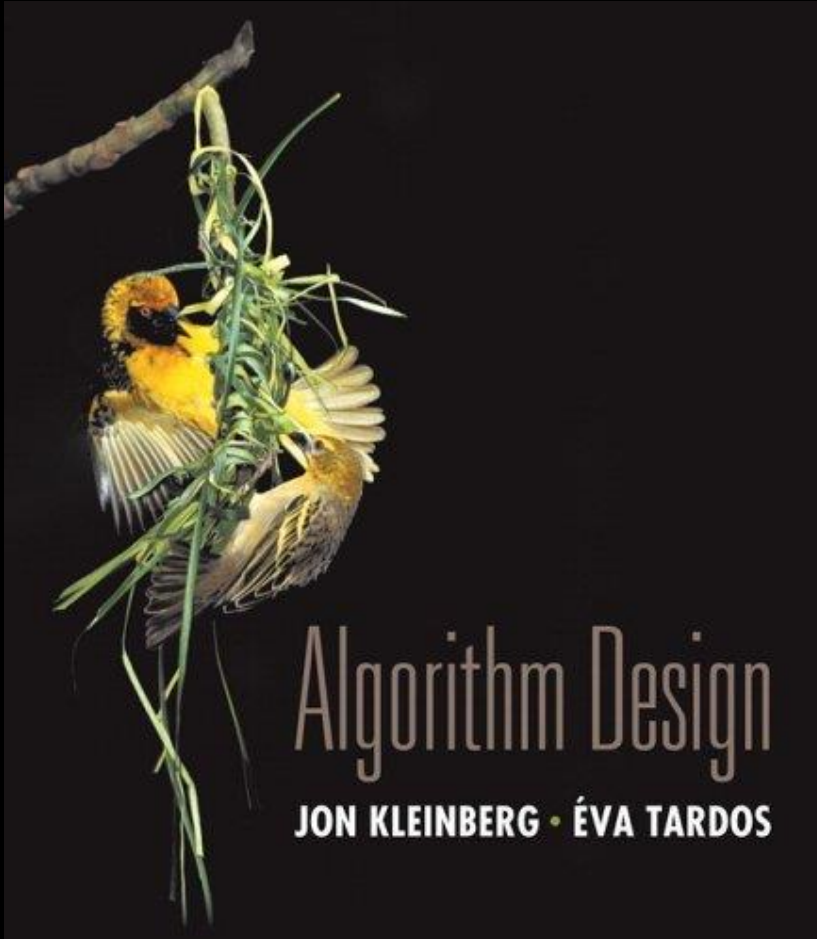


Chapter 4

Divide and Conquer



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

Divide-and-Conquer

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

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 $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

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

5.1 Mergesort

Sorting

Sorting. Given n elements, rearrange 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.

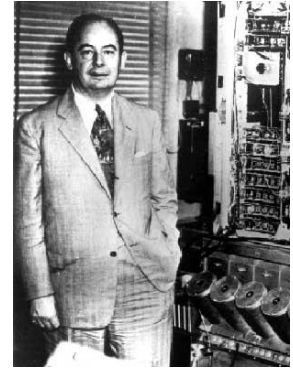
- Load balancing on a parallel computer.

- ...

Mergesort

Mergesort.

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



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 $O(1)$

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 $O(n)$

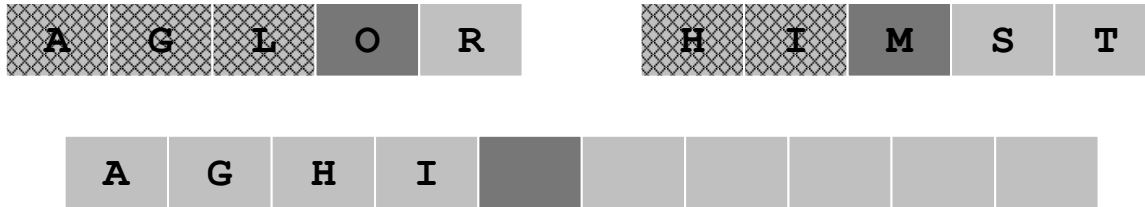
Merging

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

How to merge efficiently?



- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrud, 1969]

↑
using only a constant amount of extra storage

A Useful Recurrence Relation

Def. $T(n)$ = number of comparisons to mergesort an input of size n .

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

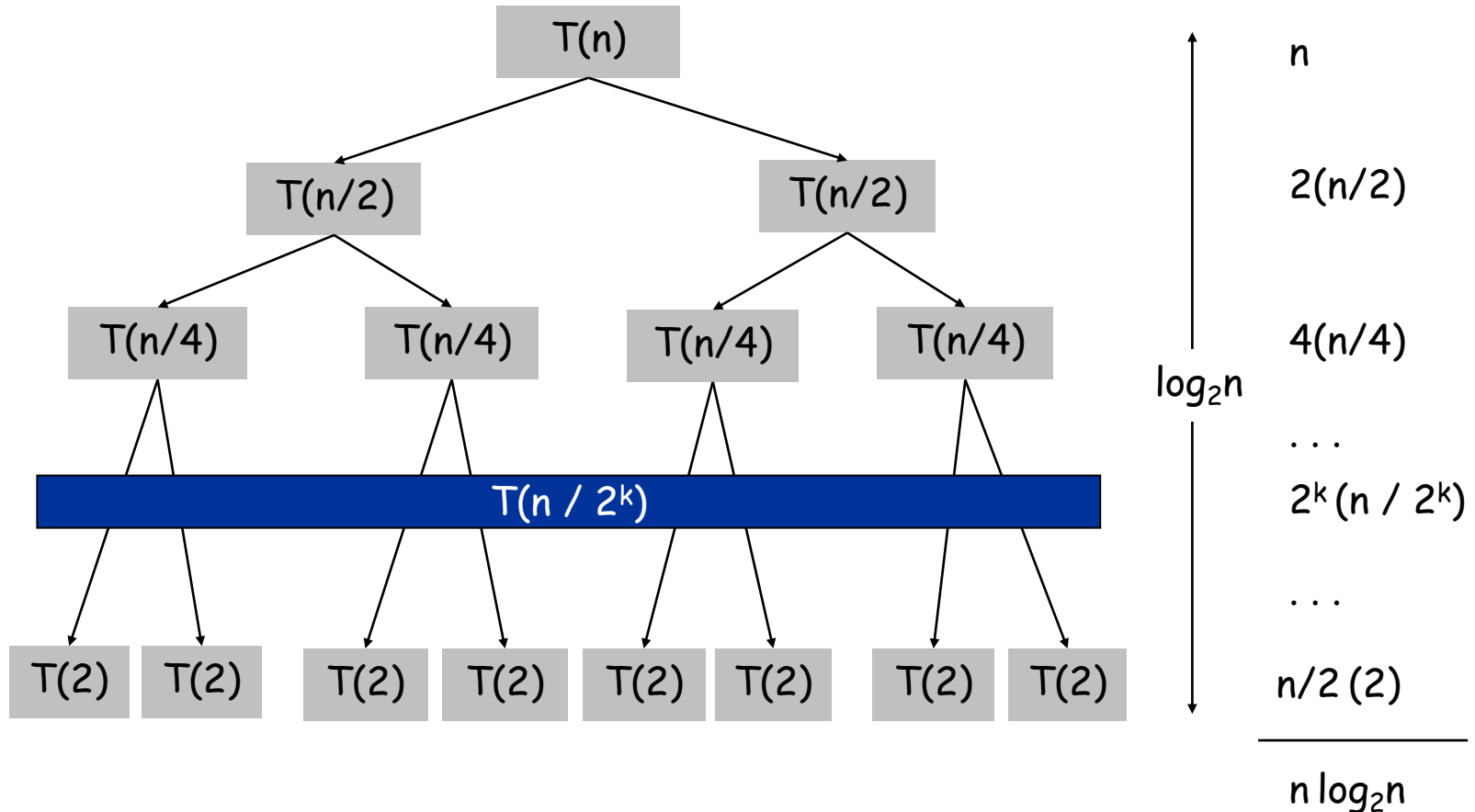
↖ between $\lfloor n/2 \rfloor$ and $n - 1$ compares

