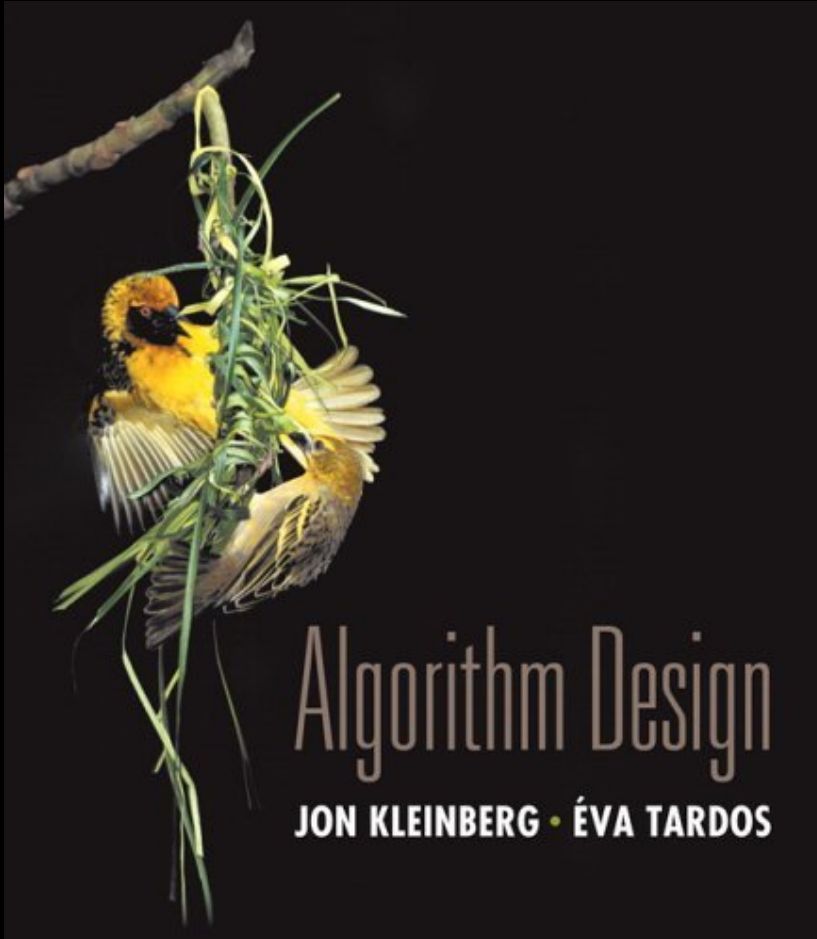


Chapter 5

Divide and Conquer



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.
Updated by T. K. Prasad

Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve several parts 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: $n \log n$.

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

5.1 Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Obvious sorting applications.

- List files in a directory.
- Organize a 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.
- ...

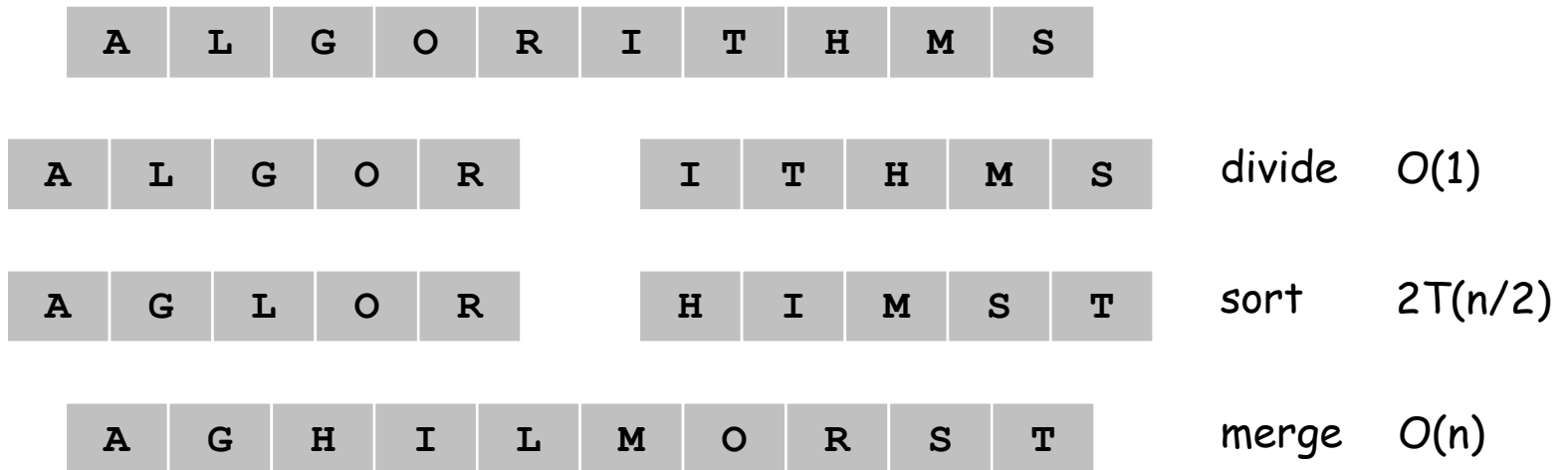
Mergesort

Mergesort.

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



John von Neumann (1945)



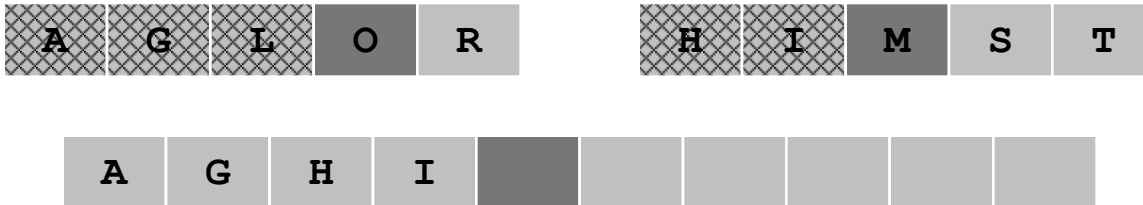
Merging

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

How to merge efficiently?



- Linear number of comparisons.
- Use temporary array.

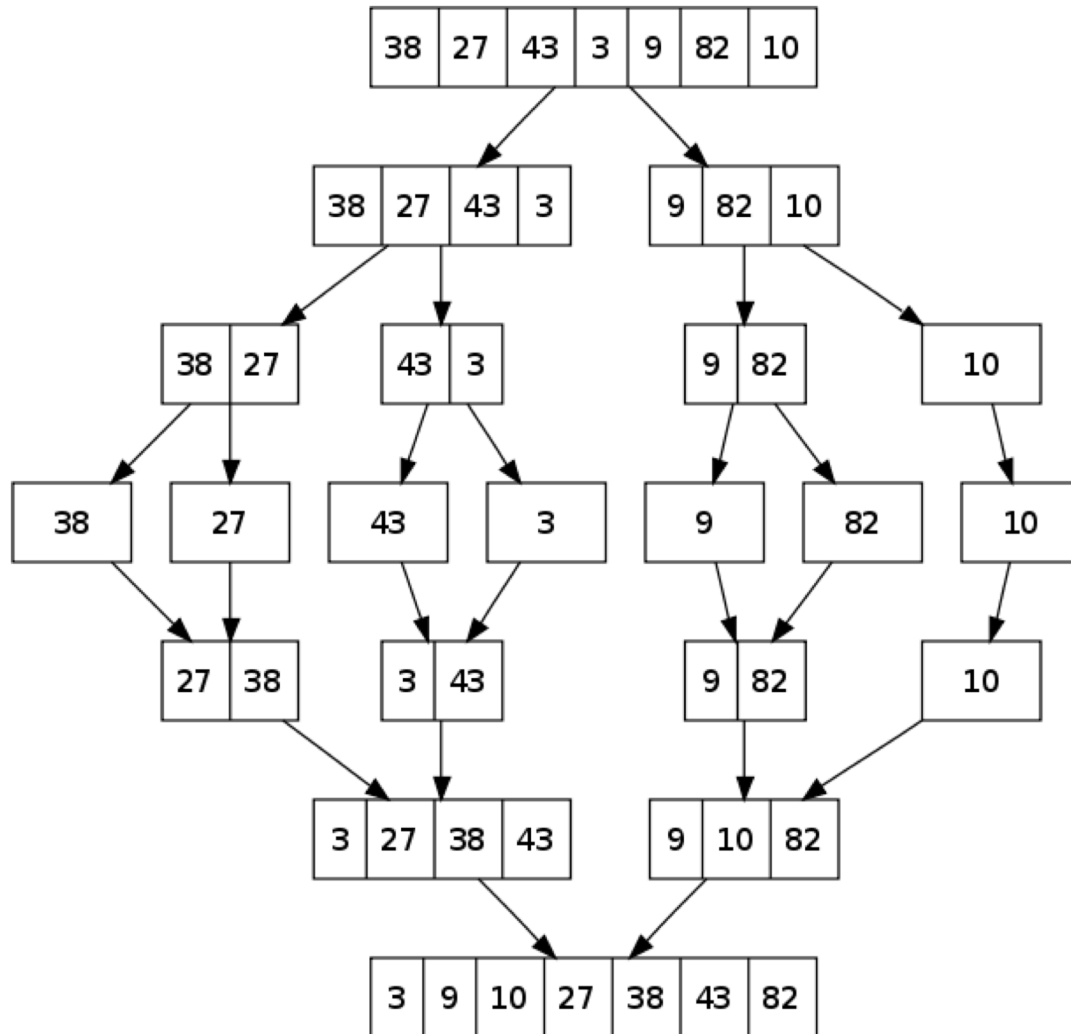


Challenge for the bored. In-place merge. [Kronrod, 1969]



