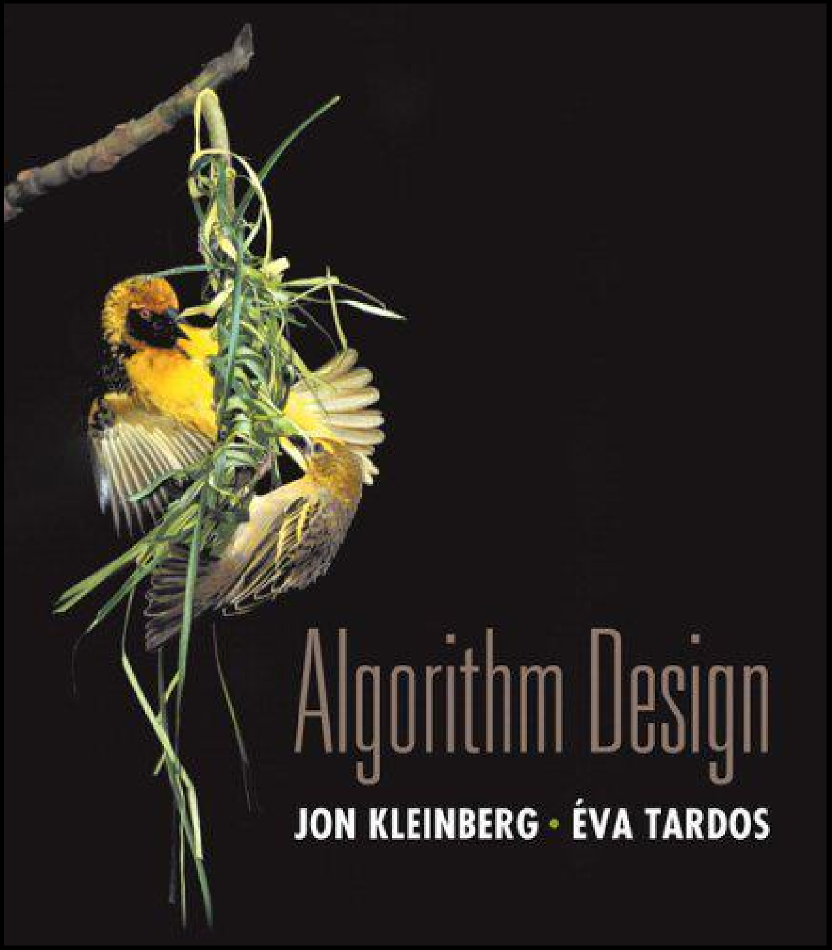


Chapter 5

Divide and Conquer

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 $\frac{1}{2}n$.
- 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.
- . . .

Non-obvious sorting applications.
- Data compression.
- Computer graphics.
- Interval scheduling.
- Minimum spanning tree.
- Supply chain management.
- Book recommendations on Amazon.
- . . .

Mergesort

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

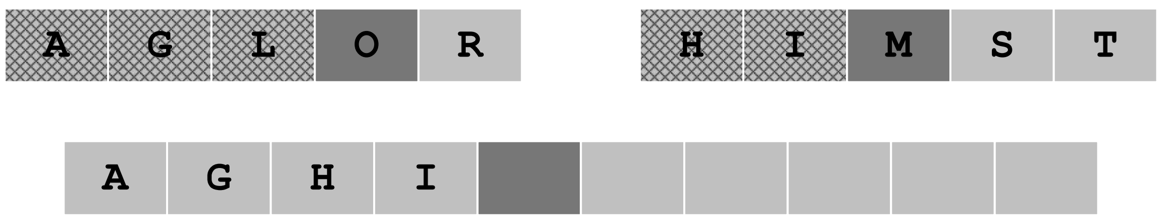
A L G O R I T H M S		
A L G O R I T H M S	divide	$O(1)$
A G L O R H I M S T	sort	$2T(n/2)$
A G H I L M O R S T	merge	$O(n)$

Merging

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

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.



A Useful Recurrence Relation

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

Mergesort recurrence.

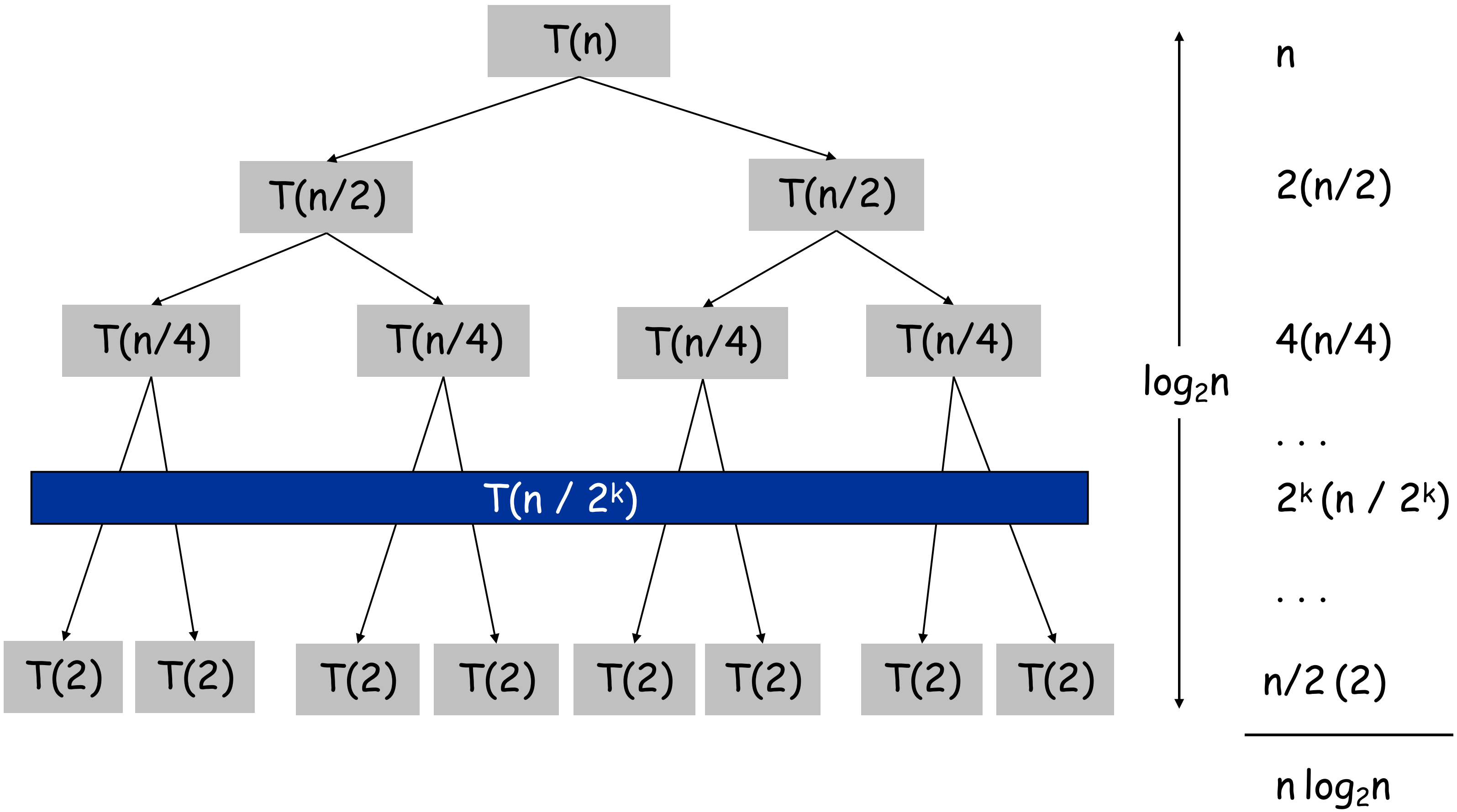
$$
\mathrm{T}(n) \;\le\; \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T\left(\lceil n/2 \rceil\right)}_{\text{solve left half}} + \underbrace{T\left(\lfloor n/2 \rfloor\right)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}
$$

