# Module 1
## General Techniques

Ira A. Fulton Schools of Engineering
Arizona State University

# Objectives

- Review the basics of divide-and-conquer technique
- Demonstrate general techniques of divide-and-conquer

# General Techniques

# Divide-and-Conquer Method

## Divide-and-conquer method.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

> Divide et impera.
> Veni, vidi, vici.
> \- *Julius Caesar*

## Most common usage.

- Break up problem of size n into two equal parts of size ½n.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

## Consequence.

- Brute force: $n^2$
- Divide-and-conquer: $n \log n$

# Sorting

Sorting: Given n elements, rearrange them in ascending order.

Obvious sorting applications.

- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

Problems become easier once sorted.

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

Non-obvious sorting applications.

- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

. . .

# Mergesort

Mergesort.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

A L G O R I T H M S

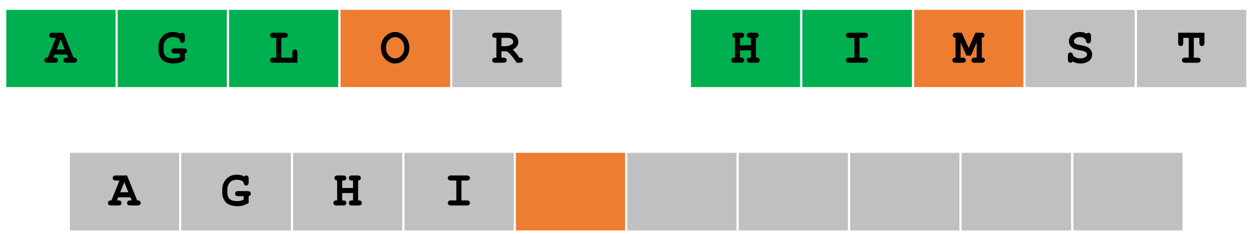
A L G O R    I T H M S    divide    O(1)

A G L O R    H I M S T    sort    2T(n/2)

A G H I L M O R S T    merge    O(n)

# Merging

Merging. Combine two pre-sorted lists into a sorted whole.



# A Useful Recurrence Relation

- Define T(n) = number of comparisons to mergesort an input of size n.

- Mergesort recurrence.

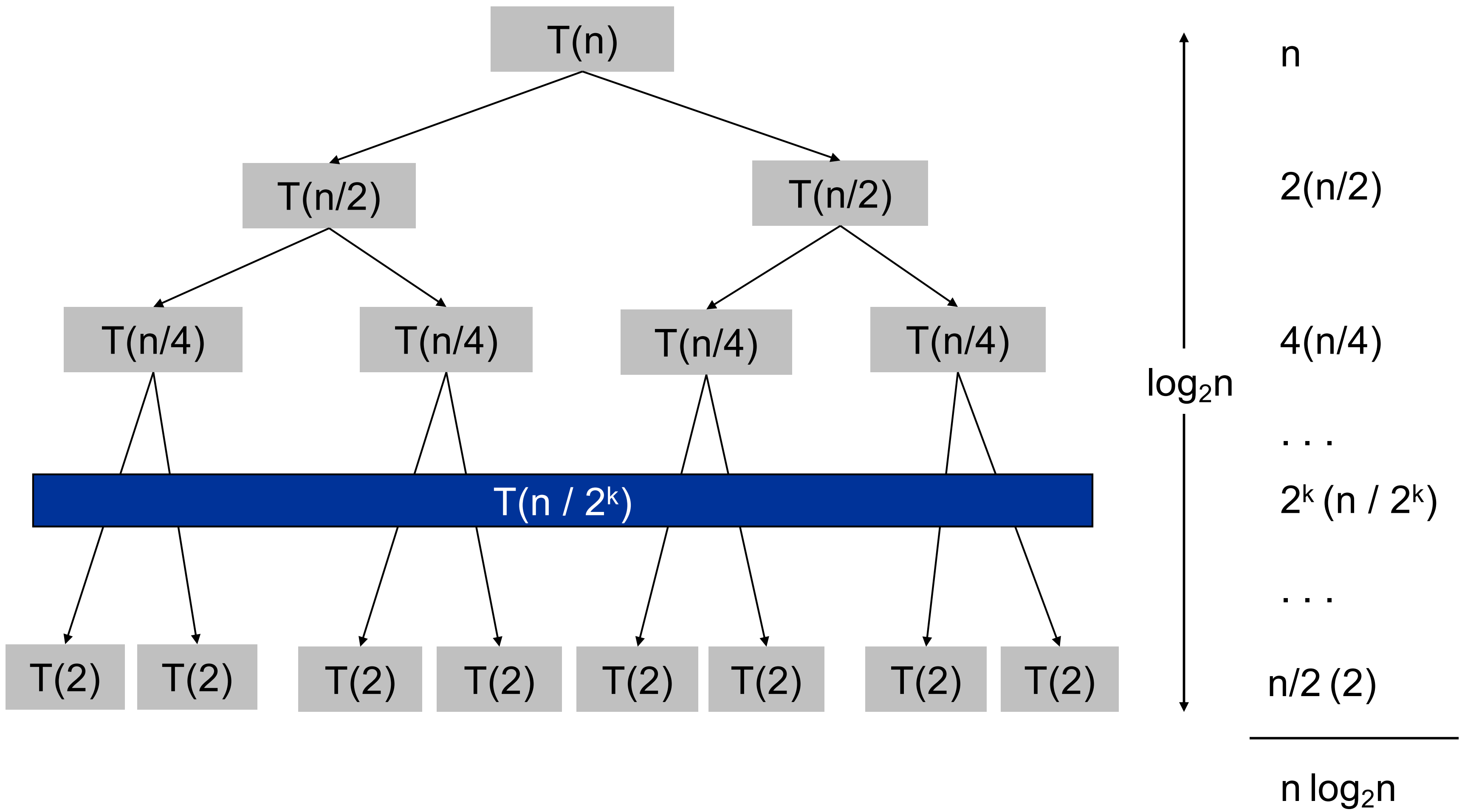
$$\mathrm{T}(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T\left(\lceil n/2 \rceil\right)}_{\text{solve left half}} + \underbrace{T\left(\lfloor n/2 \rfloor\right)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Solution. T(n) = O(n $\log_2$ n).

- Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace ≤ with =.

# 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 Telescoping

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

assumes n is a power of 2

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

- Proof. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

# Proof by Induction

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

assumes n is a power of 2

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

- Proof. (by induction on n)
  - Base case: n = 1.
  - Inductive hypothesis: $T(n) = n \log_2 n$.
  - Goal: show that $T(2n) = 2n \log_2 (2n)$.

