Plan for Today

- Finish recurrences
- Inversion Counting
- Closest Pair of Points

Divide and Conquer

Divide-and-conquer.

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

Most common usage:

- Problem of size n → two equal parts of size n/2
- Combine solutions in linear time.

Recommender Systems

Netflix tries to match your movie preferences with others.

- You rank n movies.
- Netflix consults database to find people with similar tastes.
- Netflix can recommend to you movies that they liked.

Doing this well was worth $1,000,000 to Netflix!!

Counting Inversions

Similarity metric: number of inversions between two rankings.

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

Movies

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

<u>Inversions</u>

3-2, 4-2

What is the brute force algorithm?

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

Divide and Conquer

Count inversions relative to a sorted list

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

Divide into 2 sublists of equal size

1	5	4	8	10	2

6	9	12	11	3	7

Recursively count the inversions

5 blue-blue inversions

8 red-red inversions

Combine: add recursive counts plus blue-red inversions

9 blue-red inversions

Total = 5 + 8 + 9 = 22.

Divide and Conquer

Count inversions relative to a sorted list

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

Cost

Divide into 2 sublists of equal size

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

$O(1)$

Recursively count the inversions

5 blue-blue inversions 8 red-red inversions

$2*T(n/2)$

Combine: add recursive counts plus blue-red inversions

9 blue-red inversions

Total = 5 + 8 + 9 = 22.

???

Finding Inversions

Variation of mergesort

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.

Finding Inversions

Idea: sort each half during the recursive call, then count inversions while merging the two sorted lists (merge-and-count).

Modified merge sort.

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 6

3	7	10	14	18	19

2	11	16	17	23	25

Total:

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 6

3	7	10	14	18	19

2	11	16	17	23	25

2											

Total: 6

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 5

3	7	10	14	18	19

2	11	16	17	23	25

2	3										

Total: 6

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 4

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7									

Total: 6

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 3

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10								

Total: 6

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 3

3	7	10	14	18	19

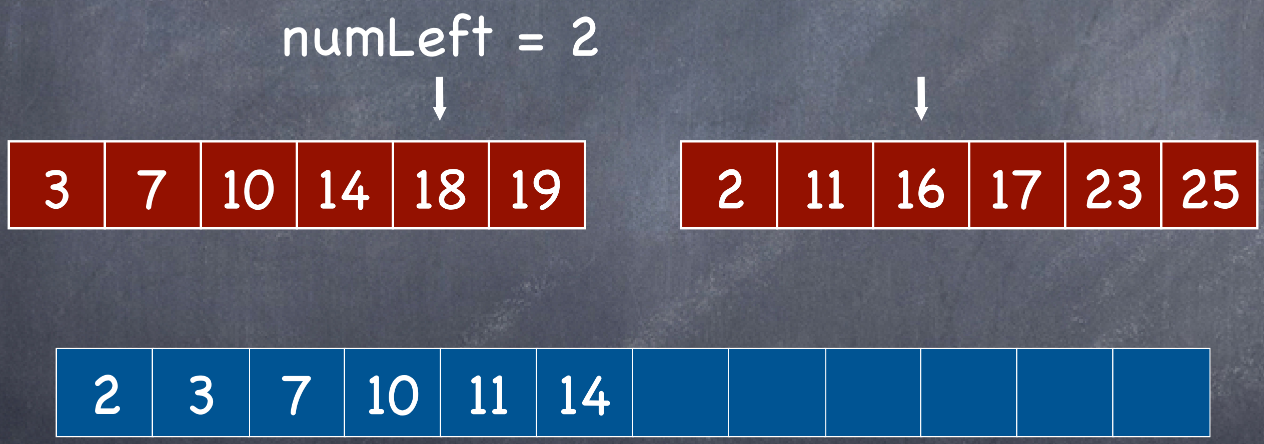
2	11	16	17	23	25

2	3	7	10	11							

Total: 6 + 3

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.