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 $\le$ 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 Induction

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

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

Pf. (by induction on n)

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

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

Analysis of Mergesort Recurrence

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

$\lg n$: $\log_2 n$

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

Pf. (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.

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

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

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, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- 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.

- Voting theory.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- 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.

- 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

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

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

8 green-green inversions

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

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

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)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$

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 **then**
    return $(0, L)$.
end if
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).
$(r_{AB}, L') \leftarrow$ MERGE-AND-COUNT(A, B).
return $r_A + r_B + r_{AB}, L'$.

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.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST.

fast closest pair inspired fast algorithms for these problems

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

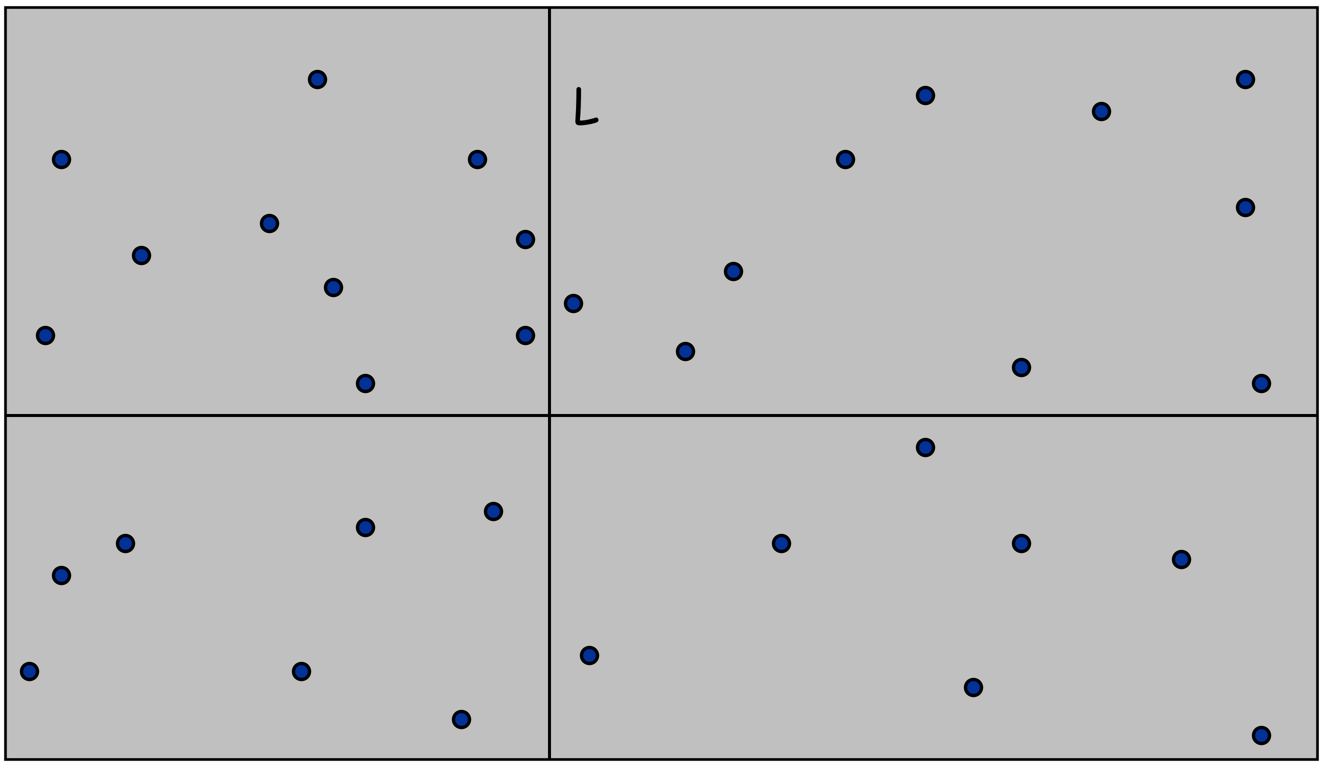
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points: First Attempt

