

# Chapter 5

## Divide and Conquer



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

# Divide-and-Conquer

## Divide-and-conquer.

Break up problem into several parts.

Solve each part recursively.

Combine solutions to sub-problems into overall solution.

## Most common usage.

Break up problem of size  $n$  into **two** equal parts of size  $\frac{1}{2}n$  in **linear time**.

Solve two parts recursively.

Combine two solutions into overall solution in **linear time**.

## Consequence.

Divide-and-conquer:  $\Theta(n \log n)$

# 5.1 Mergesort

---

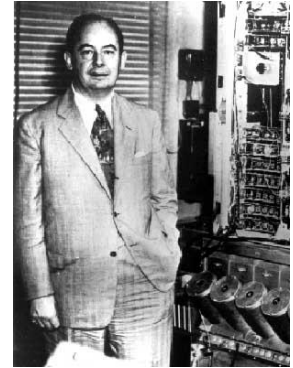
# Mergesort

## Mergesort.

Divide array into two halves.

Recursively sort each half.

Merge two halves to make sorted whole.



Jon von Neumann (1945)

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| A | L | G | O | R | I | T | H | M | S |
|---|---|---|---|---|---|---|---|---|---|

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| A | L | G | O | R | I | T | H | M | S |
|---|---|---|---|---|---|---|---|---|---|

divide

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| A | G | L | O | R | H | I | M | S | T |
|---|---|---|---|---|---|---|---|---|---|

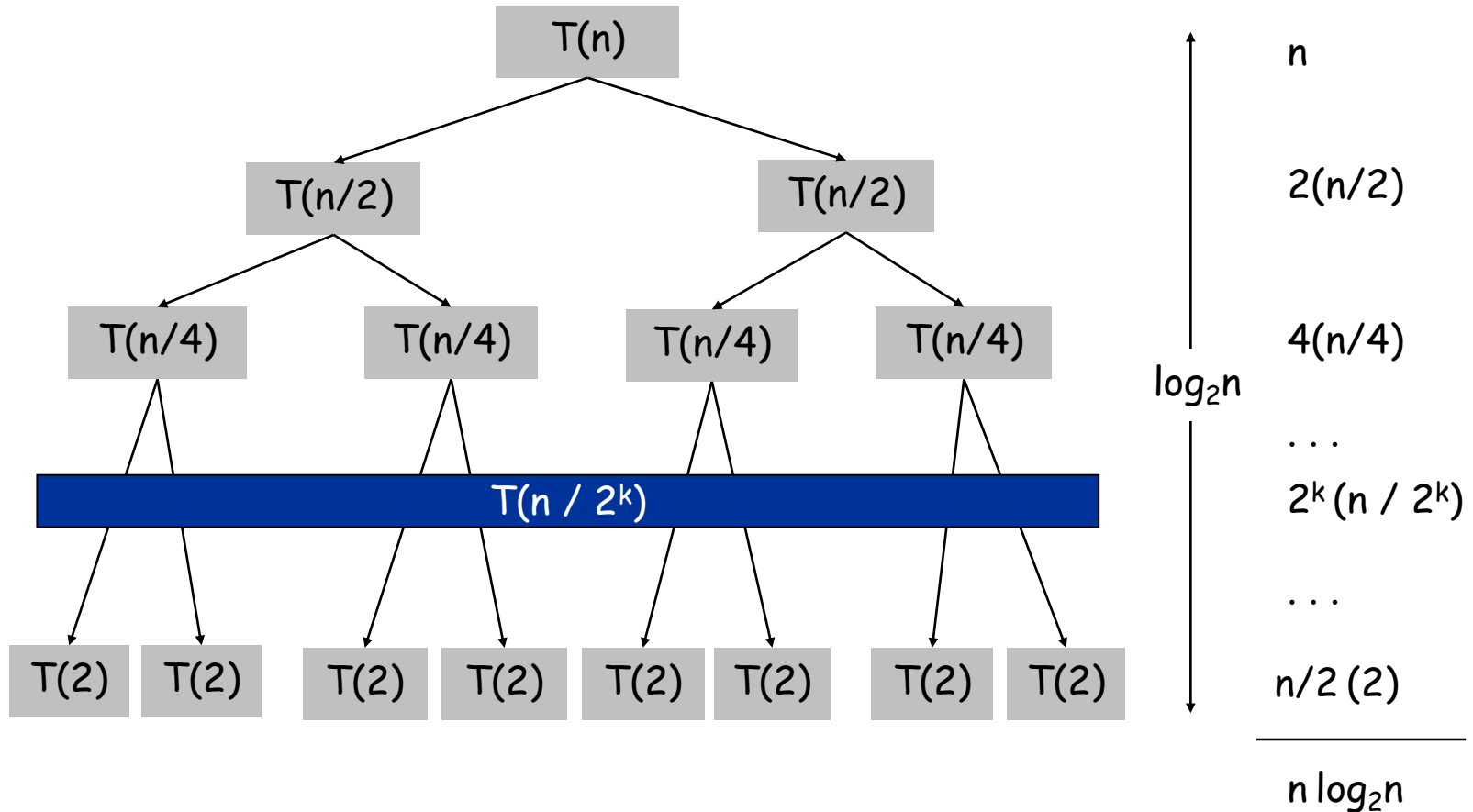
sort

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| A | G | H | I | L | M | O | R | S | T |
|---|---|---|---|---|---|---|---|---|---|

merge

## Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



# Proof by Induction

**Claim.** If  $T(n)$  satisfies this recurrence, then  $T(n) = n \log_2 n$ .

↑  
assumes  $n$  is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

**Pf.** (by induction on  $n$ )

Base case:  $n = 1$ .

Inductive hypothesis:  $T(n) = n \log_2 n$ .

Goal: show that  $T(2n) = 2n \log_2 (2n)$ .

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n(\log_2(2n) - 1) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

## 5.4 Closest Pair of Points

---

# Closest Pair of Points

**Closest pair.** Given  $n$  points in the plane, find a pair with smallest Euclidean distance between them.

**Fundamental geometric primitive.**

Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.

Special case of nearest neighbor, Euclidean MST, Voronoi.

