

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



See 05demo-merge.ppt

- 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.

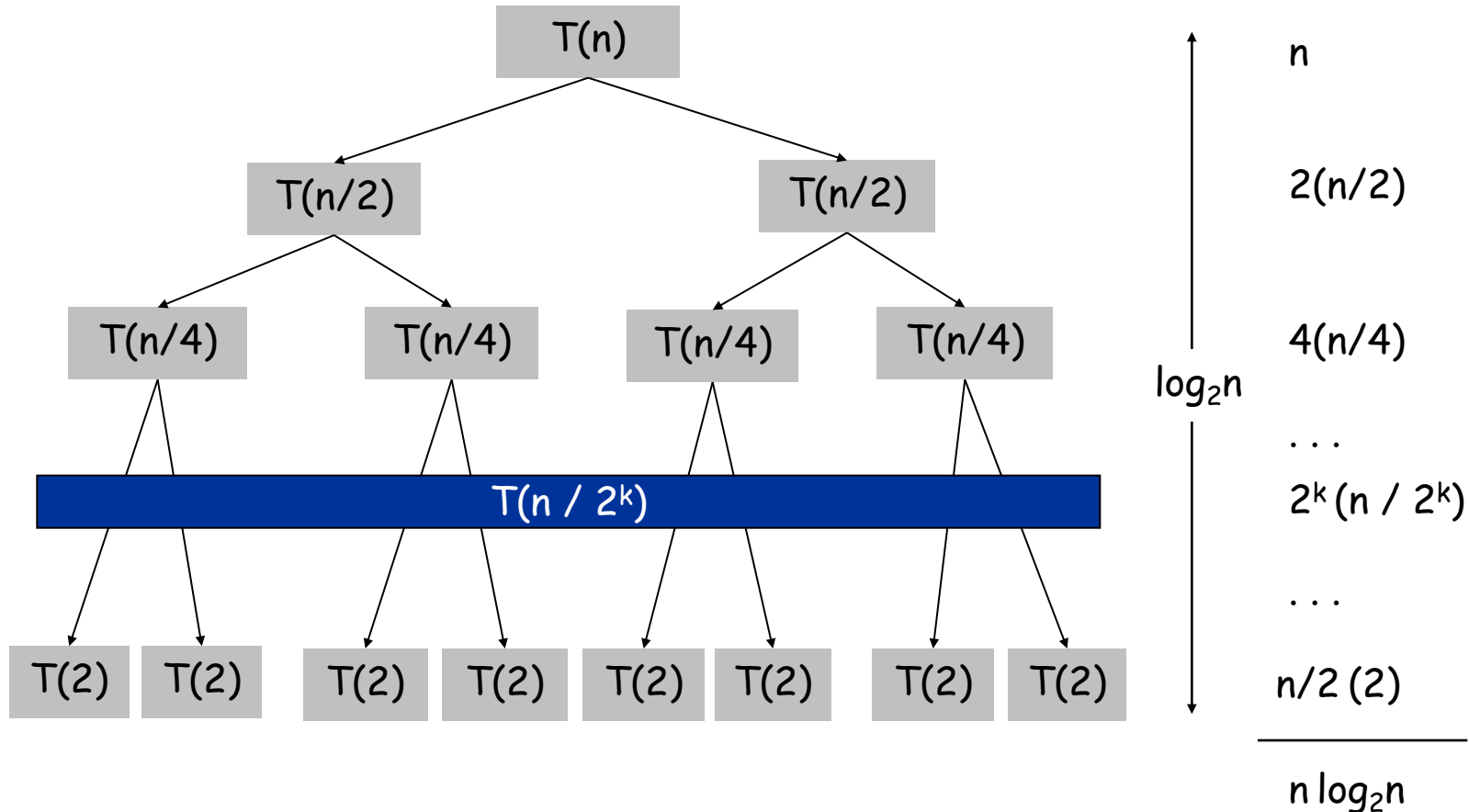
$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

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

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



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

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

Alternative proof

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

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 step: assume true for $1, 2, \dots, n-1$.

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

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

Since $n \geq 2$

5.2 Further Recurrence Relations

Note that we ignore the ceilings and floors, i.e, we approximate:

$$\underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} \approx \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} \approx T(n/2)$$

5.2. Further Recurrence Relations

$$T(n) \leq \begin{cases} c & \text{if } n = 2 \\ \underbrace{qT(n/2)}_{\text{solve half problem}} + \underbrace{cn}_{\text{merge}} & \text{if } n > 2 \end{cases}$$

Three different cases:

- a) $q=1$: $T(n) = O(n)$
- b) $q=2$: $T(n) = O(n \log_2 n)$
- c) $q>2$: $T(n) = O(n^{\log_2 q})$

Proofs: Use the recursion tree and the sum of the geometric series

$$\sum_{j=0}^{\log_2 n - 1} r^j \leq \frac{r^{\log_2 n} - 1}{r - 1} \leq \frac{r^{\log_2 n}}{r - 1} \quad \text{if } r > 1$$

$$\sum_{j=0}^{\log_2 n - 1} \frac{1}{2^j} \leq \sum_{j=0}^{\infty} \frac{1}{2^j} = 2$$

5.2. Further Recurrence Relations

$$T(n) \leq \begin{cases} c & \text{if } n = 2 \\ \underbrace{2T(n/2)}_{\text{solve half problem}} + \underbrace{cn^2}_{\text{merge}} & \text{if } n > 2 \end{cases}$$

Solution: Use the recursion tree to find the following sum:

$$T(n) \leq \sum_{j=0}^{\log_2 n - 1} \frac{cn^2}{2^j} = cn^2 \sum_{j=0}^{\log_2 n - 1} \frac{1}{2^j} \leq 2cn^2 = O(n^2)$$

5.2. Further Recurrence Relations

Note:

in Chapter 4: **Greedy Algorithms** we found $O(n)$ or $O(n \log n)$ solutions to problems whose brute force solutions would take **exponential time in n** ($O(q^n)$, $q > 2$).