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

# Analysis of Mergesort Recurrence

- Claim. If T(n) satisfies the following recurrence, then $T(n) \le n \lceil \lg n \rceil$.

$\log_2 n$

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Proof. (by induction on n)
  - Base case: n = 1.
  - Define $n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$.
  - Induction step: assume true for 1, 2, ... , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\le \lceil \lg n \rceil - 1 \end{aligned}$$

# Module 2

## Closest Pair of Points

Ira A. Fulton Schools of Engineering

Arizona State University

# Objectives

- Explain the closest pair of points algorithm
- Demonstrate the algorithm with an example

# Closest Pair of Points

# Closest Pair of Points

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

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

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

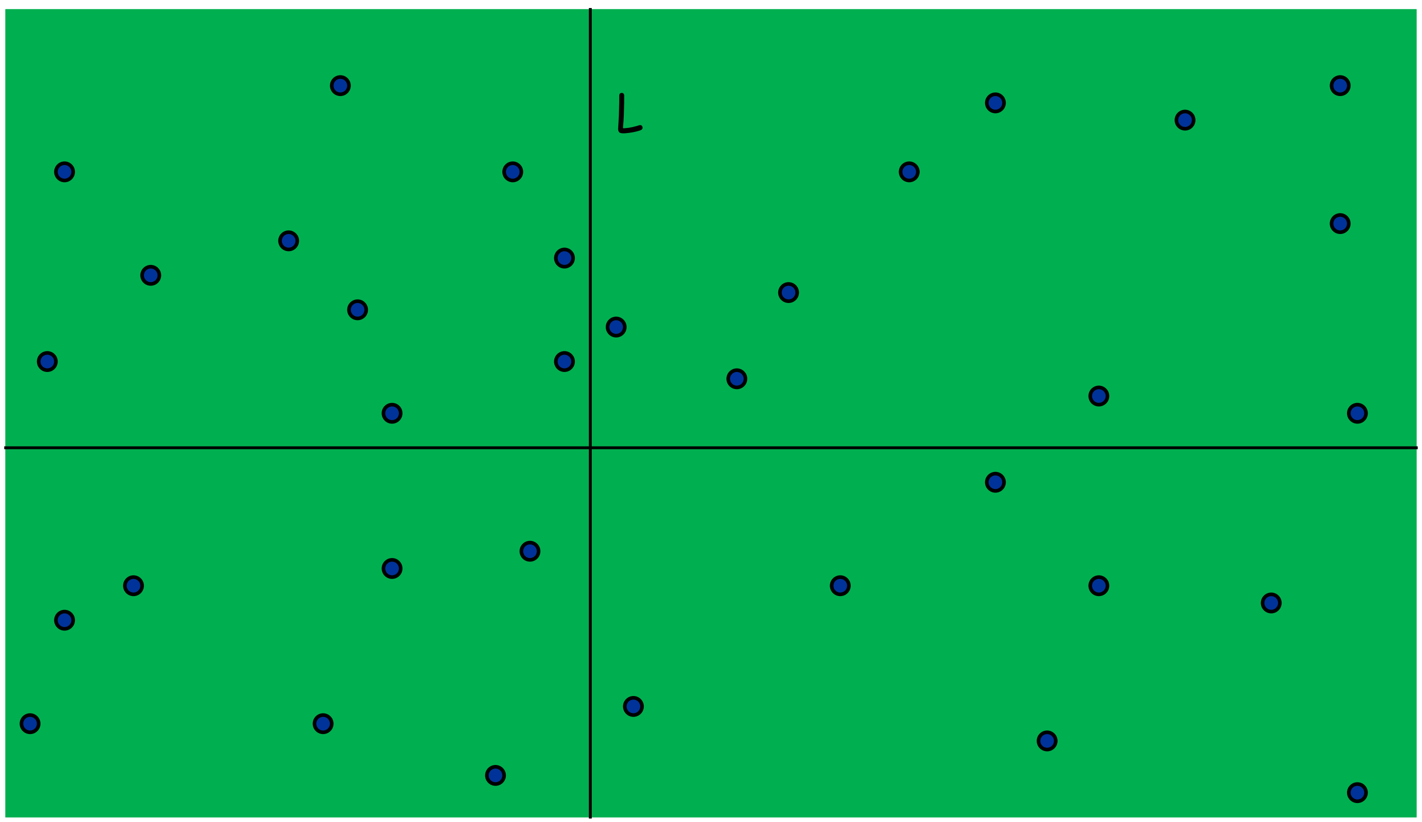
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

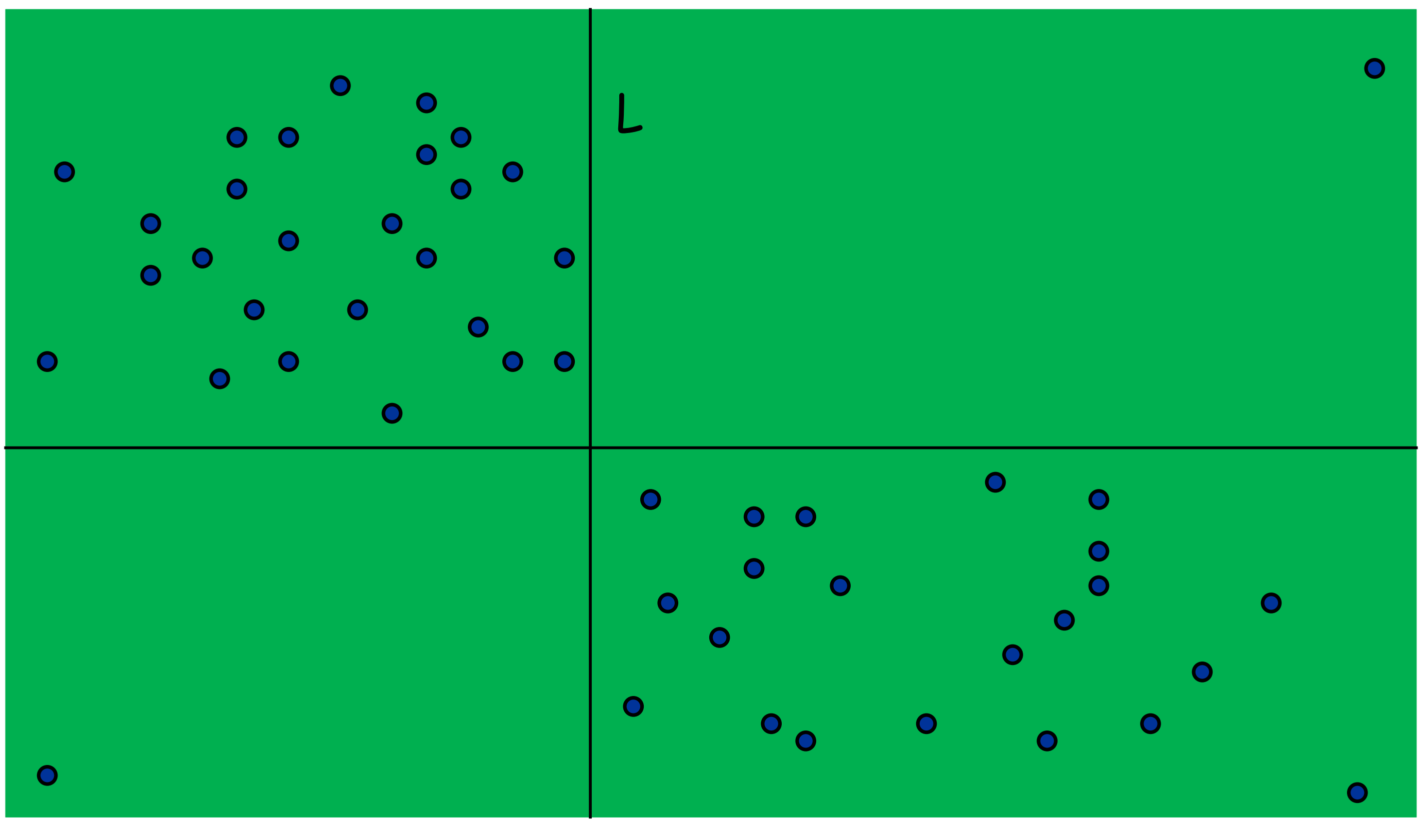
# Closest Pair of Points:  First Attempt