↖ fast closest pair inspired fast algorithms for these problems

**Brute force.** Check all pairs of points  $p$  and  $q$  with  $\Theta(n^2)$  comparisons.

**Assumption.** No two points have same  $x$  coordinate.

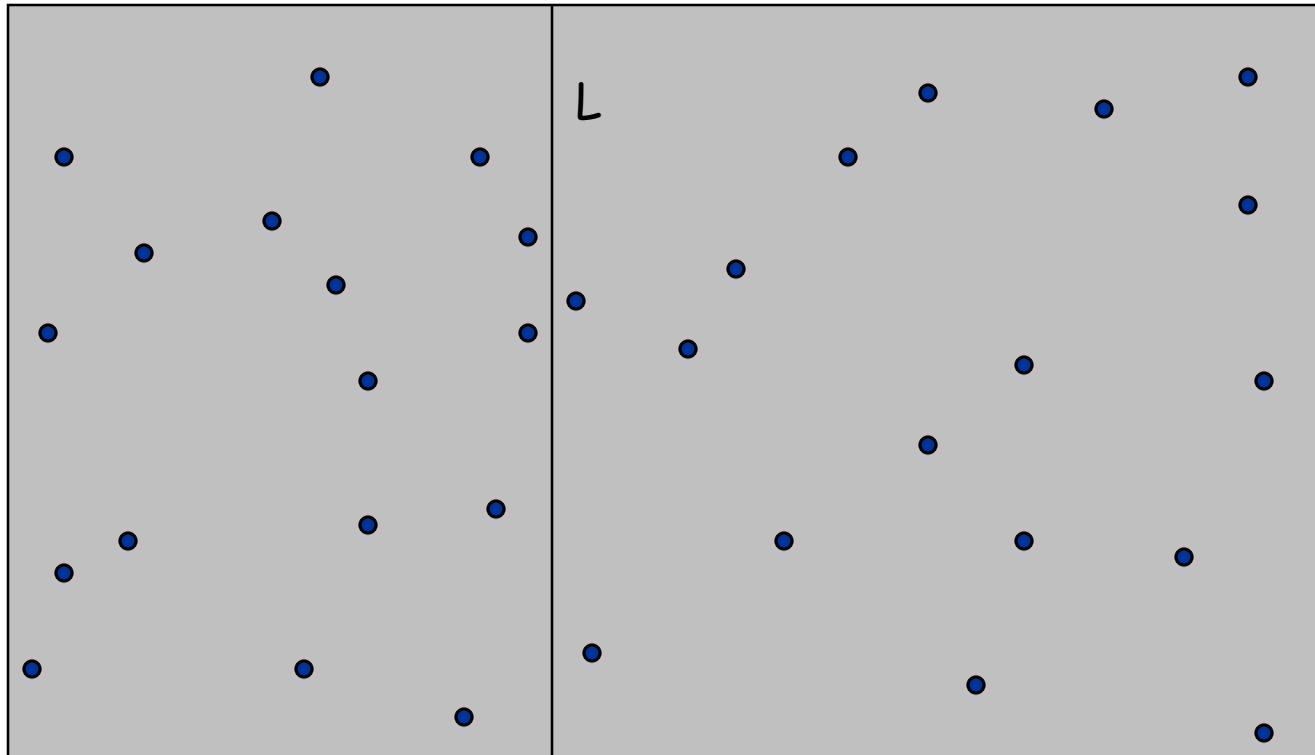
↑  
to make presentation cleaner



# Closest Pair of Points

Algorithm.

**Divide:** draw vertical line  $L$  so that roughly  $\frac{1}{2}n$  points on each side.

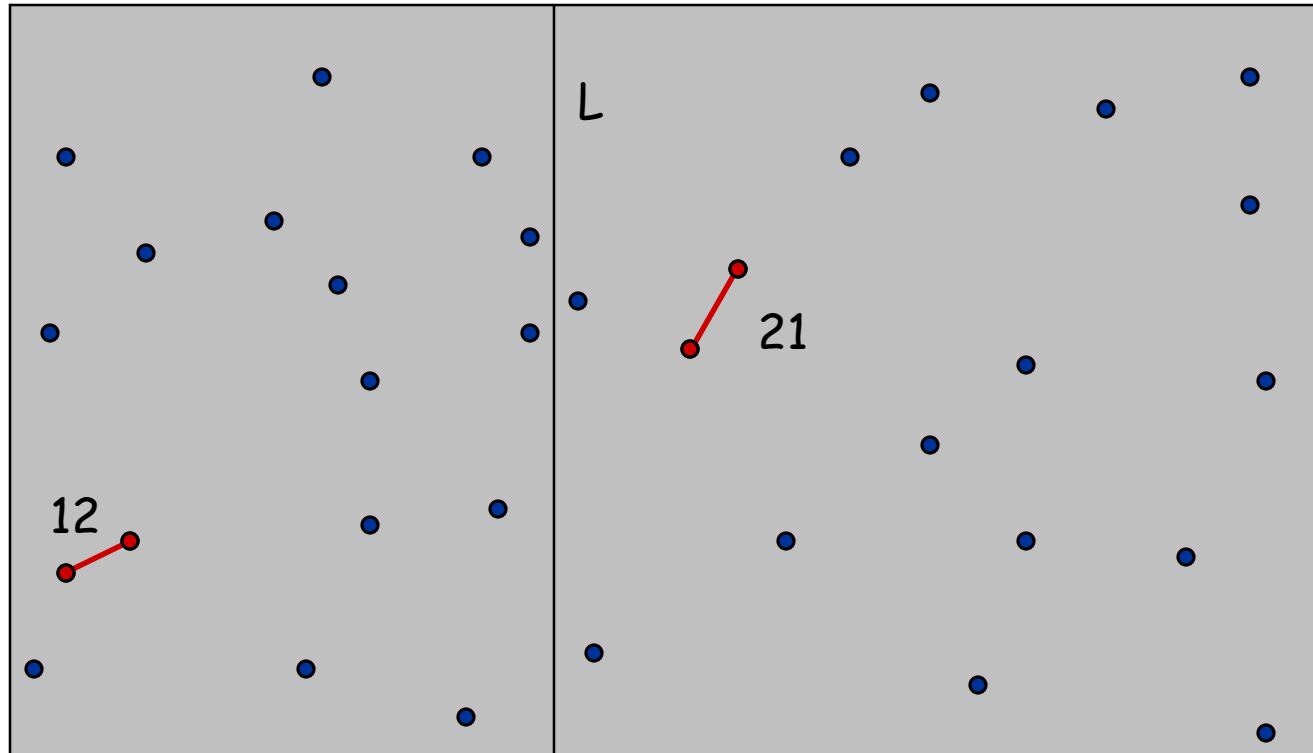


# Closest Pair of Points

## Algorithm.

**Divide:** draw vertical line  $L$  so that roughly  $\frac{1}{2}n$  points on each side.

**Conquer:** find closest pair in each side recursively.



# Closest Pair of Points

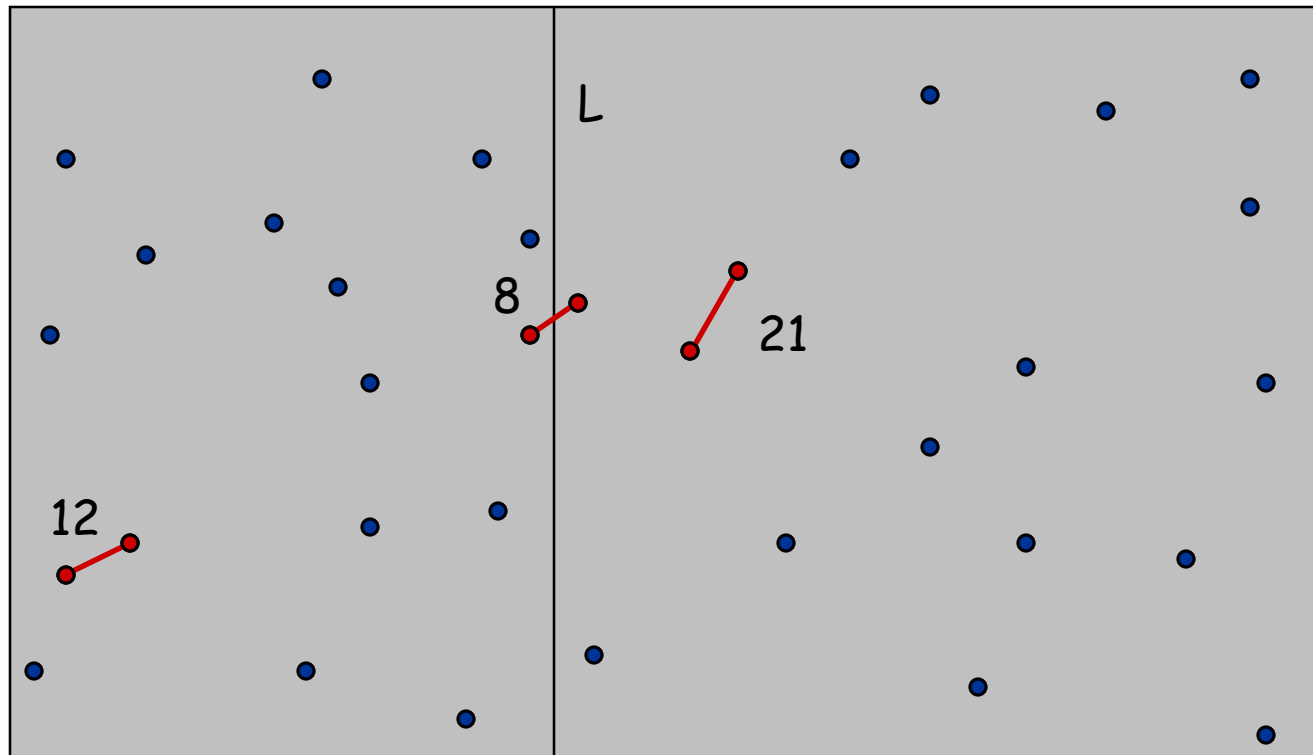
## Algorithm.

Divide: draw vertical line  $L$  so that roughly  $\frac{1}{2}n$  points on each side.

Conquer: find closest pair in each side recursively.