The **Divide and Conquer** Algorithms will compute polynomial $O(n^q)$ or $O(n \log n)$ solutions to algorithms whose brute force running time are $O(n^p)$ where $p \geq q$.

5.3 Counting Inversions

Counting Inversions

Ranking Problem:

Music site tries to match your song preferences with others.

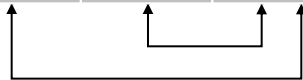
- You rank n songs.
- Music site consults database to find people with **similar** tastes.

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.

- 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).

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.

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

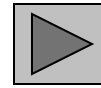
Combine: ???

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.



See: 05demo-merge-invert.ppt

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: $O(n)$

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

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  
    A contains the first  $\lceil n/2 \rceil$  elements  
    B contains the remaining  $\lfloor n/2 \rfloor$  elements  
     $(r_A, A) \leftarrow \text{Sort-and-Count}(A)$   
     $(r_B, B) \leftarrow \text{Sort-and-Count}(B)$   
     $(r, L) \leftarrow \text{Merge-and-Count}(A, B)$   
  
    return  $r_A + r_B + r$  and the sorted list L  
}
```


Counting Inversions: Implementation

[Merge-and-Count] Algorithm

Merge-and-Count(A,B)

Maintain a *Current* pointer into each list, initialized to point to the front elements.

Maintain a variable *Count* for the number of inversions, *Count* is initialized to 0.

While both lists are nonempty:

Let a_i and b_j be the elements pointed to by the *Current* pointer

Append the smaller of these to the output list

if b_j is the smaller element then

Increment *Count* by the no of elements remaining in A

Endif

Advance the *Current* pointer in the list from which the smaller element was selected.

EndWhile

Once one list is empty, append the remainder of the other list to the output.

Return *Count* and the merged list

5.4 Closest Pair of Points

(Reading Assignment)

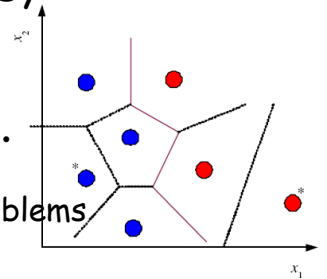
Closest Pair of Points

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.

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

Sort the points: $O(n \log n)$

Walk through the list keeping track of the min dist $O(n)$

Assumption. No two points have same x coordinate.

↑
to make presentation cleaner

Closest Pair of Points: Divide and Conquer

Notation:

Set of points: $P = \{p_1, p_2, \dots, p_n\}$

p_i has coordinates (x_i, y_i)

$d(p_i, p_j)$: Euclidean distance between p_i and p_j

Goal: Find a pair of points p_i and p_j that minimizes $d(p_i, p_j)$

Divide and Conquer Idea:

Find the closest pair of points in the left half of P

Find the closest pair of points in the right half of P