Total: 6 + 3

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 2

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16					

Total: 6 + 3 + 2

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 2

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16	17				

Total: 6 + 3 + 2 + 2

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 1

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16	17	18			

Total: 6 + 3 + 2 + 2

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 0

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16	17	18	19		

Total: 6 + 3 + 2 + 2

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 0

3	7	10	14	18	19

2	11	16	17	23	25

2	3	7	10	11	14	16	17	18	19	23	

Total: 6 + 3 + 2 + 2

Merge and Count

- Merge and count step.
  - Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
  - Combine two sorted halves into sorted whole.

numLeft = 0

3	7	10	14	18	19

2	11	16	17	23	25

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

Total: 6 + 3 + 2 + 2 = 13

Counting Inversions: Implementation

```
Sort-and-Count(L) {
   if list L has one element
      return (0, L)

   Divide the list into two halves A and B
   (r_A, A) ← Sort-and-Count(A)
   (r_B, B) ← Sort-and-Count(B)

   (r_C, L) ← Merge-and-Count(A, B)
   r = r_A + r_B + r_C
   return (r, L)
}
```

Counting Inversions: Implementation

```
Merge-and-Count (A, B) {
        curA = 0;  curB = 0;
        count = 0;
        mergedList = empty list
        while (not at end of A && not at end of B) {
            a = A[curA];  b = B[curB];
            if (a < b) {
                append a to mergedList;
                curA++;
            else {
                append b to mergedList;
                curB++;
                count = count + num elements left in A
            }
        }
        if (at end of A) append rest of B to mergedList;
        else append rest of A to mergedList;
        return (count, mergedList);
}
```

Cost of Sort-and-Count?

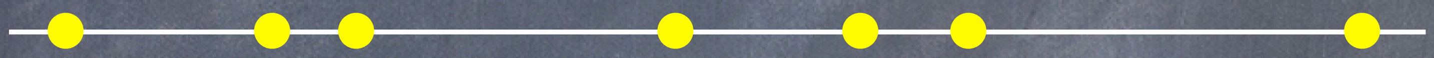
```
Sort-and-Count(L) {
   if list L has one element
      return (0, L)

   Divide the list into two halves A and B
   (r_A, A) ← Sort-and-Count(A)
   (r_B, B) ← Sort-and-Count(B)

   (r_C, L) ← Merge-and-Count(A, B)
   r = r_A + r_B + r_C
   return (r, L)
}
```

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

Closest Pair of Points

1-dimensional version



Closest Pair of Points

- 1-D version.
  - Sort points
  - For each point, find the distance between a point and the point that follows it.
  - Remember the smallest.