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 \leq with $=$.

Proof by Recursion Tree

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



Proof by Induction

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

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

↑
assumes n is a power of 2

Pf. (by induction on n)

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

$$\begin{aligned} T(2n) &= 2T(n) + 2n && \text{recurrence} \\ &\stackrel{\text{inductive hypothesis}}{\longrightarrow} = 2n \log_2 n + 2n \\ &= 2n (\log_2 (2n) - 1) + 2n \\ &= 2n \log_2 (2n). \quad \blacksquare \end{aligned}$$

Analysis of Mergesort Recurrence

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

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

\uparrow
 $\log_2 n$

Pf. (by induction on n)

- n Base case: $n = 1$.
- n Define $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- n Induction step: assume true for $1, 2, \dots, n-1$.

$$\begin{aligned}
 T(n) &\leq T(n_1) + T(n_2) + n \\
 \text{inductive hypothesis} \longrightarrow &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
 &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
 &= n \lceil \log_2 n_2 \rceil + n \\
 &\leq n (\lceil \log_2 n \rceil - 1) + n \longleftarrow \log_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\
 &= n \lceil \log_2 n \rceil. \quad \blacksquare
 \end{aligned}$$

$$\begin{aligned}
 n_2 &= \lceil n/2 \rceil \\
 &\leq \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\
 &= 2^{\lceil \log_2 n \rceil} / 2
 \end{aligned}$$

$\log_2 n_2 \leq \lceil \log_2 n \rceil - 1$
 \uparrow
 an integer

5.3 Counting Inversions

Counting Inversions


Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with **similar** tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \dots, n$.
- Your rank: a_1, a_2, \dots, a_n .
- Songs i and j **inverted** if $i < j$, but $a_i > a_j$.

Songs					
	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5



Inversions
3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j .

Applications

Applications.

- n Voting theory.
- n Collaborative filtering.
- n Measuring the "sortedness" of an array.
- n Sensitivity analysis of Google's ranking function.
- n Rank aggregation for meta-searching on the Web.
- n Nonparametric statistics (e.g., Kendall's Tau distance).

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

ⁿ **Divide**: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer**: recursively count inversions in each half.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Conquer: $2T(n / 2)$

5 blue-blue inversions

8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine**: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

5 blue-blue inversions

8 green-green inversions

Conquer: $2T(n / 2)$

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

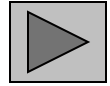
Combine: ???

Total = $5 + 8 + 9 = 22$.

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is **sorted**.
- Count inversions where a_i and a_j are in different halves.
- Merge** two sorted halves into sorted whole.



to maintain sorted invariant

3	7	10	14	18	19
---	---	----	----	----	----

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

Count: $O(n)$

2	3	7	10	11	14	16	17	18	19	23	25
---	---	---	----	----	----	----	----	----	----	----	----

Merge: $O(n)$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {  
    if list L has one element  
        return 0 and the list L  
  
    Divide the list into two halves A and B  
    ( $r_A$ , A)  $\leftarrow$  Sort-and-Count(A)  
    ( $r_B$ , B)  $\leftarrow$  Sort-and-Count(B)  
    ( $r$ , L)  $\leftarrow$  Merge-and-Count(A, B)  
  