- Sub-divide region into 4 quadrants.



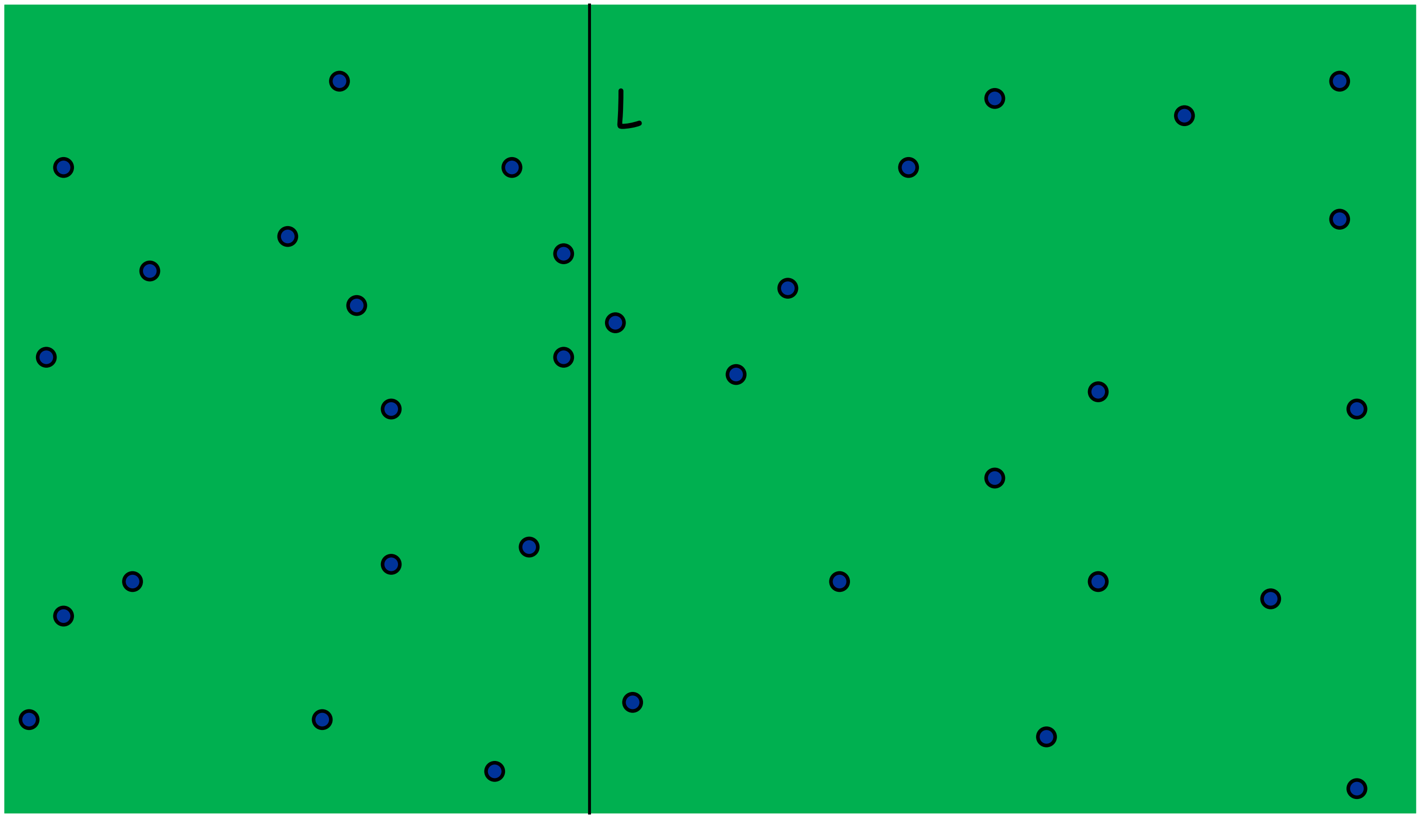
# Closest Pair of Points:  First Attempt

- Sub-divide region into 4 quadrants.
- Impossible to ensure n/4 points in each piece.