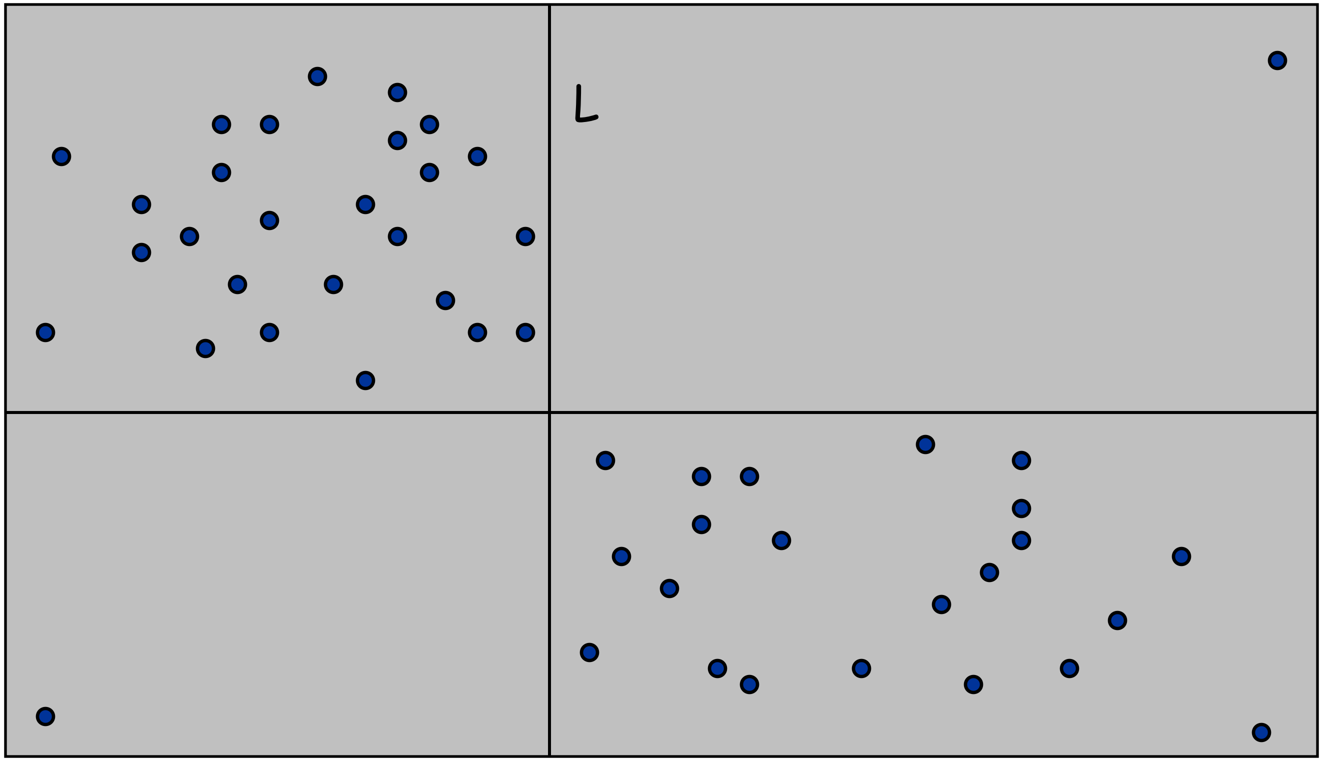
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

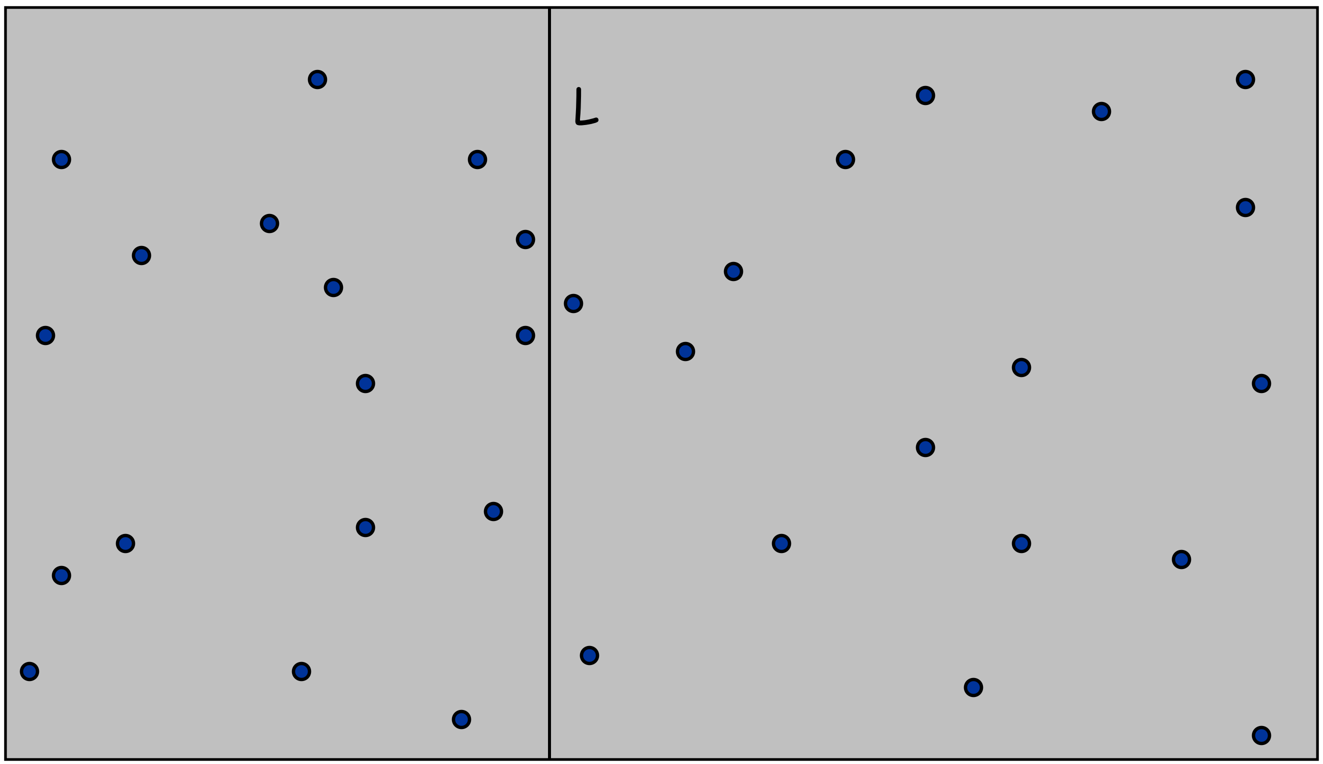
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

Algorithm.