Cost

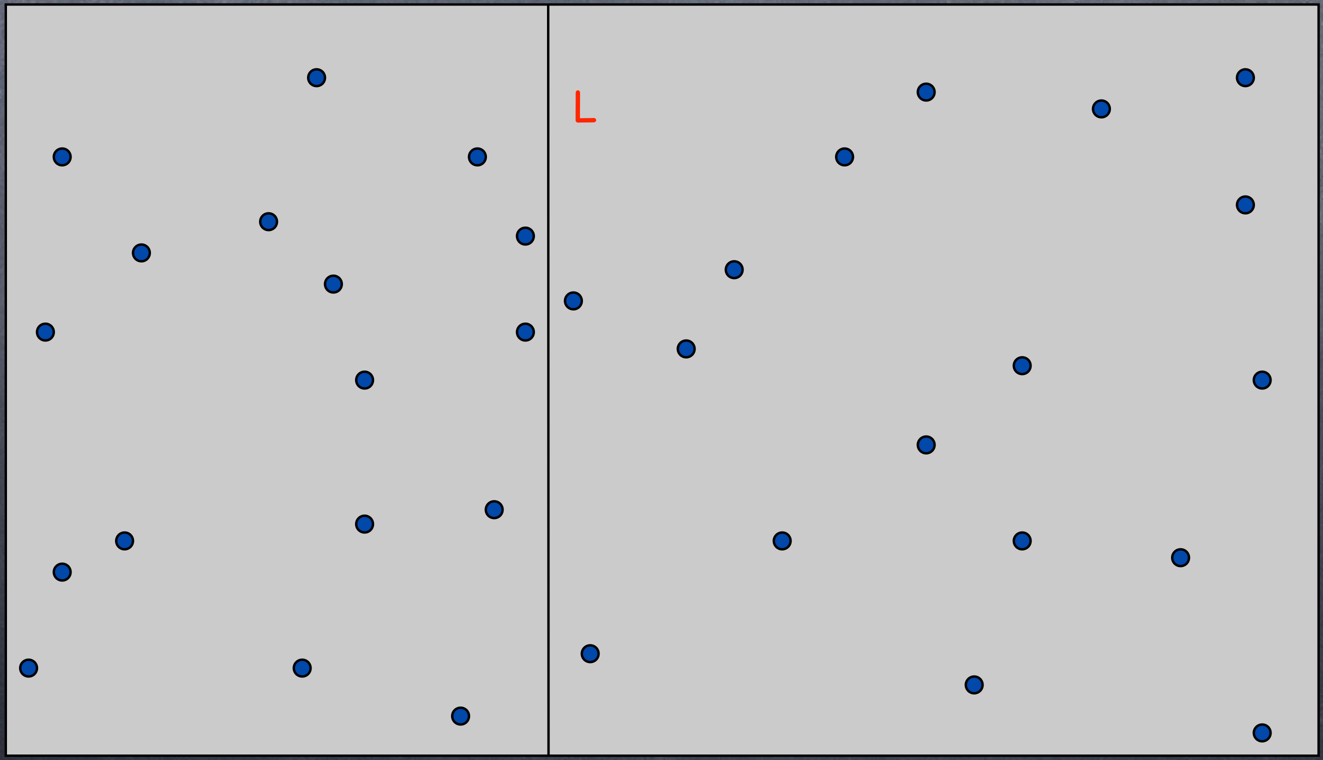
O(n log n)

O(n)

Total is O(n log n)