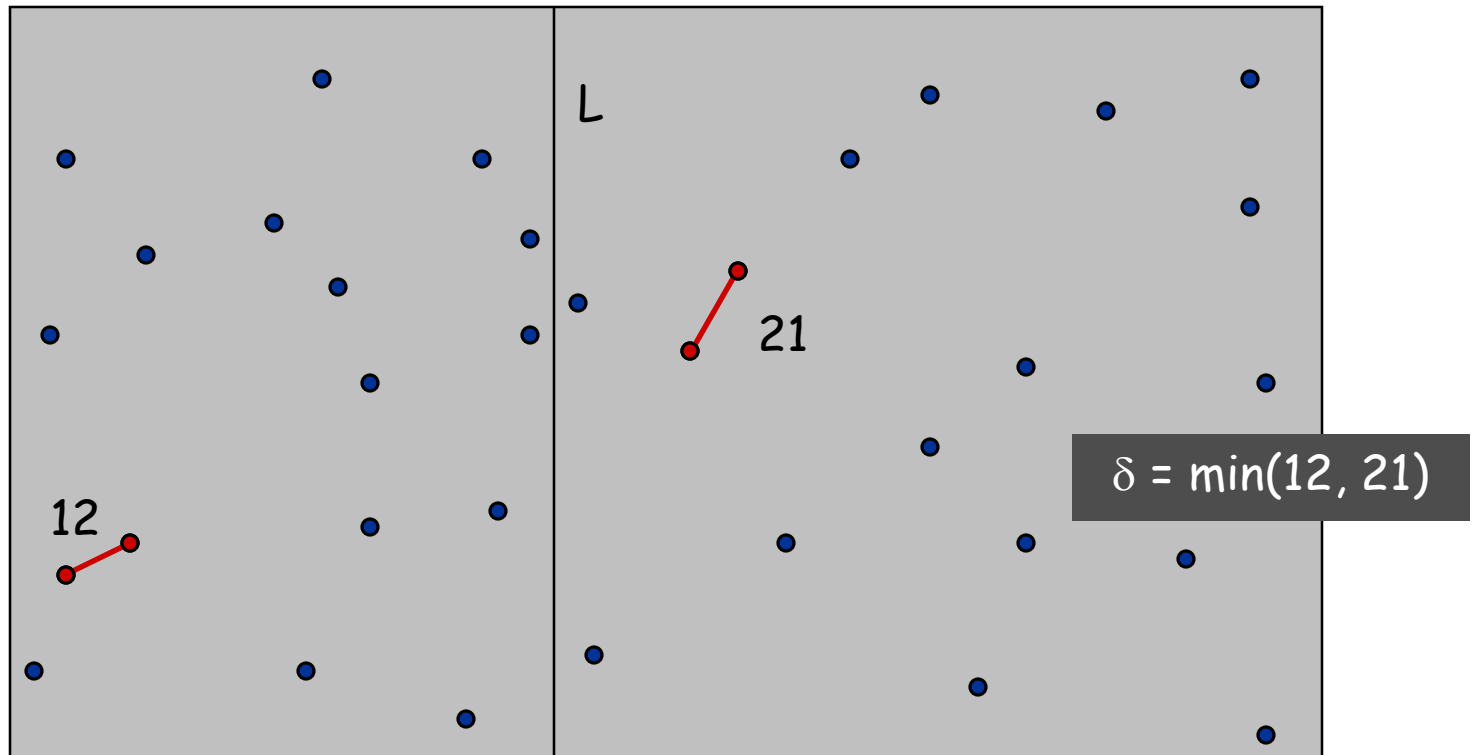
**Combine:** find closest pair with one point in each side.  $\leftarrow$  seems like  $\Theta(n^2)$

Return best of 3 solutions.



# Closest Pair of Points

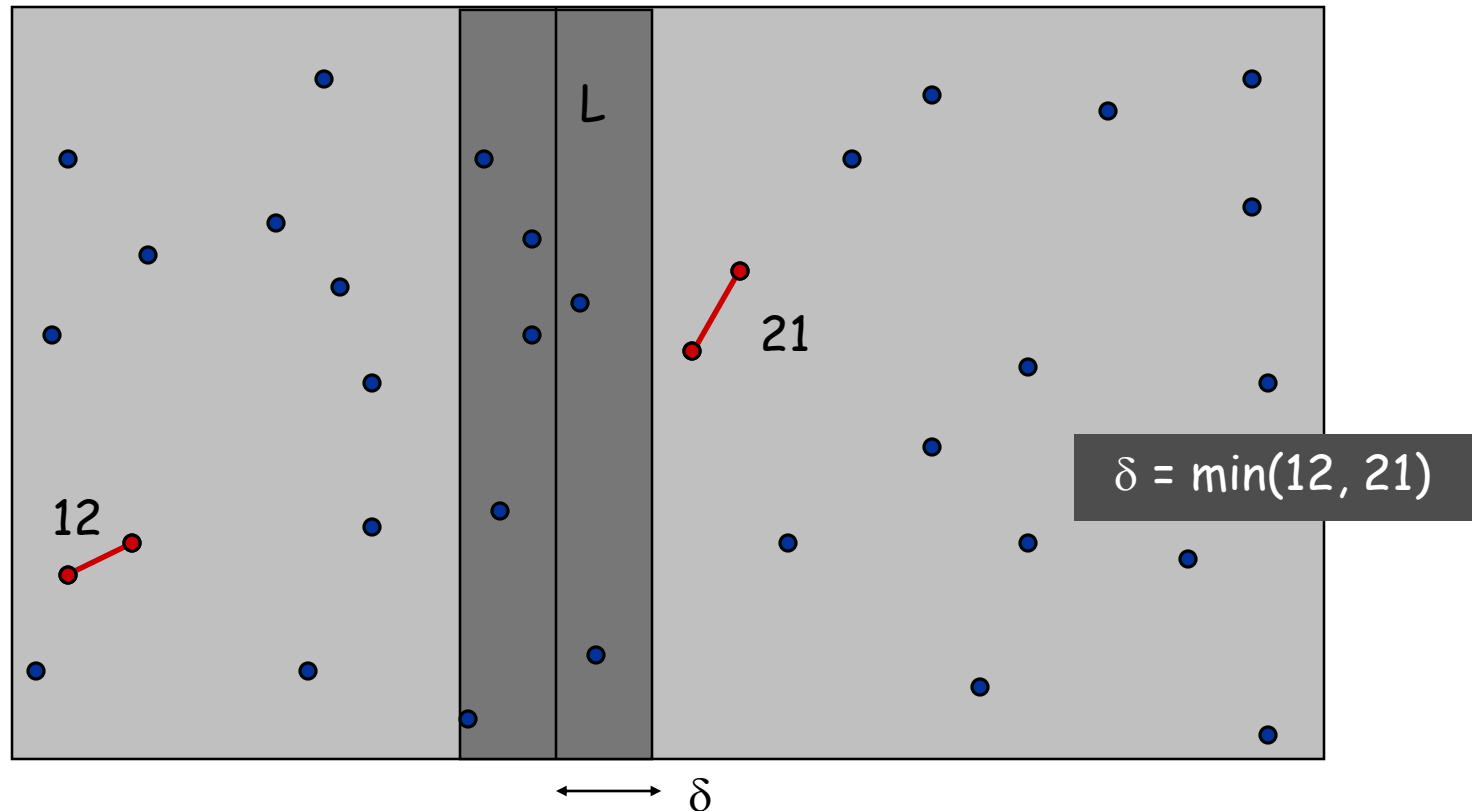
Find closest pair with one point in each side, **assuming that distance  $< \delta$** .



## Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance  $< \delta$** .

Observation: only need to consider points within  $\delta$  of line  $L$ .

