})\}$$

- Interpolation: $\mathrm{O}(n^2)$ using Lagrange's formula

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

© Dept. CS, UPC

Interpolation: Lagrange polynomials

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

$$\mathrm{A}(x) = \{(-1,-1), (0,-2), (2,2)\}$$

$$A(x) = -1\frac{(x-0)(x-2)}{(-1-0)(-1-2)} - 2\frac{(x+1)(x-2)}{(0+1)(0-2)} + 2\frac{(x+1)(x-0)}{(2+1)(2-0)}$$

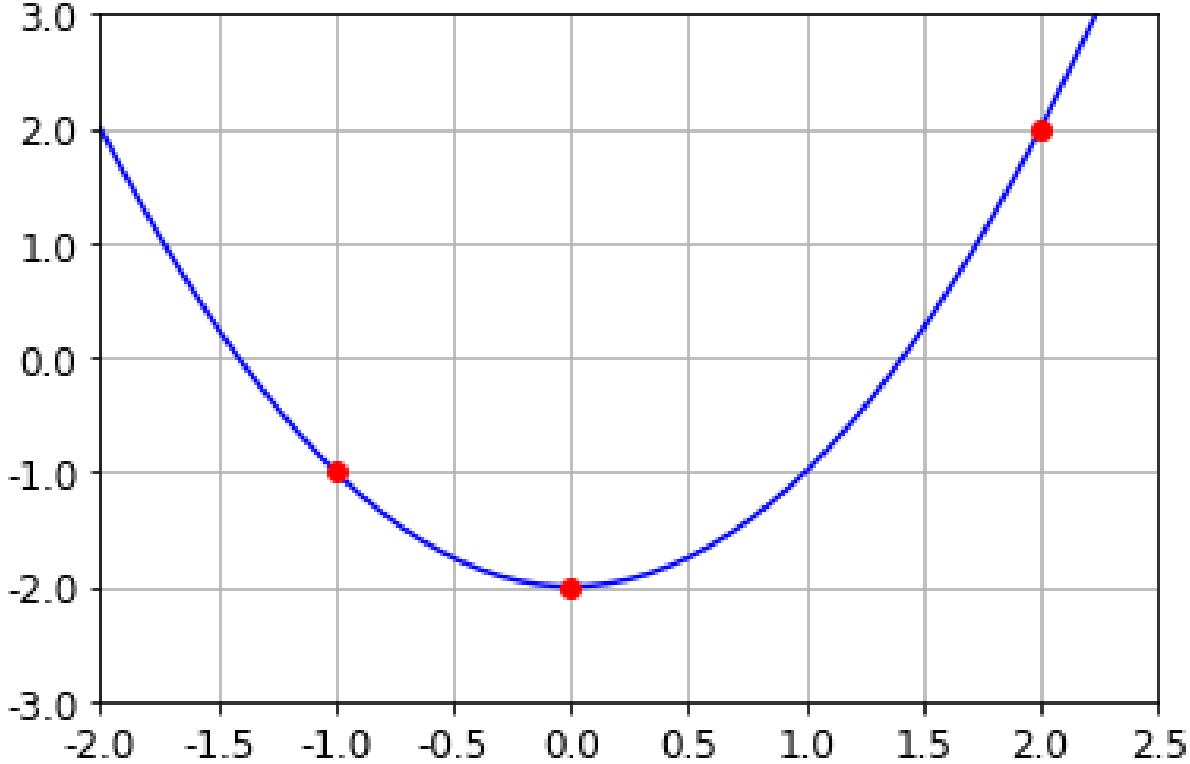
$$A(x) = -\frac{x(x-2)}{3} + (x+1)(x-2) + \frac{(x+1)x}{3}$$

$$\boxed{A(x) = x^2 - 2}$$

Interpolation: Lagrange polynomials

$$\mathrm{A}(x) = \{(-1,-1),(0,-2),(2,2)\}$$

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



Conversion between both representations

representation	addition	multiplication	evaluation
coefficient	$O(n)$	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n)$	$O(n^2)$

$$\begin{tikzcd} a_0, a_1, \cdots, a_{n-1} \arrow[r, shift left, "\text{evaluation}"] & (x_0, y_0), \cdots, (x_{n-1}, y_{n-1}) \arrow[l, shift left, "\text{interpolation}"] \end{tikzcd}$$

Coefficient representation | Point-value representation

Could we have an ***efficient*** algorithm to move from coefficient to point-value representation and vice versa?

© Dept. CS, UPC

From coefficients to point-values

Given a polynomial $a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$, evaluate it at n different points $x_0, \ldots, x_{n-1}$:

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

Runtime: $\mathrm{O}(n^2)$ matrix-vector multiplication (apply Horner n times).

Horner's rule:

$$p(x) = a_0 + x\Big(a_1 + x\big(a_2 + x(a_3 + \cdots + x(a_{n-1} + x a_n)\cdots)\big)\Big)$$

Evaluation by divide-and-conquer

- Credits: based on the intuitive explanation by Dasgupta, Papadimitriou and Vazinari, *Algorithms*, McGraw-Hill, 2008.
- We want to evaluate $A(x)$ at n different points. Let us choose them to be positive-negative pairs: $\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$
- The computations for $A(x_i)$ and $A(-x_i)$ overlap a lot.
- Split the polynomial into odd and even powers

$$3 + 4x + 6x^2 + 2x^3 + x^4 + 10x^5 = (3 + 6x^2 + x^4) + x(4 + 2x^2 + 10x^4)$$

- The terms in parenthesis are polynomials in x^2:

$$A(x) = A_e(x^2) + xA_o(x^2)$$

© Dept. CS, UPC

Evaluation by divide-and-conquer

- The calculations needed for $A(x_i)$ can be reused for computing $A(-x_i)$.

$$A(x_i) = A_e(x_i^2) + x_i A_o(x_i^2)$$
$$A(-x_i) = A_e(x_i^2) - x_i A_o(x_i^2)$$

- Evaluating $A(x)$ at n paired points

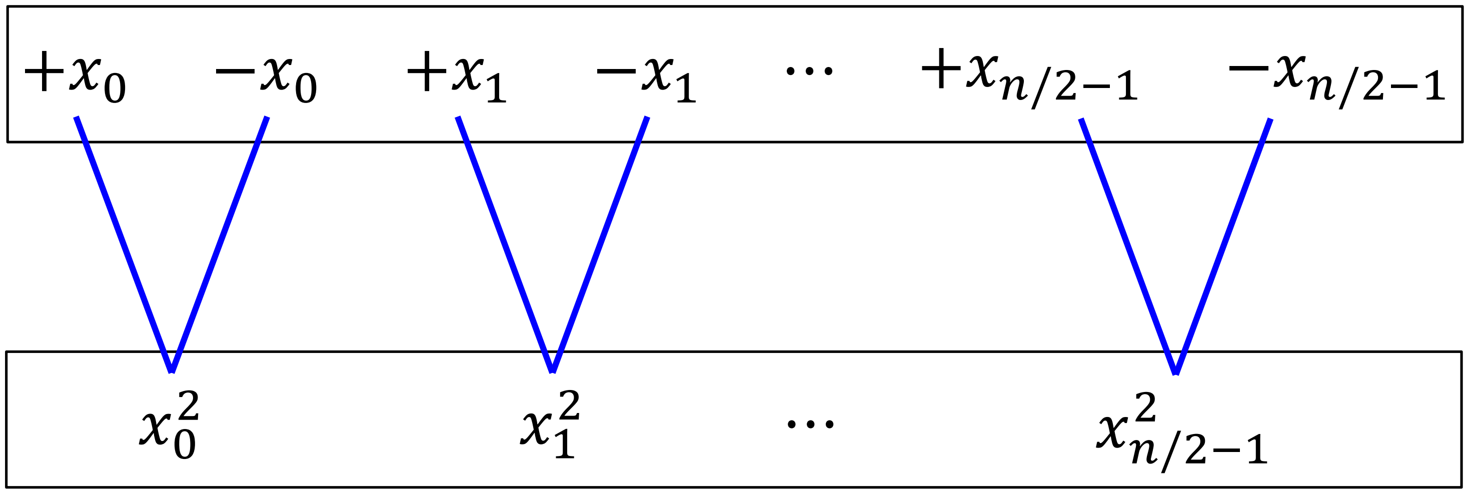
$$\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$$

reduces to evaluating $A_e(x)$ and $A_o(x)$ at just $n/2$ points: $x_0^2, \cdots, x_{n/2-1}^2$

© Dept. CS, UPC

Evaluation by divide-and-conquer

Evaluate: $A(x)$
degree $\leq n-1$



Evaluate:
$A_e(x^2)$ and $A_o(x^2)$
degree $\leq n/2-1$

If we could recurse, we would get a running time:

$$T(n) = 2 \cdot T(n/2) + O(n) = O(n \log n)$$

But can we recurse?