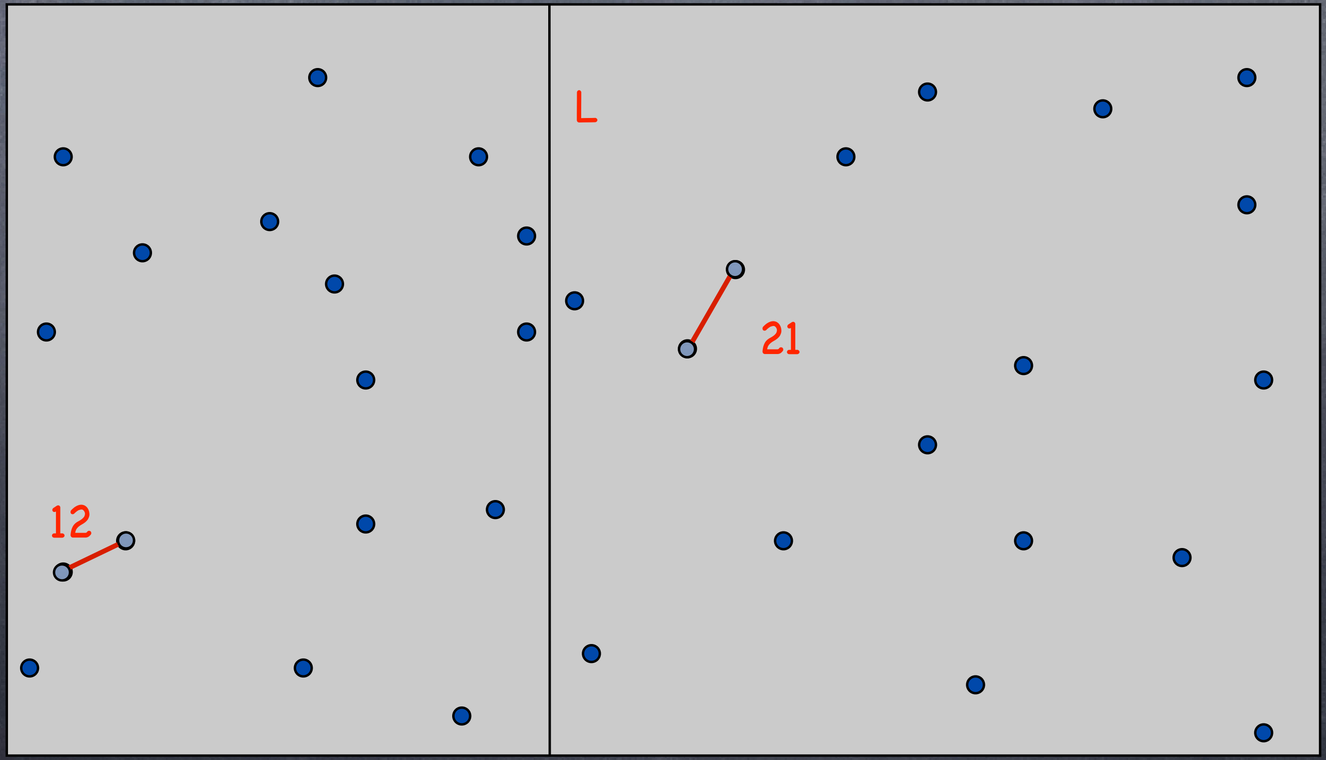
Closest Pair of Points

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



Closest Pair of Points

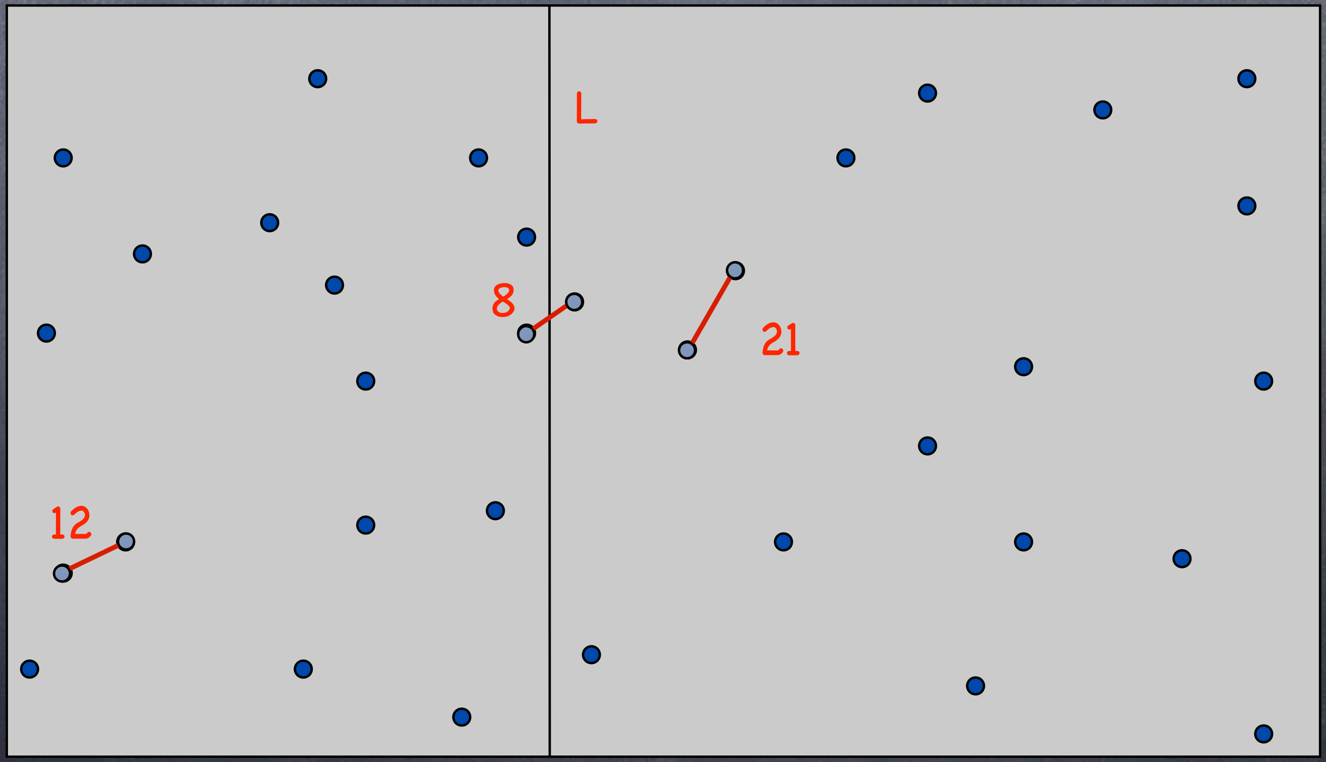
Solve: recursively find closest pair in each side.