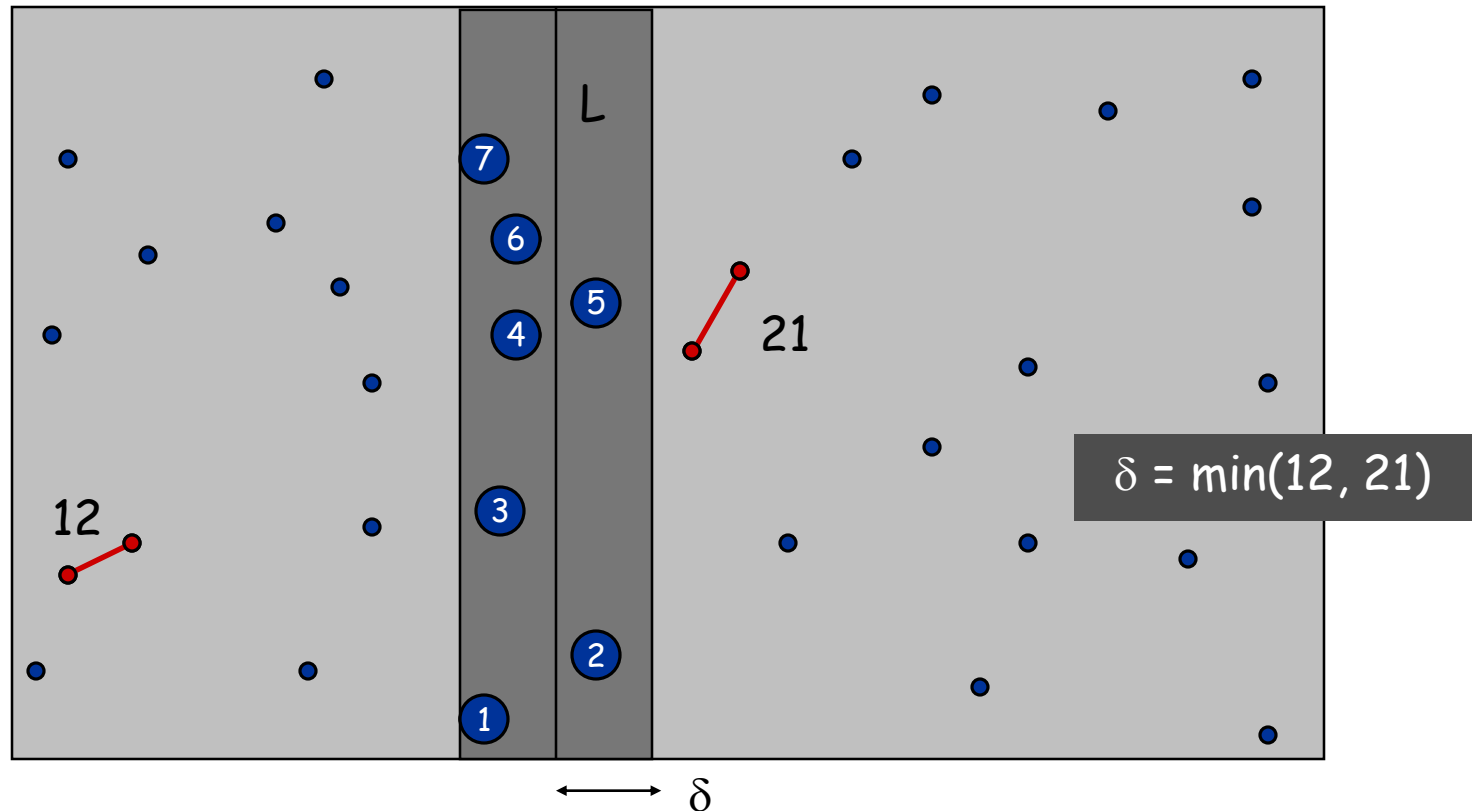


# Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance  $< \delta$** .

Observation: only need to consider points within  $\delta$  of line  $L$ .

Sort points in  $2\delta$ -strip by their  $y$  coordinate.



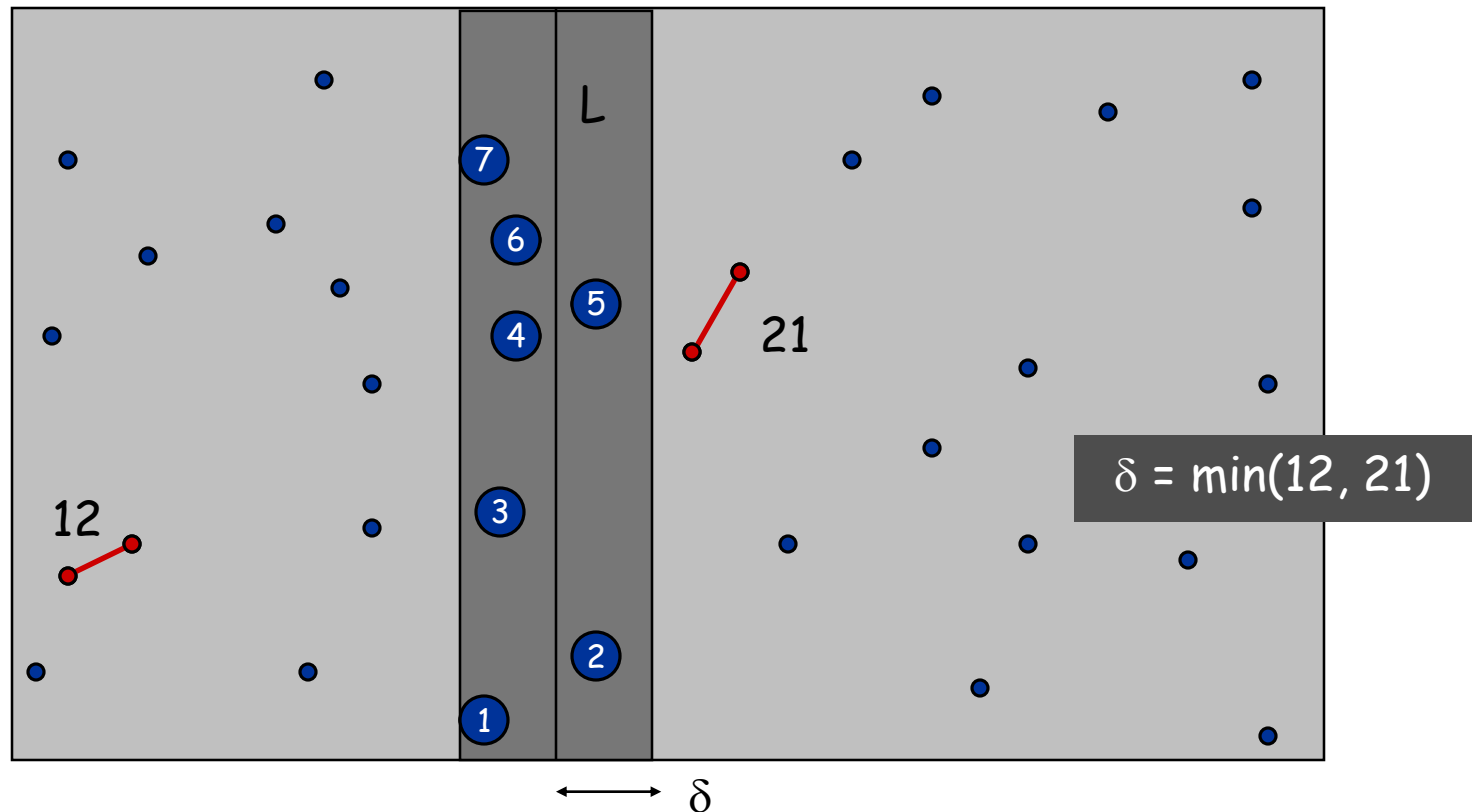
# Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance  $< \delta$** .

Observation: only need to consider points within  $\delta$  of line  $L$ .

Sort points in  $2\delta$ -strip by their  $y$  coordinate.

Only check distances of those within 11 positions in sorted list!