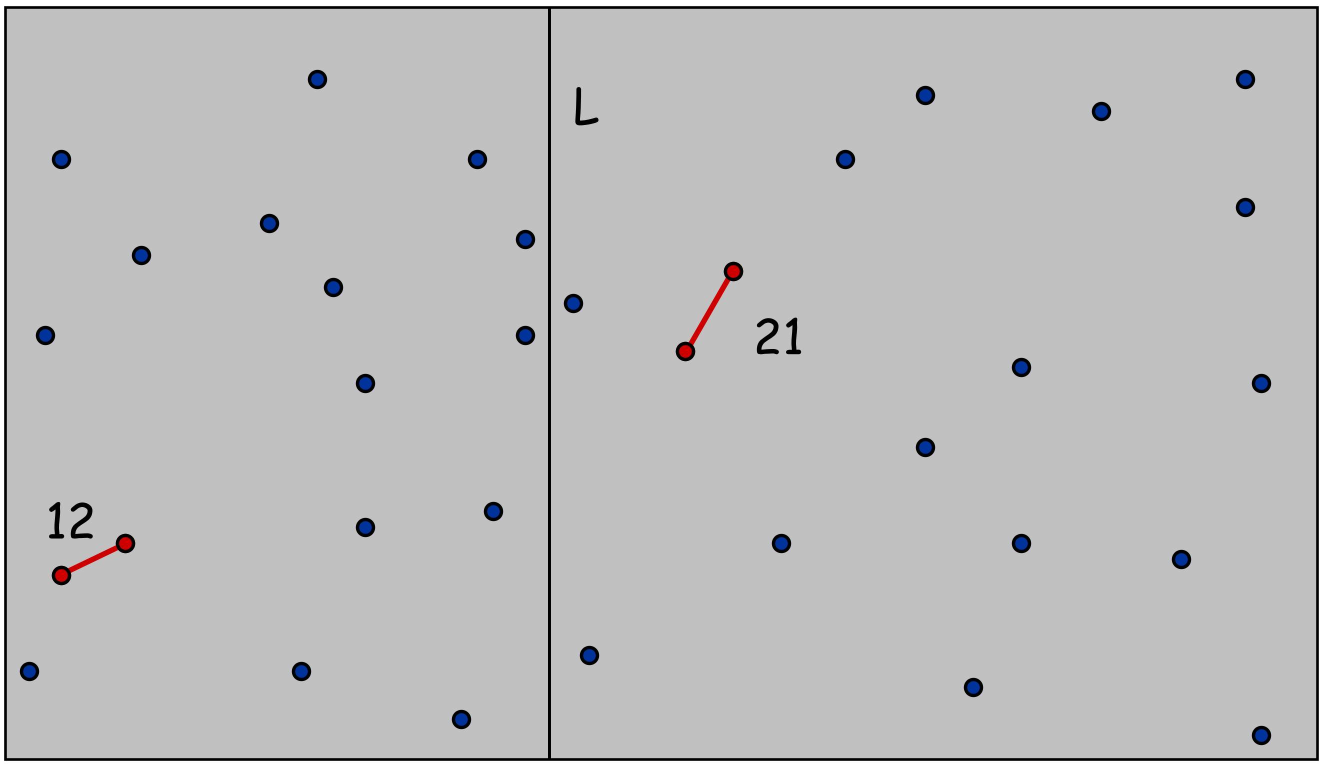
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

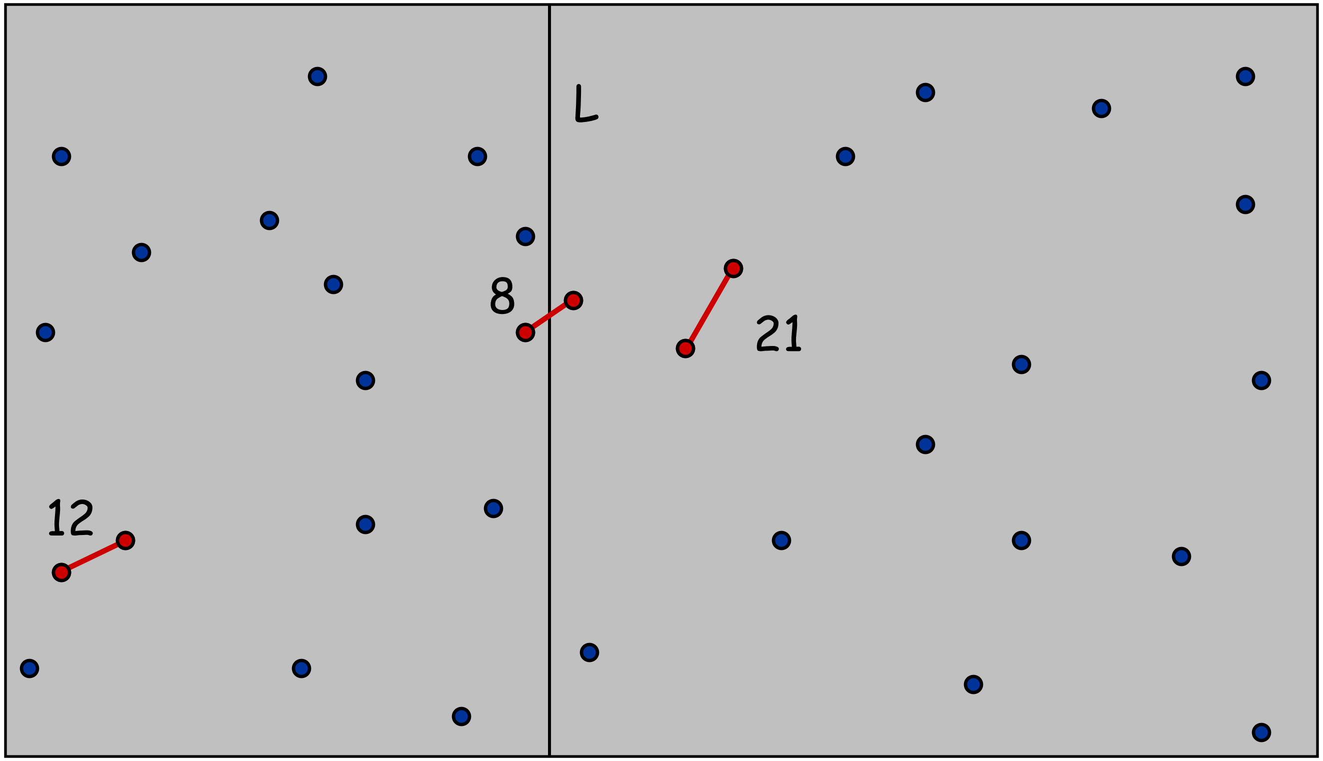
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