Use this information to get the overall solution (**combine**) in linear time.

Combination part is tricky. Distances that have not been considered, the ones between the left and right half are $\Omega(n^2)$, but we need to find an $O(n)$ algorithm to find the smallest one!

Closest Pair of Points: Divide and Conquer

Preprocessing: $O(n \log n)$

We maintain two lists P_x and P_y

P_x sort all points in P by x coordinate

P_y sort all points in P by y coordinate

For each point in P_x and P_y attach the position of the point in both lists.

Divide and Conquer:

Let:

Q : set of $\lfloor n/2 \rfloor$ points in the first (left) half of P_x

R : set of $\lfloor n/2 \rfloor$ points in the remaining (right) half of P_x

Produce lists Q_x , Q_y and R_x , R_y analogous to P_x and P_y

Assume:

q_0^* and q_1^* are the closest pair of points in Q and

r_0^* and r_1^* are the closest pair of points in R .

Closest Pair of Points: Divide and Conquer

Combine:

Let: $\delta = \min(d(q_0^*, q_1^*), d(r_0^*, r_1^*))$

Are there points $q \in Q$ and $r \in R$ such that $d(q, r) < \delta$?

If no: then we already found the closest pair of points,
they are either (q_0^*, q_1^*) or (r_0^*, r_1^*)

If yes: Let L be the vertical line with equation $x = x^*$ where x^* is the
rightmost point in Q . L separates Q from R .

Claim 5.8.: If (q, r) are the closest points, then each of them lies within
a δ distance of L .