# Closest Pair of Points

**Def.** Let  $s_i$  be the point in the  $2\delta$ -strip, with the  $i^{\text{th}}$  smallest  $y$ -coordinate.

**Claim.** If  $|i - j| \geq 12$ , then the distance between  $s_i$  and  $s_j$  is at least  $\delta$ .

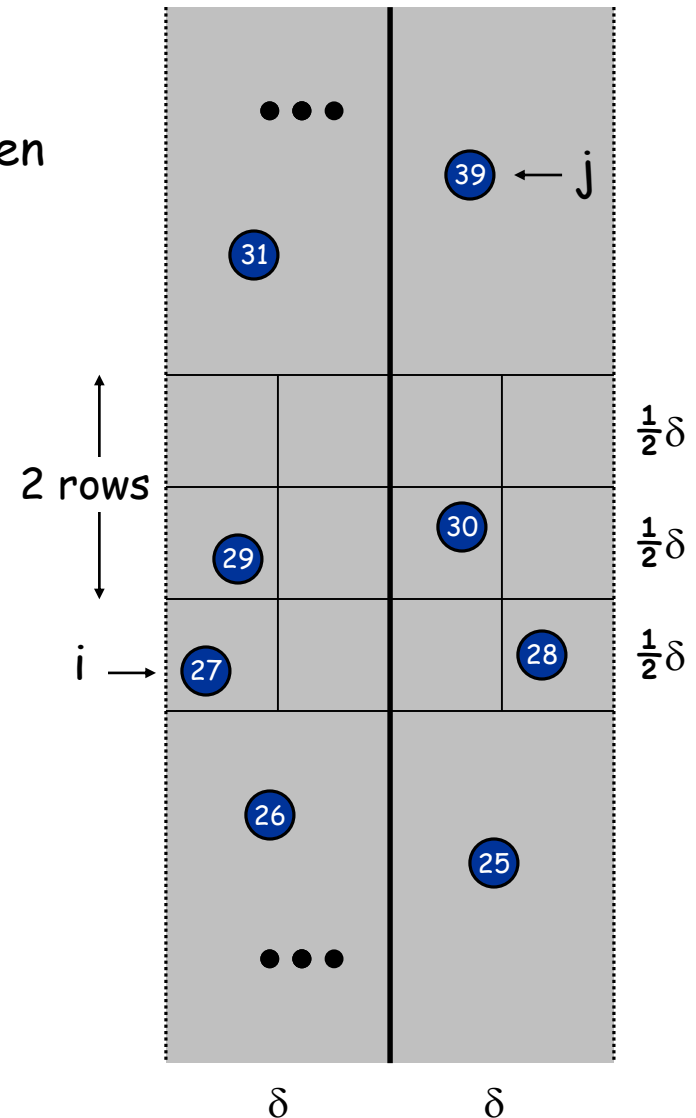
**Pf.**

No two points lie in same  $\frac{1}{2}\delta$ -by- $\frac{1}{2}\delta$  box.

Two points at least 2 rows apart

have distance  $\geq 2(\frac{1}{2}\delta)$ . ■

**Fact.** Still true if replace 12 with 8. Why?





# Closest Pair Algorithm

```
Closest-Pair( $p_1, \dots, p_n$ ) {
```

```
  Compute separation line  $L$  such that half the points  
  are on one side and half on the other side.
```

```
   $\delta_1$  = Closest-Pair(left half)
```

```
   $\delta_2$  = Closest-Pair(right half)
```

```
   $\delta$  =  $\min(\delta_1, \delta_2)$ 
```

```
  Delete all points further than  $\delta$  from separation line  $L$ 
```

```
  Sort remaining points by y-coordinate.
```

```
  Scan points in y-order and compare distance between  
  each point and next 11 neighbors. If any of these  
  distances is less than  $\delta$ , update  $\delta$ .
```

```
  return  $\delta$ .
```

```
}
```

sort

$O(n \log n)$

$2T(n/2)$

$O(n)$

$O(n \log n)$

$O(n)$

# Closest Pair of Points: Analysis

Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Can we achieve  $O(n \log n)$ ?

A. Yes. Don't sort points from scratch each time.

Sort all the points twice before recursive call, once by x coordinate and once by y coordinate

Reuse the sorted sequences when needed (linear time)

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

## 5.5 Integer Multiplication

---

# Integer Arithmetic

**Add.** Given two  $n$ -digit integers  $a$  and  $b$ , compute  $a + b$ .  
 $O(n)$  bit operations.

**Multiply.** Given two  $n$ -digit integers  $a$  and  $b$ , compute  $a \times b$ .  
Brute force solution:  $\Theta(n^2)$  bit operations.

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
|   | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| + | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
|   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

Add

[illegible]

## Multiply

# Divide-and-Conquer Multiplication: Warmup

To multiply two  $n$ -digit integers:

Multiply four  $\frac{1}{2}n$ -digit integers.

Add two  $\frac{1}{2}n$ -digit integers, and shift to obtain result.

$$\begin{array}{ccccccc} x = & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ & \underbrace{\phantom{1000}}_{x_1} & & \underbrace{\phantom{1101}}_{x_0} \end{array}$$

$$x = 2^{n/2} \cdot x_1 + x_0$$

$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0$$

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

↑  
assumes  $n$  is a power of 2

# Karatsuba Multiplication

To multiply two  $n$ -digit integers:

Add two  $\frac{1}{2}n$  digit integers.

Multiply **three**  $\frac{1}{2}n$ -digit integers.

Add, subtract, and shift  $\frac{1}{2}n$ -digit integers to obtain result.

$$\begin{aligned}x &= 2^{n/2} \cdot x_1 + x_0 \\y &= 2^{n/2} \cdot y_1 + y_0 \\xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\&= \underbrace{2^n \cdot x_1 y_1}_A + 2^{n/2} \cdot \underbrace{((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0)}_B + \underbrace{x_0 y_0}_C\end{aligned}$$

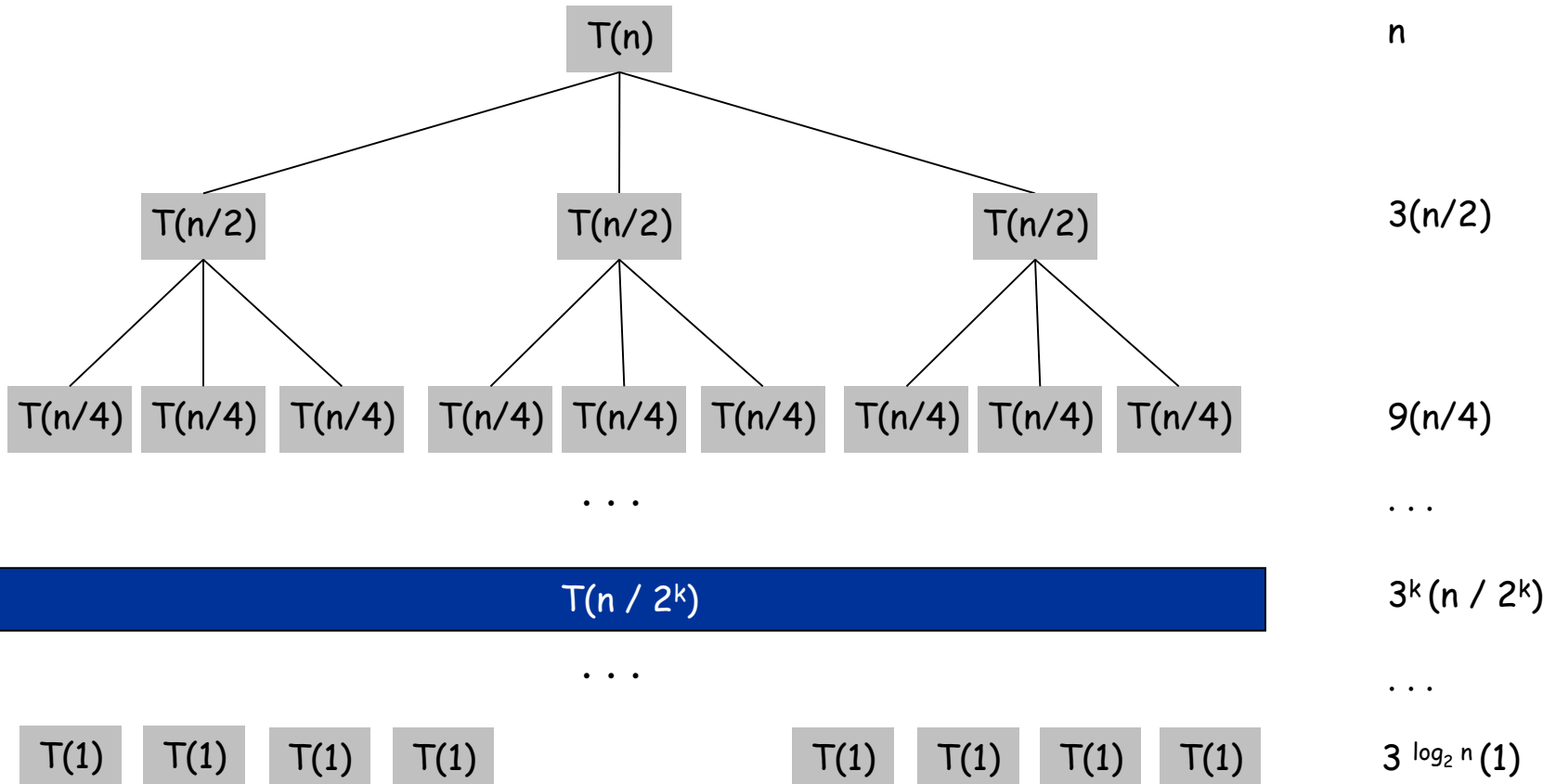
**Theorem.** [Karatsuba-Ofman, 1962] Can multiply two  $n$ -digit integers in  $O(n^{1.585})$  bit operations.