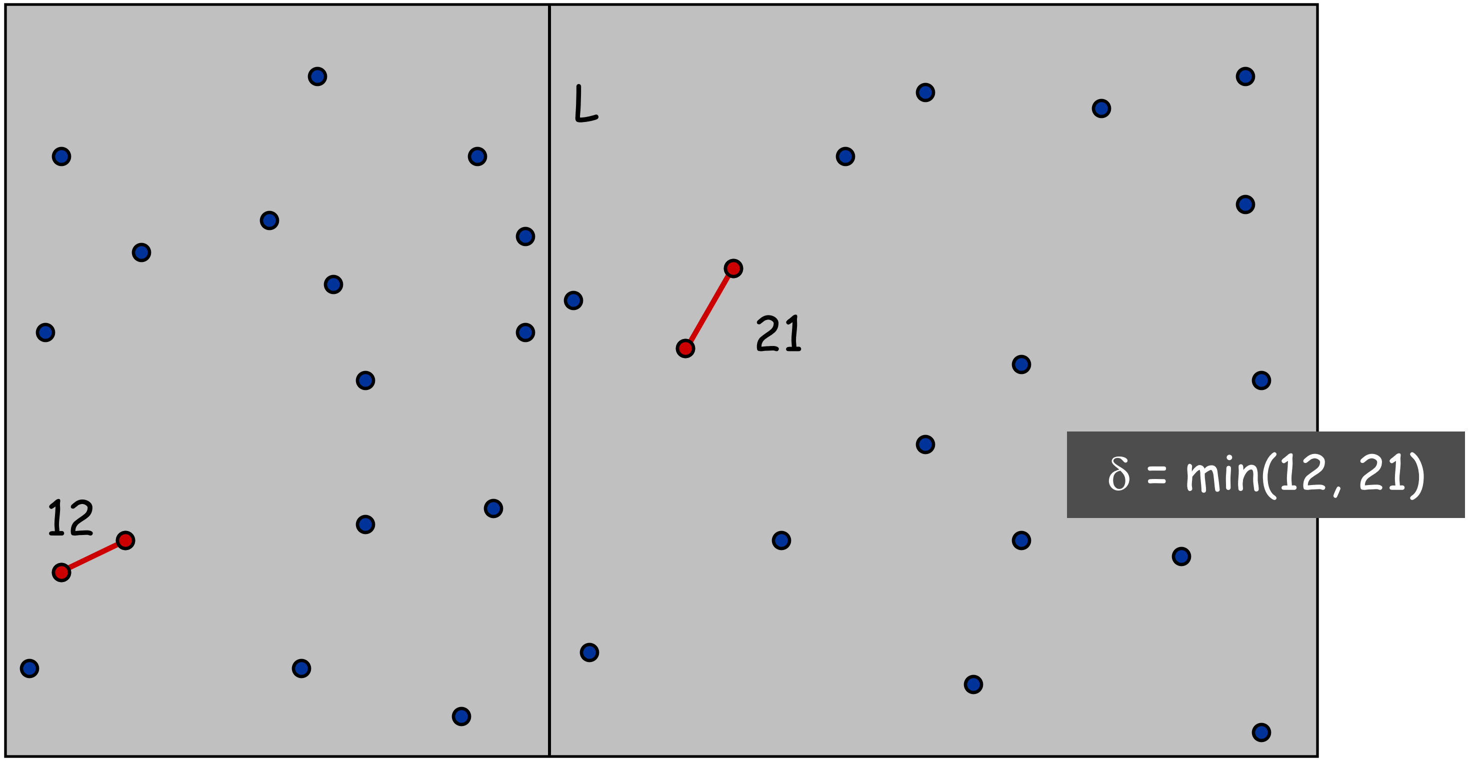
Algorithm.