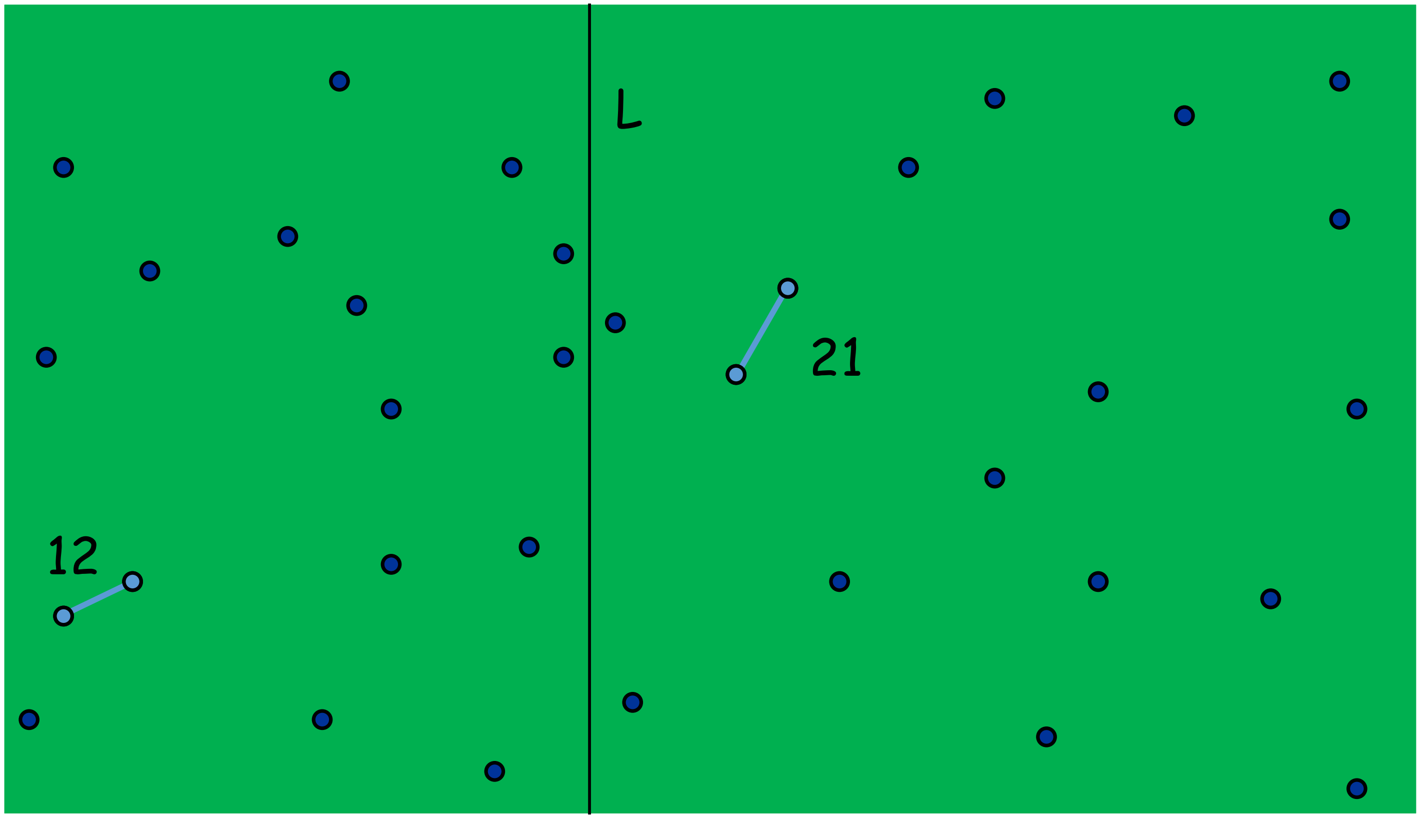
# Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly n/2 points on each side.



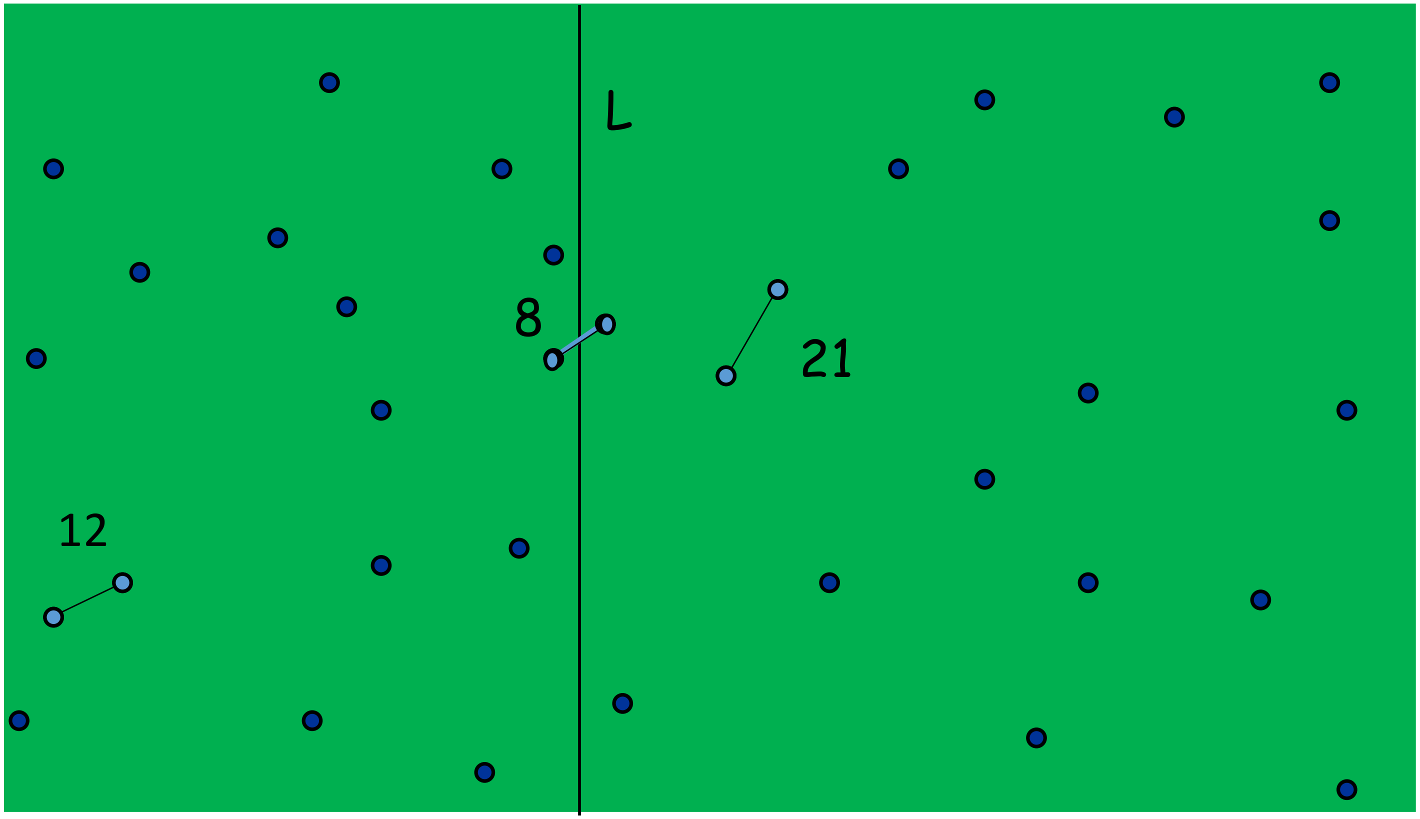
# Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.
  - Conquer: find closest pair in each side recursively.