using only a constant amount of extra storage

Mergesort



A Useful Recurrence Relation

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

Mergesort recurrence.

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

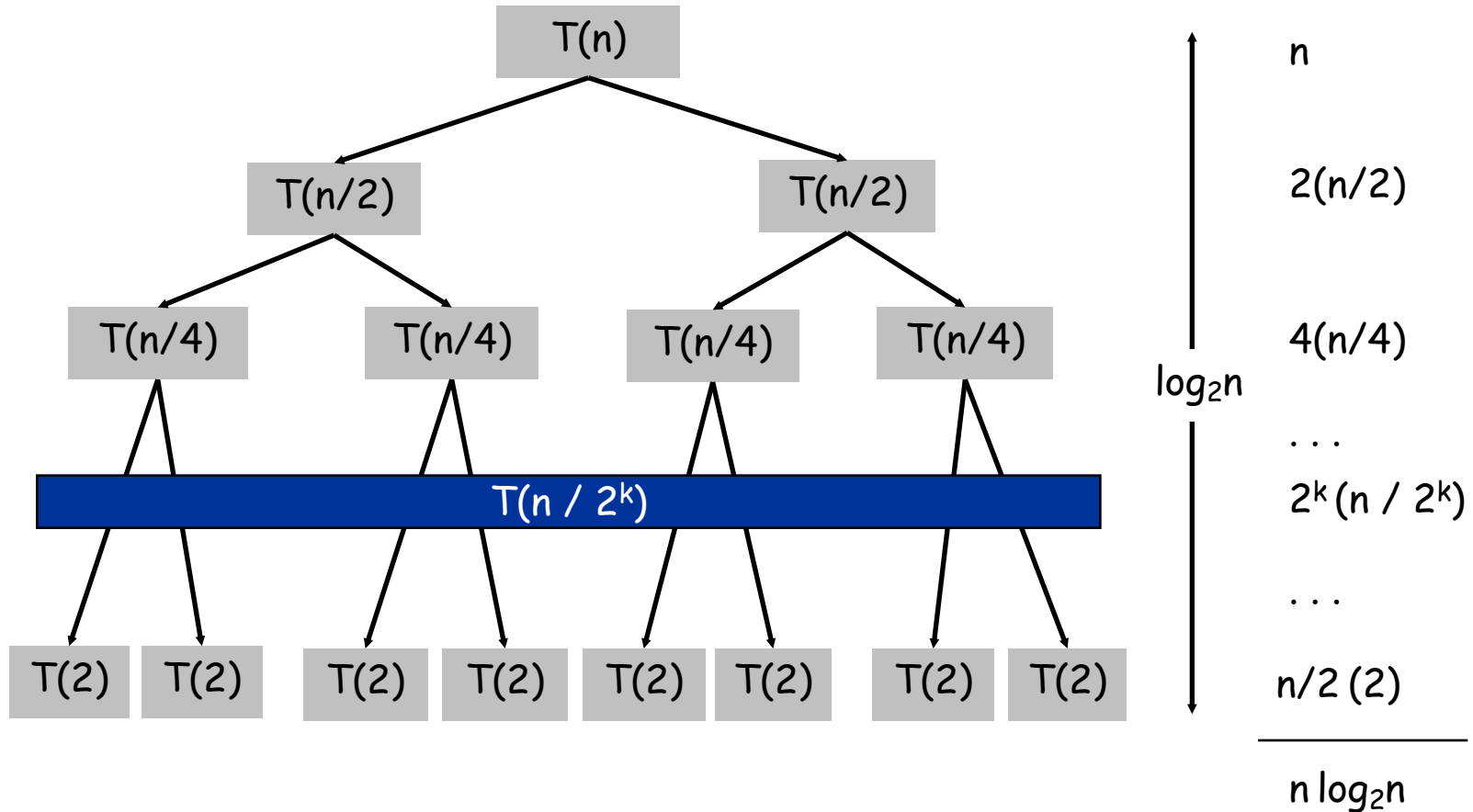
Solution. $T(n) = O(n \log_2 n)$.

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

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

assumes n is a power of 2



Proof by Telescoping

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

↑
assumes n is a power of 2

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

Pf. For $n > 1$:

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

Proof by Induction

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

↑
assumes n is a power of 2

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

Pf. (by induction on n)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Induction Step:** Show that $T(2n) = 2n \log_2 (2n)$.

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

Analysis of Mergesort Recurrence

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

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

↑
 $\log_2 n$

Pf. (by induction on n)

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

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

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

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

<i>Songs</i>					
	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.

- Voting theory.
- Collaborative filtering.
- 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).

Observations

- In the worst case, there are quadratic number of inversions ($O(n^2)$).
 - E.g., consider a list sorted in descending order.
- So to improve upon $O(n^2)$ bound asymptotically (e.g., $O(n \log n)$) an algorithm must *count inversions without ever looking at each inversion individually.*
- Key “combine” Idea:

The cross-inversions between the two *sorted* halves A and B are precisely due to pairs

$$(a_i, b_j), (a_{i+1}, b_j), \dots \in A \times B \text{ where } a_i > b_j .$$

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
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

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



Divide: $O(1)$.



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

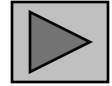
Combine: ???

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: $O(n)$

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

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  
        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 on a 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.

↖ closest pair inspired fast algorithms for these problems

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

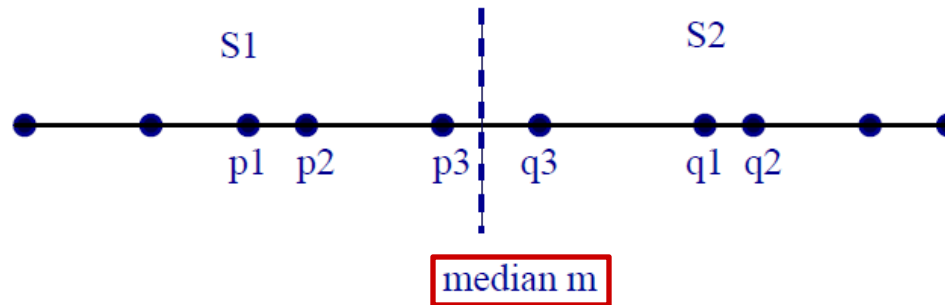
1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

↑
to make presentation cleaner

1D version : Divide and Conquer

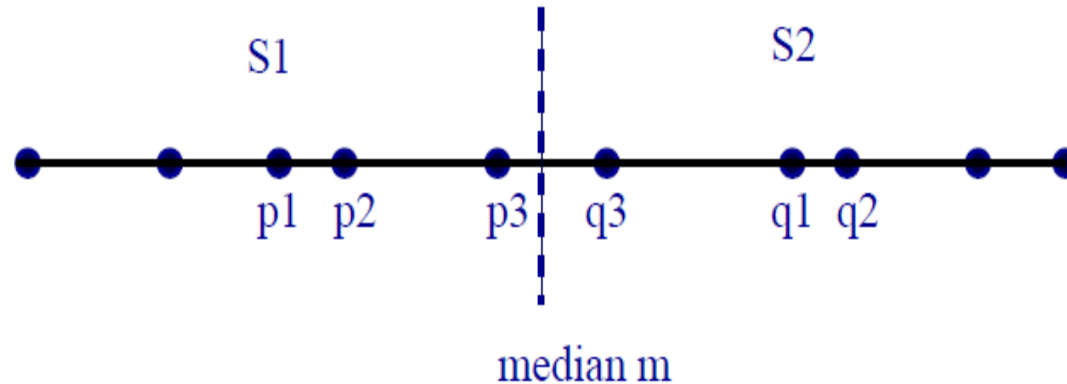
- Divide the points S into two sets S_1, S_2 by some x -coordinate so that $p < q$ for all $p \in S_1$ and $q \in S_2$.
- Recursively compute closest pair (p_1, p_2) in S_1 and (q_1, q_2) in S_2 .