Evaluation by divide-and-conquer

Evaluate: $A(x)$
degree $\leq n-1$

$+x_0 \quad -x_0 \quad +x_1 \quad -x_1 \quad \cdots \quad +x_{n/2-1} \quad -x_{n/2-1}$

Evaluate:
$A_e(x^2)$ and $A_o(x^2)$
degree $\leq n/2-1$

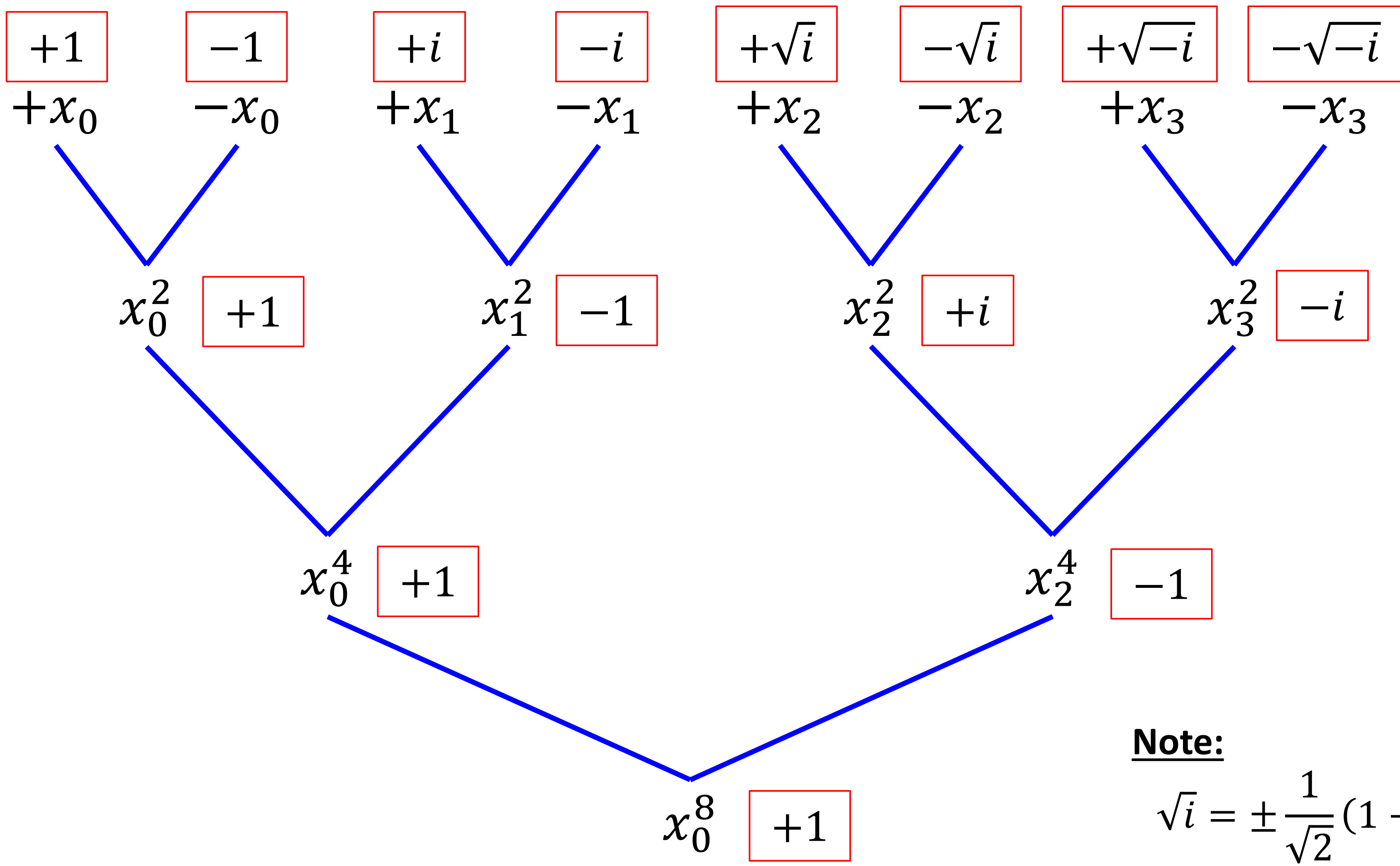
$x_0^2 \quad x_1^2 \quad \cdots \quad x_{n/2-1}^2$

The problem: **?**

We need x_0^2 and x_1^2 to be a plus-minus pair.
But a square cannot be negative !

Not if we use real numbers. How about complex numbers?

Selection of the evaluation points



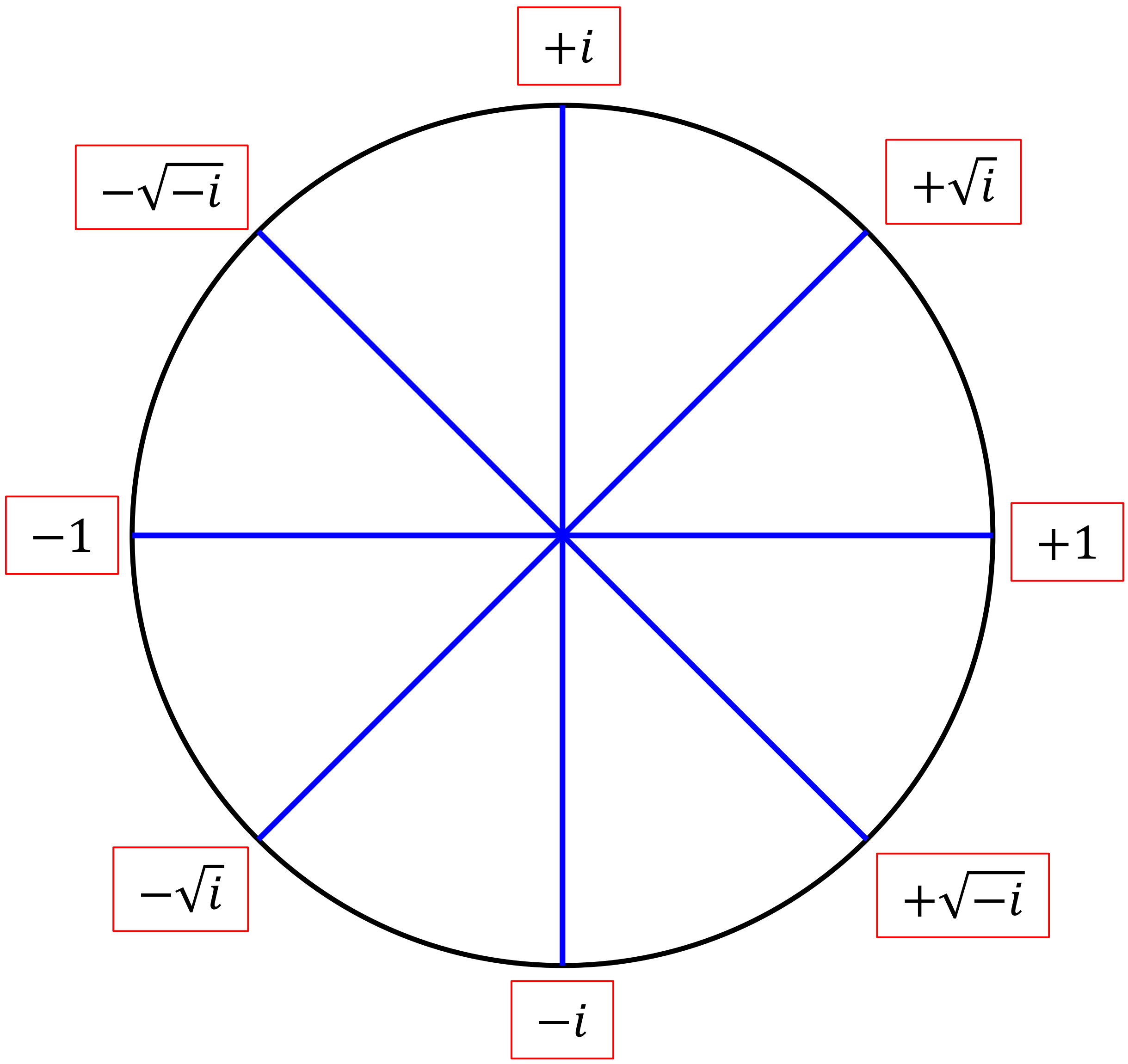
Note:

$$\sqrt{i} = \pm \frac{1}{\sqrt{2}}(1 + i)$$

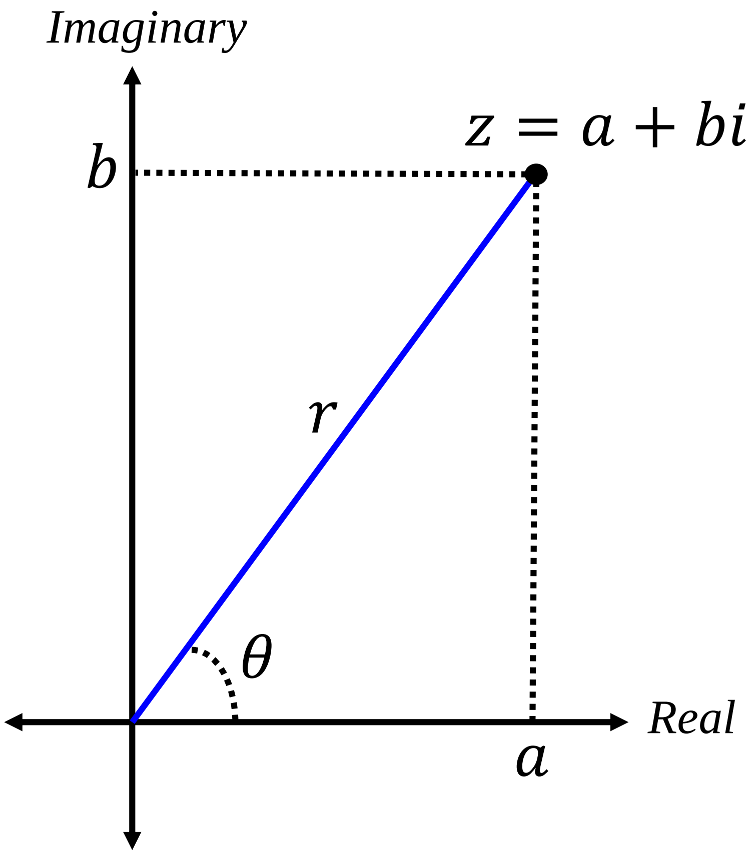
$$\sqrt{-i} = \pm \frac{1}{\sqrt{2}}(1 - i)$$