    return  $r = r_A + r_B + r$  and the sorted list L  
}
```

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.

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

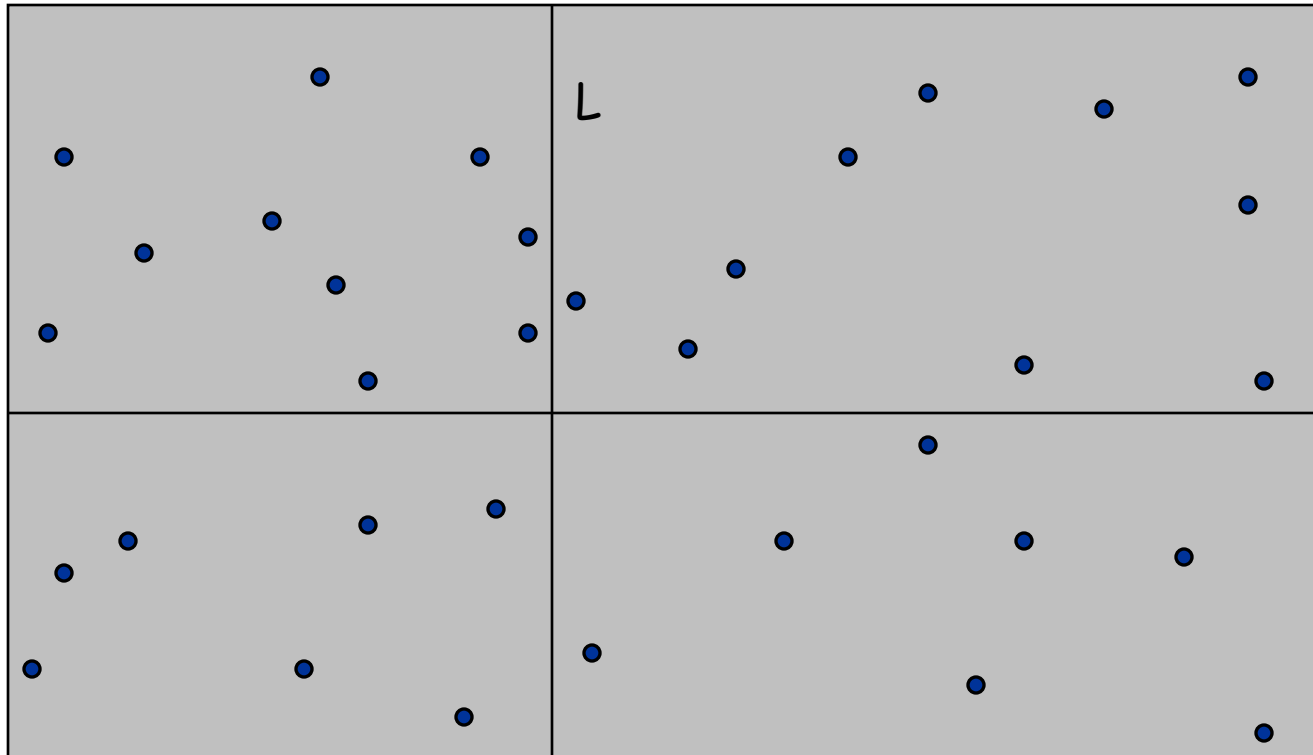
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

↑
to make presentation cleaner

Closest Pair of Points: First Attempt

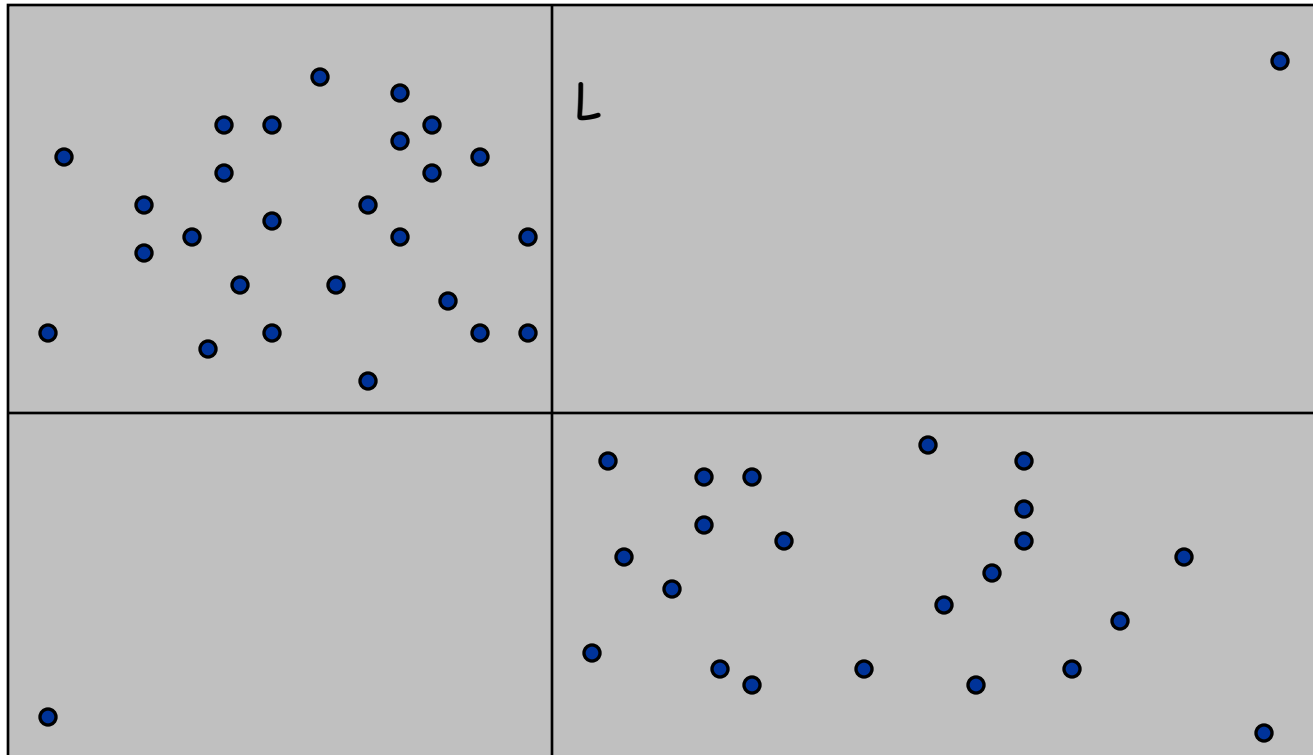
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

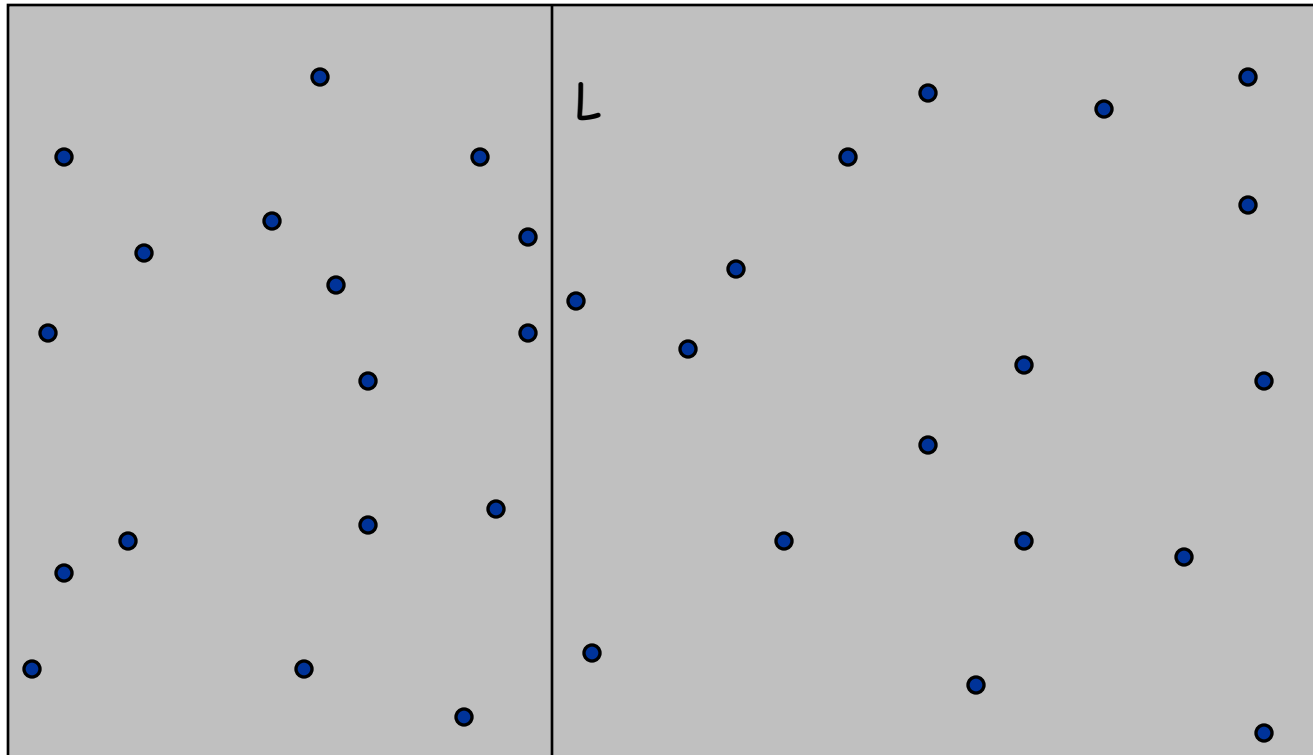