- 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

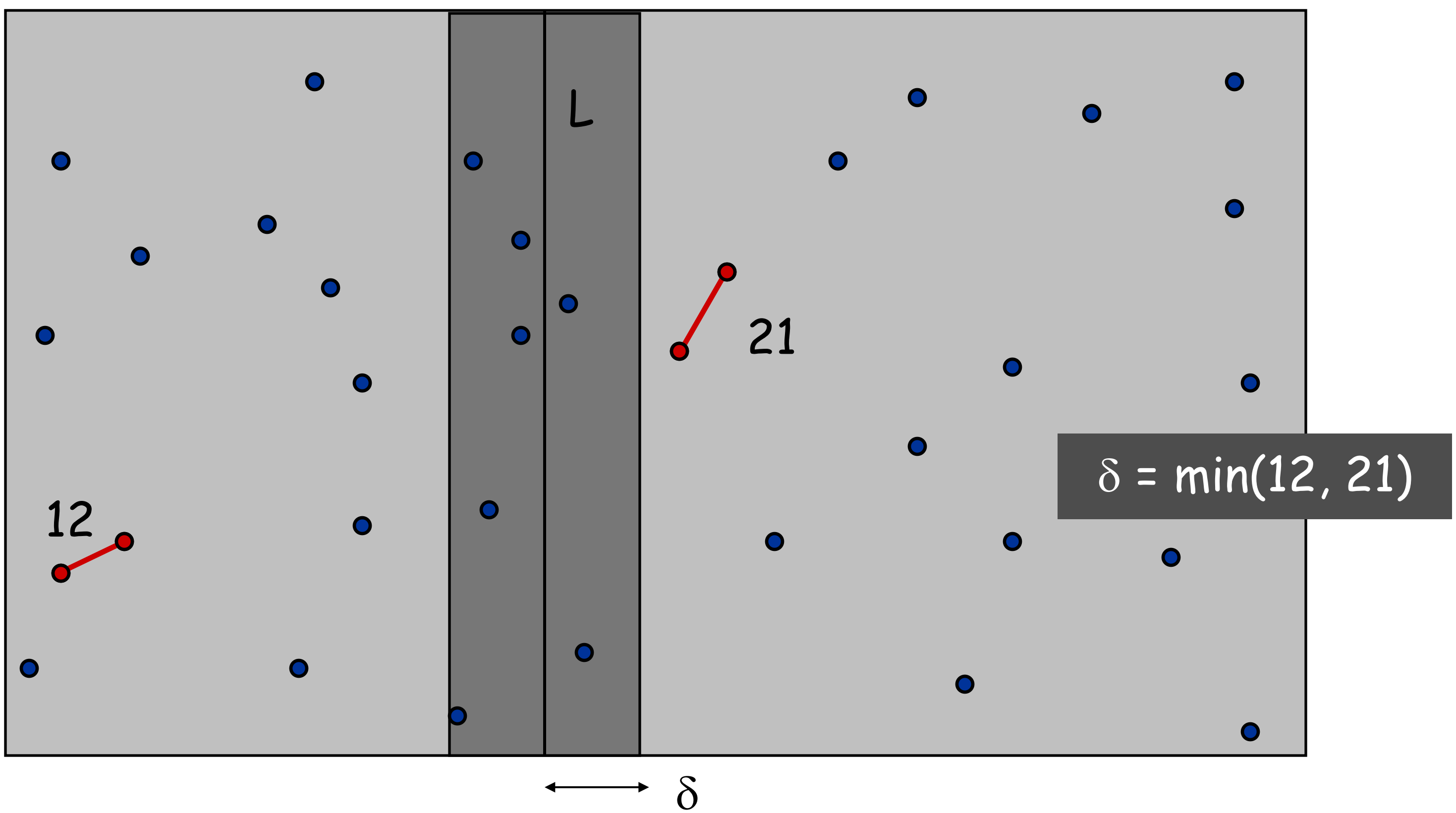
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

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

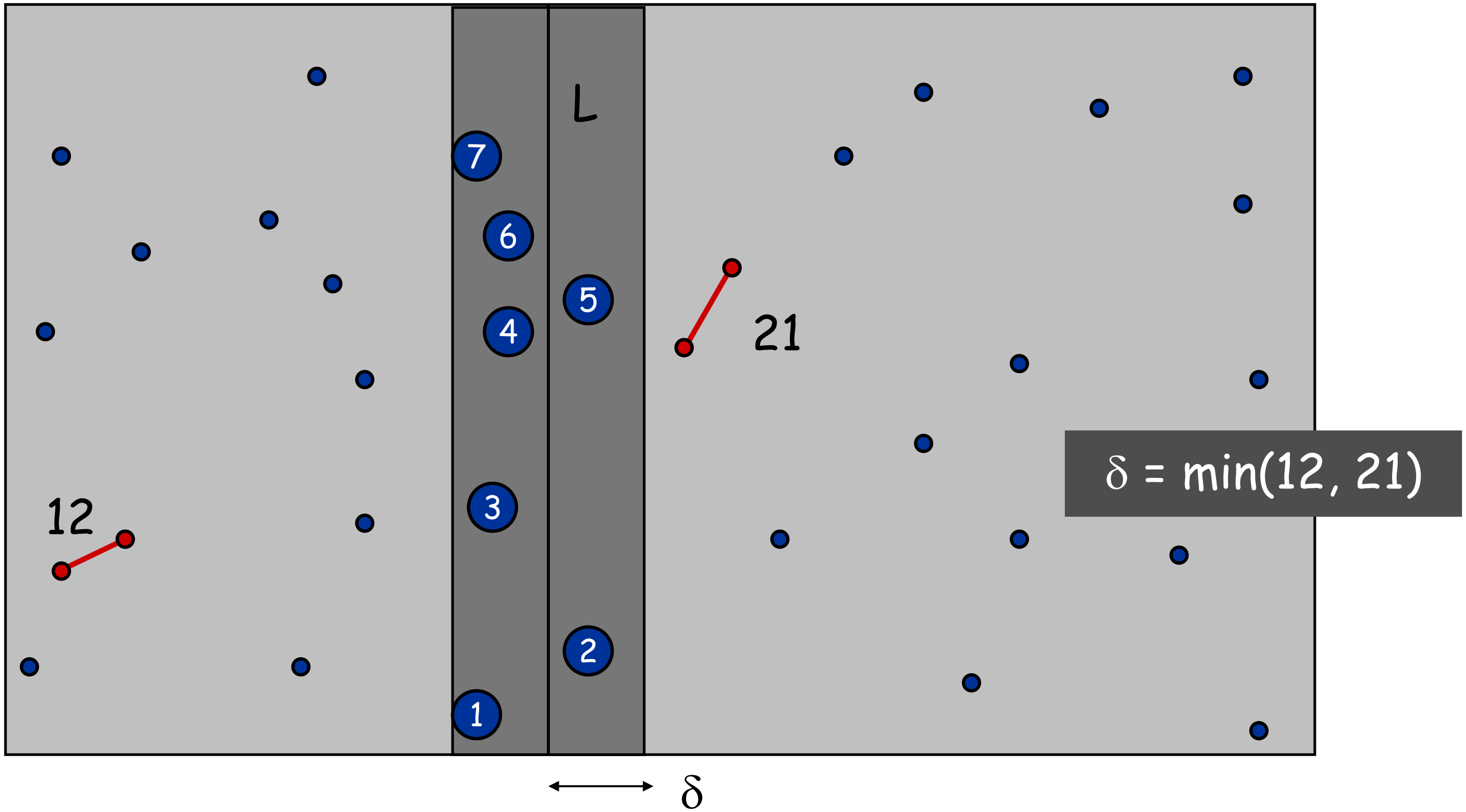
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