© Dept. CS, UPC

Selection of the evaluation points



Complex numbers: review



$$z = r(\cos\theta + i\sin\theta) = re^{i\theta}$$

Polar coordinates: (r, θ)

Length: $r = \sqrt{a^2 + b^2}$

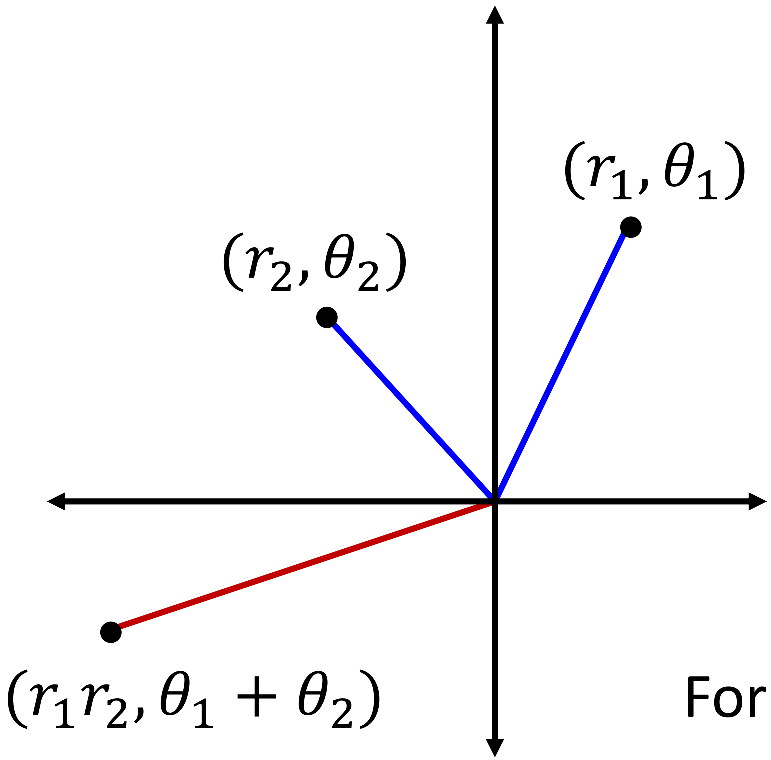
Angle $\theta \in [0,2\pi)$: $\cos\theta = \frac{a}{r}, \sin\theta = \frac{b}{r}$

θ can always be reduced modulo 2π

Some examples:

Number	-1	i	$5 + 5i$
Polar coords	$(1, \pi)$	$(1, \pi/2)$	$(5\sqrt{2}, \pi/4)$

Complex numbers: multiplication



$$(r_1, \theta_1) \times (r_2, \theta_2) = (r_1 r_2, \theta_1 + \theta_2)$$

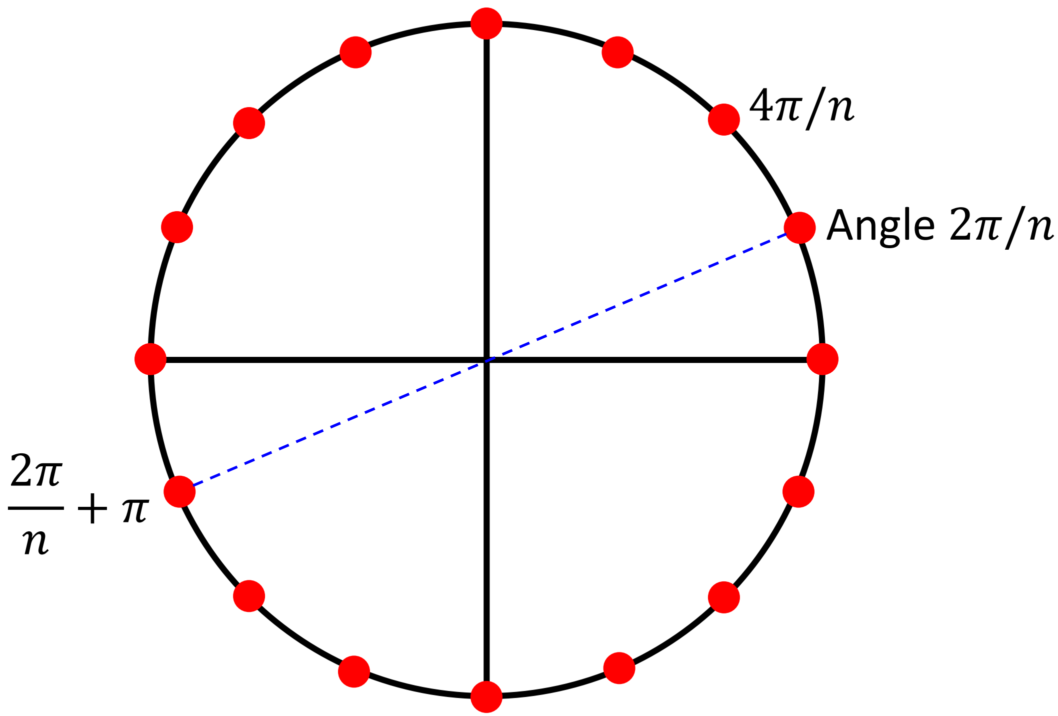
For any $z = (r, \theta)$:

$$-z = (r, \theta + \pi), \text{ since } -1 = (1, \pi)$$

If z is on the unit circle, then $z^n = (1, n\theta)$

Complex numbers: the nth roots of unity

Solutions to the equation $z^n = 1$
$(n = 16)$



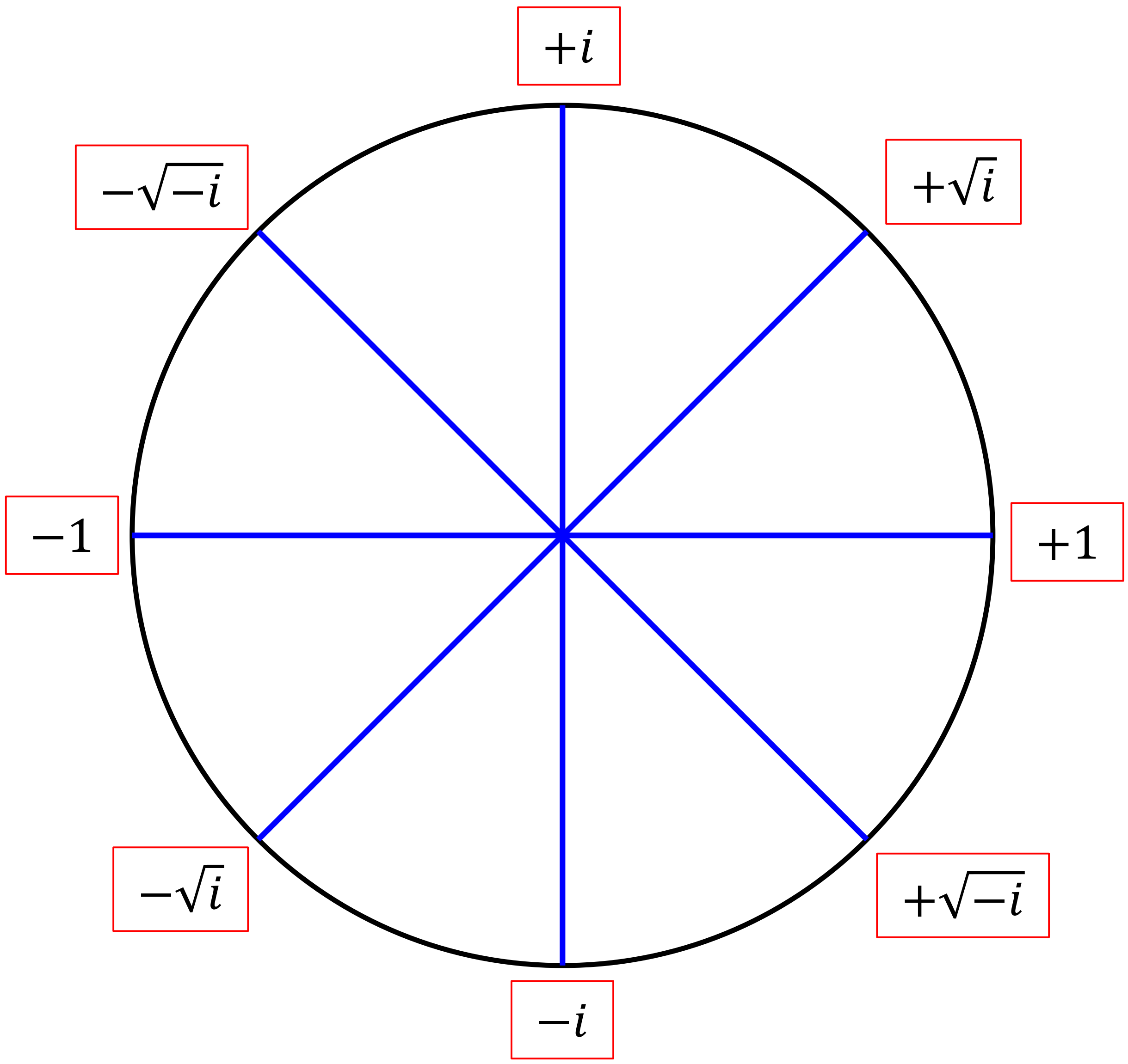
Solutions are $z = (1, \theta)$,
for θ a multiple of $2\pi/n$

All roots are plus-minus paired:

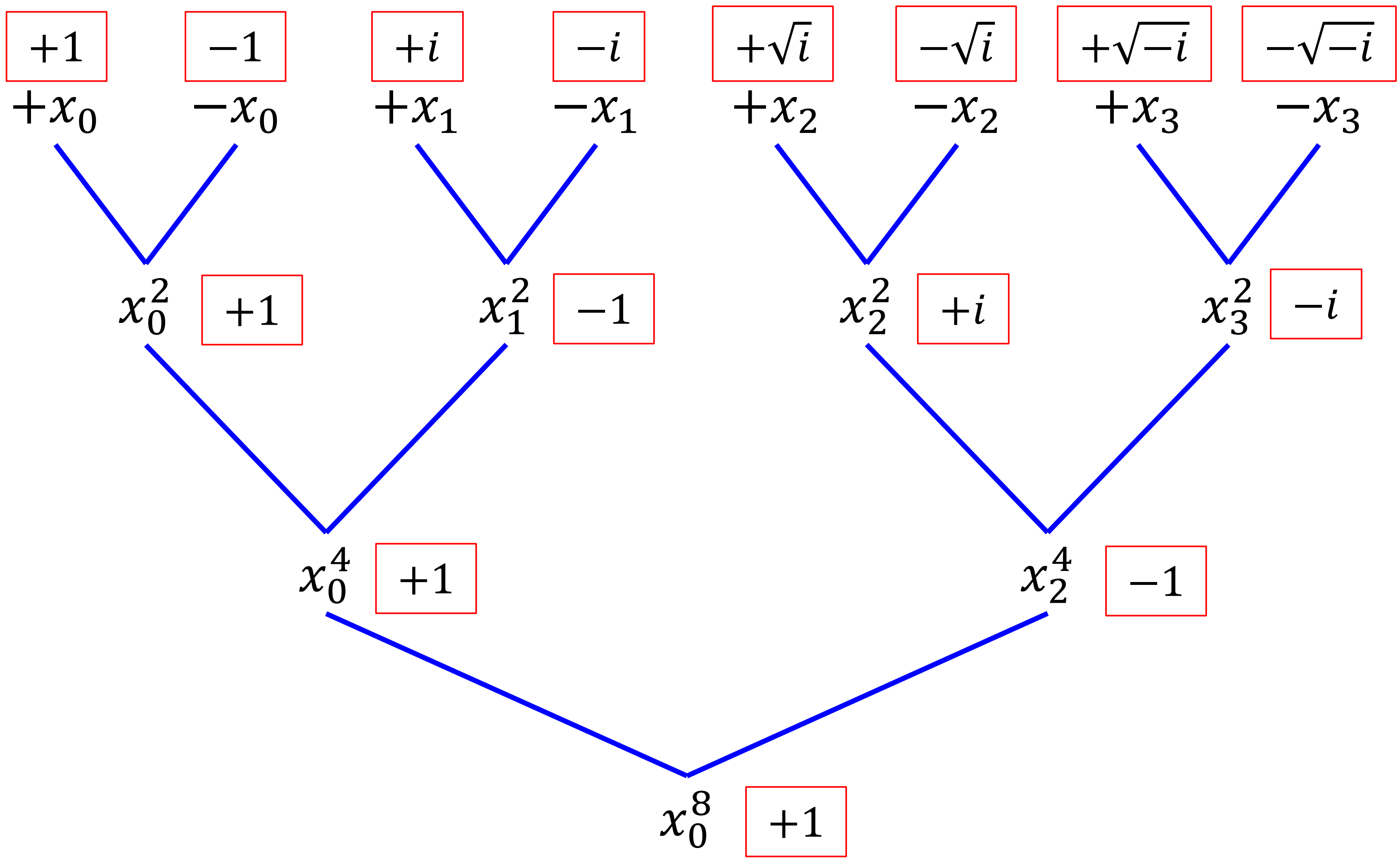
$$-(1, \theta) = (1, \theta + \pi)$$

© Dept. CS, UPC

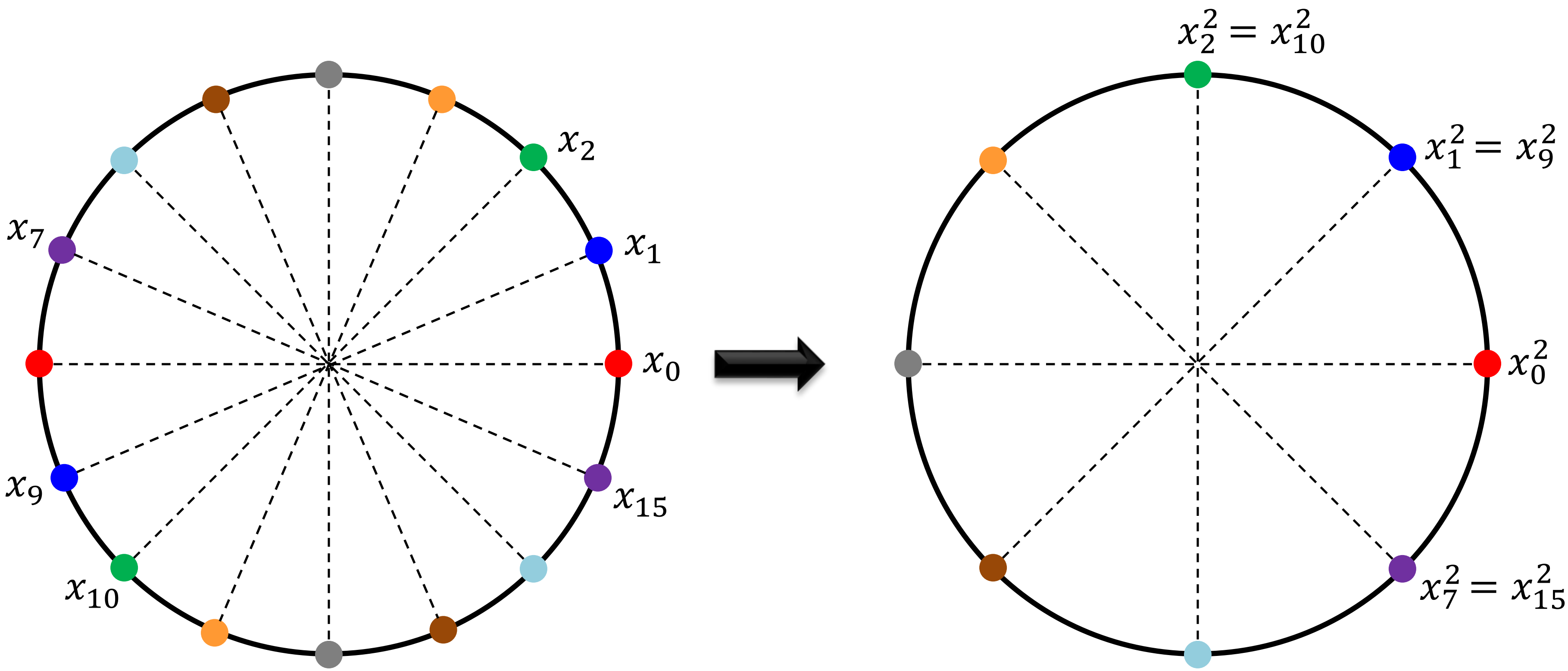
Roots of unity for $n = 8$



Recursive divide-and-conquer



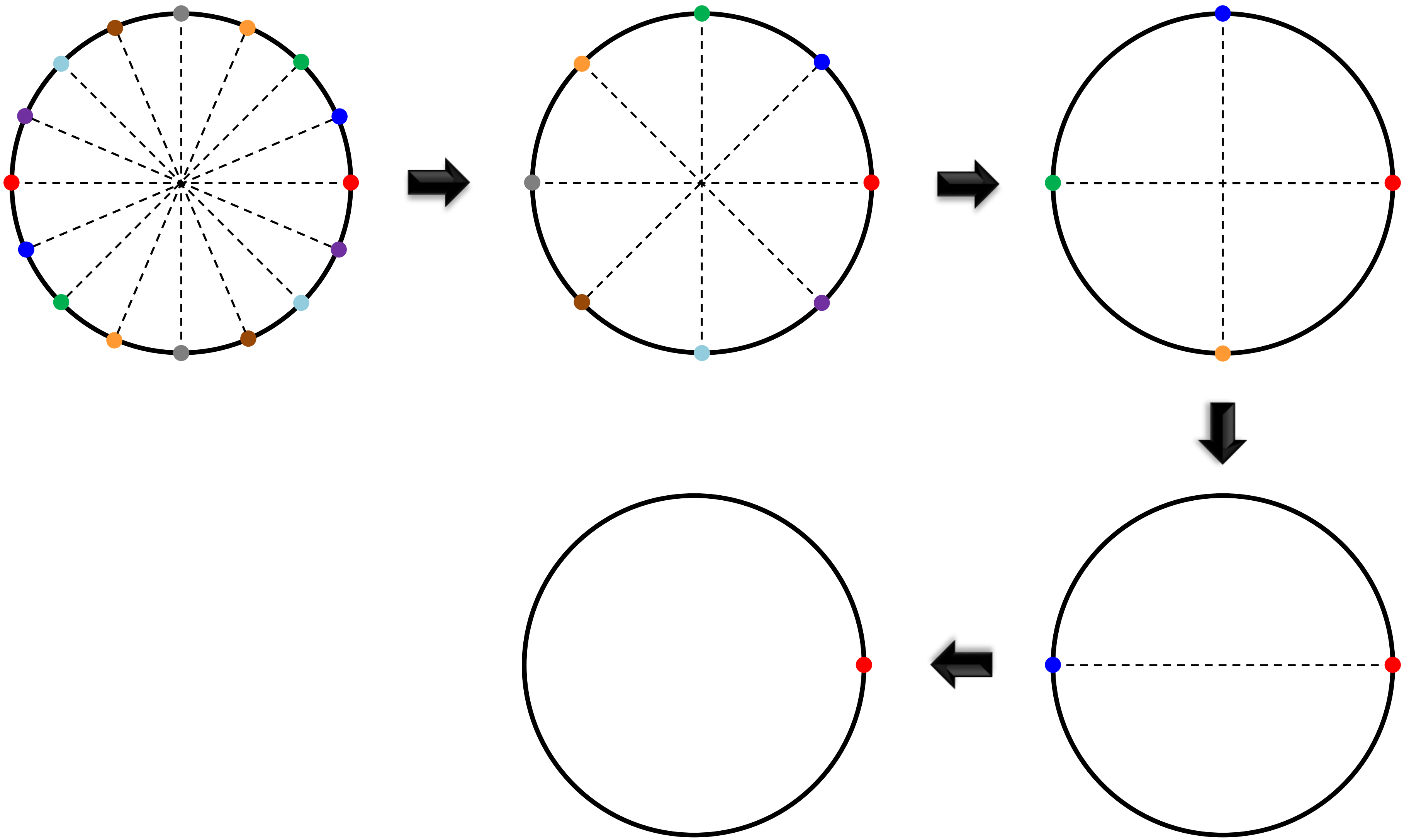
Divide-and-conquer step



Evaluate $A(x)$ at nth roots of unity

Evaluate $A_e(x^2)$ and $A_o(x^2)$ at $(n/2)$nd roots of unity

Divide-and-conquer steps



FFT algorithm

function FFT(A,$\boxed{\omega}$)