- Let δ be the smallest separation found so far:

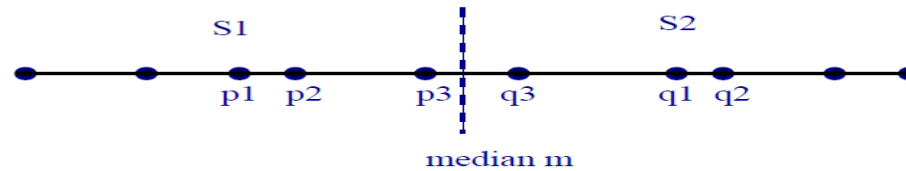
$$\delta = \min(|p_2 - p_1|, |q_2 - q_1|)$$

1D version : Divide and Conquer



- The closest pair is $\{p_1, p_2\}$, or $\{q_1, q_2\}$, or some $\{p_3, q_3\}$ where $p_3 \in S_1$ and $q_3 \in S_2$.
- **Key Observation:** If m is the dividing coordinate, then p_3, q_3 must be within δ of m .

1D version : Divide and Conquer



How many points of S_1 can lie in the interval $(m - \delta, m]$?

By definition of δ , at most one. Same holds for S_2 .

So, we have just one pair (p_3, q_3) to check!

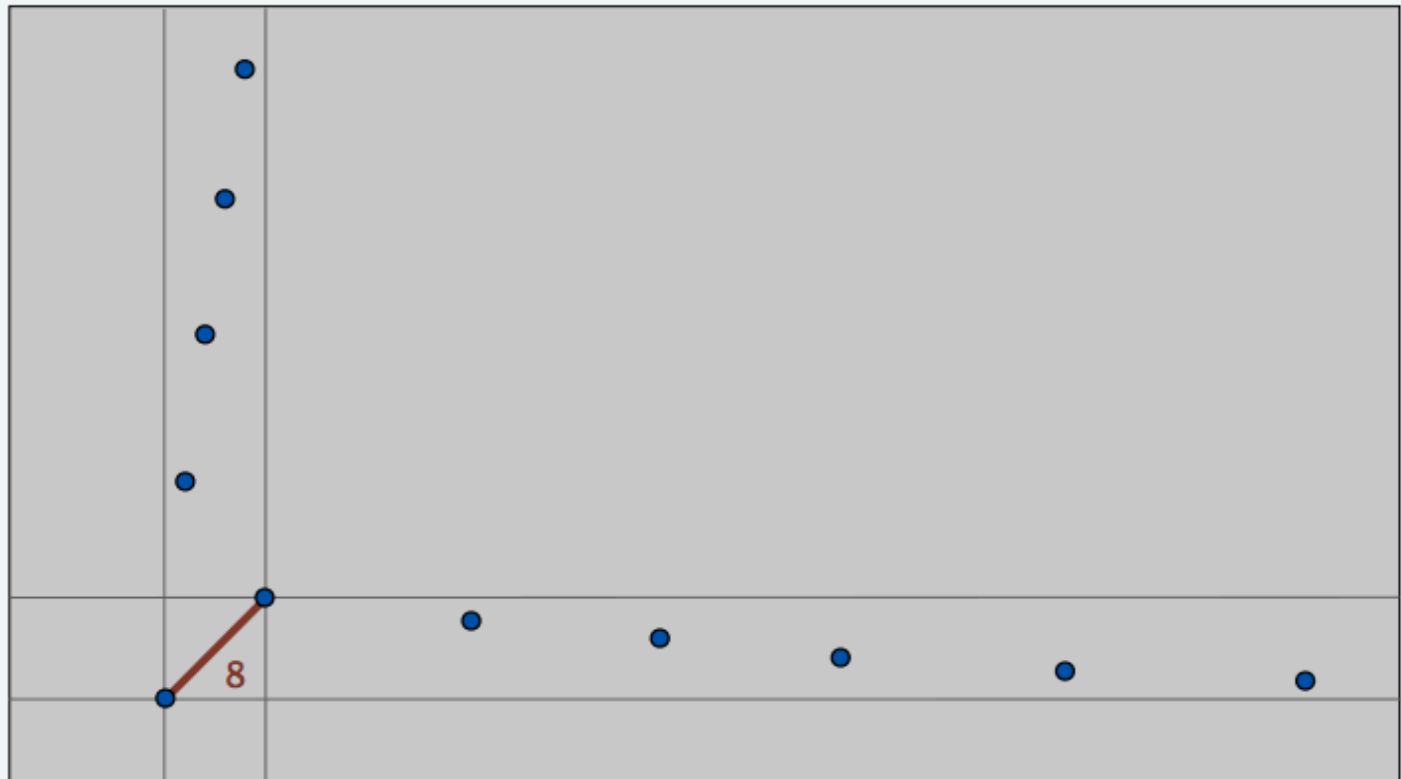
Note, problem decomposition (median computation) takes linear time and solution composition (max-min computation) takes linear time.

Recurrence is $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

Closest pair of points: first attempt

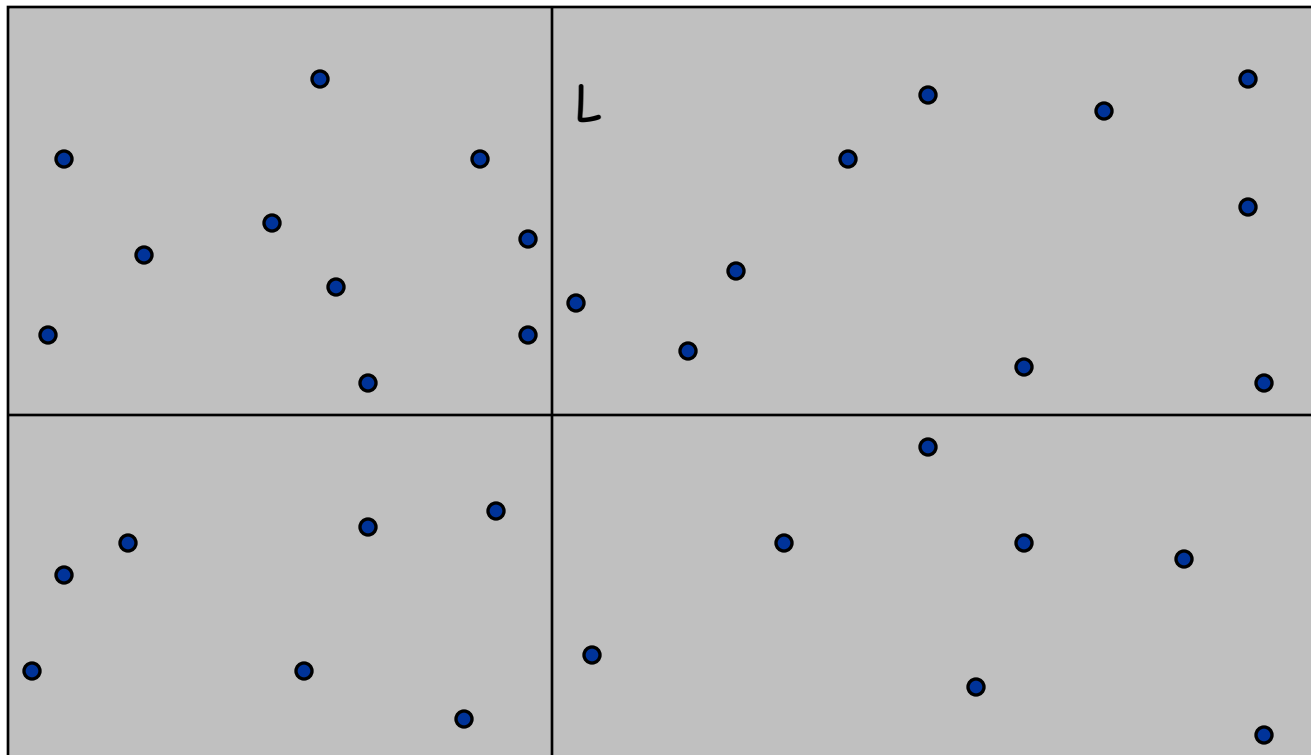
Sorting solution.

- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.



Closest Pair of Points: Second Attempt

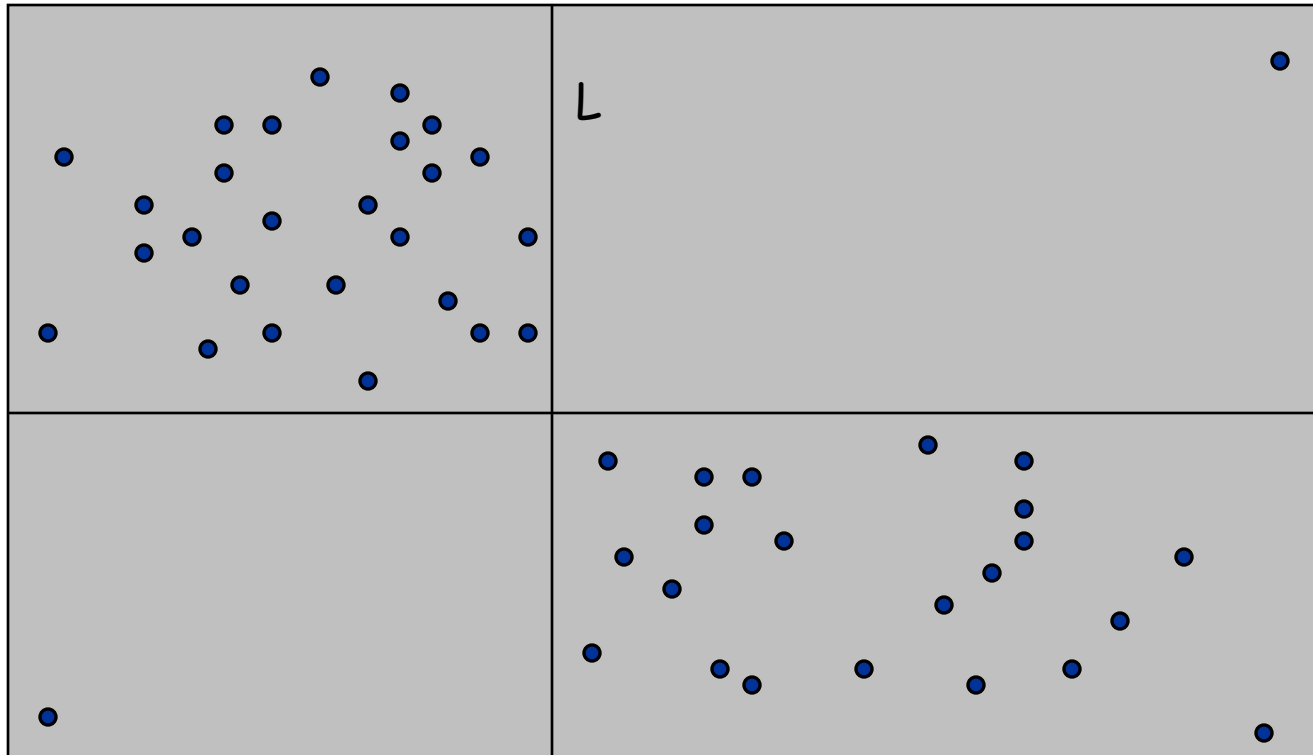
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: Second Attempt

Divide. Sub-divide region into 4 quadrants.

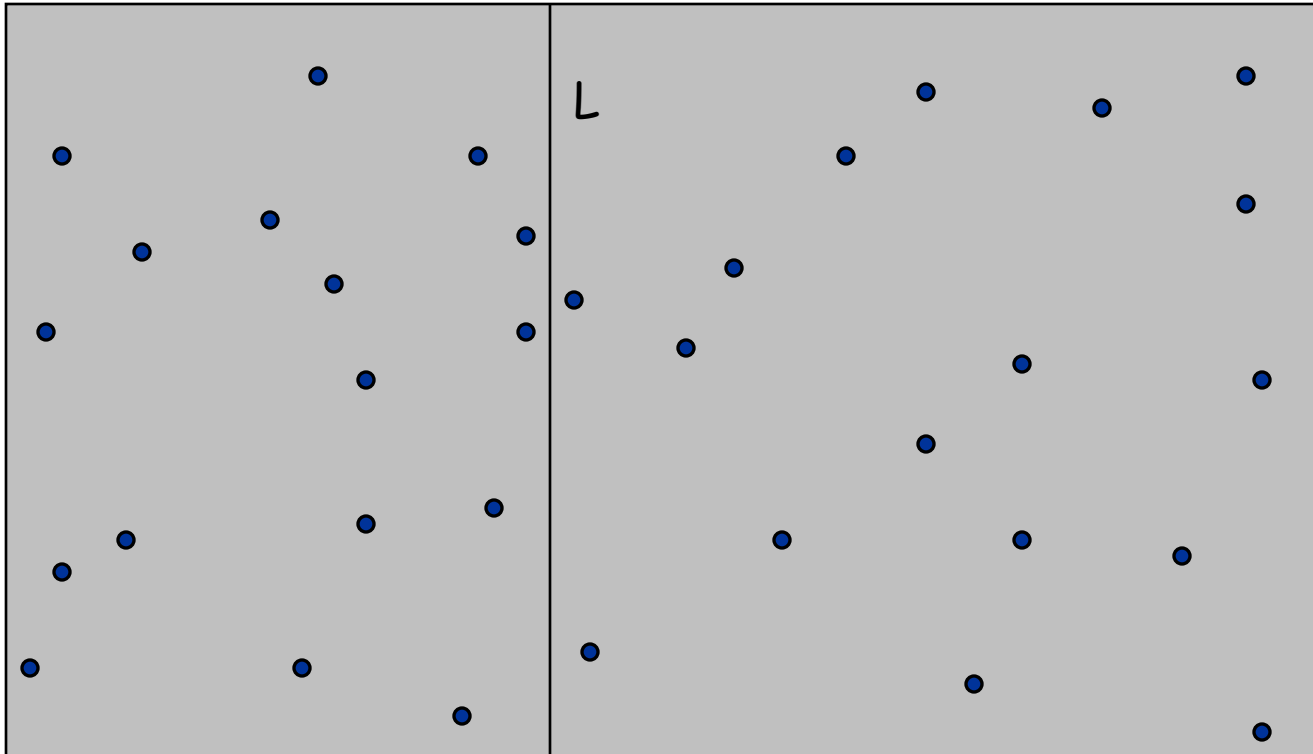
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest Pair of Points : Final Count Down

Algorithm.

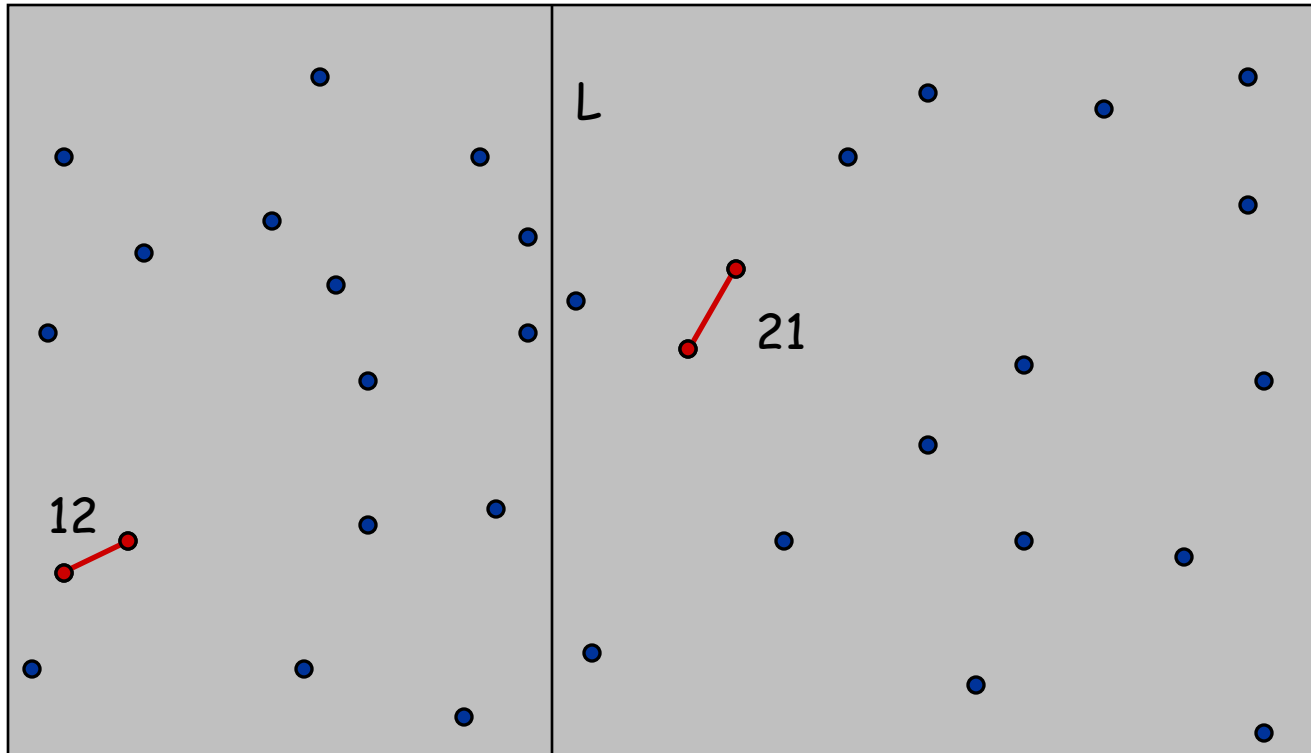
- **Divide:** draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