$$\begin{aligned}T(n) &\leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \\&\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})\end{aligned}$$

# Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = \frac{\left(\frac{3}{2}\right)^{1+\log_2 n} - 1}{\frac{3}{2} - 1} n = 3n^{\log_2 3} - 2n$$



# Matrix Multiplication

---



# Matrix Multiplication

**Matrix multiplication.** Given two  $n$ -by- $n$  matrices  $A$  and  $B$ , compute  $C = AB$ .

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

**Brute force.**  $\Theta(n^3)$  arithmetic operations.

**Fundamental question.** Can we improve upon brute force?

# Matrix Multiplication: Warmup

## Divide-and-conquer.

Divide: partition  $A$  and  $B$  into  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks.

Conquer: multiply 8  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  recursively.

Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

# Matrix Multiplication: Key Idea

Key idea. multiply 2-by-2 block matrices with only **7** multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$P_1 = A_{11} \times (B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12}) \times B_{22}$$

$$P_3 = (A_{21} + A_{22}) \times B_{11}$$

$$P_4 = A_{22} \times (B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

7 multiplications.

18 = 10 + 8 additions (or subtractions).

# Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

Divide: partition  $A$  and  $B$  into  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks.

Compute: 14  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  matrices via 10 matrix additions.

Conquer: multiply 7  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  matrices recursively.

Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

Assume  $n$  is a power of 2.

$T(n)$  = # arithmetic operations.

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

# Fast Matrix Multiplication in Practice

**Common misperception:** "Strassen is only a theoretical curiosity."

Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when  $n \sim 2,500$ .

Range of instances where it's useful is a subject of controversy.

**Remark.** Can "Strassenize"  $Ax=b$ , determinant, eigenvalues, and other matrix ops.

# Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?

A. Yes! [Strassen, 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.81})$$

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?

A. Impossible. [Hopcroft and Kerr, 1971]

$$\Theta(n^{\log_2 6}) = O(n^{2.59})$$

Q. Two 3-by-3 matrices with only 21 scalar multiplications?

A. Also impossible.

$$\Theta(n^{\log_3 21}) = O(n^{2.77})$$

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?

A. Yes! [Pan, 1980]

$$\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$$

# Fast Matrix Multiplication in Theory

## Decimal wars.

December, 1979:  $O(n^{2.521813})$ .

January, 1980:  $O(n^{2.521801})$ .

...

1987:  $O(n^{2.375477})$ .

2010:  $O(n^{2.374})$ .

2011:  $O(n^{2.3728642})$ .

2014:  $O(n^{2.3728639})$ .

**Best known.**  $O(n^{2.3728639})$  [François Le Gall, 2014.]

**Conjecture.**  $O(n^{2+\varepsilon})$  for any  $\varepsilon > 0$ .

**Caveat.** Theoretical improvements to Strassen are progressively less practical.

## 5.6 Convolution and FFT

---



# Fast Fourier Transform: Applications

## Applications.

Optics, acoustics, quantum physics, telecommunications, control systems, signal processing, speech recognition, data compression, image processing.

DVD, JPEG, MP3, MRI, CAT scan.

Numerical solutions to Poisson's equation.

The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. *-Charles van Loan*

# Polynomials: Coefficient Representation

Polynomial. [coefficient representation]

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

$$B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$$

Add:  $O(n)$  arithmetic operations.

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

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

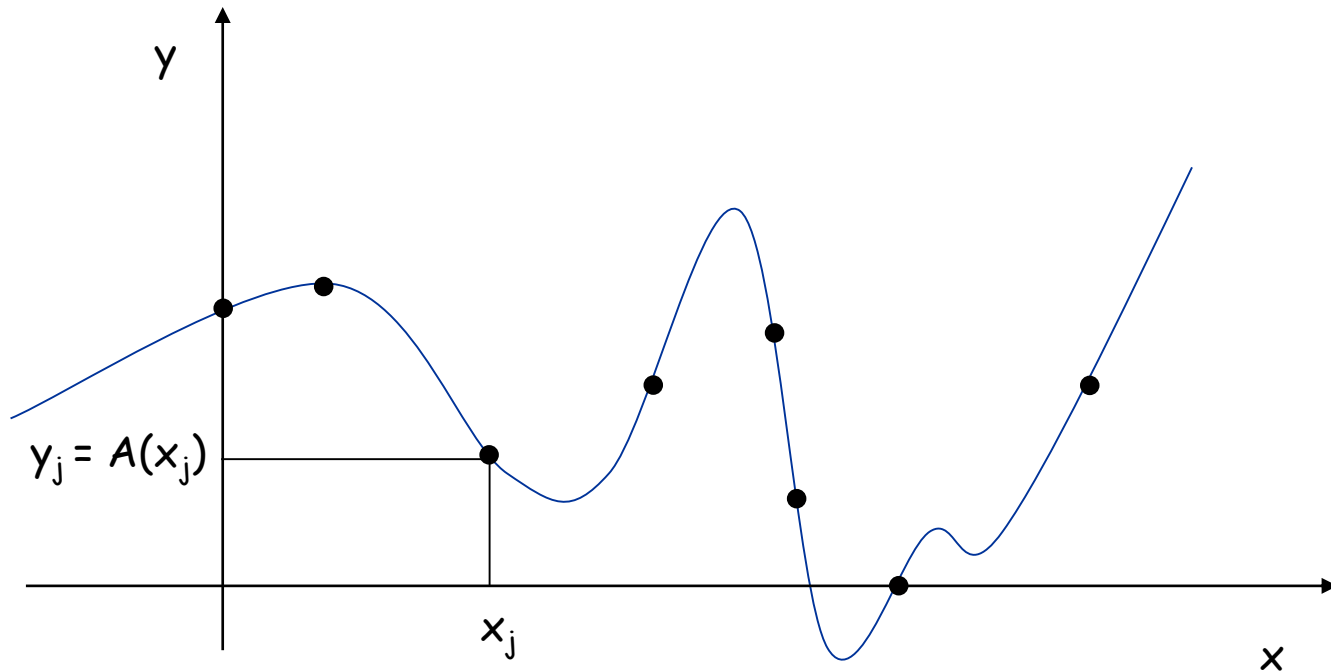
$$A(x) = a_0 + (x(a_1 + x(a_2 + \cdots + x(a_{n-2} + x(a_{n-1}))) \cdots))$$

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

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

## Polynomials: Point-Value Representation

A degree  $n-1$  polynomial  $A(x)$  is uniquely specified by its evaluation at  $n$  distinct values of  $x$ .



# Polynomials: Point-Value Representation

Polynomial. [point-value representation]

$$A(x): (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

$$B(x): (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

Add:  $O(n)$  arithmetic operations.

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

Multiply:  $O(n)$ , but need  $2n-1$  points.

$$A(x) \times B(x): (x_0, y_0 \times z_0), \dots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluate:  $O(n^2)$  using Lagrange's formula.

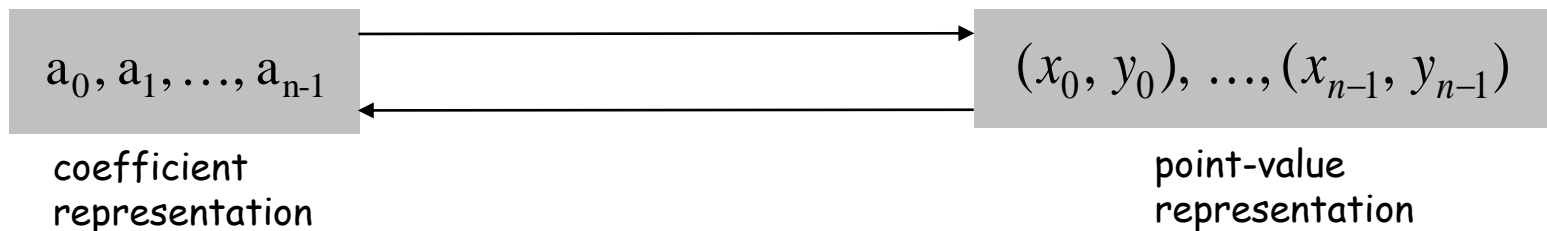
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

# Converting Between Two Polynomial Representations

**Tradeoff.** Fast evaluation or fast multiplication. We want both!

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

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



# Converting Between Two Polynomial Representations: Brute Force

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

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

**Running time.**  $O(n^2)$  for matrix-vector multiplication or  $n$  times Horner's method

# Converting Between Two Polynomial Representations: Brute Force

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

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

**Running time.**  $O(n^3)$  for Gaussian elimination.

# Coefficient to Point-Value Representation: Intuition

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

We can choose which points!