  Inputs: $A = (a_0, a_1, \dots, a_{n-1})$, for n a power of 2

  ω: A primitive nth root of unity

  Output: $\left(A(1), A(\omega), A(\omega^2), \dots, A(\omega^{n-1})\right)$

  if ω=1: **return** A // Only 1 coef. (constant)

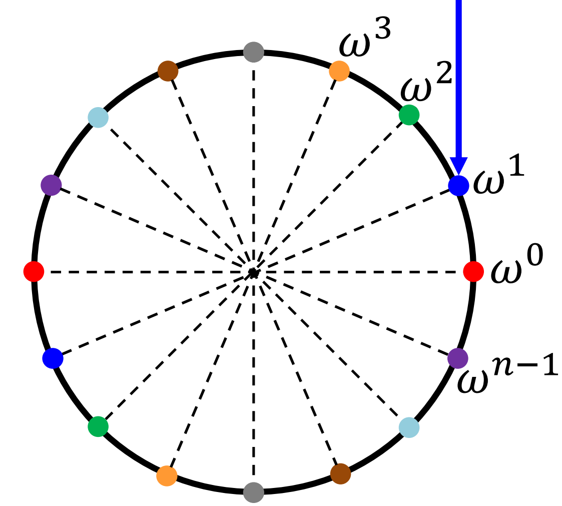
  $\left(A_e(\omega^0), A_e(\omega^2), \dots, A_e(\omega^{n-2})\right) = \text{FFT}(A_e, \omega^2)$

  $\left(A_o(\omega^0), A_o(\omega^2), \dots, A_o(\omega^{n-2})\right) = \text{FFT}(A_o, \omega^2)$

  for $k = 0$ **to** $n - 1$:

    $A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})$

  return $\left(A(1), A(\omega), A(\omega^2), \dots, A(\omega^{n-1})\right)$

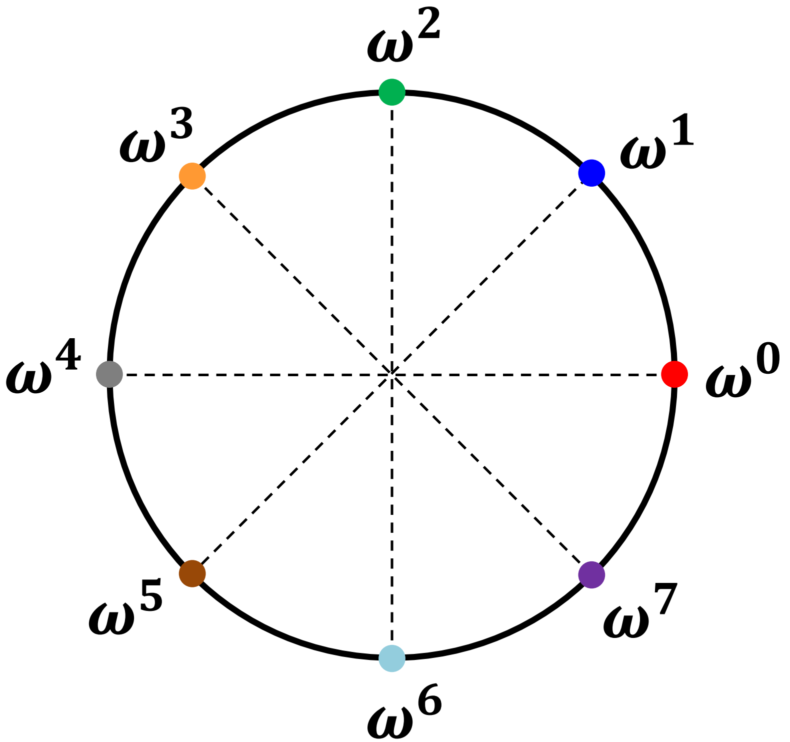


FFT algorithm

$$\boxed{\texttt{for}\ \ k = 0\ \ \texttt{to}\ \ n-1\texttt{:}\ \ A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})}$$

Example ($n = 8$):

$$A(\omega^0) = A_e(\omega^0) + \omega^0 A_o(\omega^0)$$
$$A(\omega^1) = A_e(\omega^2) + \omega^1 A_o(\omega^2)$$
$$A(\omega^2) = \boxed{A_e(\omega^4)} + \boxed{\omega^2} A_o(\omega^4)$$
$$A(\omega^3) = A_e(\omega^6) + \omega^3 A_o(\omega^6)$$
$$A(\omega^4) = A_e(\omega^8) + \omega^4 A_o(\omega^8)$$
$$A(\omega^5) = A_e(\omega^{10}) + \omega^5 A_o(\omega^{10})$$
$$A(\omega^6) = \boxed{A_e(\omega^{12})} + \boxed{\omega^6} A_o(\omega^{12})$$
$$A(\omega^7) = A_e(\omega^{14}) + \omega^7 A_o(\omega^{14})$$



$$\boxed{\omega^4 = \omega^{12}} \qquad \boxed{\omega^2 = -\omega^6}$$

FFT algorithm

$$\texttt{for } k = 0 \texttt{ to } n-1\text{:}\quad A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})$$

Example ($n = 8$):

$$\begin{aligned}
A(\omega^0) &= A_e(\omega^0) + \omega^0 A_o(\omega^0)\\
A(\omega^1) &= A_e(\omega^2) + \omega^1 A_o(\omega^2)\\
A(\omega^2) &= \boxed{A_e(\omega^4)} + \boxed{\omega^2} A_o(\omega^4)\\
A(\omega^3) &= A_e(\omega^6) + \omega^3 A_o(\omega^6)\\
A(\omega^4) &= A_e(\omega^8) + \omega^4 A_o(\omega^8)\\
A(\omega^5) &= A_e(\omega^{10}) + \omega^5 A_o(\omega^{10})\\
A(\omega^6) &= \boxed{A_e(\omega^{12})} + \boxed{\omega^6} A_o(\omega^{12})\\
A(\omega^7) &= A_e(\omega^{14}) + \omega^7 A_o(\omega^{14})
\end{aligned}$$

$$\Rightarrow$$

$$\begin{aligned}
A(\omega^0) &= A_e(\omega^0) + \omega^0 A_o(\omega^0)\\
A(\omega^1) &= A_e(\omega^2) + \omega^1 A_o(\omega^2)\\
A(\omega^2) &= A_e(\omega^4) + \omega^2 A_o(\omega^4)\\
A(\omega^3) &= A_e(\omega^6) + \omega^3 A_o(\omega^6)\\
A(\omega^4) &= A_e(\omega^0) - \omega^0 A_o(\omega^0)\\
A(\omega^5) &= A_e(\omega^2) - \omega^1 A_o(\omega^2)\\
A(\omega^6) &= A_e(\omega^4) - \omega^2 A_o(\omega^4)\\
A(\omega^7) &= A_e(\omega^6) - \omega^3 A_o(\omega^6)
\end{aligned}$$

$$\boxed{\omega^4 = \omega^{12}} \qquad \boxed{\omega^2 = -\omega^6}$$

© Dept. CS, UPC

FFT algorithm

$$\boxed{\texttt{for } k = 0 \texttt{ to } n-1\texttt{: } A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})}$$

Example ($n = 8$):

$$\begin{aligned}
A(\omega^0) &= A_e(\omega^0) + \omega^0 A_o(\omega^0) \\
A(\omega^4) &= A_e(\omega^0) - \omega^0 A_o(\omega^0) \\
A(\omega^1) &= A_e(\omega^2) + \omega^1 A_o(\omega^2) \\
A(\omega^5) &= A_e(\omega^2) - \omega^1 A_o(\omega^2) \\
A(\omega^2) &= A_e(\omega^4) + \omega^2 A_o(\omega^4) \\
A(\omega^6) &= A_e(\omega^4) - \omega^2 A_o(\omega^4) \\
A(\omega^3) &= A_e(\omega^6) + \omega^3 A_o(\omega^6) \\
A(\omega^7) &= A_e(\omega^6) - \omega^3 A_o(\omega^6)
\end{aligned}
\quad \Leftarrow \quad
\begin{aligned}
A(\omega^0) &= A_e(\omega^0) + \omega^0 A_o(\omega^0) \\
A(\omega^1) &= A_e(\omega^2) + \omega^1 A_o(\omega^2) \\
A(\omega^2) &= A_e(\omega^4) + \omega^2 A_o(\omega^4) \\
A(\omega^3) &= A_e(\omega^6) + \omega^3 A_o(\omega^6) \\
A(\omega^4) &= A_e(\omega^0) - \omega^0 A_o(\omega^0) \\
A(\omega^5) &= A_e(\omega^2) - \omega^1 A_o(\omega^2) \\
A(\omega^6) &= A_e(\omega^4) - \omega^2 A_o(\omega^4) \\
A(\omega^7) &= A_e(\omega^6) - \omega^3 A_o(\omega^6)
\end{aligned}$$