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest Pair of Points

Algorithm.

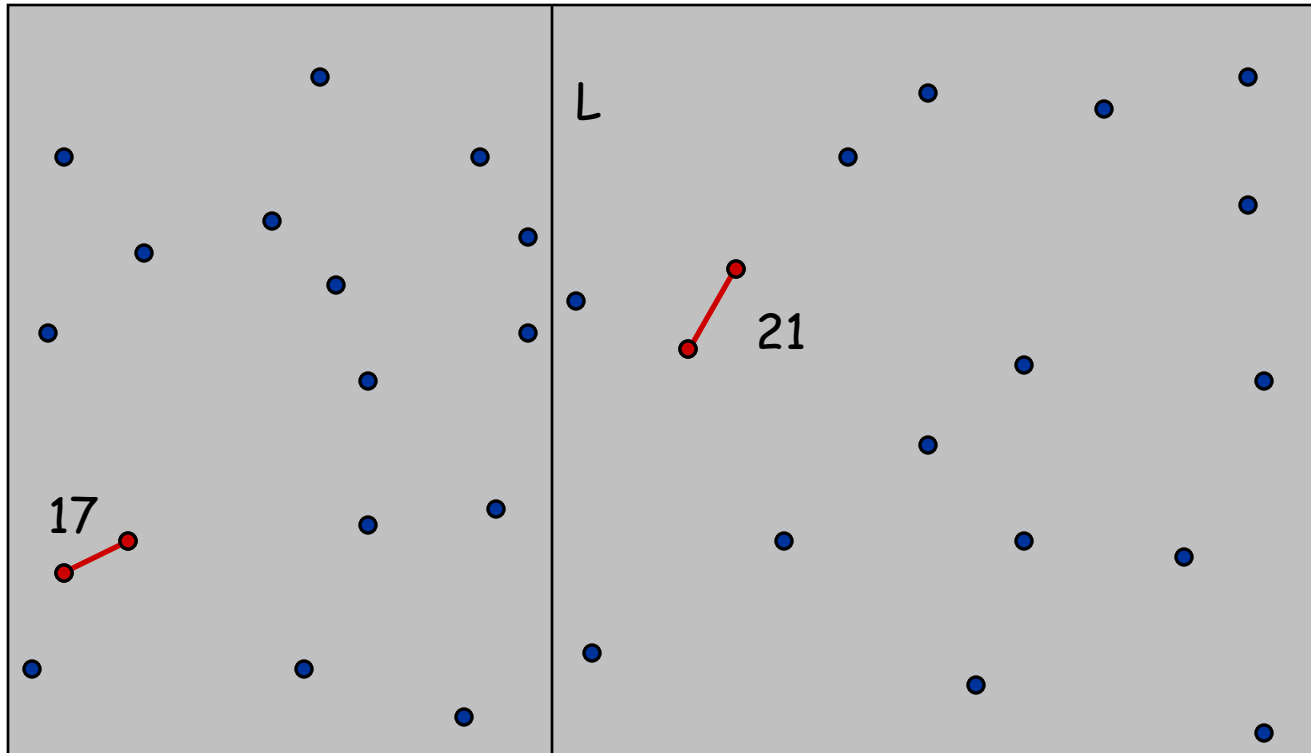
n **Divide:** draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

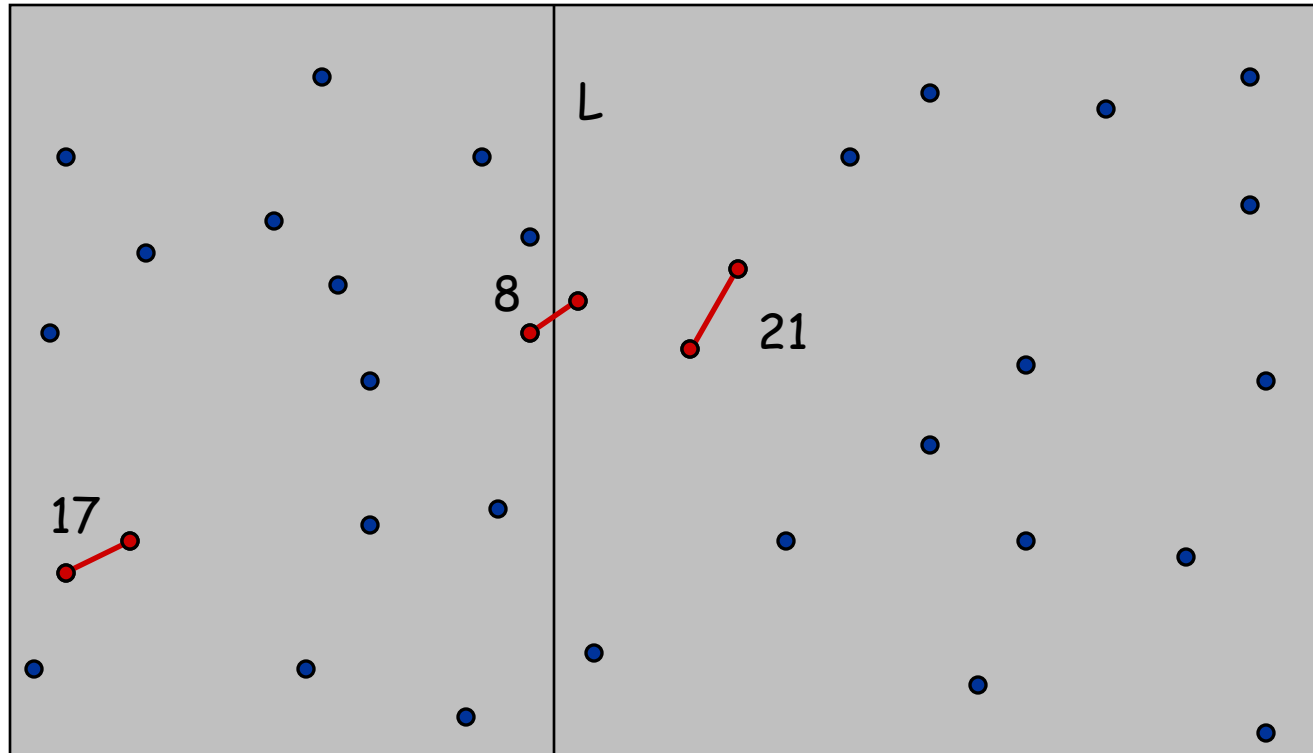
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer:** find closest pair in each side recursively.



Closest Pair of Points

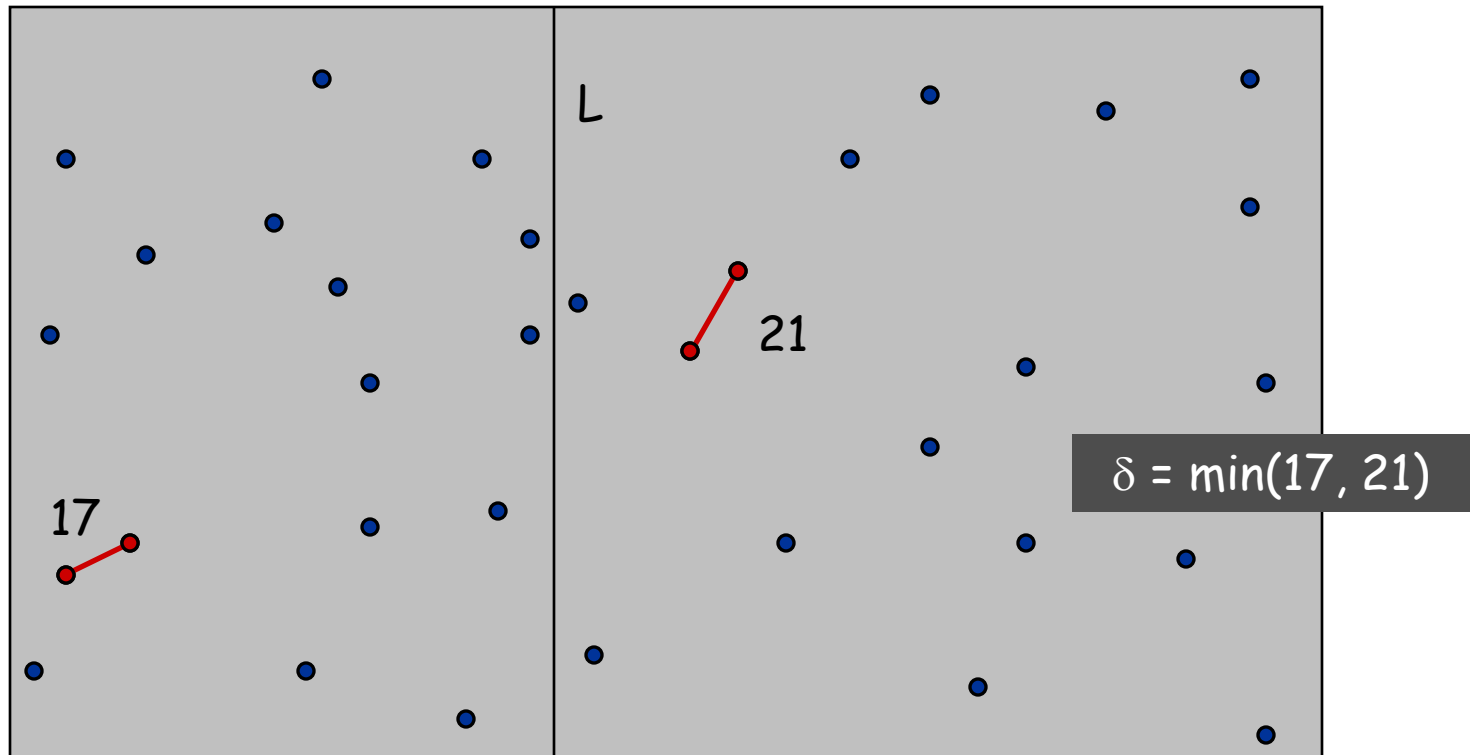
Algorithm.

- 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. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