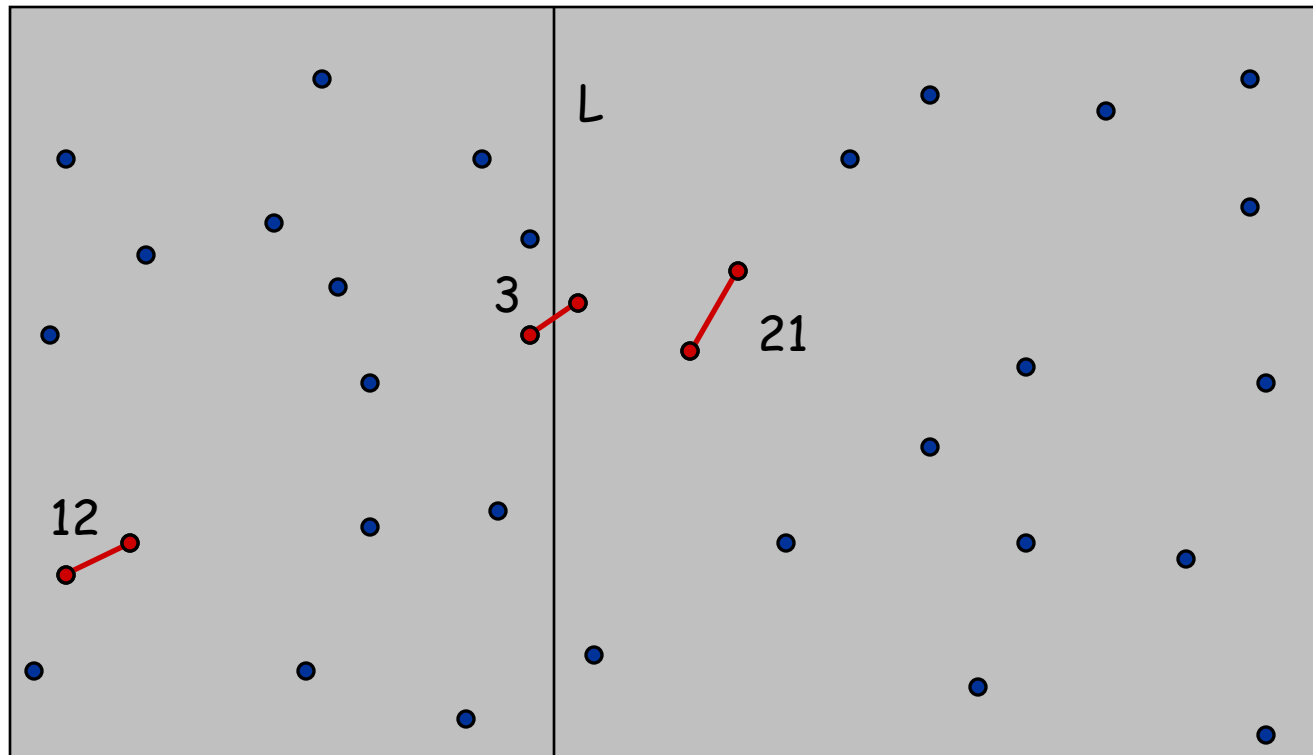
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair on each side recursively.



Closest Pair of Points

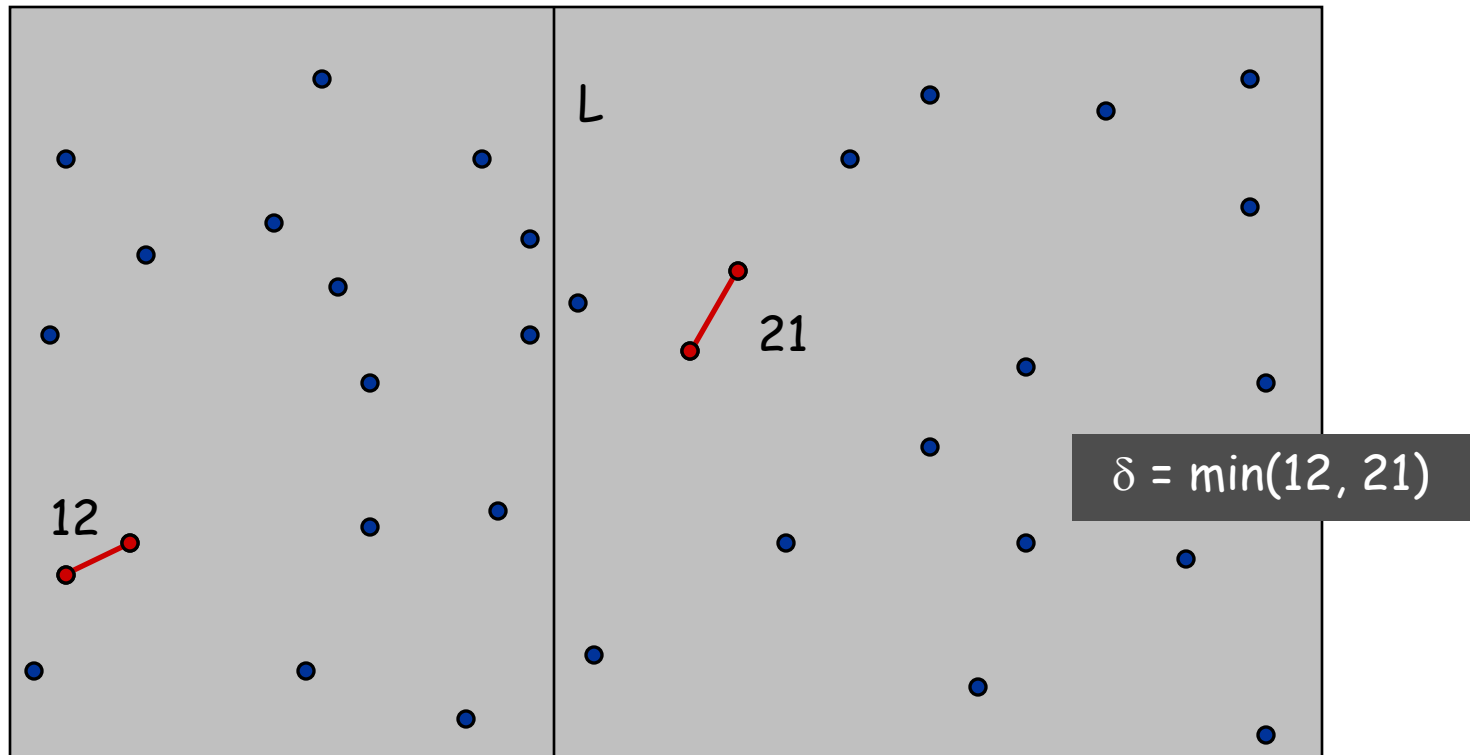
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair on each side recursively.
- **Combine**: find closest pair with one point on each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