FFT shuffling

FFT algorithm

$$\texttt{for } k = 0 \texttt{ to } n-1\text{:}\quad A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})$$

Example ($n = 8$):

$$A(\omega^0) = A_e(\omega^0) + \omega^0 A_o(\omega^0)$$
$$A(\omega^4) = A_e(\omega^0) - \omega^0 A_o(\omega^0)$$
$$A(\omega^1) = A_e(\omega^2) + \omega^1 A_o(\omega^2)$$
$$A(\omega^5) = A_e(\omega^2) - \omega^1 A_o(\omega^2)$$
$$A(\omega^2) = A_e(\omega^4) + \omega^2 A_o(\omega^4)$$
$$A(\omega^6) = A_e(\omega^4) - \omega^2 A_o(\omega^4)$$
$$A(\omega^3) = A_e(\omega^6) + \omega^3 A_o(\omega^6)$$
$$A(\omega^7) = A_e(\omega^6) - \omega^3 A_o(\omega^6)$$

$\texttt{for } k = 0 \texttt{ to } n/2-1\text{:}$

$$A(\omega^k) = A_e(\omega^{2k}) + \omega^k A_o(\omega^{2k})$$
$$A\left(\omega^{k+\frac{n}{2}}\right) = A_e(\omega^{2k}) - \omega^k A_o(\omega^{2k})$$

FFT algorithm

```
function FFT(a,ω)
  Inputs: a = (a_0, a_1, …, a_{n-1}), for n a power of 2
          ω: A primitive nth root of unity
  Output: (a(1), a(ω), a(ω^2), …, a(ω^{n-1}))
  if ω=1: return a // n = 1, a has only one element

  (s_0, s_1, …, s_{n/2-1}) = FFT((a_0, a_2, …, a_{n-2}), ω^2)
  (s'_0, s'_1, …, s'_{n/2-1}) = FFT((a_1, a_3, …, a_{n-1}), ω^2)

  for k = 0 to n/2 - 1: // FFT shuffling
    r_k = s_k + ω^k s'_k
    r_{k+n/2} = s_k - ω^k s'_k

  return (r_0, r_1, …, r_{n-1})
```

FFT: asymptotic complexity

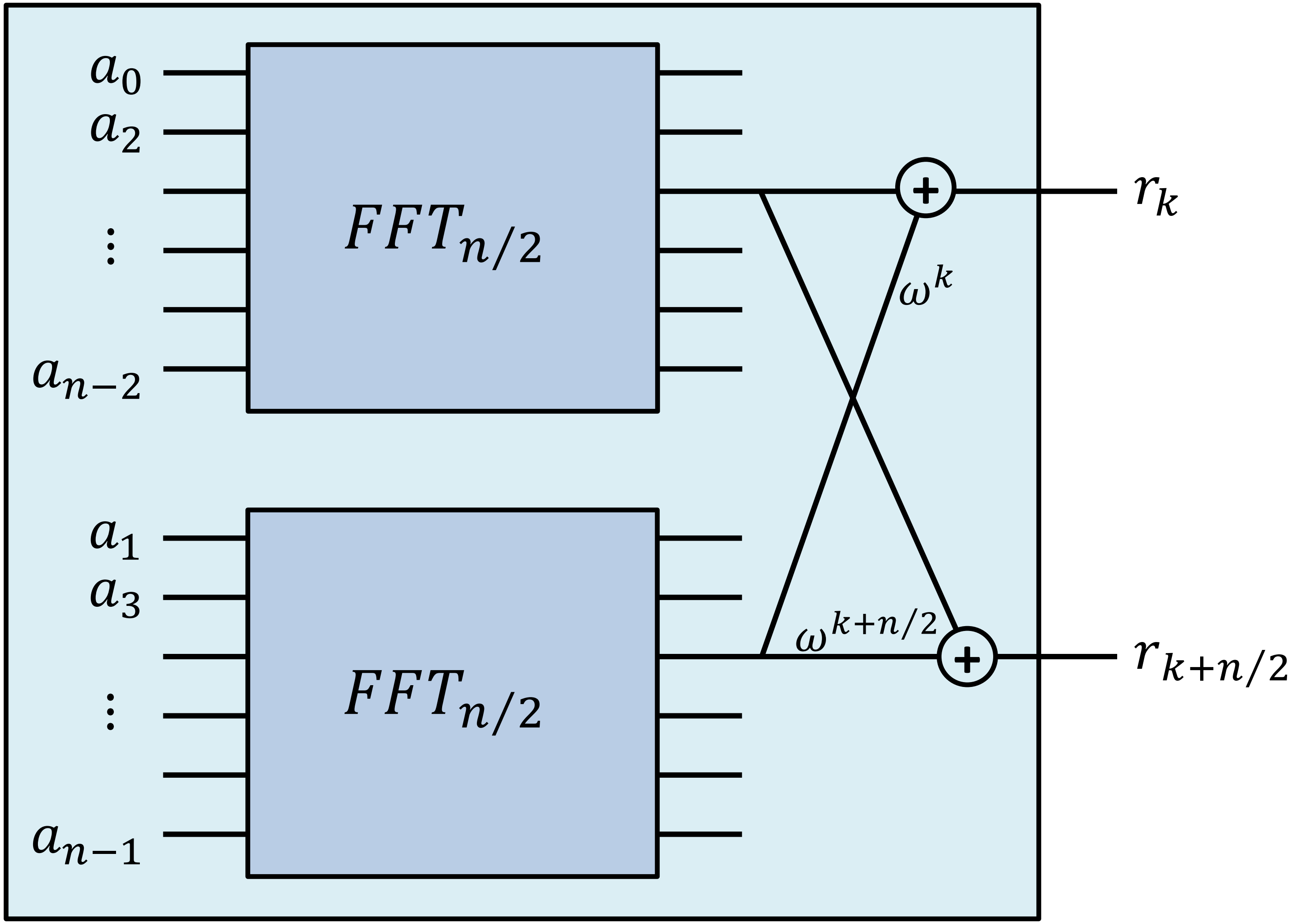
- The runtime of the FFT can be expressed as:

$$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n)$$

- Using the *Master Theorem* we conclude:

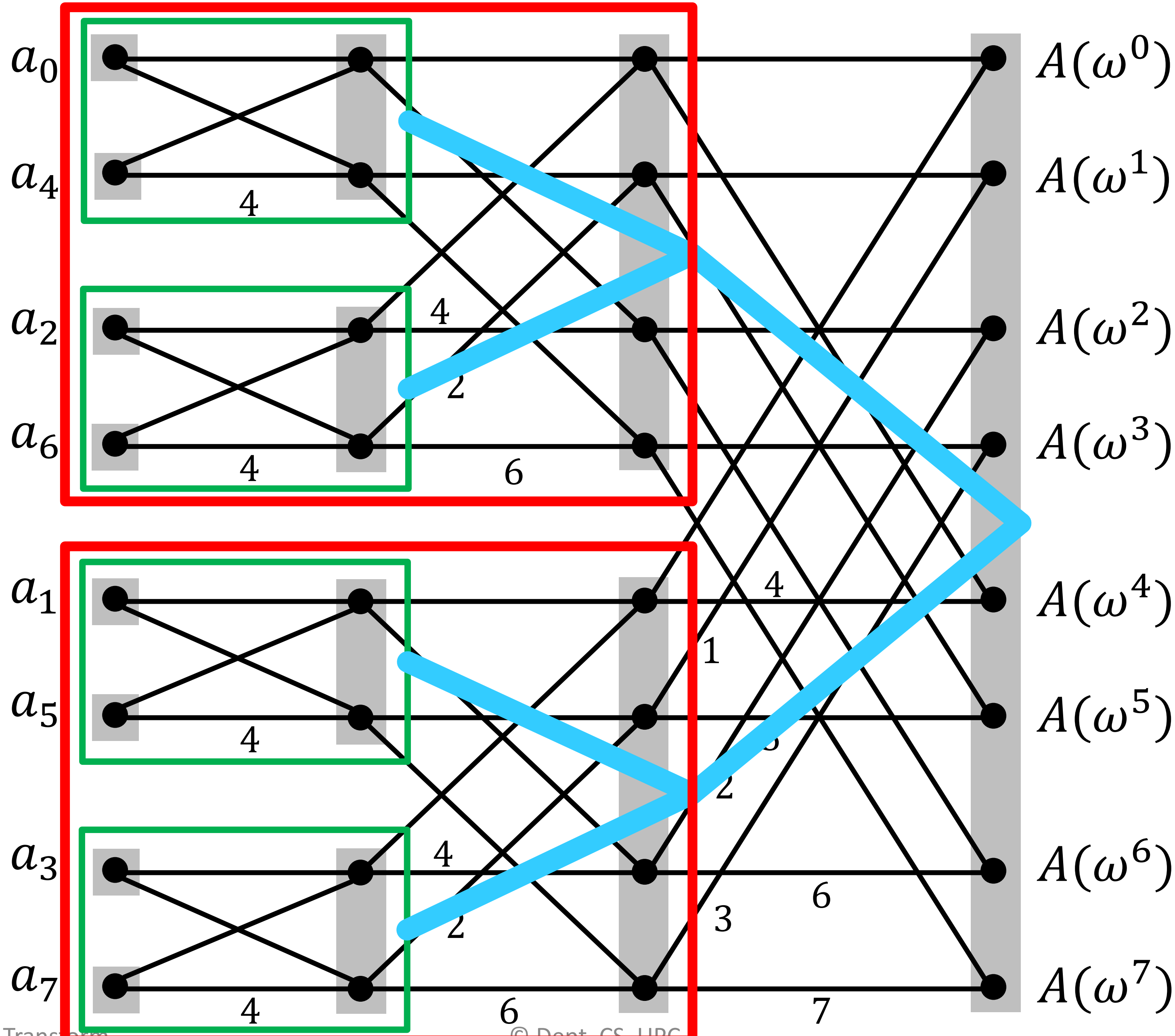
$$\text{Runtime FFT}(n) = \mathrm{O}(n \log n)$$