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

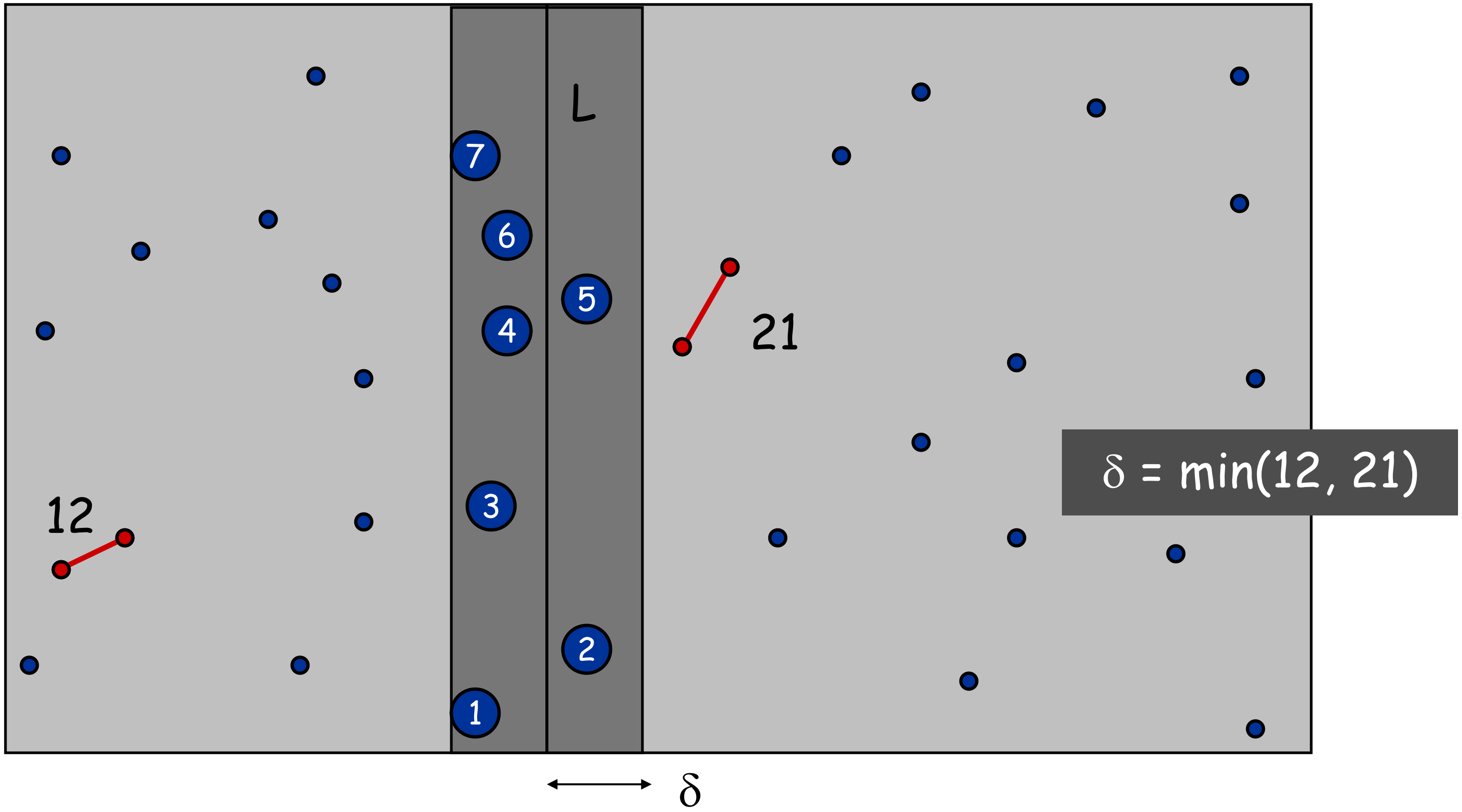
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



Closest Pair of Points

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

- 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

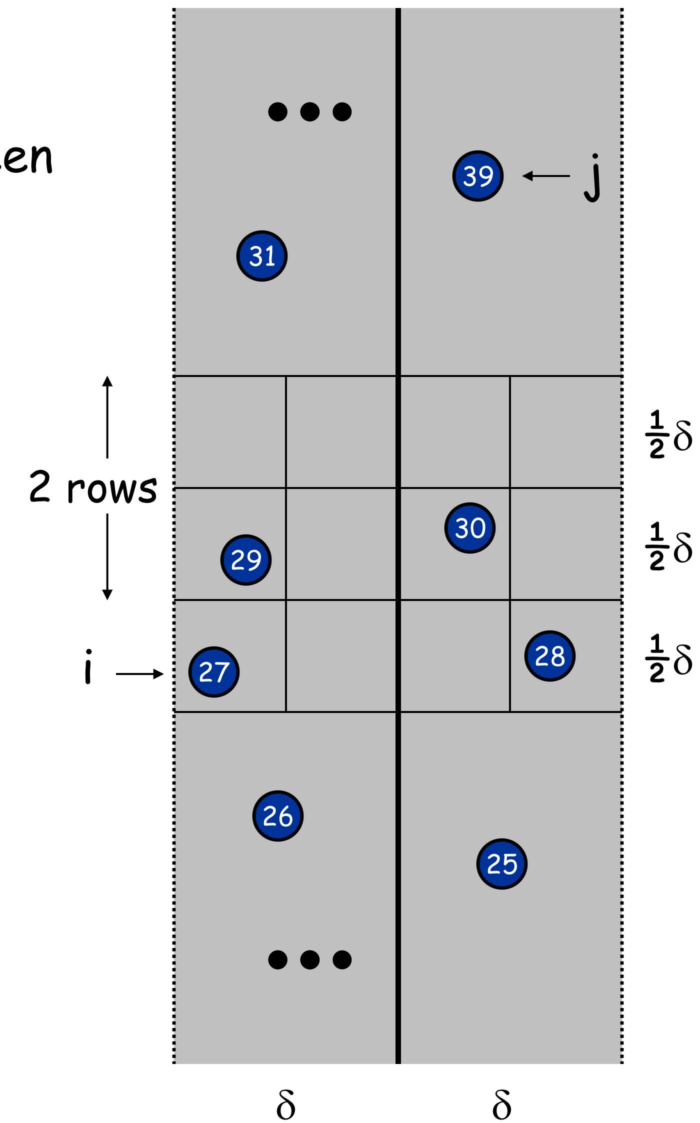
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