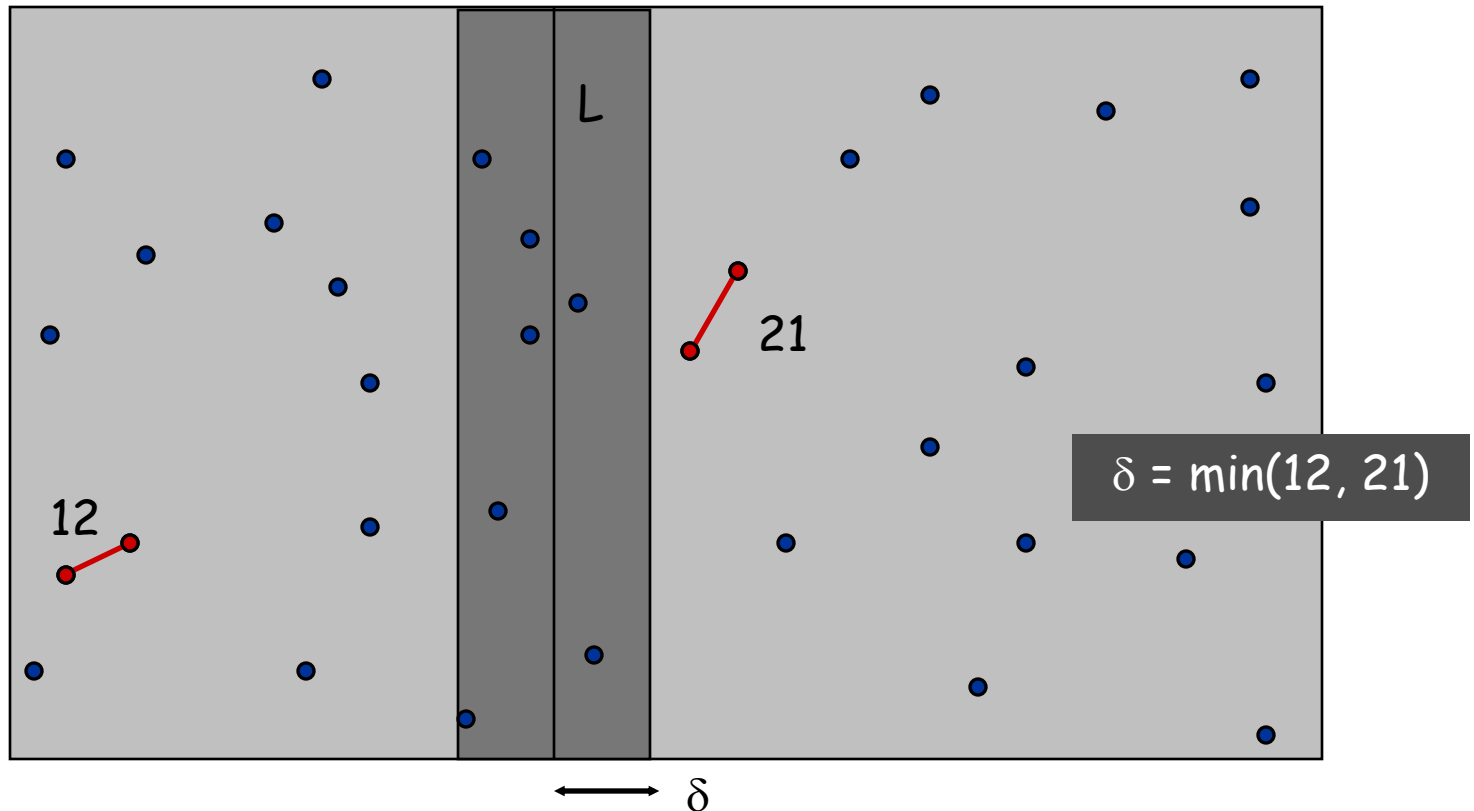
Find closest pair with one point on each side, assuming that distance $< \delta$.



Closest Pair of Points

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

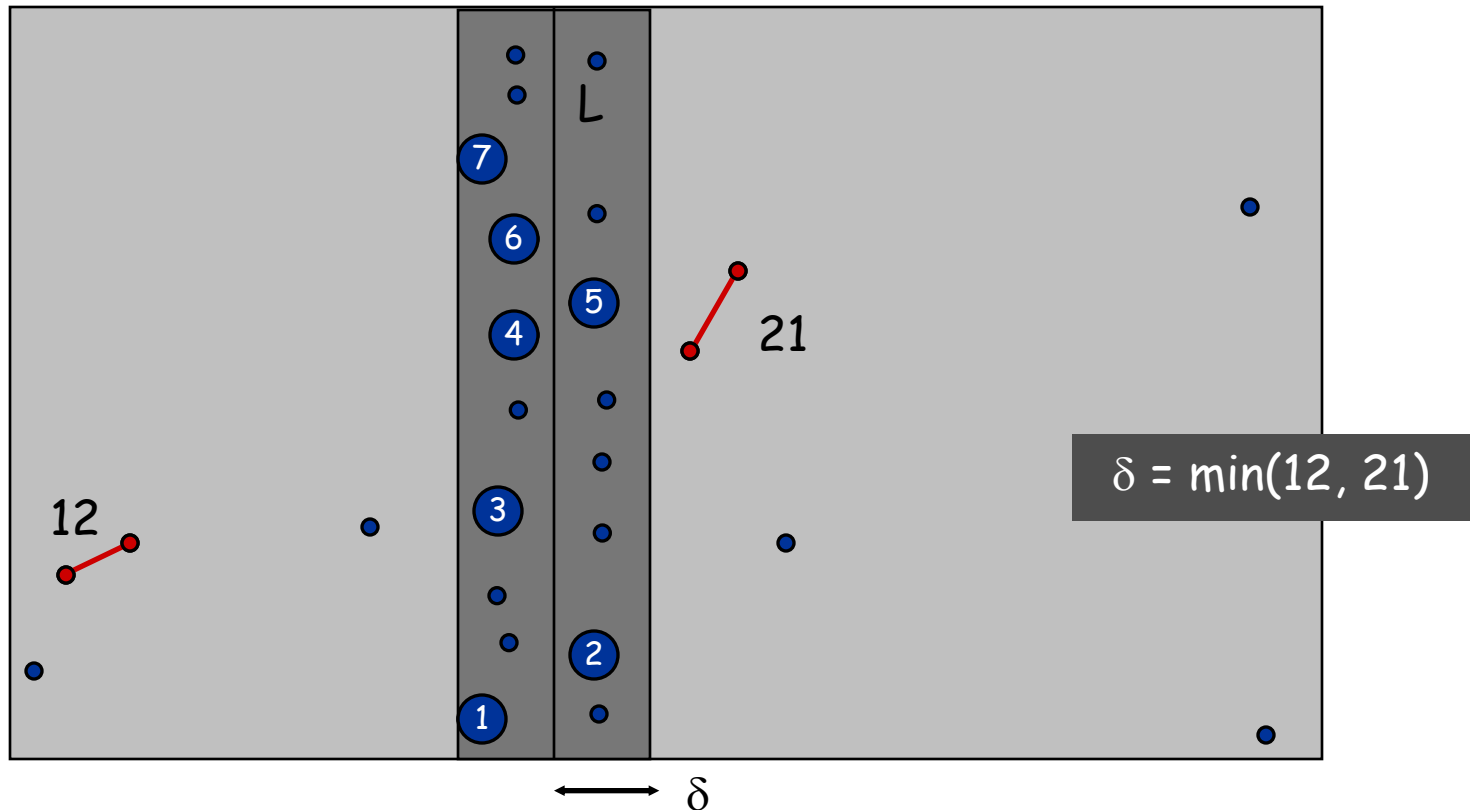
- **Observation:** need to consider points only within δ of line L .