Unfolding the FFT

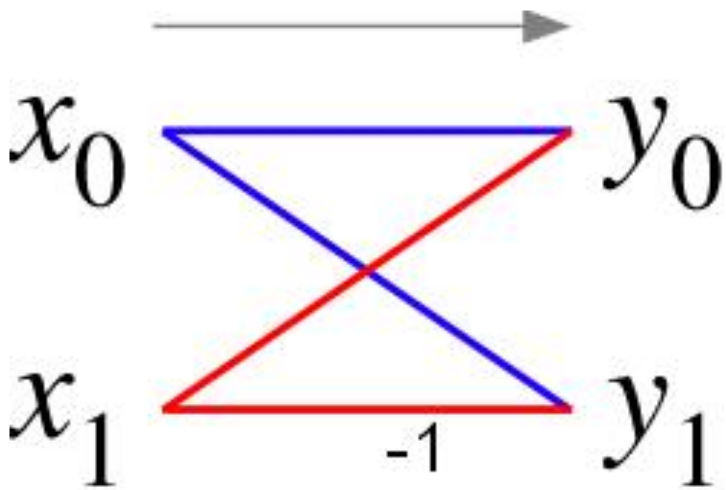


© Dept. CS, UPC

Unfolding the FFT (butterfly diagram)



Why is it called a butterfly diagram?



Conversion between both representations

representation	addition	multiplication	evaluation
coefficient	$O(n)$	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n)$	$O(n^2)$

$$\langle \text{values} \rangle = \text{FFT}(\langle \text{coefficients} \rangle, \omega)$$

$$a_0, a_1, \cdots, a_{n-1} \quad \xrightarrow[\mathbf{O(n \log n)}]{\text{evaluation}} \quad (x_0, y_0), \cdots, (x_{n-1}, y_{n-1})$$

$$a_0, a_1, \cdots, a_{n-1} \quad \xleftarrow[\text{interpolation}]{} \quad (x_0, y_0), \cdots, (x_{n-1}, y_{n-1})$$

Coefficient representation

Point-value representation

From point-values to coefficients

The Fast Fourier Transform computes:

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

where $\omega = e^{2\pi i/n} = (1, 2\pi/n)$.

Let us call $F_n(\omega)$ the Fourier matrix. Thus,

$$y = F_n(\omega) \cdot a$$

How about if we know y and we want to obtain a?

© Dept. CS, UPC

From point-values to coefficients

$$
\begin{aligned}
y &= F_n(\omega) \cdot a \\
&\Downarrow \\
[F_n(\omega)]^{-1} \cdot y &= a
\end{aligned}
$$

$F_n(\omega)$ is a unitary matrix and has the following property:

$$[F_n(\omega)]^{-1} = \frac{1}{n} \cdot F_n(\omega^{-1})$$

and also

> If ω is a primitive nth root of unit,
> then $1/\omega$ is also a primitive nth root of unit.

Note: The inverse of unitary matrix is its conjugate transpose

© Dept. CS, UPC

Conversion between both representations

representation	addition	multiplication	evaluation
coefficient	$O(n)$	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n)$	$O(n^2)$

$$\langle\text{values}\rangle = \text{FFT}(\langle\text{coefficients}\rangle, \omega)$$

$$\boxed{a_0, a_1, \cdots, a_{n-1}} \underset{\text{interpolation}}{\overset{\text{evaluation}}{\rightleftarrows}} \boxed{(x_0, y_0), \cdots, (x_{n-1}, y_{n-1})}$$

Coefficient representation — $\mathbf{O(n \log n)}$ — Point-value representation

$$\langle\text{coefficients}\rangle = \frac{1}{n}\text{FFT}(\langle\text{values}\rangle, \omega^{-1})$$

Polynomial multiplication

Input: Coefficients of two polynomials $A(x)$ and $B(x)$, of degree d_A and d_B, respectively. Let $d = d_A + d_B$.

Output: The product $C = A \cdot B$.

1. Selection:
   - Pick $\omega = (1, 2\pi/n)$, such that $n \geq d + 1$ and n is a power of two.

2. Evaluation (FFT):
   - Compute $A(1), A(\omega), A(\omega^2), \dots, A(\omega^{n-1})$.
   - Compute $B(1), B(\omega), B(\omega^2), \dots, B(\omega^{n-1})$.

3. Multiplication:
   - Compute $C\left(\omega^k\right) = A\left(\omega^k\right) \cdot B(\omega^k)$, for all $k = 0, \dots, n-1$.

4. Interpolation (inverse FFT):
   - Recover $C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_d x^d$.

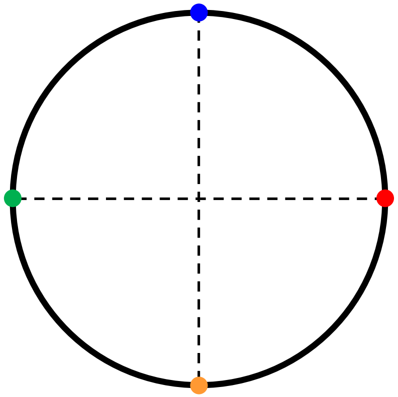
Example: from values to coefficients

- Let us consider a polynomial:

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

- We have $n = 4$ and $\omega = i$. Let us assume that the evaluation at four different points is:

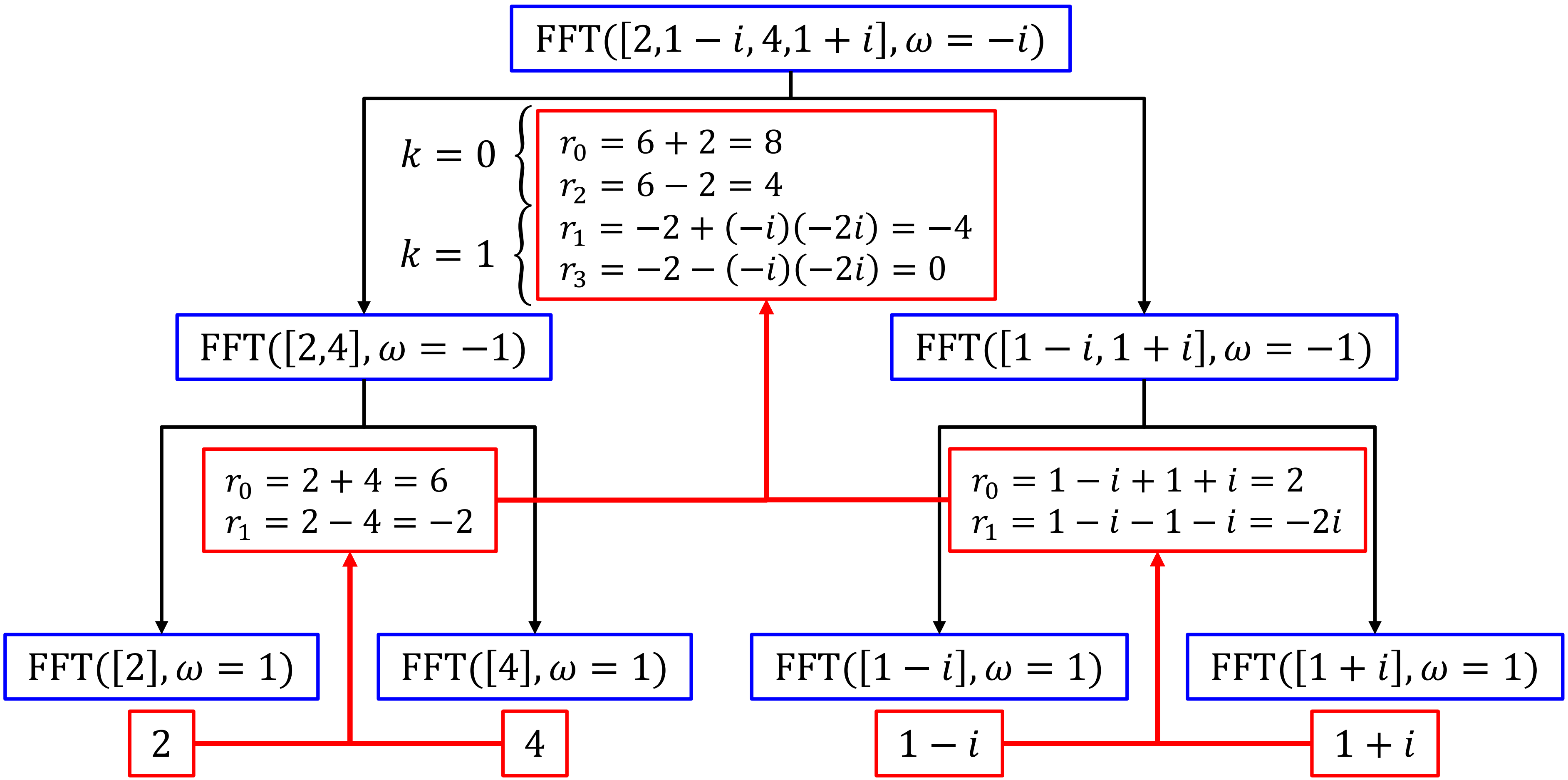
$$\begin{aligned} P(1) &= 2 \\ P(i) &= 1 - i \\ P(-1) &= 4 \\ P(-i) &= 1 + i \end{aligned}$$