Closest Pair of Points

Combine: find closest pair with one point from each side.

Return closest of three pairs.



Running Time?

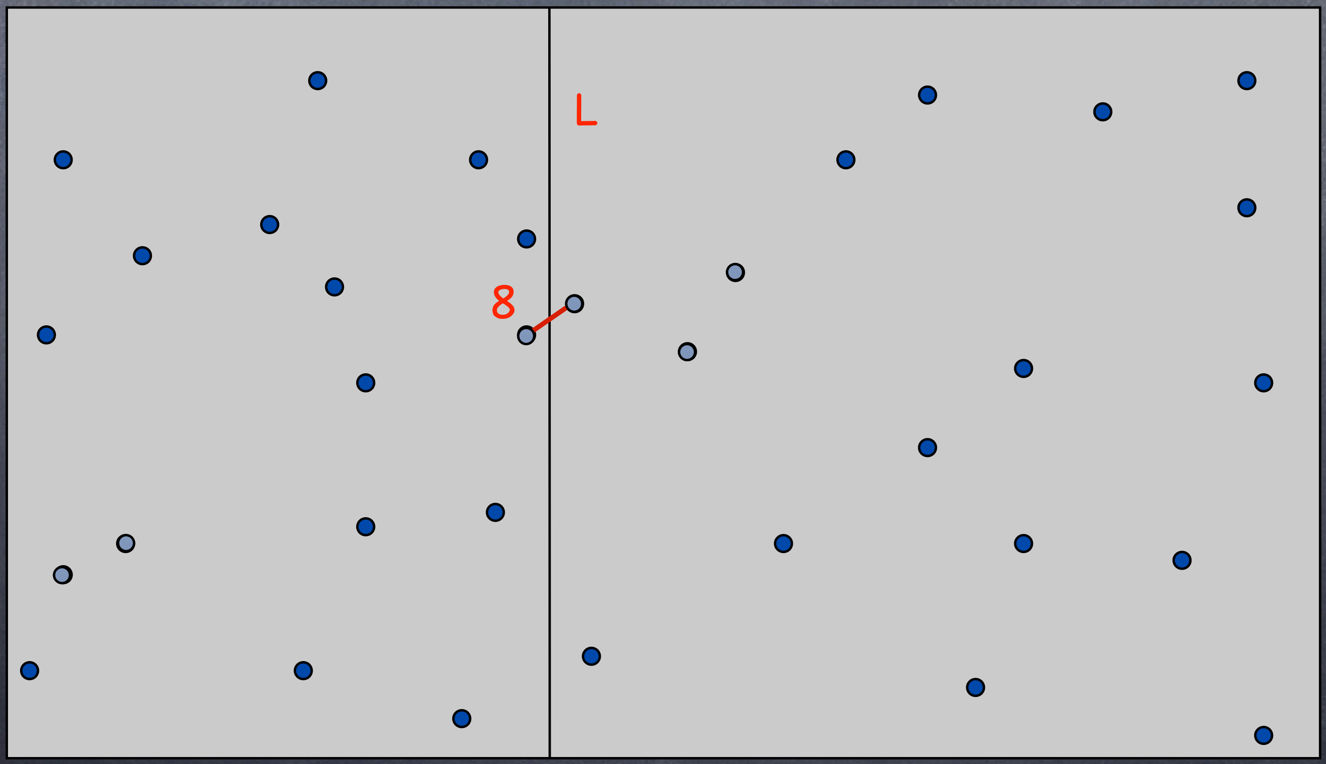
$T(n) \leq 2\ T(n/2) + ???$

Time for combine?