Proof: Let $q = (q_x, q_y)$ and $r = (r_x, r_y)$. By defn of x^* , $q_x \leq x^* \leq r_x$

Then: $x^* - q_x \leq r_x - q_x \leq d(q, r) < \delta$ and $r_x - x^* \leq r_x - q_x \leq d(q, r) < \delta$

Closest Pair of Points: Divide and Conquer

Idea: Narrow search to points that lie within δ distance of L .

Let S set of points that lie within δ distance of L .

Let S_y be list of points in S in increasing order of y ($O(n)$ time using P_y)

Claim 5.9 (Restate 5.8) There exists $q \in Q$ and $r \in R$ for which $d(q,r) < \delta$ if and only if there exists s and s' for which $d(s,s') < \delta$

Claim 5.10. If s and s' have the property that $d(s,s') < \delta$ then s and s' are within 15 positions of each other in the sorted list S_y .

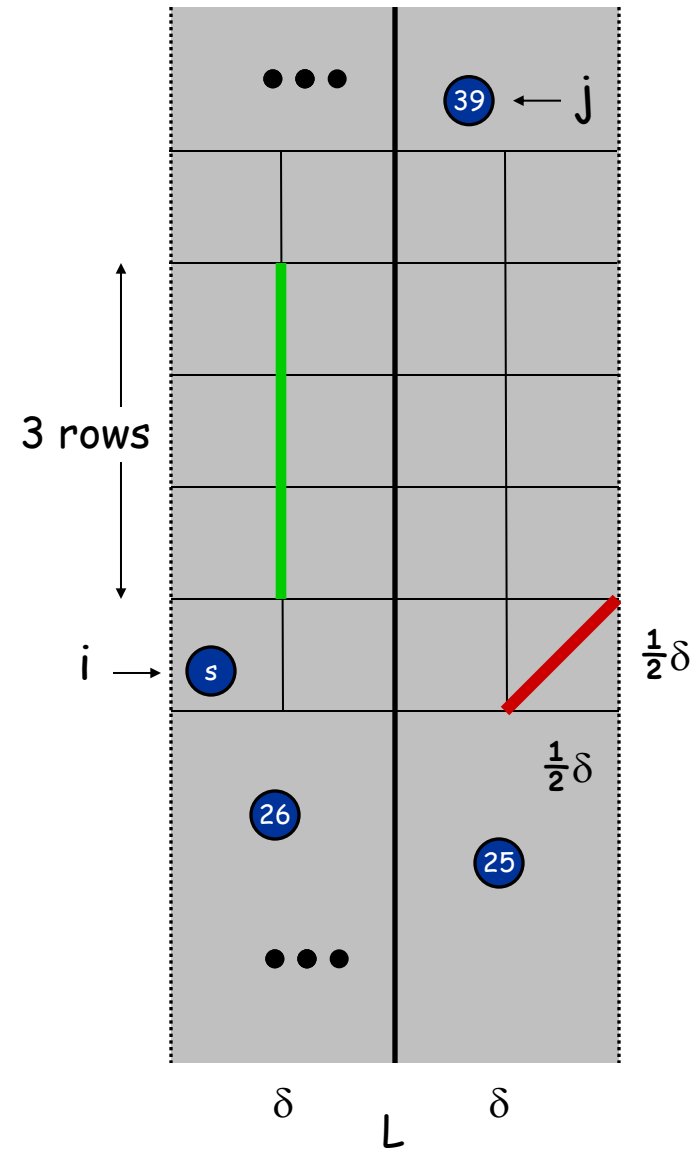
Combination Algorithm: Go through the points in S_y in order, for each point compute the distance between it and the next 15 points. $O(15*n)$

Claim 5.10. If s and s' have the property that $d(s,s') < \delta$ then s and s' are within 15 positions of each other in the sorted list S_y .

Proof:

Suppose two points s and s' of S lie in the same box. Since all boxes are in the same side of L , then s and s' either both belong to Q or both belong to R . But any two points in the same box are within distance $\delta \cdot \sqrt{2}/2 < \delta$ which contradicts the definition of δ as the min dist between any two pair of points in Q or R .