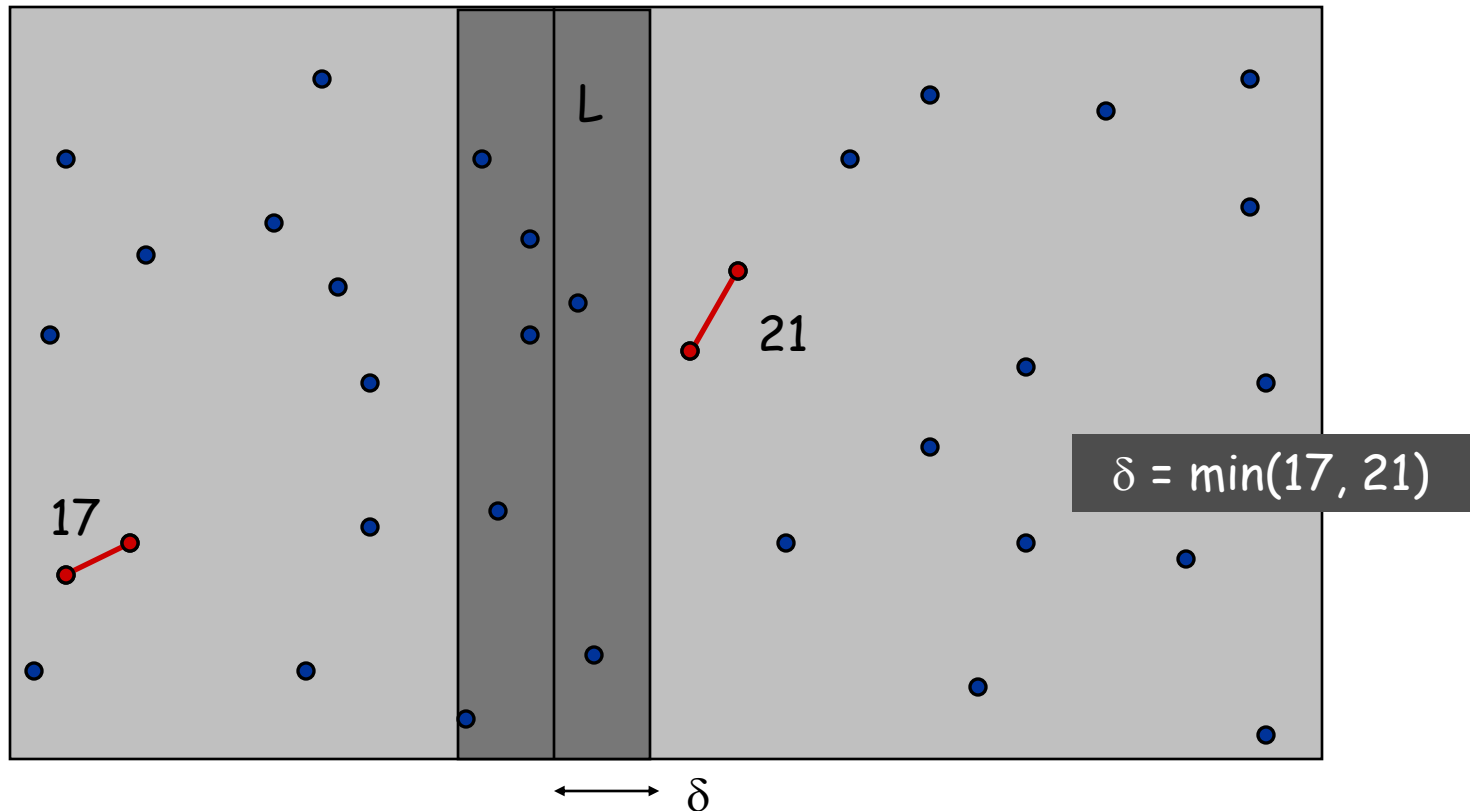
Find closest pair with one point in each side, **assuming that distance $< \delta$** .



Closest Pair of Points

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

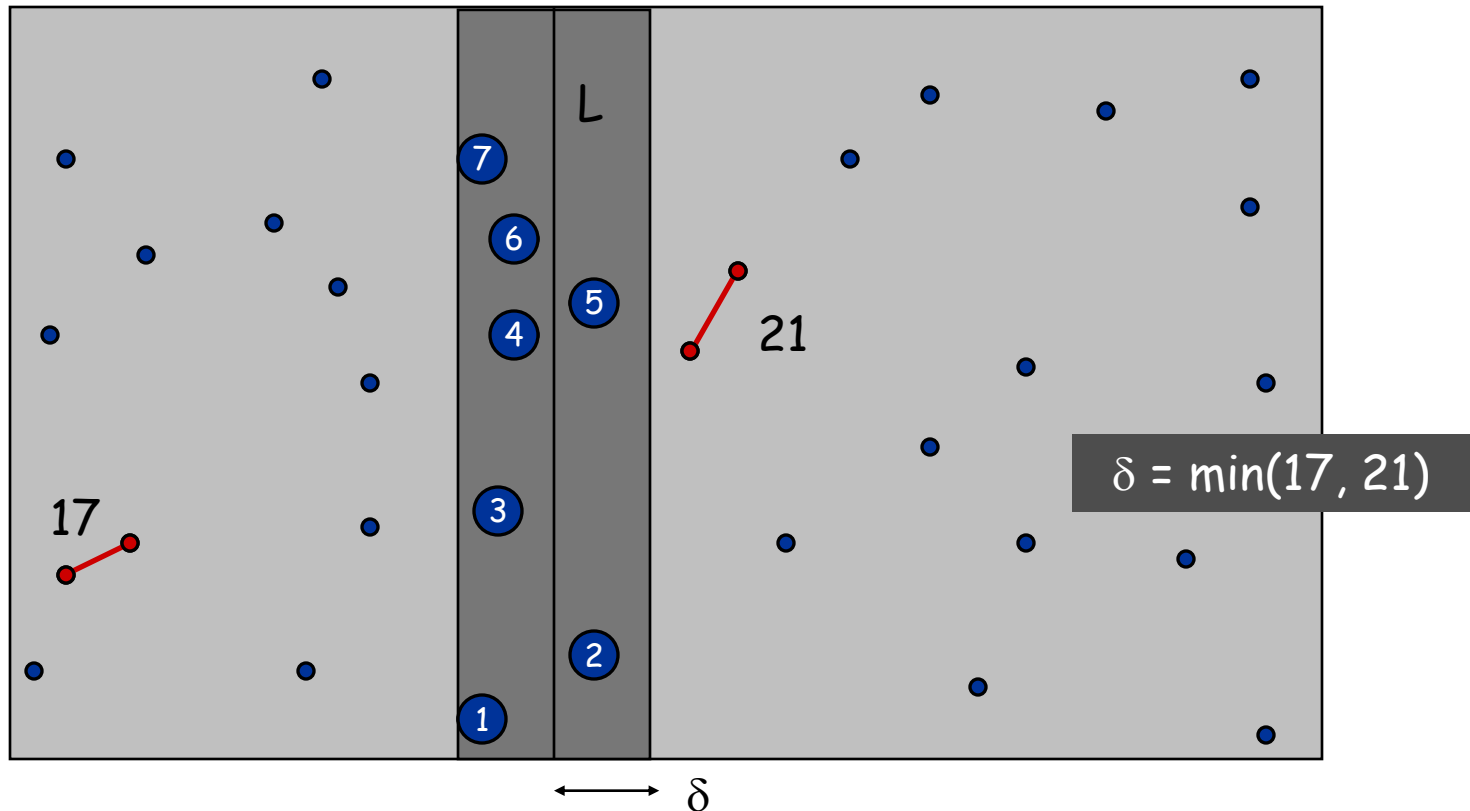
Observation: only need to consider points within δ of line L .



Closest Pair of Points

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

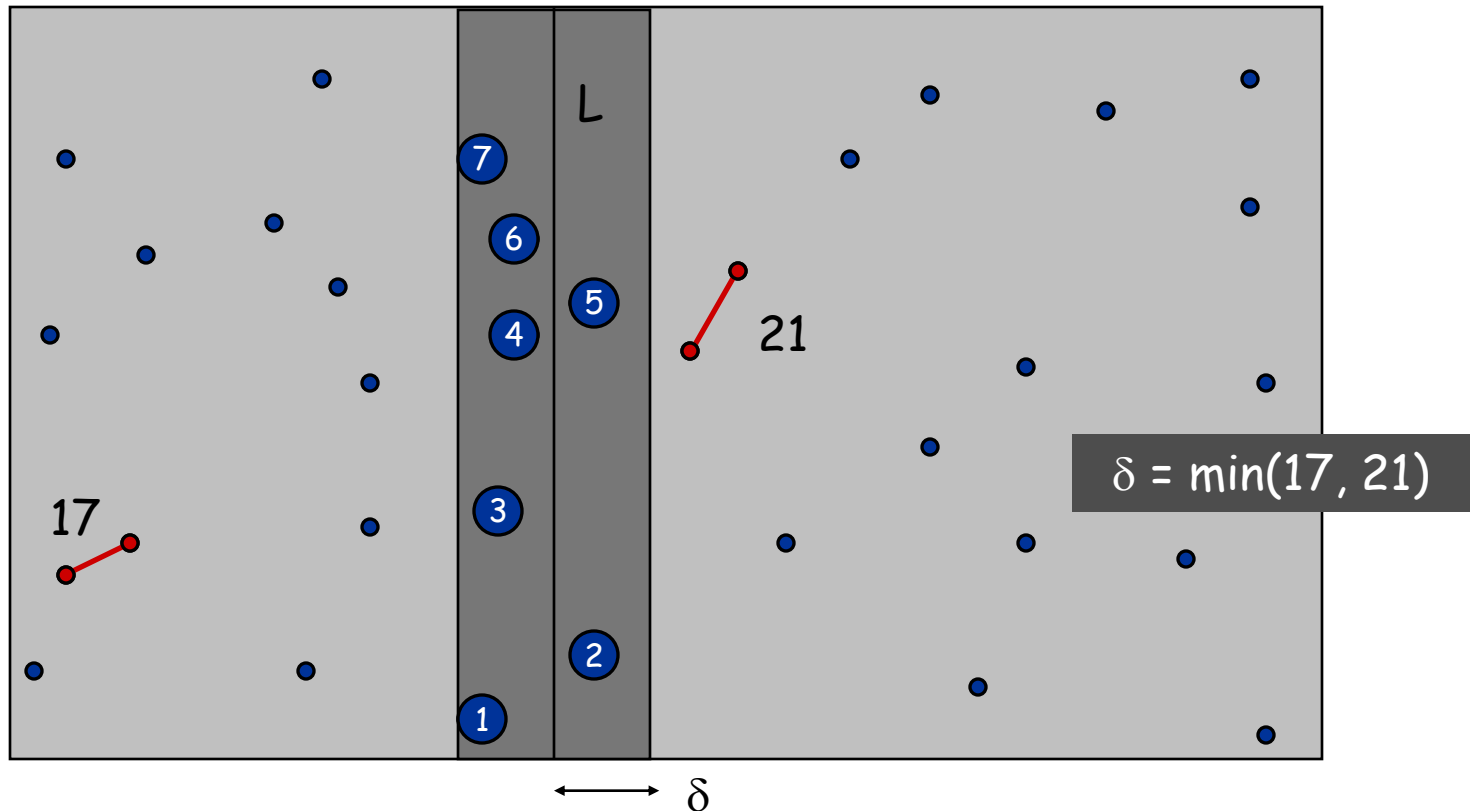
- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.