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

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

$$A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2).$$

**Intuition.** Choose two points to be  $\pm 1$ .

$$A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1).$$

$$A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1).$$

Can evaluate polynomial of degree  $\leq n$  at 2 points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at 1 point.



# Coefficient to Point-Value Representation: Intuition

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

We can choose which points!

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

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

$$A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2).$$

**Intuition.** Choose four **complex** points to be  $\pm 1, \pm i$ .

$$A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1).$$

$$A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1).$$

$$A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(-1).$$

$$A(-i) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1).$$

Can evaluate polynomial of degree  $\leq n$  at 4 points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at 2 point.

# Coefficient to Point-Value Representation: Intuition

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

We can choose which points!

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

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

$$A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2).$$

**Goal.** Choose  $n$  points s.t.

Can evaluate polynomial of degree  $\leq n$  at  $n$  points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at  $\frac{1}{2}n$  point.

But also: can evaluate polynomial of degree  $\leq \frac{1}{2}n$  at  $\frac{1}{2}n$  points by evaluating two polynomials of degree  $\leq \frac{1}{4}n$  at  $\frac{1}{4}n$  point, and so on.

# Roots of Unity

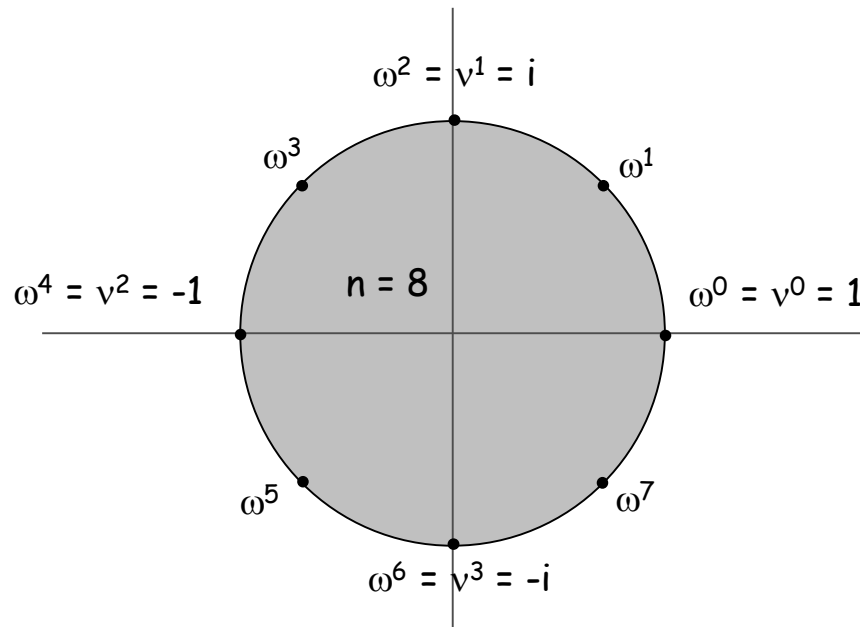
**Def.** An  $n^{\text{th}}$  root of unity is a complex number  $x$  such that  $x^n = 1$ .

**Fact.** The  $n^{\text{th}}$  roots of unity are:  $\omega^0, \omega^1, \dots, \omega^{n-1}$  where  $\omega = e^{2\pi i / n}$ .

**Pf.**  $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$ .

**Fact.** The  $\frac{1}{2}n^{\text{th}}$  roots of unity are:  $v^0, v^1, \dots, v^{n/2-1}$  where  $v = e^{4\pi i / n}$ .

**Fact.**  $\omega^2 = v$  and  $(\omega^2)^k = v^k$ .



# Discrete Fourier Transform

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

**Key idea:** choose  $x_k = \omega^k$  where  $\omega$  is principal  $n^{\text{th}}$  root of unity.

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

$\uparrow$                        $\uparrow$

Discrete Fourier transform      Fourier matrix  $F_n$

# Fast Fourier Transform

**Goal.** Evaluate a degree  $n-1$  polynomial  $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$  at its  $n^{\text{th}}$  roots of unity:  $\omega^0, \omega^1, \dots, \omega^{n-1}$ .

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

$$A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n/2-2} x^{(n-1)/2}.$$

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

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

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

**Combine.**

$$A(\omega^k) = A_{\text{even}}(v^k) + \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2$$

$$A(\omega^{k+n/2}) = A_{\text{even}}(v^k) - \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2$$

$$\begin{array}{c} \uparrow \\ \omega^{k+n/2} = -\omega^k \end{array}$$

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

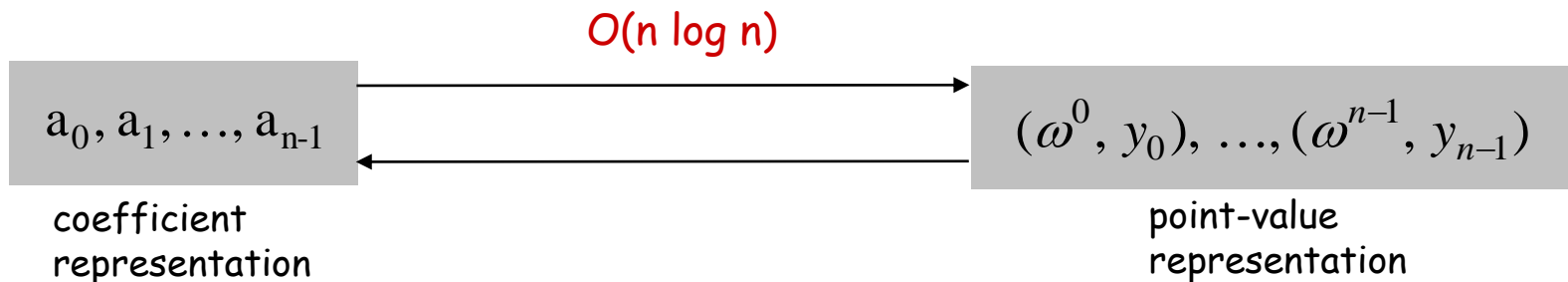
# FFT Algorithm

```
FFT(n, a0, a1, ..., an-1) {  
    if (n == 1) return a0  
  
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a2, a4, ..., an-2)  
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, a5, ..., an-1)  
  
    for k = 0 to n/2 - 1 {  
        ωk ← e2πik/n  
        yk ← ek + ωk dk  
        yk+n/2 ← ek - ωk dk  
    }  
  
    return (y0, y1, ..., yn-1)  
}
```

# FFT Summary

**Theorem.** FFT algorithm evaluates a degree  $n-1$  polynomial at each of the  $n^{\text{th}}$  roots of unity in  $O(n \log n)$  steps.  $\uparrow$   
assumes  $n$  is a power of 2

**Pf.**  $T(2n) = 2T(n) + O(n) \Rightarrow T(n) = O(n \log n)$ .



# Point-Value to Coefficient Representation: Inverse DFT

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

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

$\uparrow$  Inverse DFT                       $\uparrow$  Fourier matrix inverse  $(F_n)^{-1}$



# Inverse FFT

**Claim.** Inverse of Fourier matrix is given by following formula.

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

# Inverse FFT: Proof of Correctness

**Claim.**  $F_n$  and  $G_n$  are inverses.

**Pf.**

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} \underset{\substack{\uparrow \\ \text{summation lemma}}}{=} \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

**Summation lemma.** Let  $\omega$  be a principal  $n^{\text{th}}$  root of unity. Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

**Pf.**

If  $k$  is a multiple of  $n$  then  $\omega^k = 1 \Rightarrow$  sums to  $n$ .