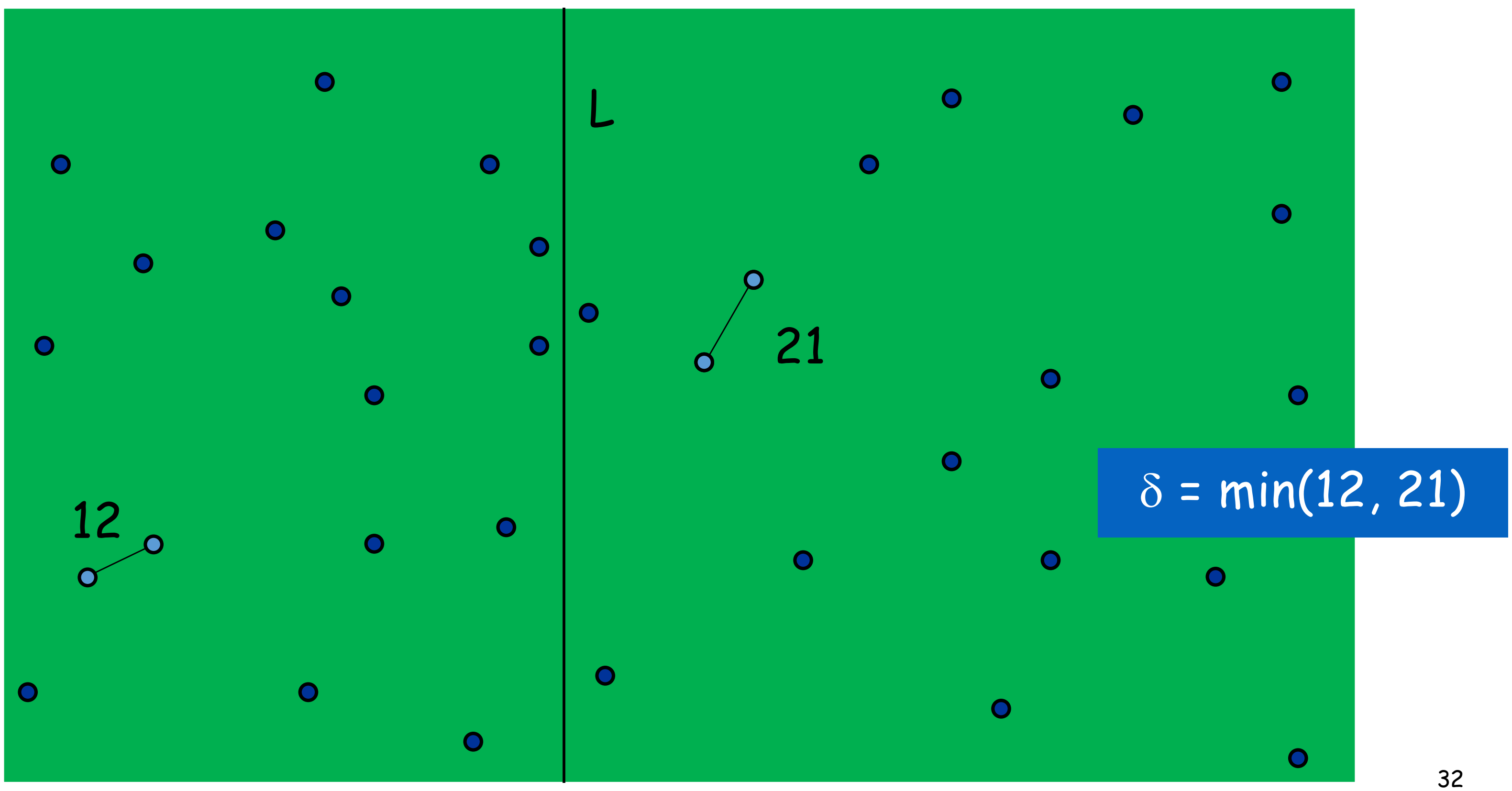
# Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.
  - Conquer: find closest pair in each side recursively.
  - Combine: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
  - Return best of 3 solutions.



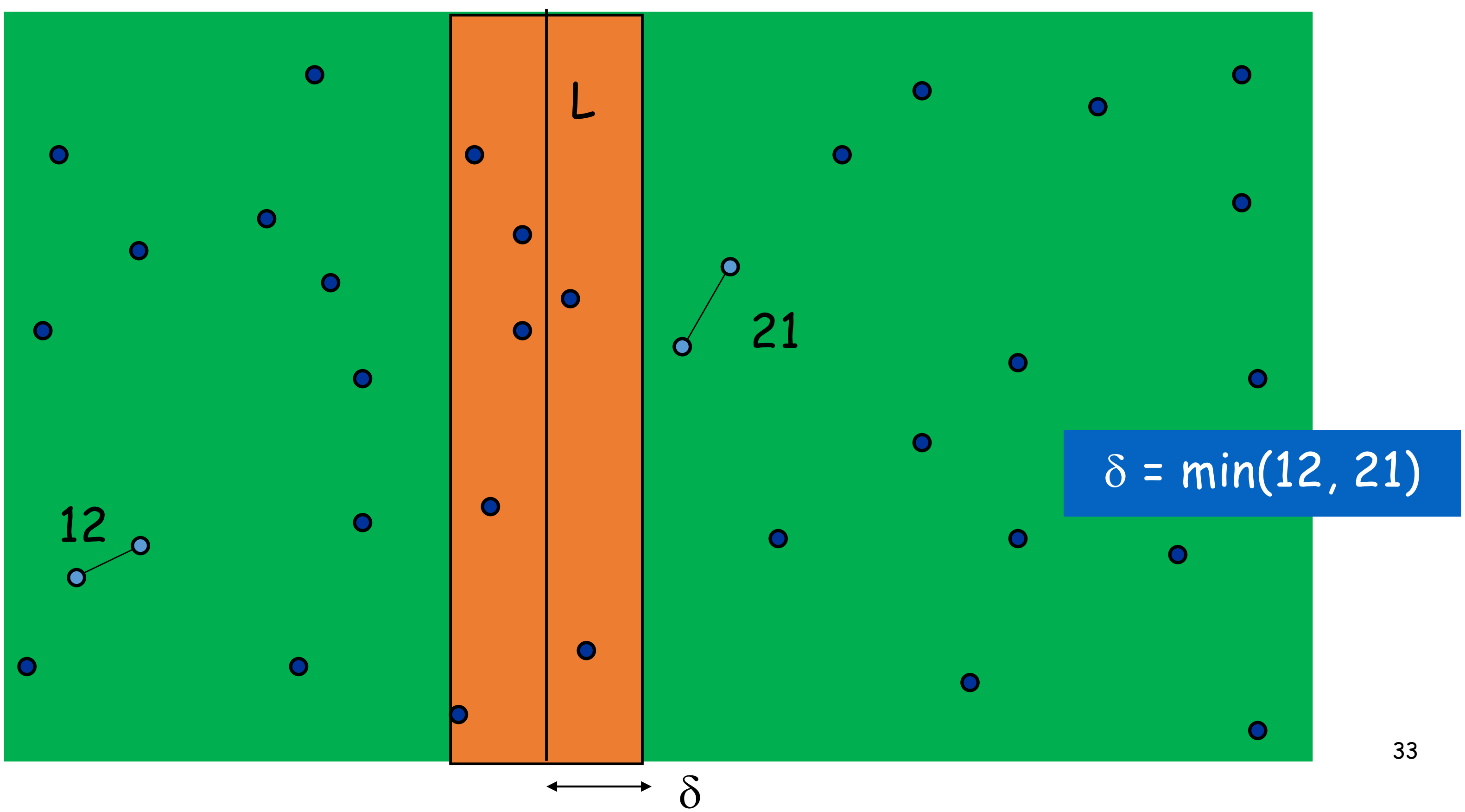
# Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.