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

Pf.

- 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 Algorithm

$CLOSEST - PAIR(p_1, p_2, \cdots, p_n)$

1: Compute separation line L such that half the points are on each side of the line. $O(n \log n)$
2: $\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).
3: $\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half). $2T(n/2)$
4: $\delta \leftarrow \min\{\delta_1, \delta_2\}$.
5: Delete all points further than δ from line L. $O(n)$
6: Sort remaining points by y-coordinate. $O(n \log n)$
7: 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. $O(n)$
8: **return** δ.

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 in strip from scratch each time.

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

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

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

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

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

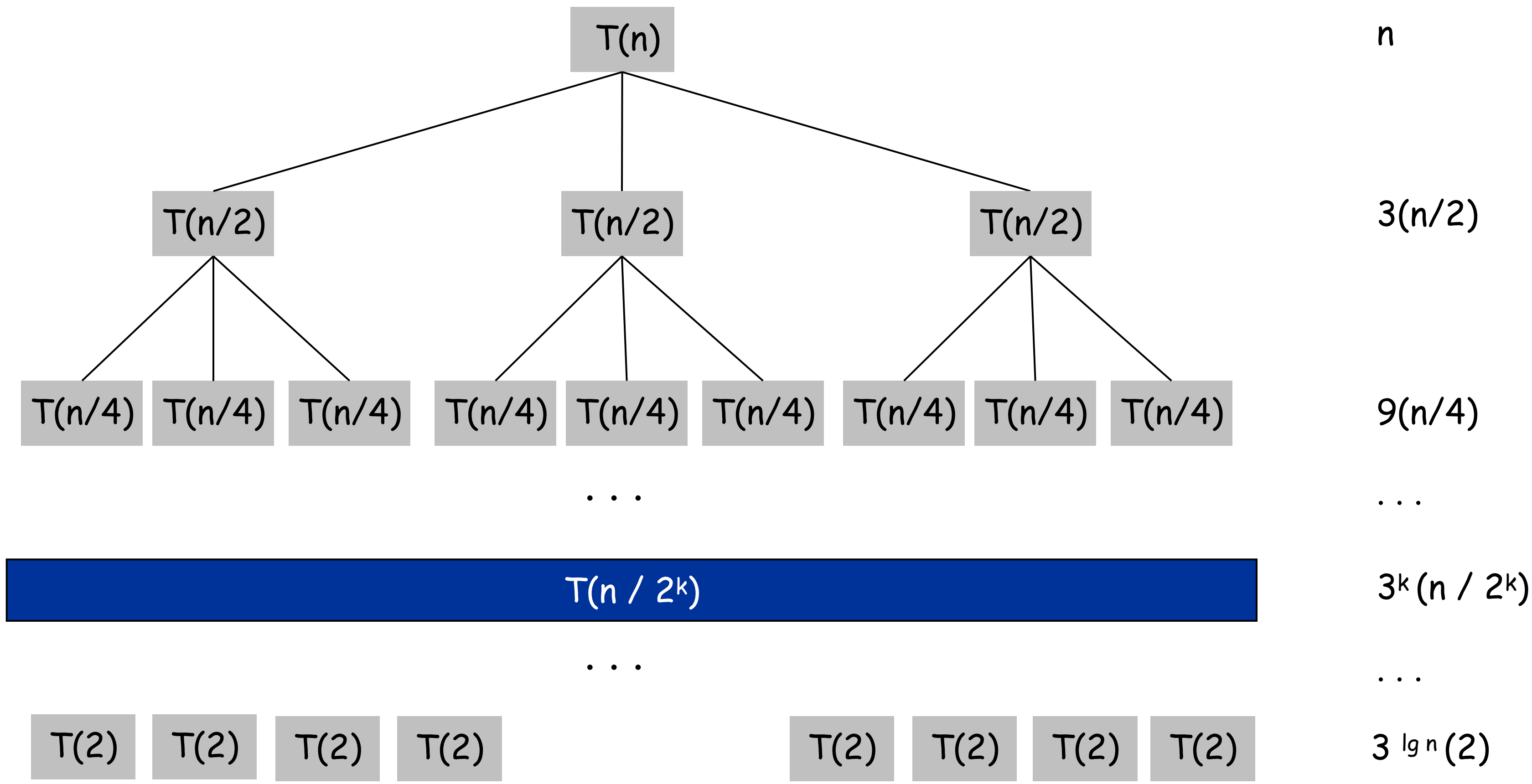
$$
\mathrm{T}(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
$$

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

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

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 ½n-by-½n blocks.
- Conquer: multiply 8 ½n-by-½n recursively.
- Combine: add appropriate products using 4 matrix additions.

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

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

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

Matrix Multiplication: Key Idea

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

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

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

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

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

Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

- Divide: partition A and B into ½n-by-½n blocks.
- Compute: 14 ½n-by-½n matrices via 10 matrix additions.
- Conquer: multiply 7 ½n-by-½n matrices recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

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

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

Fast Matrix Multiplication in 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. Unknown.

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

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. $O(n^{2.373})$ [Williams, 2011.]

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

Homework

- Read Chapter 5 of the textbook.

- Exercises 1, 2, 3 & 4 in Chapter 5.