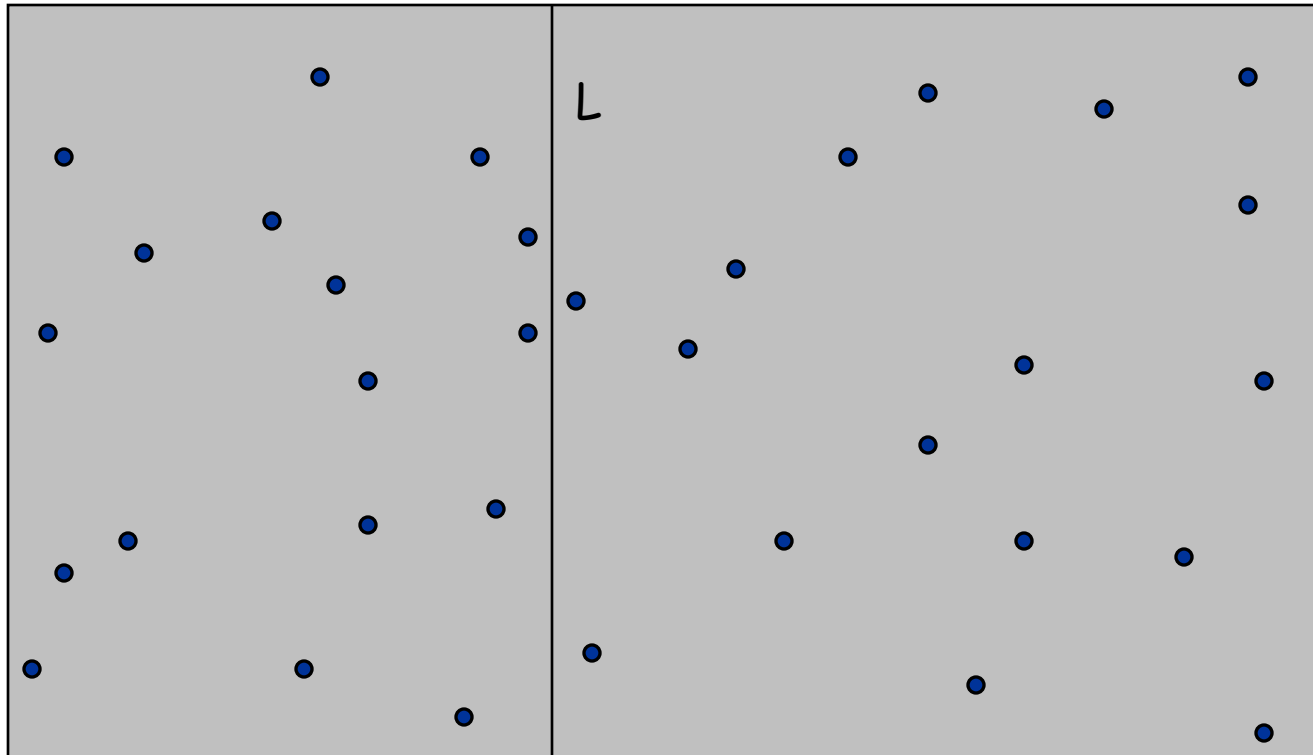
Suppose s and s' have the property that $d(s,s') < \delta$ and they are 16 positions apart in S_y . w.o.l.o.f.gen assume that s has smaller y coordinate. Since there are at most one point per box, there are at least three rows of Z lying between s and s' . But any two points in Z separated by at least three rows must be a distance of at least $3\delta/2$ apart- a contradiction.



Closest Pair of Points

Algorithm.

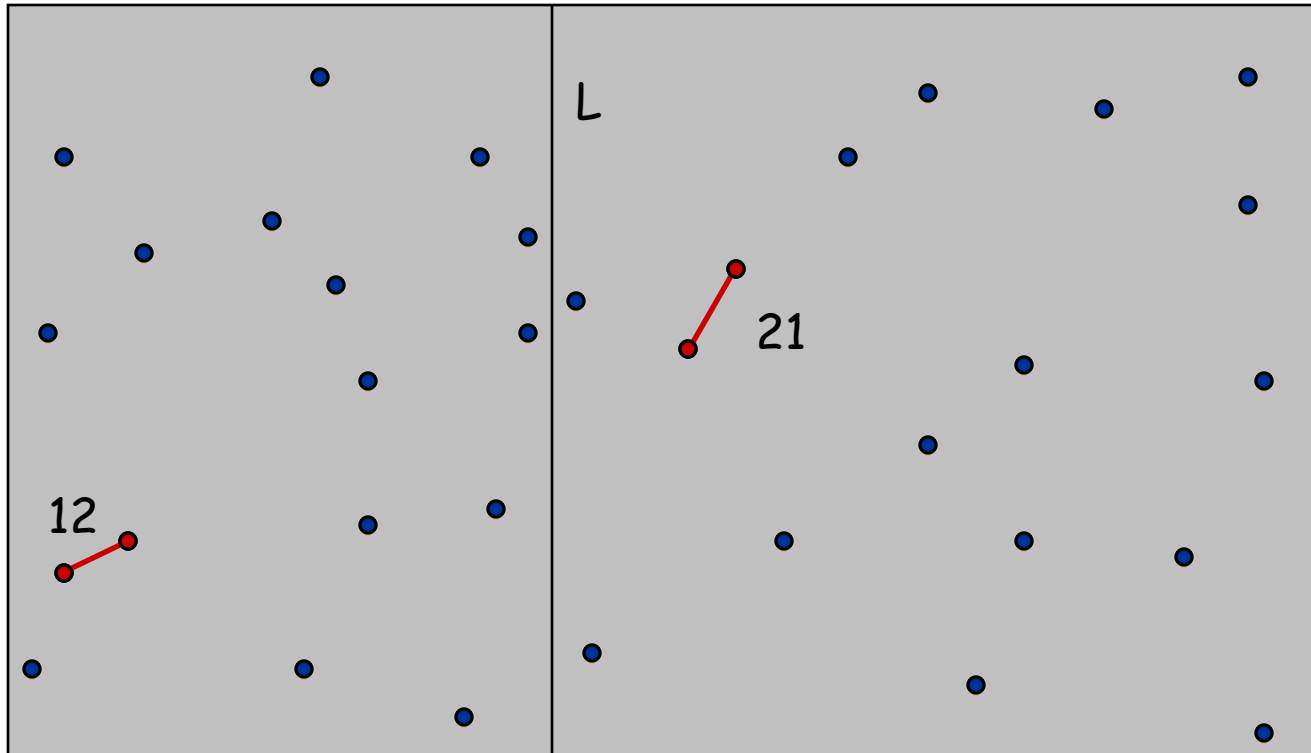
- **Divide:** draw vertical line L so that $\frac{1}{2}n$ points on each side



Closest Pair of Points

Algorithm.

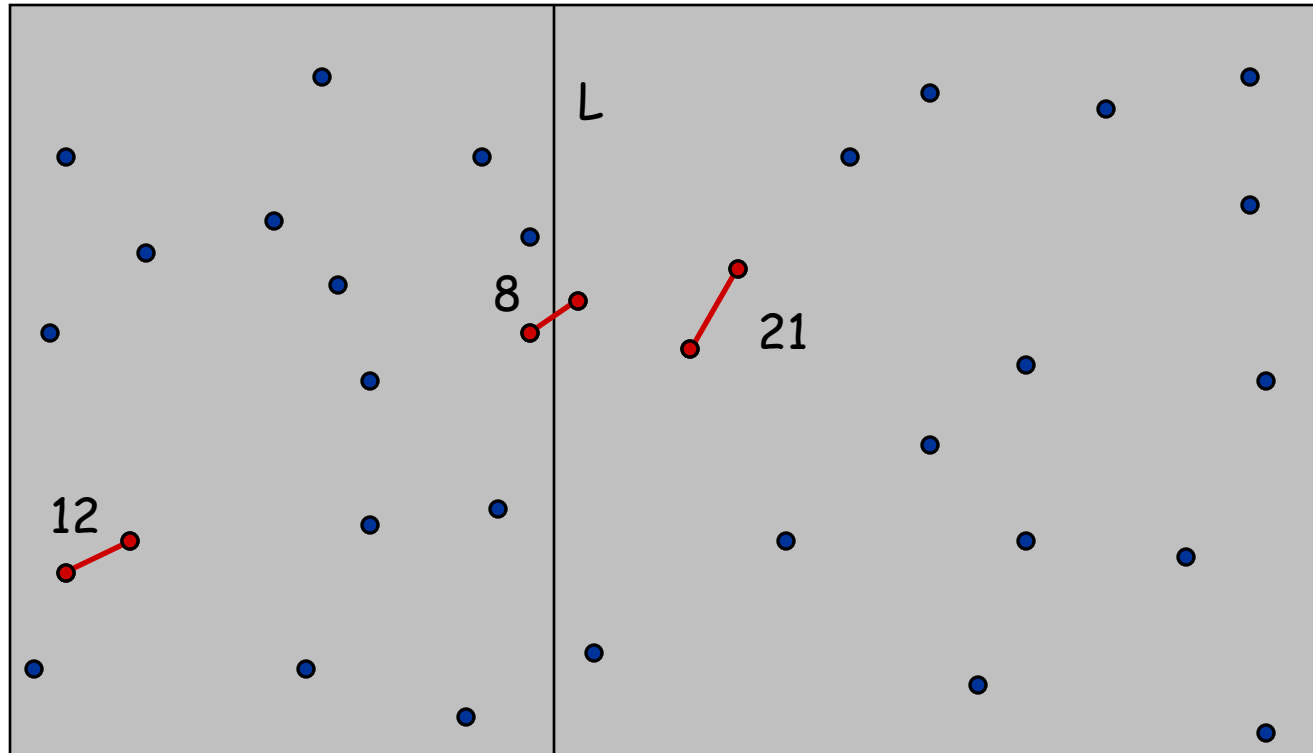
- Divide: draw vertical line L so that $\frac{1}{2}n$ points on each side
- **Conquer**: find closest pair in each side recursively.



Closest Pair of Points

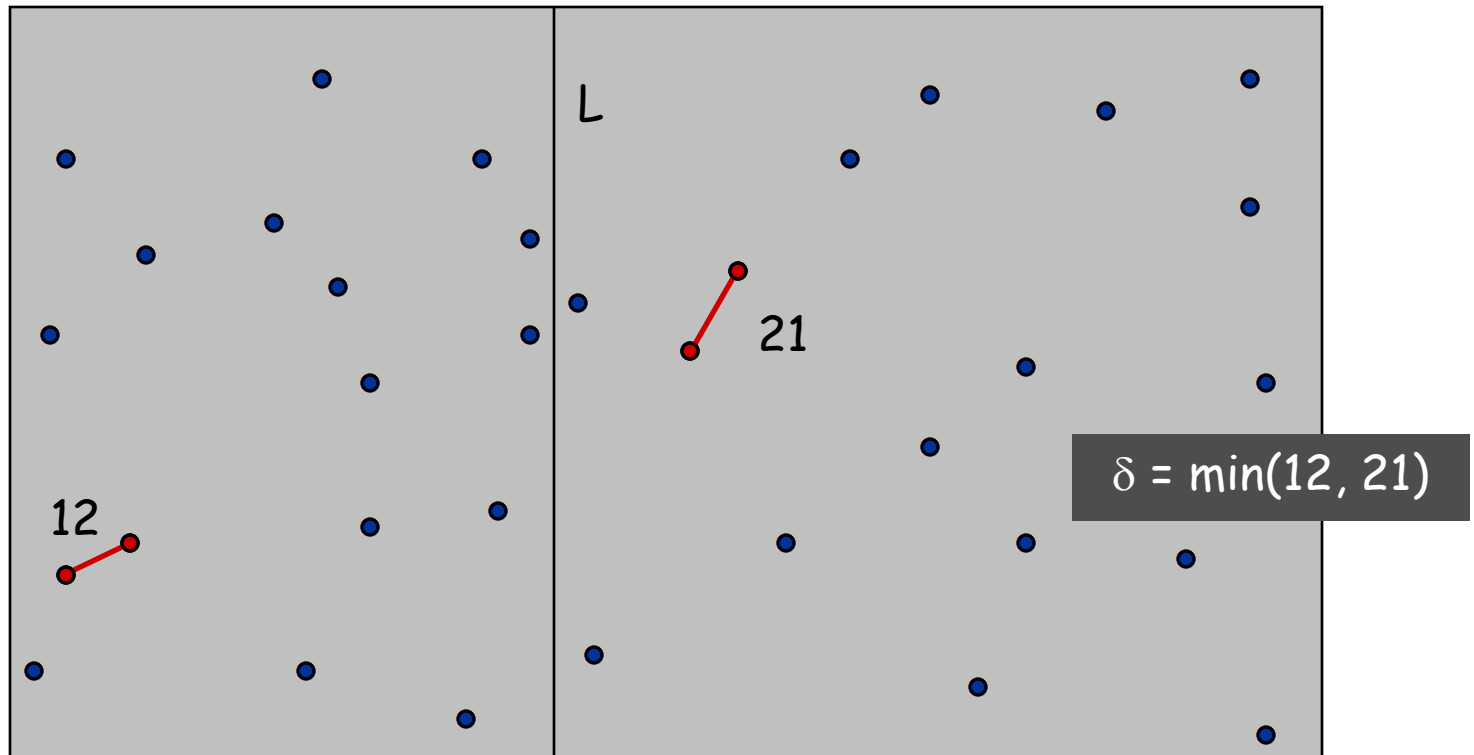
Algorithm.

- Divide: draw vertical line L so that $\frac{1}{2}n$ 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

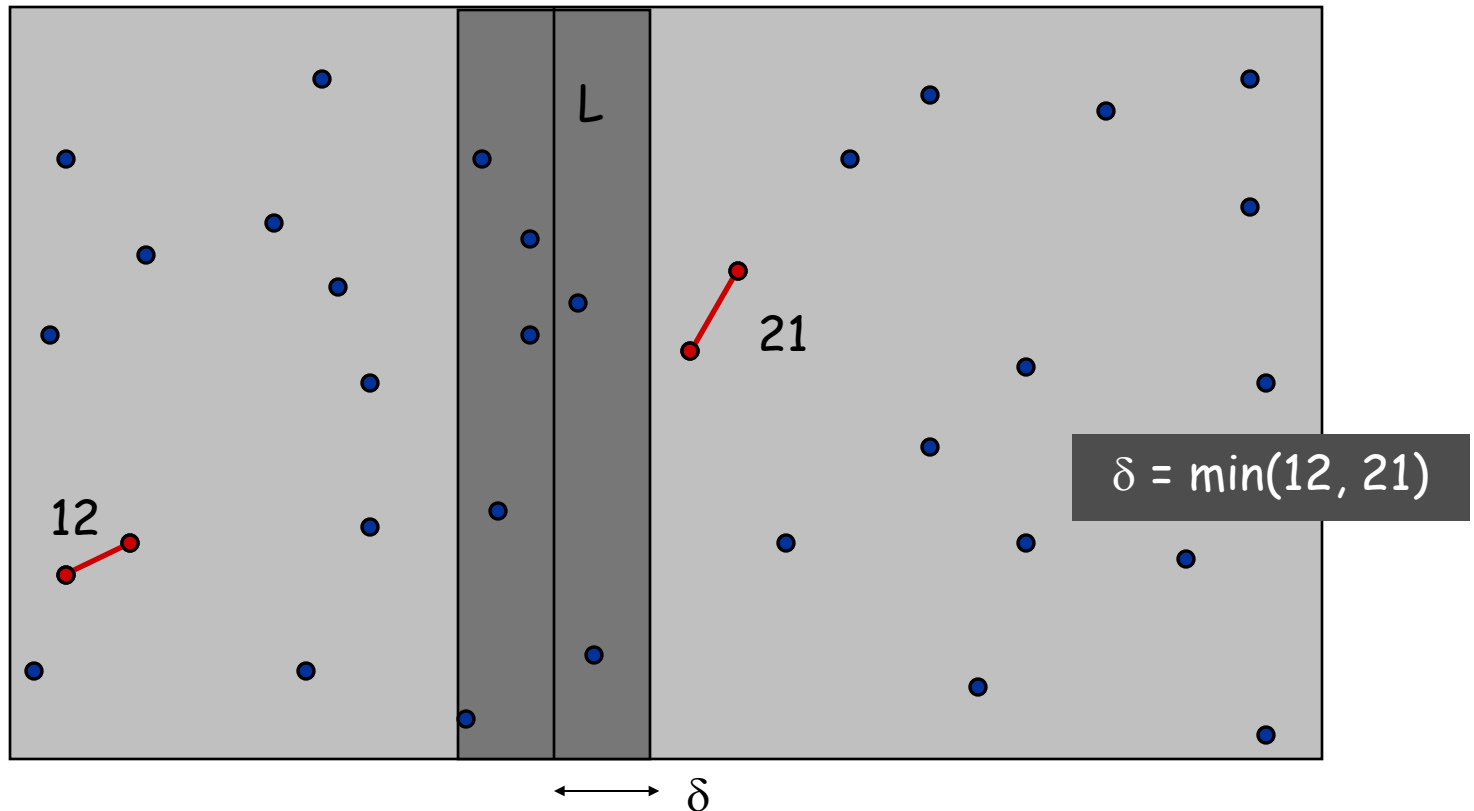
Find closest pair with one point in each side, **assuming that distance $< \delta$** .



Closest Pair of Points

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

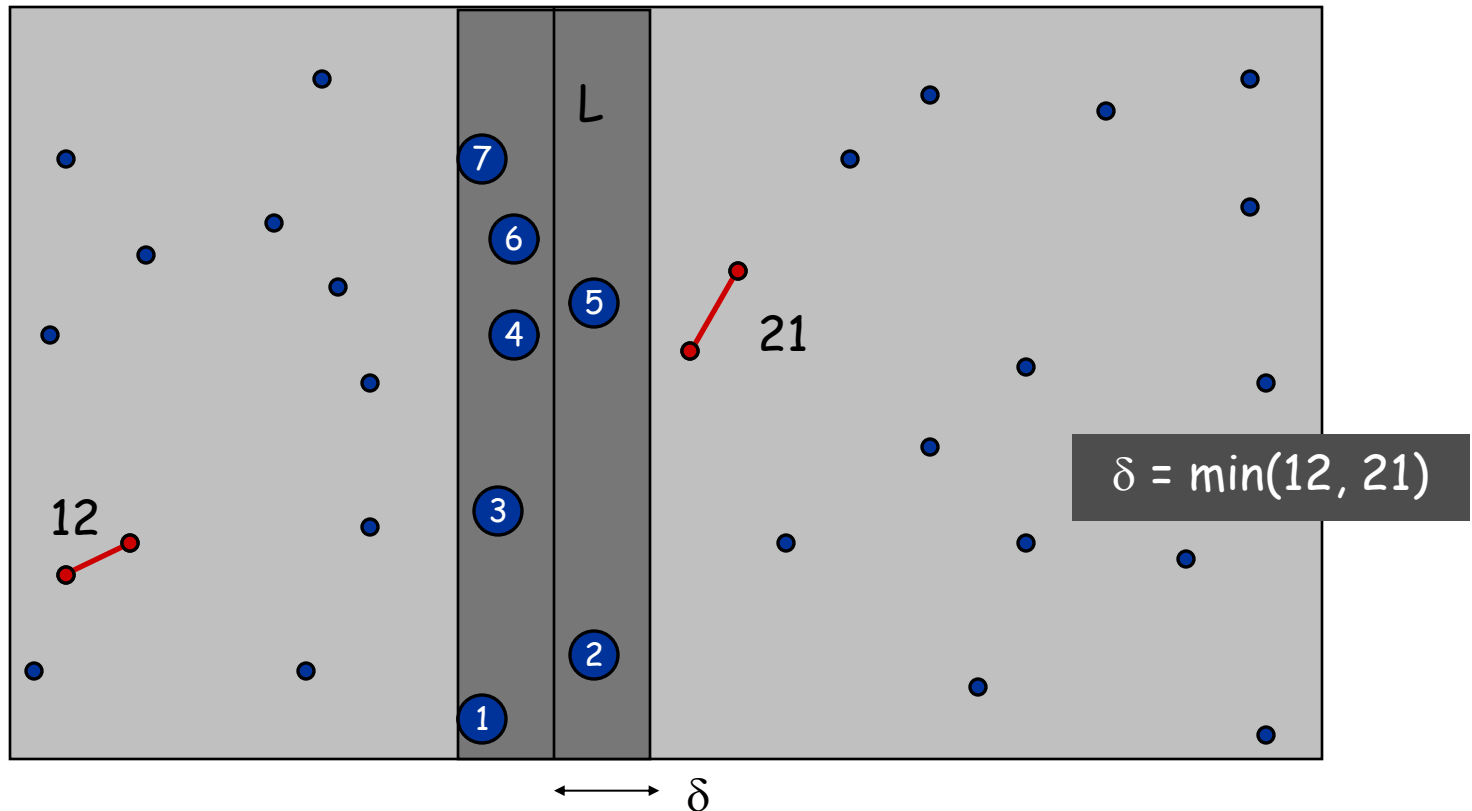
- Observation: only need to consider points within δ of line L .



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 (list S_y).

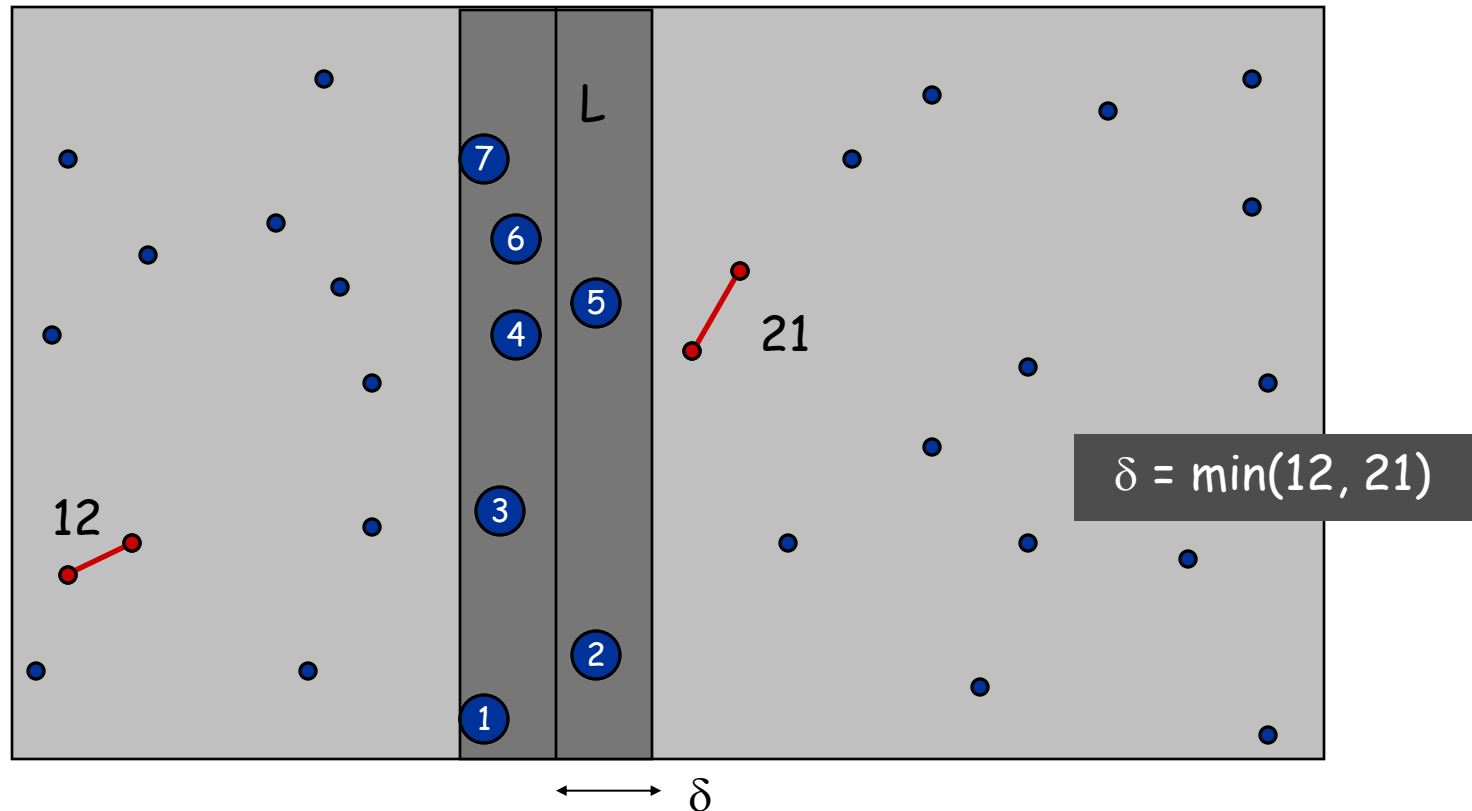