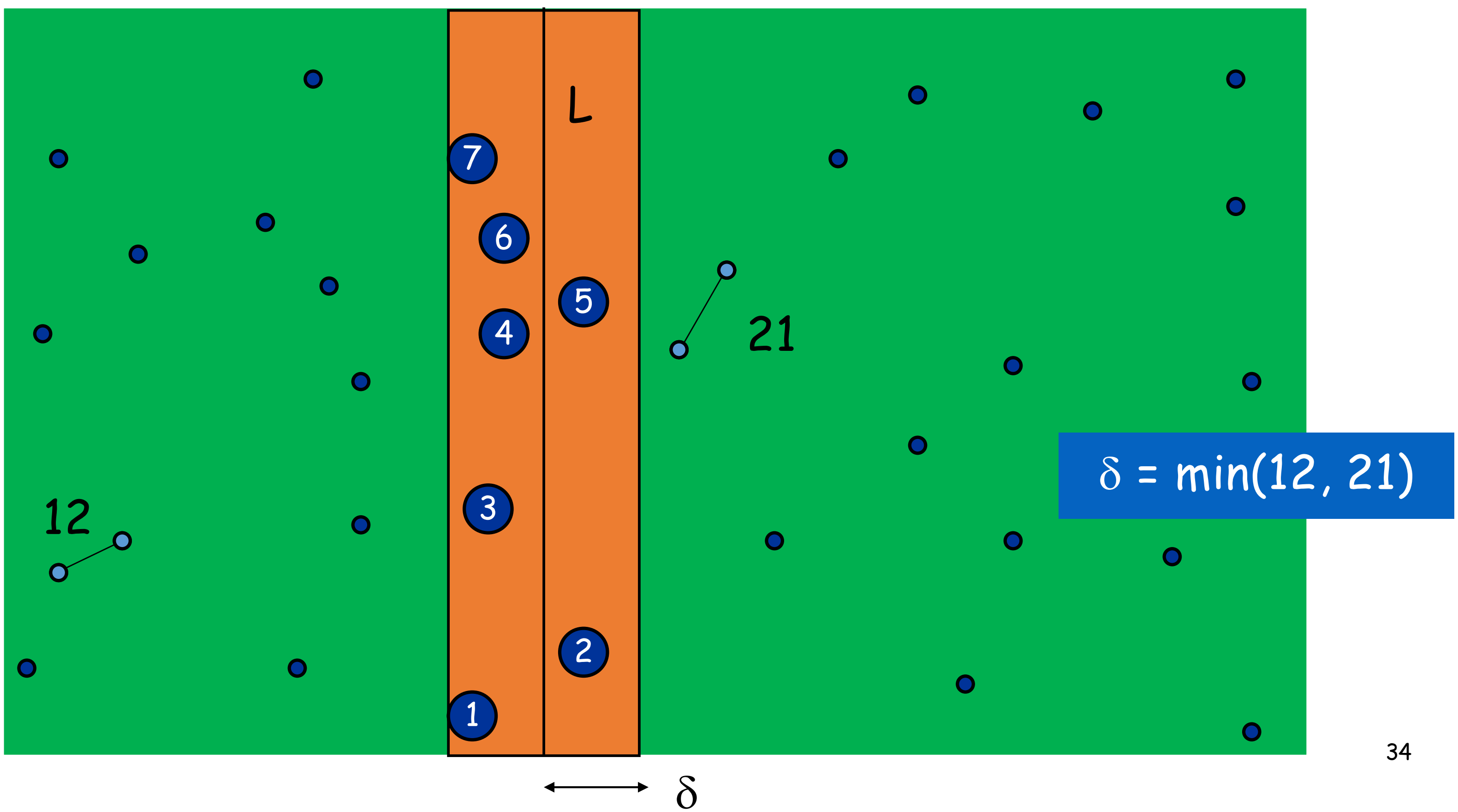
# Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < $\delta$.
  - ✓ Only need to consider points within $\delta$ of line L.