Closest Pair of Points

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

- Observation: 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

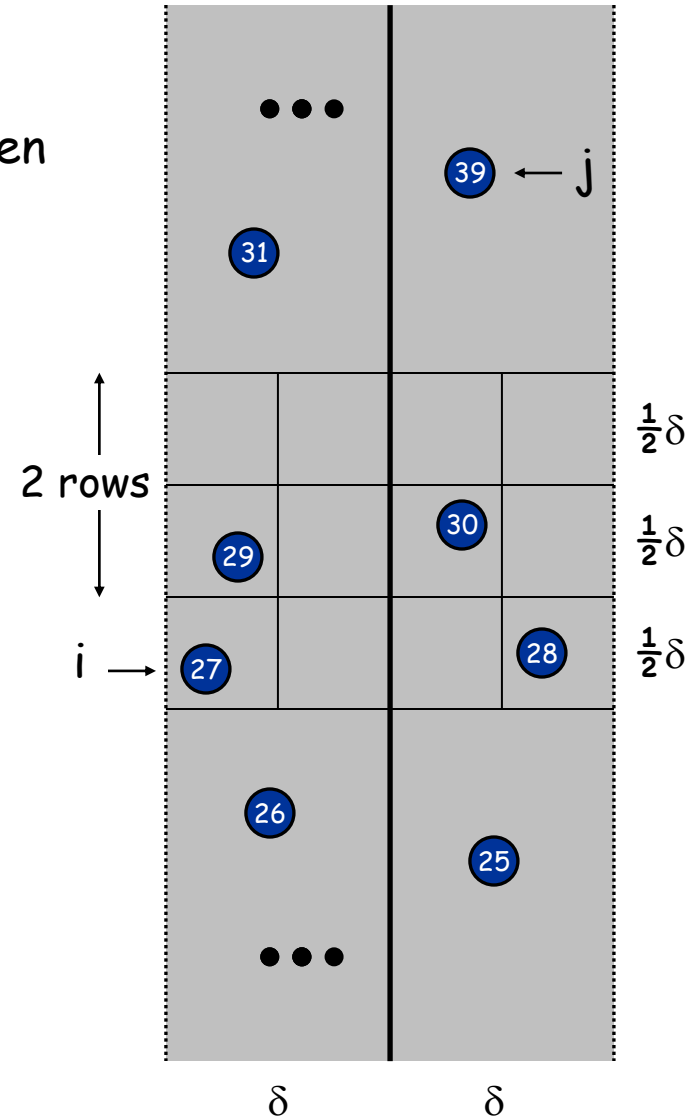
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

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

Pf.

- \cdot No two points lie in same $\frac{1}{2}\delta$ -by- $\frac{1}{2}\delta$ box.
- \cdot 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 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.  $O(n \log n)$   
  
   $\delta_1 = \text{Closest-Pair}(\text{left half})$   
   $\delta_2 = \text{Closest-Pair}(\text{right half})$   $2T(n / 2)$   
   $\delta = \min(\delta_1, \delta_2)$   
  
  Delete all points further than  $\delta$  from separation line  $L$   $O(n)$   
  
  Sort remaining points by  $y$ -coordinate.  $O(n \log n)$   
  
  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  $\delta$ .  $O(n)$   
  
  return  $\delta$ .  
}
```