- We want to calculate the coefficients (a_0, a_1, a_2, a_3) using the inverse FFT, i.e.,

$$[\mathrm{a}_0, \mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3] = \frac{1}{4} \mathrm{FFT}([2, 1 - i, 4, 1 + i], \omega^{-1})$$

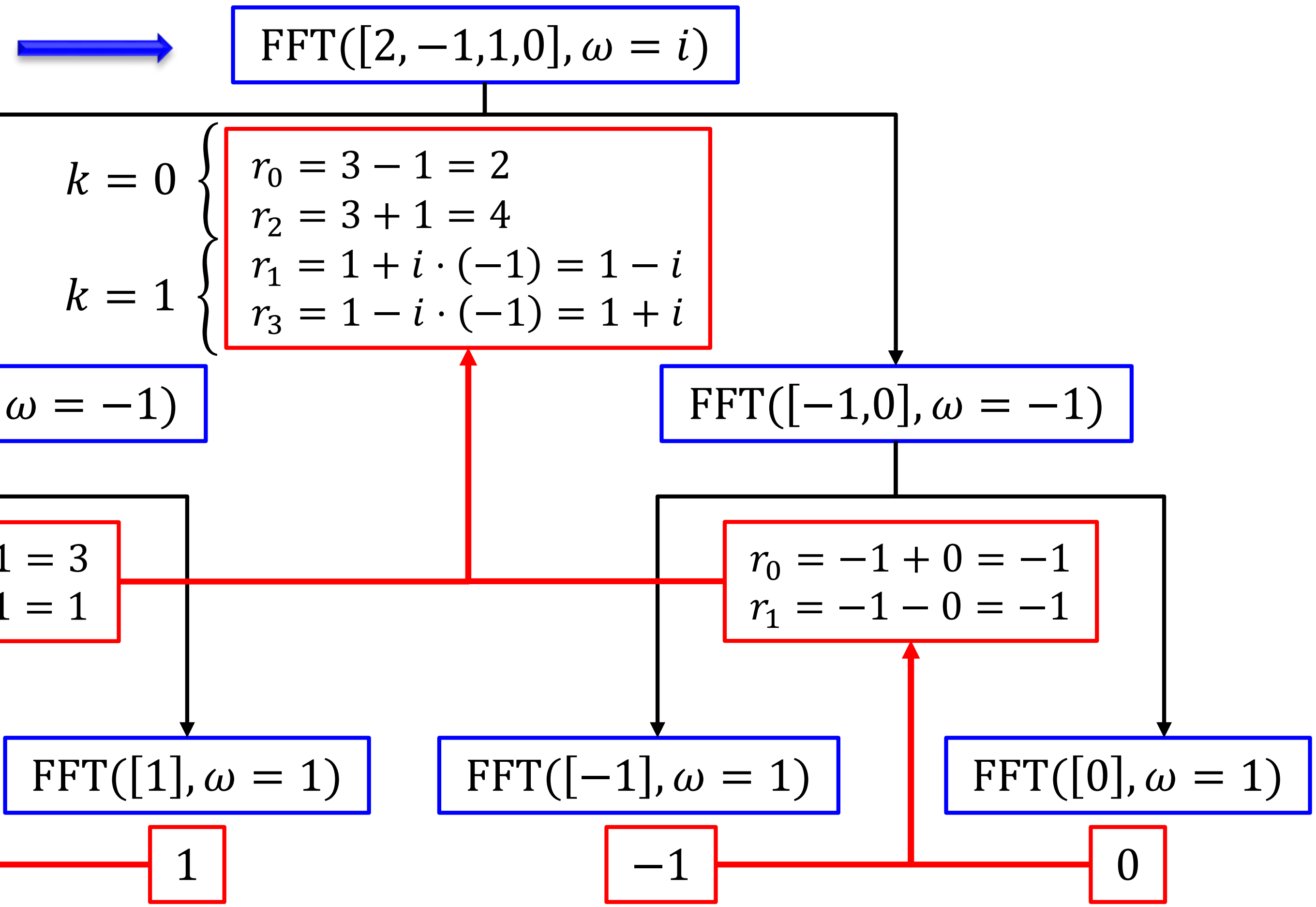
Example: from values to coefficients



$$[a_0, a_1, a_2, a_3] = \frac{1}{4}[8, -4, 4, 0] \quad \Rightarrow \quad P(x) = x^2 - x + 2$$

Example: from coefficients to values

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



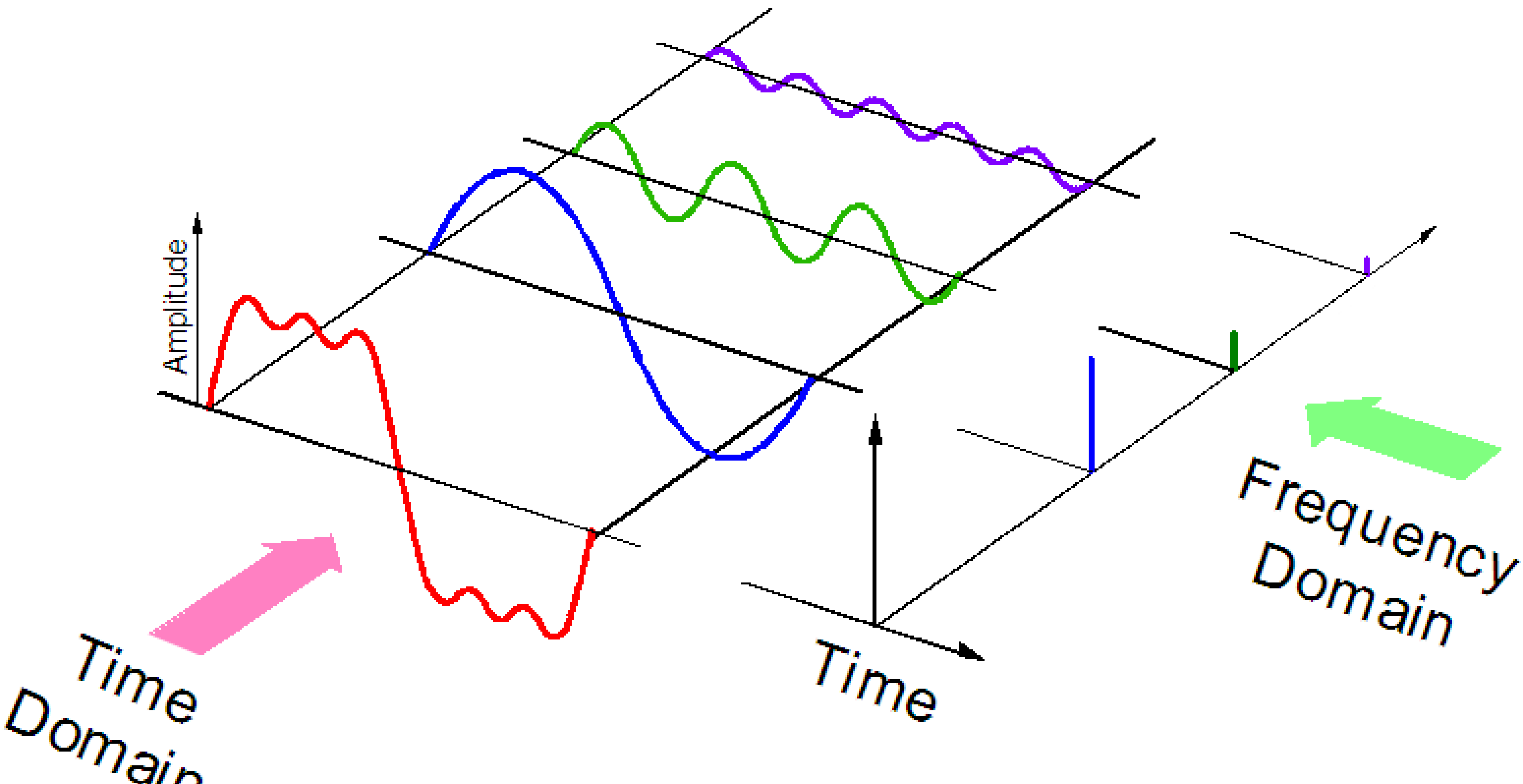
$[y_0, y_1, y_2, y_3] = \mathrm{FFT}([a_0, a_1, a_2, a_3], \omega = i)$

$\boxed{[y_0, y_1, y_2, y_3] = [2, 1 - i, 4, 1 + i]}$

Conclusions

- Gilbert Strang (MIT, 1994):
  "the most important numerical algorithm of our lifetime".

- **Reference:** Cooley, James W., and Tukey, John W., 1965,
  "An algorithm for the machine calculation of complex Fourier series," Mathematics of Computation 19: 297-301.



EXERCISES

Multiplication

Consider the polynomials $1 + x - 2x^2 + x^3$ and $-1 + x^2$:

- Choose an appropriate power of two to execute the FFT for the polynomial multiplication. Find the value of ω.

- Give the result of the FFT for $x^2 - 1$ using the value of ω required for the multiplication (no need to execute the FFT).

Polynomial evaluation

Consider the FFT of the polynomial $x^2 + 2x + 1$:

- Find the value of ω to execute the FFT.
- In which points the polynomial must be evaluated?
- Execute the FFT and give the point-value representation of the polynomial.

© Dept. CS, UPC

Multiplication using FFT

Consider the polynomials $-1 + 2x + x^2$ and $1 + 2x$:

- Choose an appropriate power of two to execute the FFT. Find the value of ω.
- Calculate their point-value representation using the FFT (execute the FFT algorithm manually).
- Calculate the product of the point-value representations.
- Execute the inverse FFT to obtain the coefficients of the product.