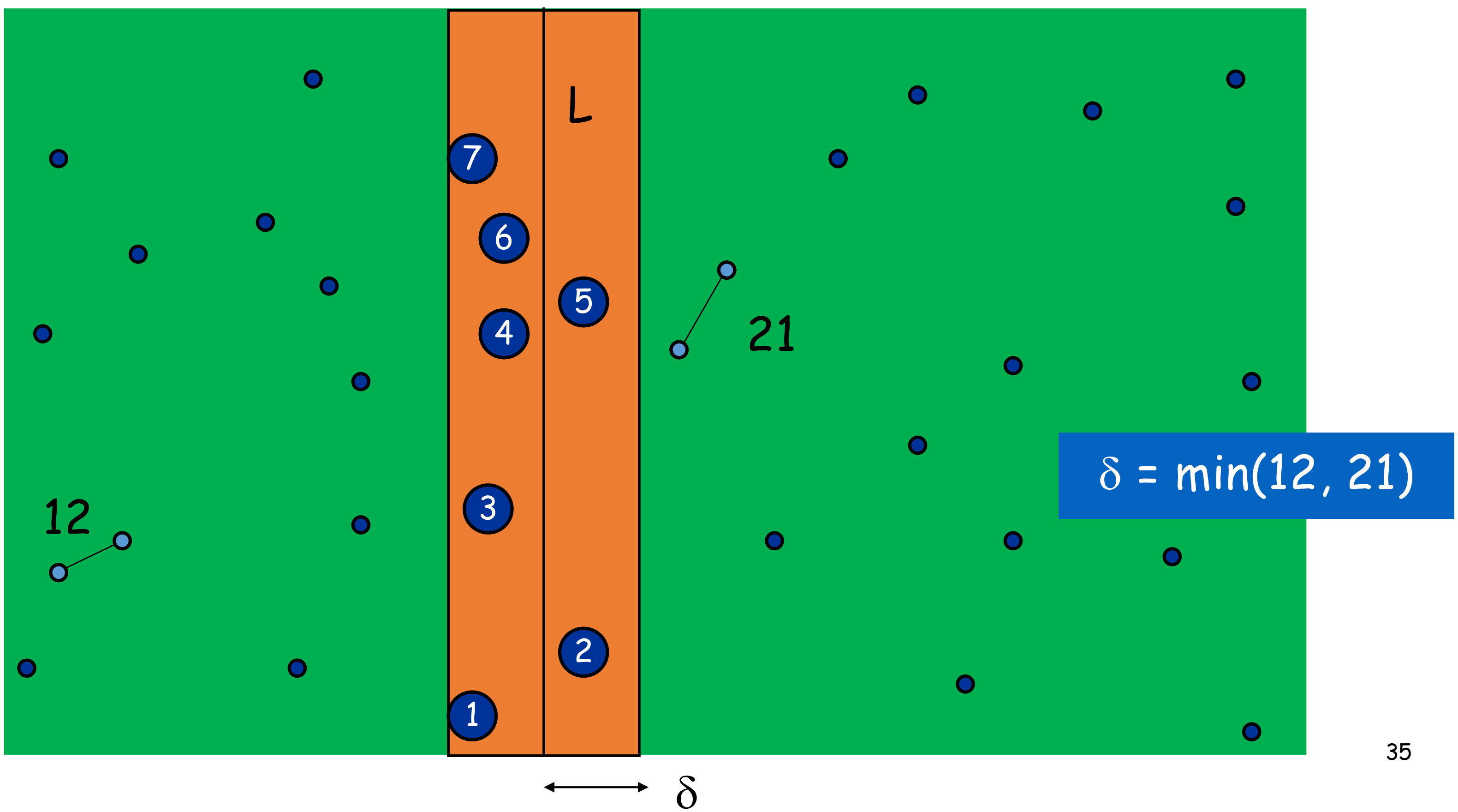
# Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.
  - ✓ Only need to consider points within δ of line L.
  - ✓ Sort points in 2δ-strip by their y coordinate.



# Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.
  - ✓ Only need to consider points within δ of line L.
  - ✓ Sort points in 2δ-strip by their y coordinate.
  - ✓ Only check distances of those within 11 positions in sorted list!



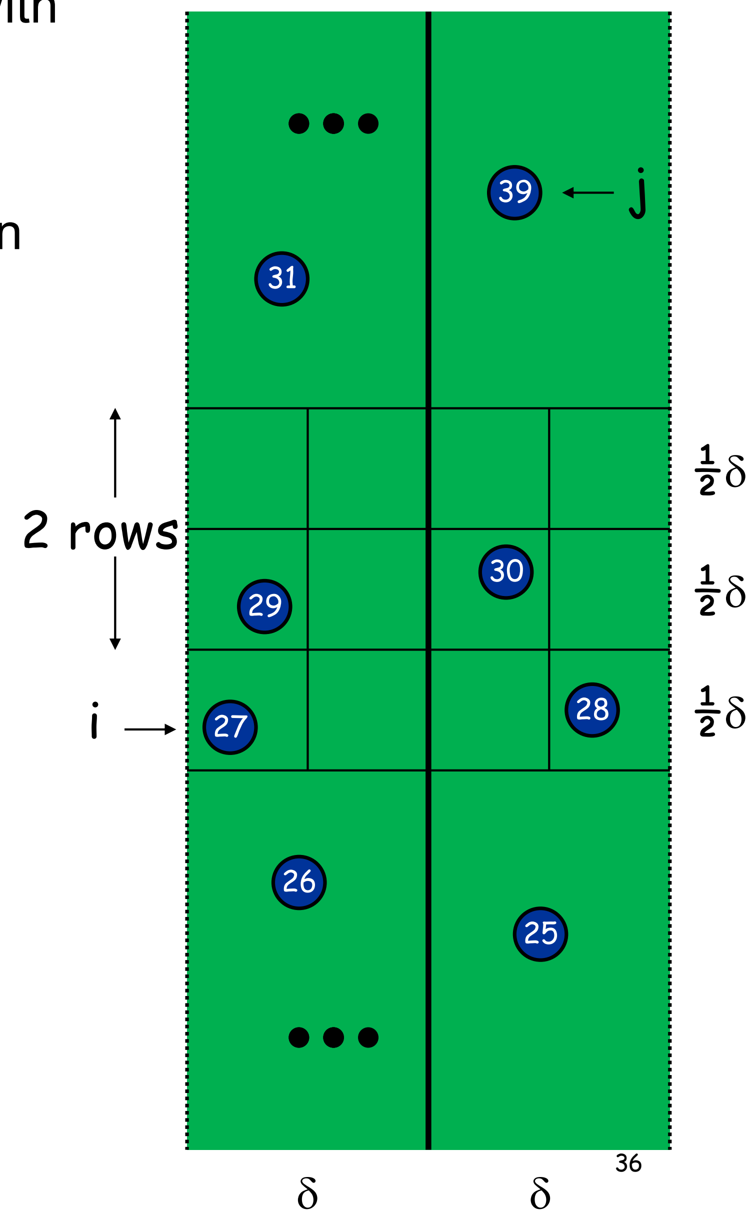
# Closest Pair of Points

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

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

- Proof.
  - 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 we replace 12 with 7.



# Closest Pair of Points Algorithm

```
Closest-Pair(p1, …, pn) {
   If n ≤ 1 return ∞
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Let S be the set of points at distance at most δ from
   separation line L.

   Sort S 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 δ, update δ.

   return δ.
}
```

O(n log n)

2T(n / 2)

O(n)

O(n log n)

O(n)

# Closest Pair of Points Analysis

Running time.

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

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

- Yes. Always keep two sorted lists of a set of points P, P_x and P_y, sorted by x- and y-coordinates, respectively.
- Before each recursive call, precompute R_x, R_y, L_x, and L_y (where L and R are the points to the left and to the right of the separation line, respectively) in O(n) time. How?
- Also compute S_y in O(n) time. How?