Closest Pair of Points

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 15 positions in sorted list!

Note: since δ is small, there can not be too many points within each δ strip.
Actually, there are max C_d points only, a const depending on no of dimensions, $d=2$



Closest Pair Algorithm

Closest-Pair(P)

Construct P_x and P_y

$(p0^*, p1^*) = \text{Closest-Pair-Rec}(P_x, P_y)$

$O(n \log n)$

Closest-Pair-Rec(P_x, P_y)

If $|P| \leq 3$ then find closest pair measuring all pairwise distances.

Endif

Construct Q_x, Q_y, R_x, R_y

$(q0^*, q1^*) = \text{Closest-Pair-Rec}(Q_x, Q_y)$

$O(n)$

$(r0^*, r1^*) = \text{Closest-Pair-Rec}(R_x, R_y)$

$2T(n/2)$

$\delta = \min(d(q0^*, q1^*), d(r0^*, r1^*))$

$x^* = \max$ x-coordinate of a point in set Q

$L = \{(x, y) : x = x^*\}$

S : Points in P within distance δ of L

Construct S_y

$O(n)$

For each point s in S_y , compute distance from s to each of next 15 points in S_y .

$O(n)$

Let s, s' pair achieve the min distance

If $d(s, s') < \delta$ then return (s, s')

Else if $d(q0^*, q1^*) < d(r0^*, r1^*)$ then return $(q0^*, q1^*)$

Else return $(r0^*, r1^*)$

Closest Pair of Points: Analysis

Running time:

Preprocessing time to produce P_x and P_y : $O(n \log n)$

The rest of the code ($\text{Closest-Pair-Rec}(P_x, P_y)$):

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

Total: $O(n \log n)$

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.