Else:  $\omega^k \neq 1$

- $x^n - 1 = (x - 1)(1 + x + x^2 + \dots + x^{n-1})$
- Let  $x = \omega^k$ , we have  $0 = \omega^{kn} - 1 = (\omega^k - 1)(1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)})$
- Therefore,  $1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)} = 0 \Rightarrow$  sums to 0. ▀

# Point-Value to Coefficient Representation: Inverse DFT

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

**Observation.** Almost the same form as DFT.

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

# Inverse FFT: Algorithm

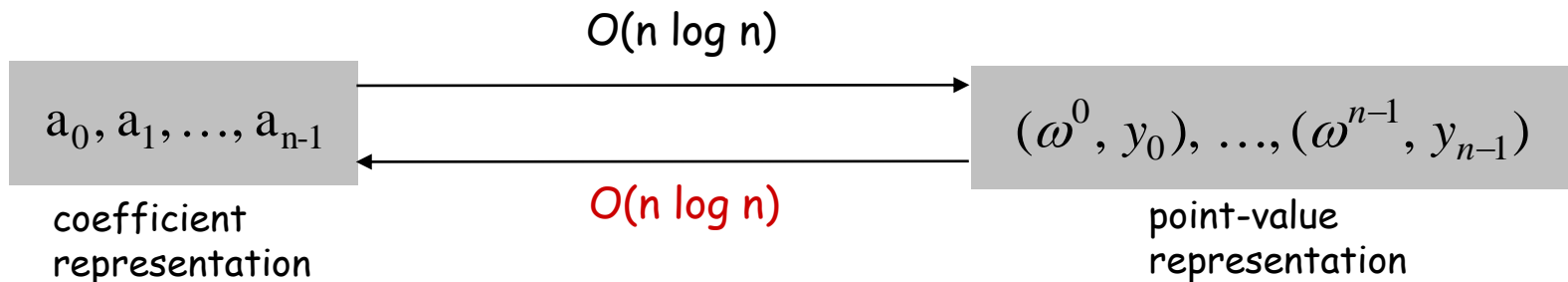
```
IFFT(n, a0, a1, ..., an-1) {  
    if (n == 1) return a0  
  
    (e0, e1, ..., en/2-1) ← IFFT(n/2, a0, a2, a4, ..., an-2)  
    (d0, d1, ..., dn/2-1) ← IFFT(n/2, a1, a3, a5, ..., an-1)  
  
    for k = 0 to n/2 - 1 {  
        ωk ← e-2πik/n  
        yk      ← (ek + ωk dk)  
        yk+n/2 ← (ek - ωk dk)  
    }  
  
    return (y0, y1, ..., yn-1)  
}
```

Note. Need to divide the final result by n.

# Inverse FFT Summary

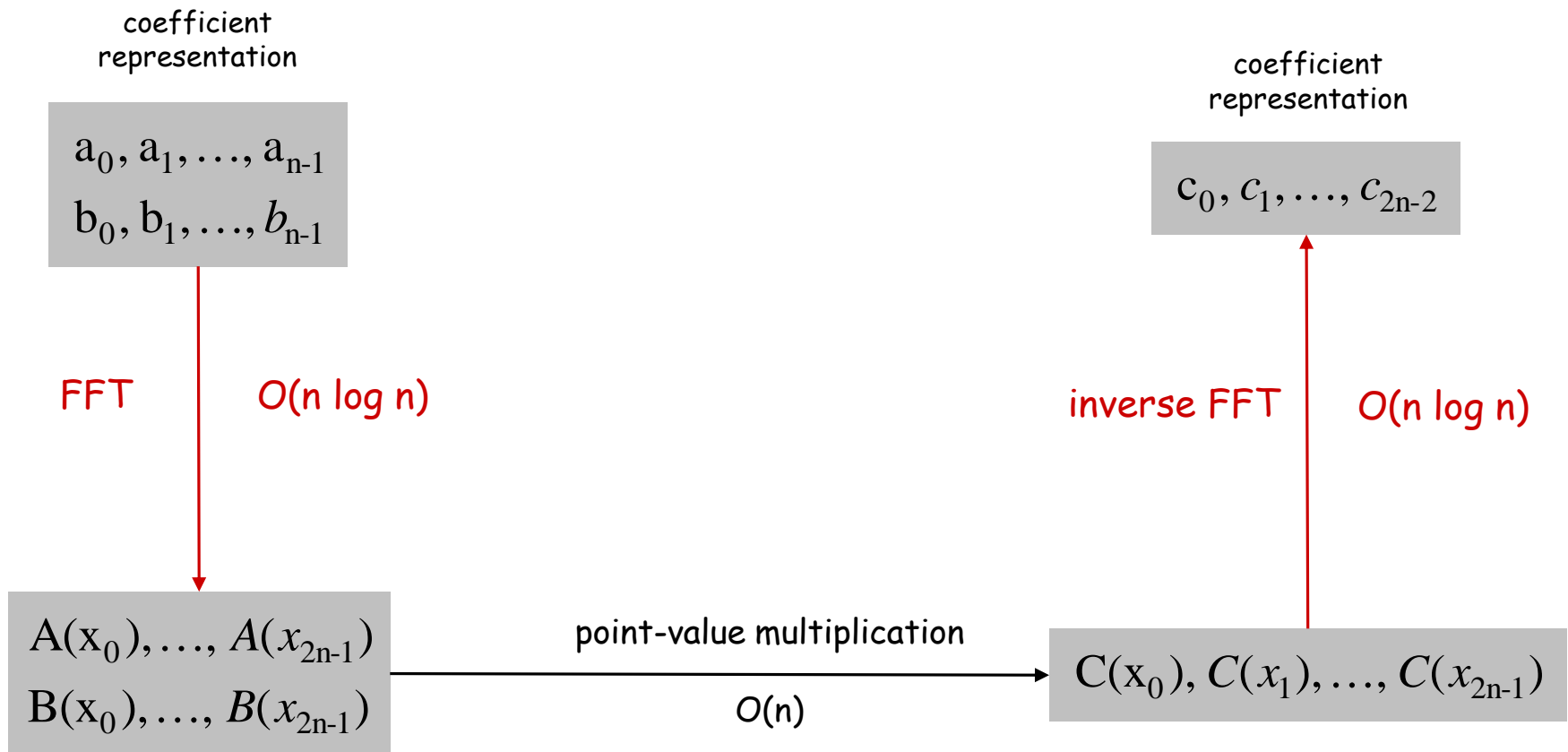
**Theorem.** Inverse FFT algorithm interpolates a degree  $n-1$  polynomial given values at each of the  $n^{\text{th}}$  roots of unity in  $O(n \log n)$  steps.

↑  
assumes  $n$  is a power of 2



# Polynomial Multiplication

**Theorem.** Can multiply two degree  $n-1$  polynomials in  $O(n \log n)$  steps.



# Integer Multiplication

**Integer multiplication.** Given two  $n$  bit integers  $a = a_{n-1} \dots a_1 a_0$  and  $b = b_{n-1} \dots b_1 b_0$ , compute their product  $c = a \times b$ .

**Convolution algorithm.**

Form two polynomials.

Note:  $a = A(2)$ ,  $b = B(2)$ .

Compute  $C(x) = A(x) \times B(x)$ .

Evaluate  $C(2) = a \times b$ .

Running time:  $O(n \log n)$  complex arithmetic steps.

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

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

**Practice.** [GNU Multiple Precision Arithmetic Library] *GMP* proclaims to be "the fastest bignum library on the planet." It uses brute force, Karatsuba, and FFT, depending on the size of  $n$ .

# Divide-and-Conquer: Chapter Summary

---



# Divide-and-Conquer

## Basic idea

Break up problem into several parts.

Solve each part recursively.

Combine solutions to sub-problems into overall solution.

## Algorithms

Mergesort

- Divide a sequence into two of same size

Closest Pair of Points

- Vertically divide the space

Integer Multiplication

- Divide each  $n$ -digit integer into two  $\frac{1}{2}n$ -digit integers

Matrix Multiplication

- Divide each  $n$ -by- $n$  matrix into four  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks

Fast Fourier Transform

- Divide a polynomial into two with even and odd powers