Goal: implement combine in linear time, to get O(n log n) overall

Closest Pair of Points

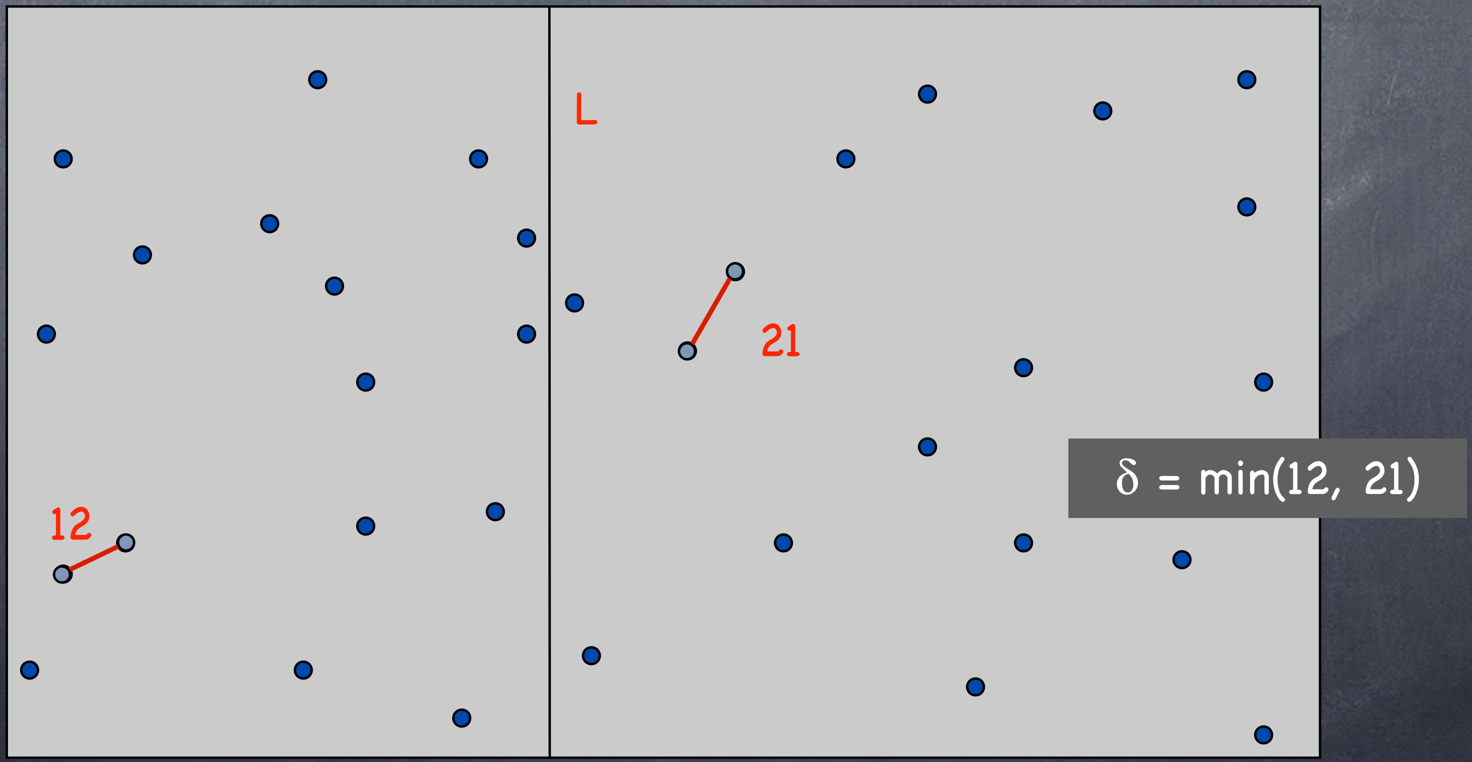
Combine: how to do this without comparing each point on left to each point on right?