Closest Pair of Points

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

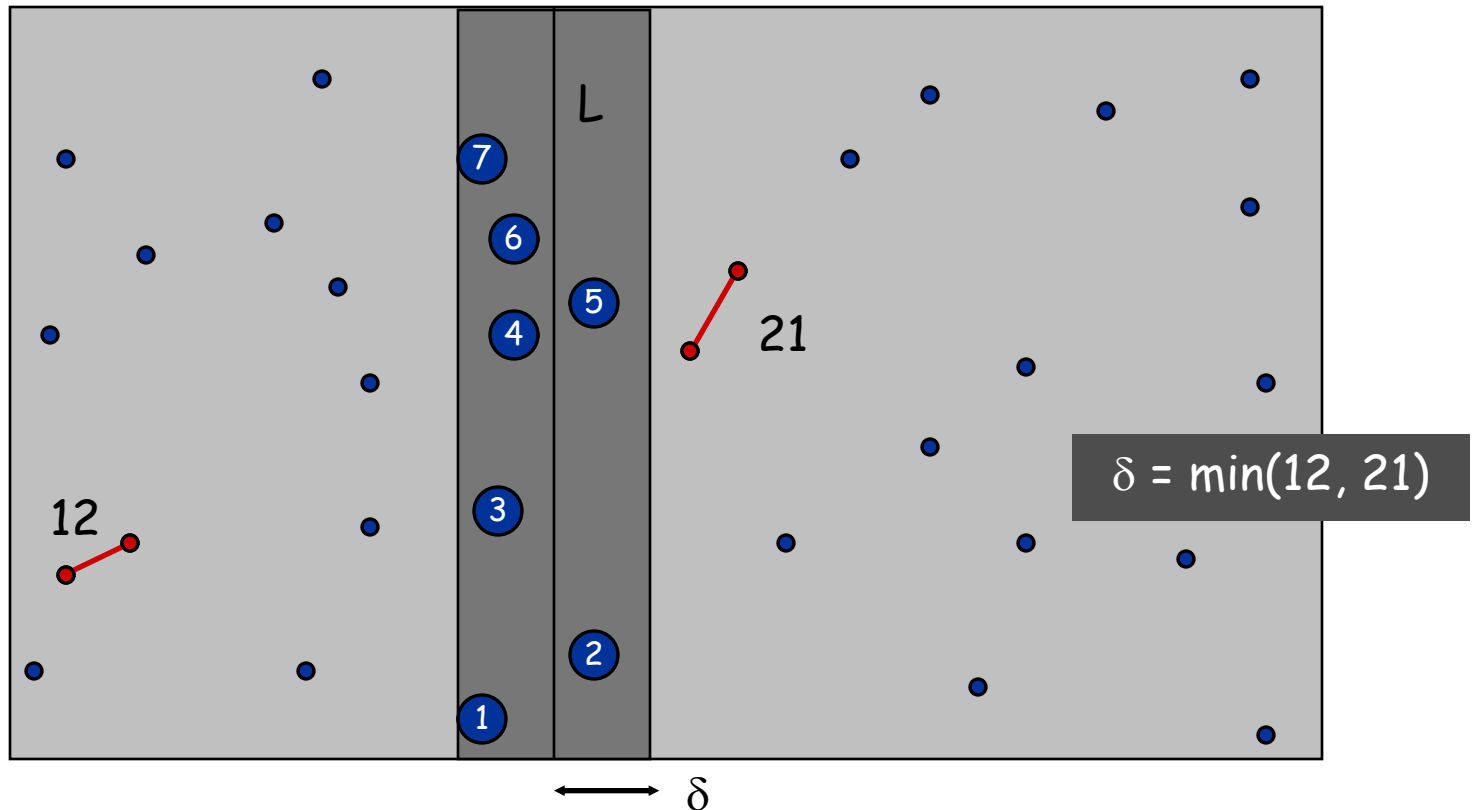
- **Observation:** only need to consider points within δ of line L .
- Unfortunately, this can degenerate into determining closest points among $O(n/2) \times O(n/2)$ point pairs in the worst case.



Closest Pair of Points

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

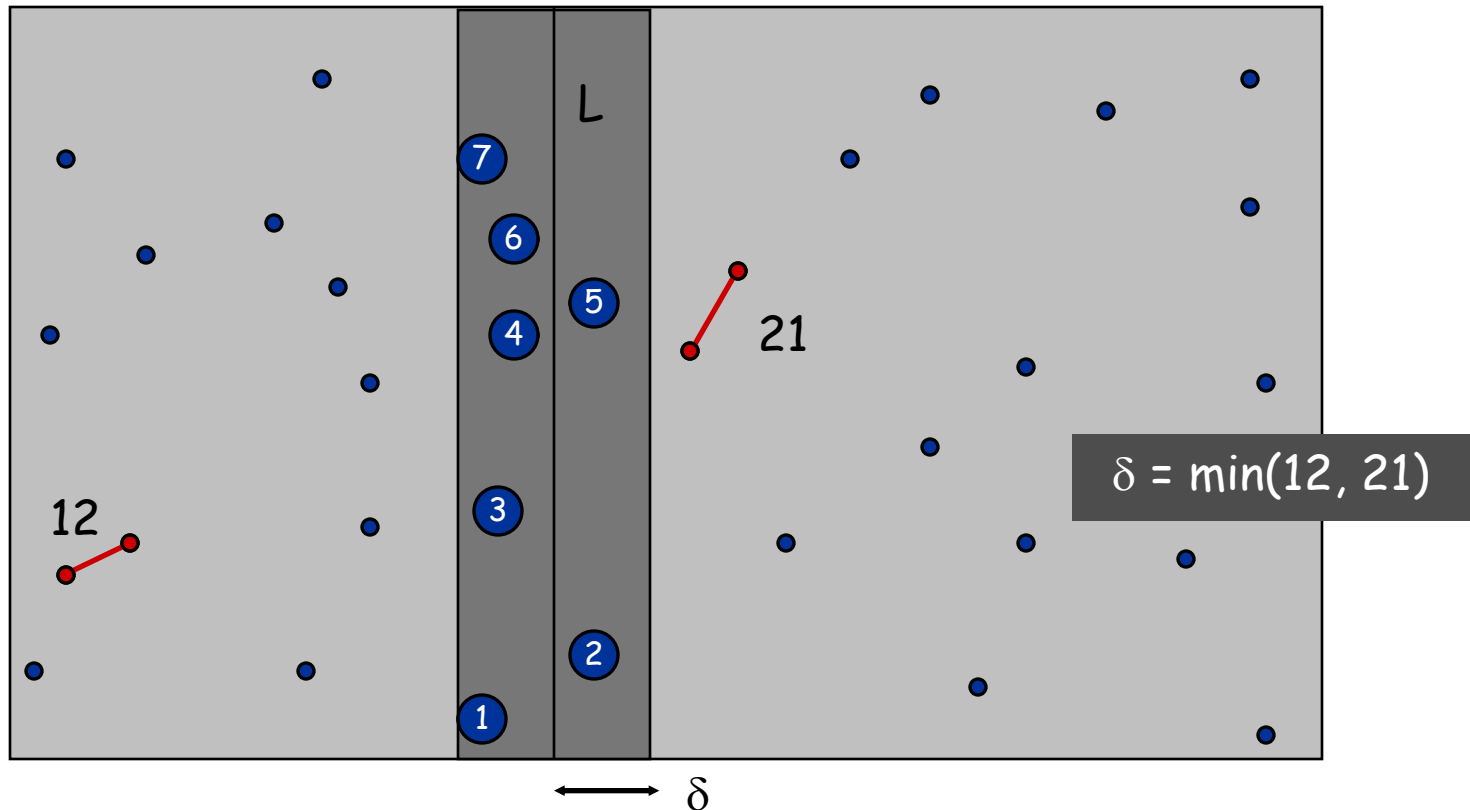
- **Observation:** only need to consider points within δ of line L .
- **Unfortunately,** this can degenerate into determining closest points among $O(n/2) \times O(n/2)$ point pairs in the worst case.
- **Sort points in 2δ -strip by their y coordinate.**



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.
- **Key Insight:** Check distances of only those points within 11 positions of each point in sorted list! (linear-time)