Closest Pair of Points: Analysis

Running time.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n) & \text{if } n > 1 \end{cases}$$

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

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

- Each recursive call returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by **merging** two pre-sorted lists.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

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	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

Multiply

[illegible]

Divide-and-Conquer Multiplication: Warmup

To multiply two n -digit integers:

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

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

← use bit shifting
to compute 4 terms

$$x y = (2^m a + b) (2^m c + d) = \underbrace{2^{2m} ac}_1 + \underbrace{2^m (bc + ad)}_2 + \underbrace{bd}_4$$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

↑
assumes n is a power of 2

Karatsuba Multiplication

To multiply two n -digit integers:

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

$$\begin{aligned}xy &= (2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd \\&= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd\end{aligned}$$

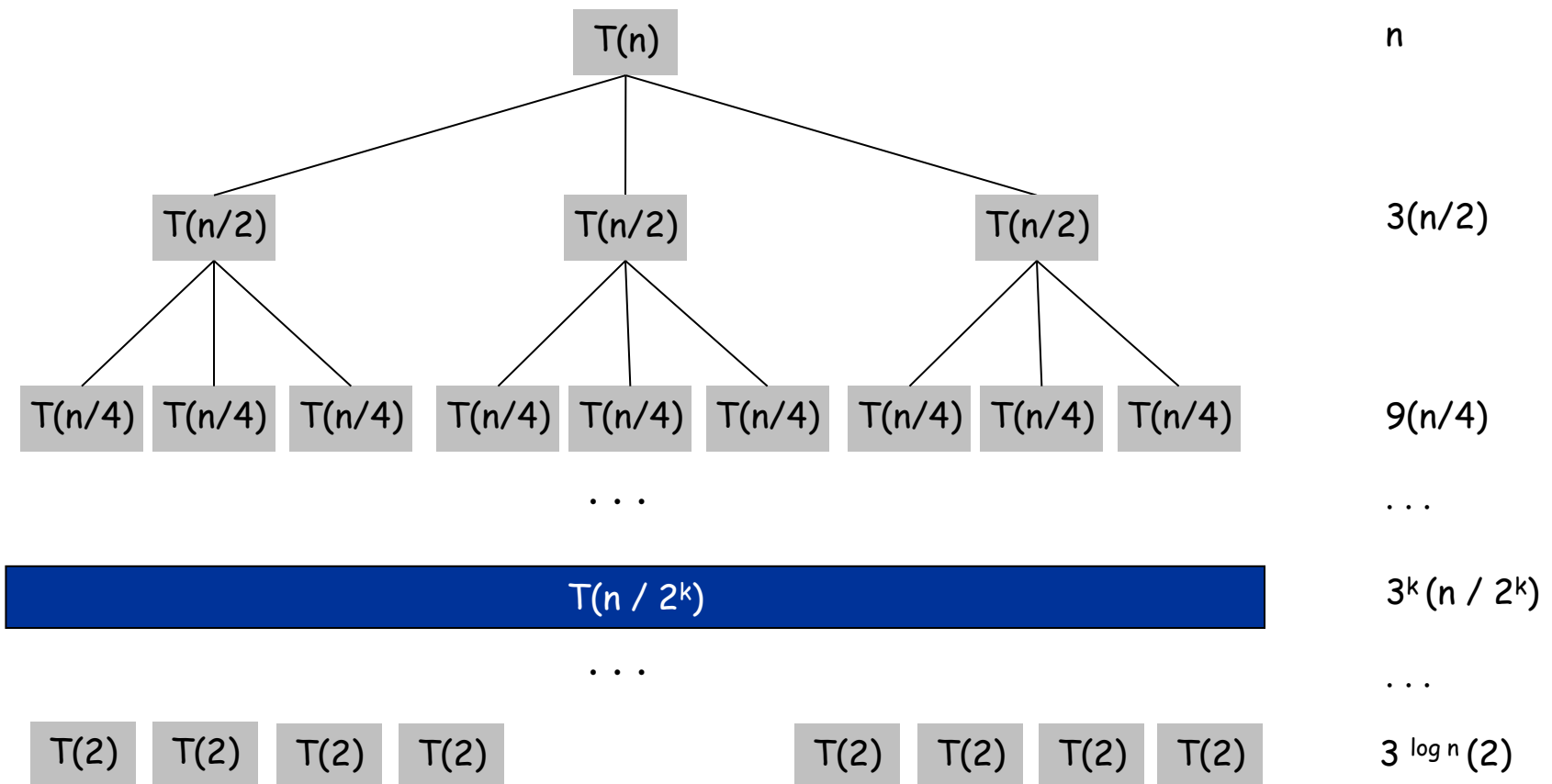
11323

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

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

$$\Rightarrow T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

Karatsuba: Recursion Tree



History of asymptotic complexity of integer multiplication

year	algorithm	bit operations
12xx	grade school	$O(n^2)$
1962	Karatsuba-Ofman	$O(n^{1.585})$
1963	Toom-3, Toom-4	$O(n^{1.465})$, $O(n^{1.404})$
1966	Toom-Cook	$O(n^{1+\varepsilon})$
1971	Schönhage-Strassen	$O(n \log n \cdot \log \log n)$
2007	Fürer	$n \log n 2^{O(\log^* n)}$
2019	Harvey-van der Hoeven	$O(n \log n)$
	???	$O(n)$

Matrix Multiplication

Matrix Multiplication

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

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

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Implementation issues.


- n Sparsity.
- n Caching effects.
- n Numerical stability.
- n Odd matrix dimensions.
- n Crossover to classical algorithm around $n = 128$.

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

- n Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \sim 2,500$.
- n Range of instances where it's useful is a subject of controversy.

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

History of arithmetic complexity of matrix multiplication

year	algorithm	arithmetic operations
1858	“grade school”	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith–Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith–Winograd	$O(n^{2.3755})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.372873})$
2014	Le Gall	$O(n^{2.372864})$
		$O(n^{2 + \epsilon})$

number of arithmetic operations to multiply two n-by-n matrices