```
  12
  13
X-----
  36
  12
+-----
156
```

Decimal Multiplication

```
  1100
  1101
X-----
  1100
  0000
  1100
  1100
+-----
10011100
```

Binary Multiplication

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.

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

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

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

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

is not any better
than brute force!

↑
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.

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

A

B

A

C

C

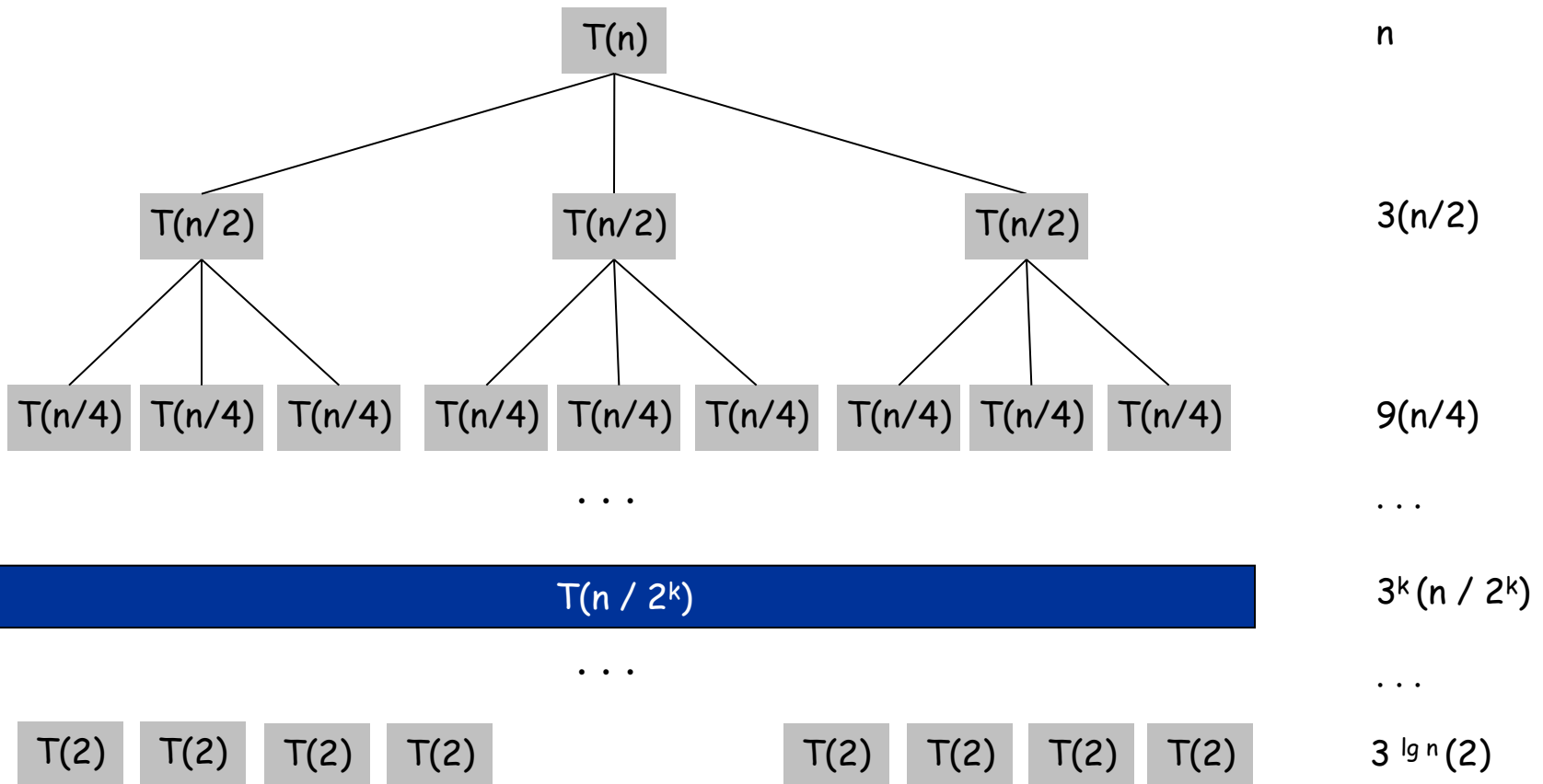
Theorem. [Karatsuba-Ofman, 1962] Can multiply two n -digit integers in $O(n^{1.585})$ bit operations.

$$T(n) = \underbrace{3T(n/2)}_{\text{recursive calls}} + \underbrace{cn}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Karatsuba: Recursion Tree

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

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



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.

$$\begin{pmatrix} \acute{C}_{11} & \grave{C}_{12} \\ \hat{C}_{21} & \check{C}_{22} \end{pmatrix} = \begin{pmatrix} \acute{A}_{11} & \grave{A}_{12} \\ \hat{A}_{21} & \check{A}_{22} \end{pmatrix} \cdot \begin{pmatrix} \acute{B}_{11} & B_{12} \\ \hat{B}_{21} & \check{B}_{22} \end{pmatrix}$$

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

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

Matrix Multiplication: Key Idea

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

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

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

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

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

Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

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

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