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

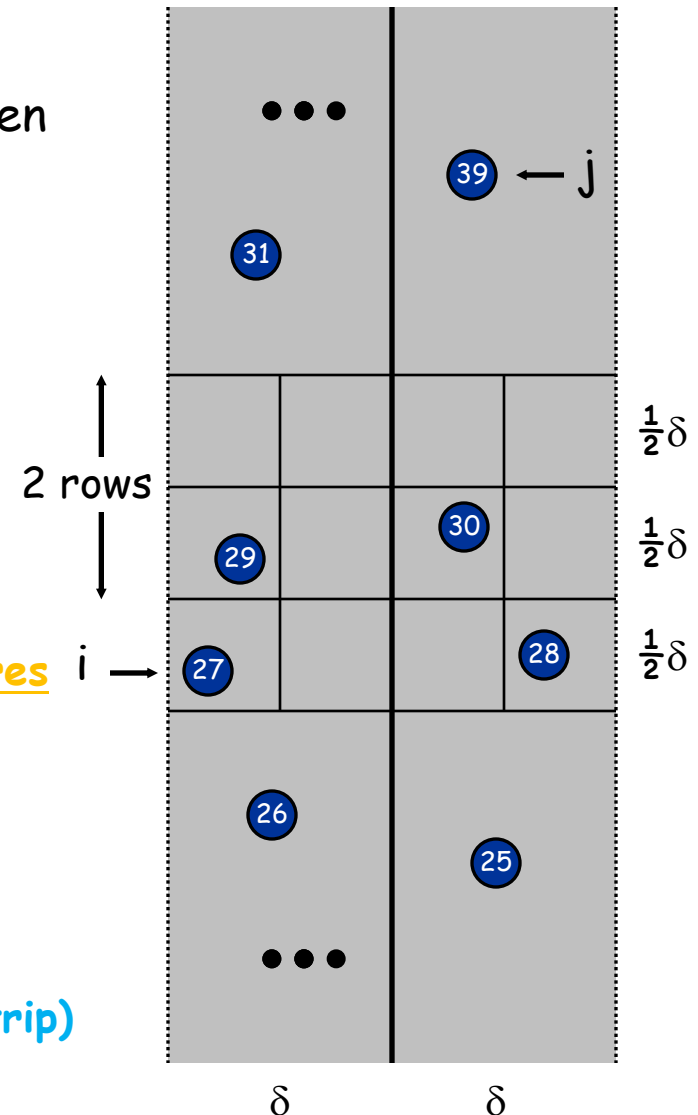
Fact. Points can require less than 12 pts.

- E.g. For pt. 28, need to check with three squares on the other side of the median line, or for pt. 30, six squares.

E.g., For pt. 27, check < 7 points in 2 rows.

-(Constant time for each point $\times O(n)$

Using y -coordinate sorted points in the 2δ -strip)



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 farther 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  $\delta$ .  
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

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 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.

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

5.5 Integer Multiplication

Integer Arithmetic

Add. Given two n -bit integers a and b , compute $a + b$.

- $O(n)$ bit operations.

Multiply. Given two n -bit 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

[illegible]

Divide-and-Conquer Multiplication: Warmup

To multiply two n -bit integers:

- **Multiply Four** pairs of $\frac{1}{2}n$ -bit integers.
- **Add Three** pairs of $\frac{1}{2}n$ -bit integers, and **shift** to obtain result.

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

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



assumes n is a power of 2

Karatsuba Multiplication

To multiply two n-bit integers:

- Add Two pairs of $\frac{1}{2}n$ -bit integers.
- Multiply Three pairs of $\frac{1}{2}n$ -bit integers.
- Add Two pairs, subtract Two pairs, and shift Two n-bit integers to obtain result.

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

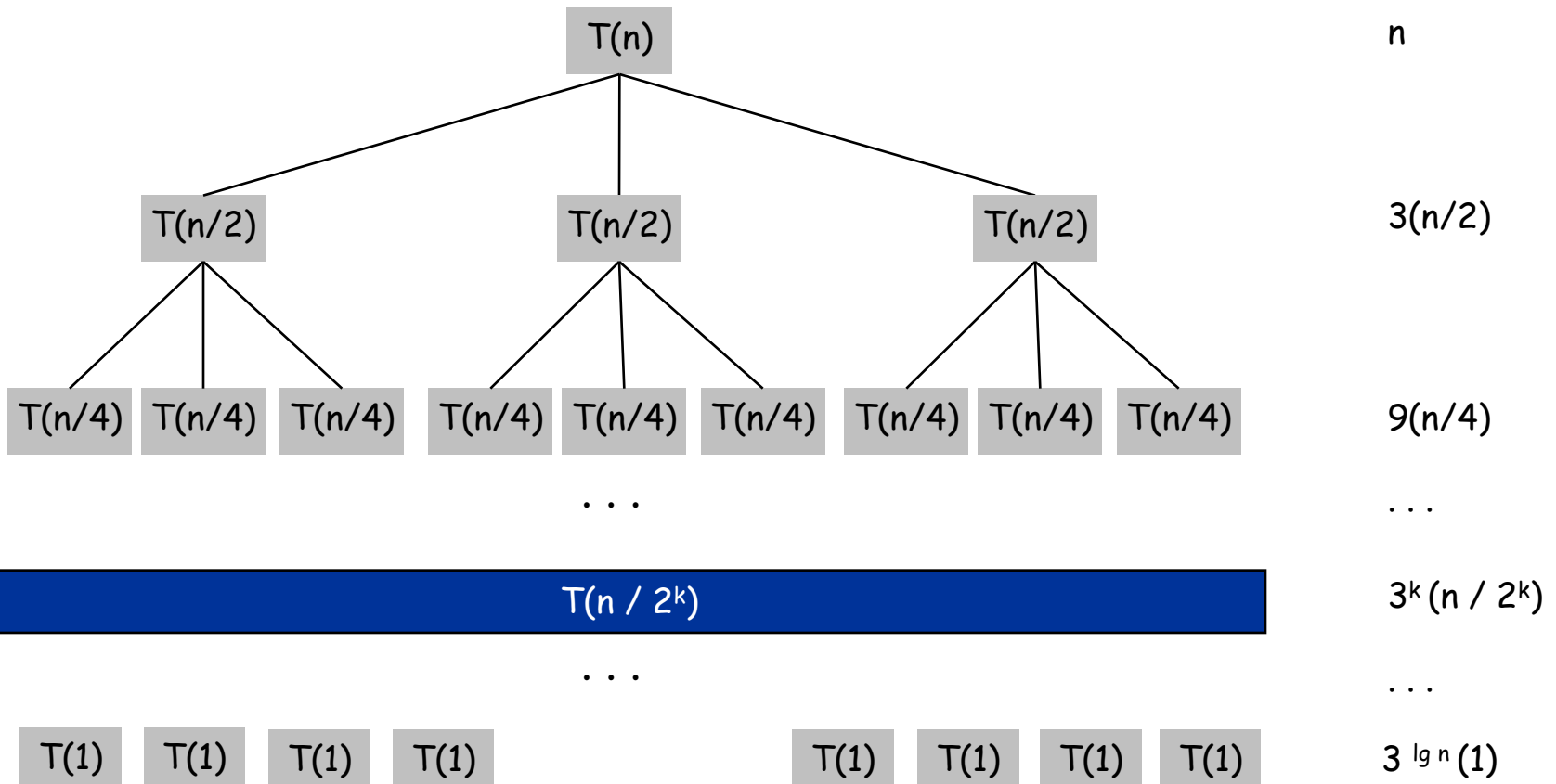
Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-bit integers in $O(n^{1.585})$ bit operations.

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

Karatsuba: Recursion Tree

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

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



Karatsuba Multiplication

KARATSUBA-MULTIPLY(x, y, n)

IF ($n = 1$)

RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{KARATSUBA-MULTIPLY}(a, c, m)$.

$f \leftarrow \text{KARATSUBA-MULTIPLY}(b, d, m)$.

$g \leftarrow \text{KARATSUBA-MULTIPLY}(a - b, c - d, m)$.

RETURN $2^{2m} e + 2^m (e + f - g) + f$.

Practice: Faster than grade-school algorithm for about 320-640 bits.

History of asymptotic complexity of integer multiplication

year	algorithm	order of growth
?	brute force	$\Theta(n^2)$
1962	Karatsuba-Ofman	$\Theta(n^{1.585})$
1963	Toom-3, Toom-4	$\Theta(n^{1.465}), \Theta(n^{1.404})$
1966	Toom-Cook	$\Theta(n^{1+\epsilon})$
1971	Schönhage–Strassen	$\Theta(n \log n \log \log n)$
2007	Fürer	$n \log n 2^{O(\log^* n)}$
?	?	$\Theta(n)$

number of bit operations to multiply two n -bit integers

used in Maple, Mathematica, gcc, cryptography, ...

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

GMP
«Arithmetic without limitations»

integer arithmetic problems with same complexity:

integer division, integer square, integer square root

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 = 11 additions + 7 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:** 7 $\frac{1}{2}n$ -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.

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

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.

Linear algebra reductions

Matrix multiplication. Given two n -by- n matrices, compute their product.

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$ A $	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L U$	$\Theta(MM(n))$
least squares	$\min \ Ax - b\ _2$	$\Theta(MM(n))$

numerical linear algebra problems with the same complexity as matrix multiplication

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: $O(n^{2.521801})$.

History of asymptotic complexity of matrix multiplication

year	algorithm	order of growth
?	brute force	$O(n^3)$
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	$O(n^{2.376})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.3727})$
?	?	$O(n^{2+\epsilon})$

number of floating-point operations to multiply two n -by- n matrices