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

# Module 3

## Integer Multiplication and Matrix Multiplication

Ira A. Fulton Schools of Engineering
Arizona State University

# Objectives

- Explain Karatsuba's integer multiplication algorithm
- Explain Strassen's fast matrix multiplication algorithm

# Integer Multiplication and Matrix 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 × b.
  - Brute force solution: $\Theta(n^2)$ bit operations.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | | | | | | | * | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | |
| | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | |
| | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | | |
| | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | | | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | | | |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Multiply

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

Add

# Divide-and-Conquer Multiplication: Warmup

- To multiply two n-digit integers:
  - Multiply four n/2-digit integers.
  - Add two n/2-digit integers, and shift to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

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

assumes n is a power of 2

# Karatsuba Multiplication

- To multiply two n-digit integers:
  - Add two n/2-digit integers.
  - Multiply three n/2-digit integers.
  - Add, subtract, and shift n/2-digit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot \left( x_1 y_0 + x_0 y_1 \right) + x_0 y_0 \\
&= 2^{n} \cdot \underbrace{x_1 y_1}_{A} + 2^{n/2} \cdot \left( \underbrace{(x_1 + x_0)(y_1 + y_0)}_{B} - \underbrace{x_1 y_1}_{A} - \underbrace{x_0 y_0}_{C} \right) + \underbrace{x_0 y_0}_{C}
\end{aligned}
$$

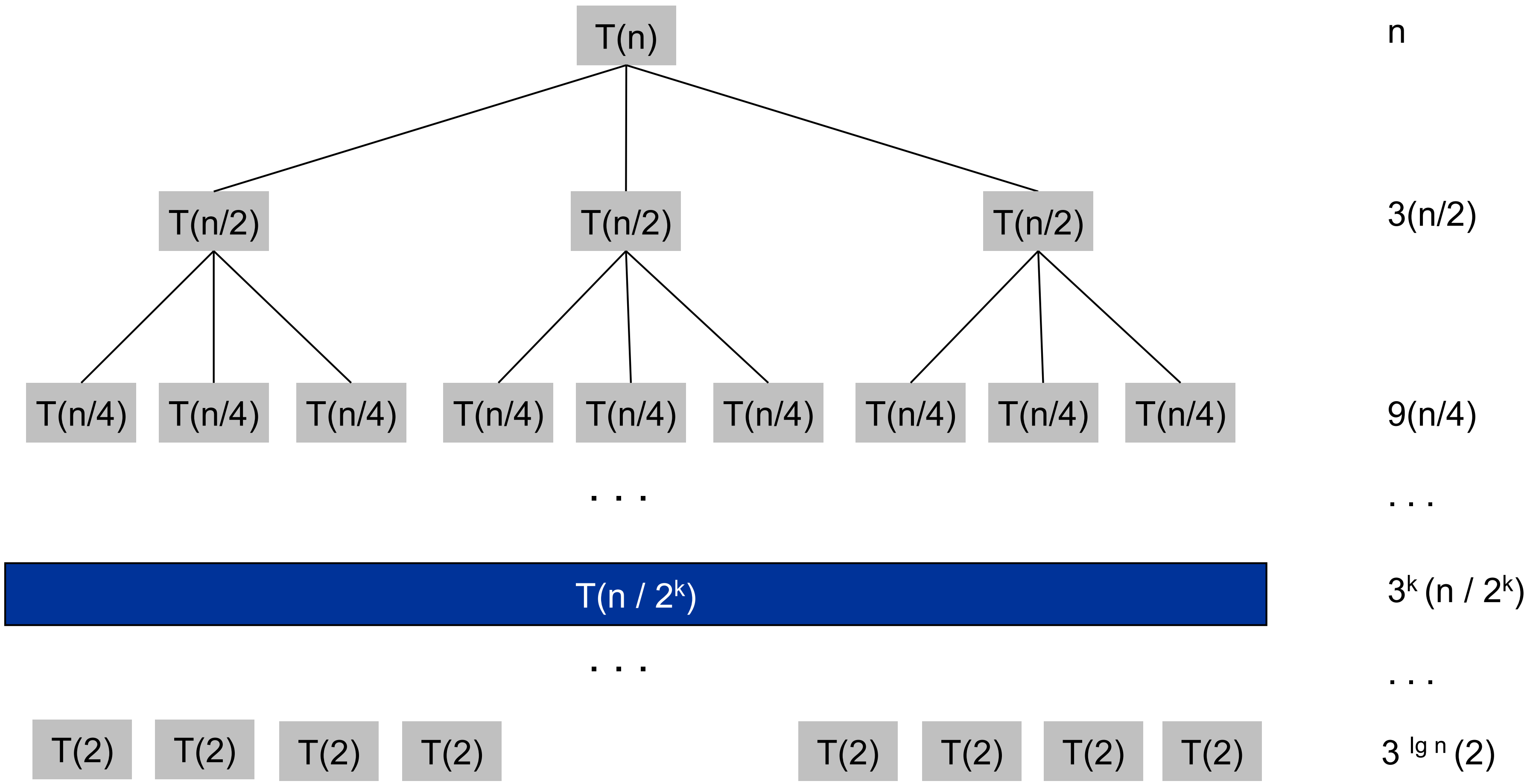
- Theorem [Karatsuba-Ofman, 1962]. One can multiply two n-digit integers in $O(n^{1.585})$ bit operations.

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

# Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

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



# Matrix Multiplication

- Matrix multiplication. Given two n x 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 n/2 x n/2 blocks.
- Conquer: perform 8 (n/2 x n/2) matrix multiplications recursively.
- Combine: add appropriate products using 4 matrix additions.

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

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

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

# Matrix Multiplication: Key Idea

Key idea. Multiply 2 x 2 block matrices with only 7 multiplications.

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).

# Fast Matrix Multiplication

- Fast matrix multiplication (Strassen, 1969).
  - Divide: partition A and B into n/2 x n/2 blocks.
  - Compute: 14 n/2 x n/2 matrices via 10 matrix additions.
  - Conquer: perform 7 n/2 x n/2 matrix multiplications recursively.
  - Combine: 7 products into 4 terms using 8 matrix additions.

- Analysis.
  - Assume n is a power of 2.
  - T(n) = # arithmetic operations.

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

# Fast Matrix Multiplication in Practice

- Implementation issues

  Sparsity.

  Caching effects.

  Numerical stability.

  Odd matrix dimensions.

  Crossover to classical algorithm around n = 128.

- Controversies
  - Is Strassen only a theoretical curiosity?
  - Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when n ~ 2,500.
  - Range of instances where it is useful is a subject of controversy.

- Remark. Can "Strassenize" Ax=b, determinant, eigenvalues, 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] $\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$

Decimal wars.

- December, 1979: $O(n^{2.521813})$
- January, 1980: $O(n^{2.521801})$

# Fast Matrix Multiplication in Theory

- Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987]
- Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.
- Caveat. Theoretical improvements to Strassen are progressively less practical.