Closest Pair of Points

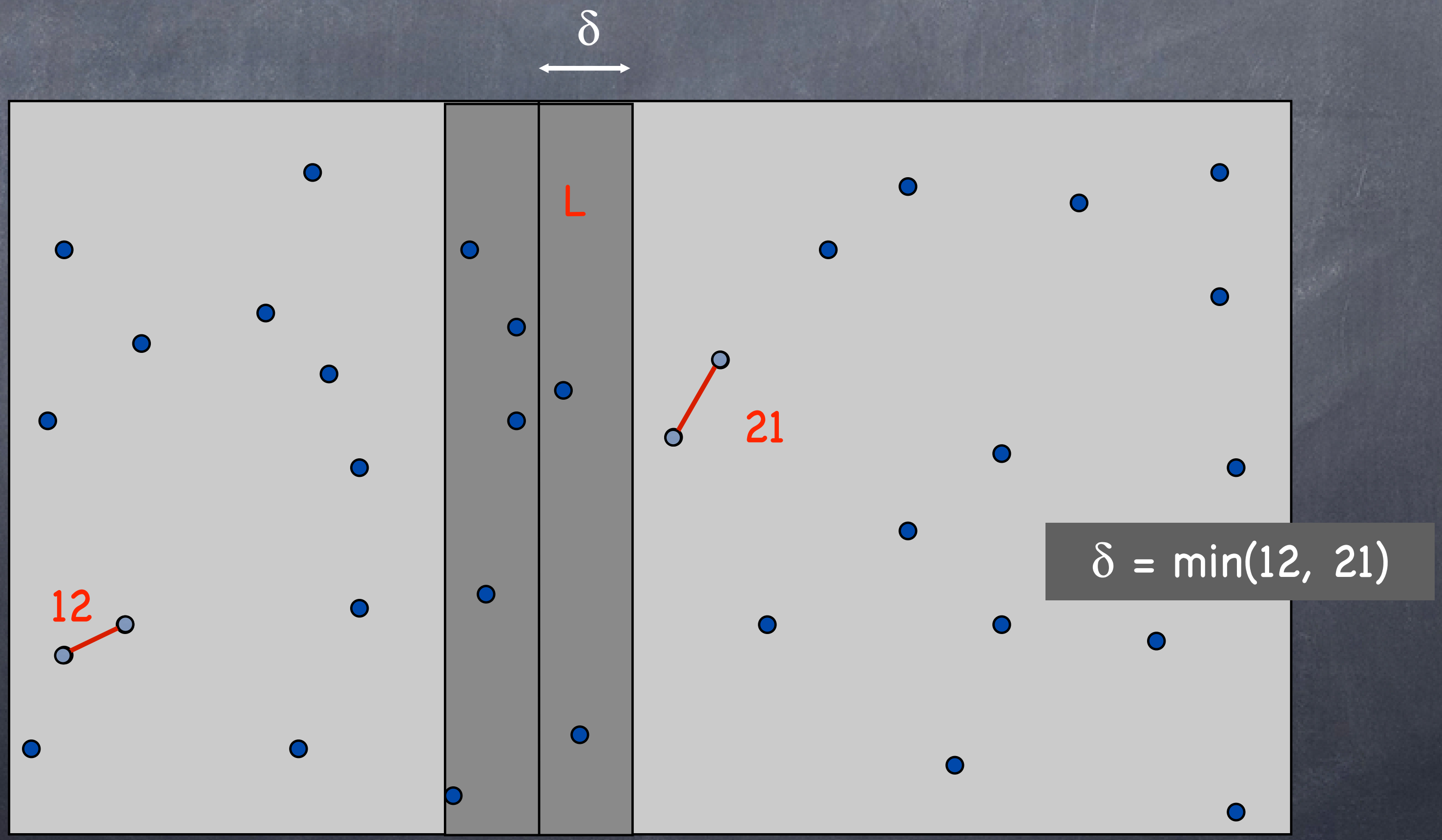
Let δ be the minimum between pair on left and pair on right

If there exists a pair with one point in each side and whose distance < δ, find that pair.



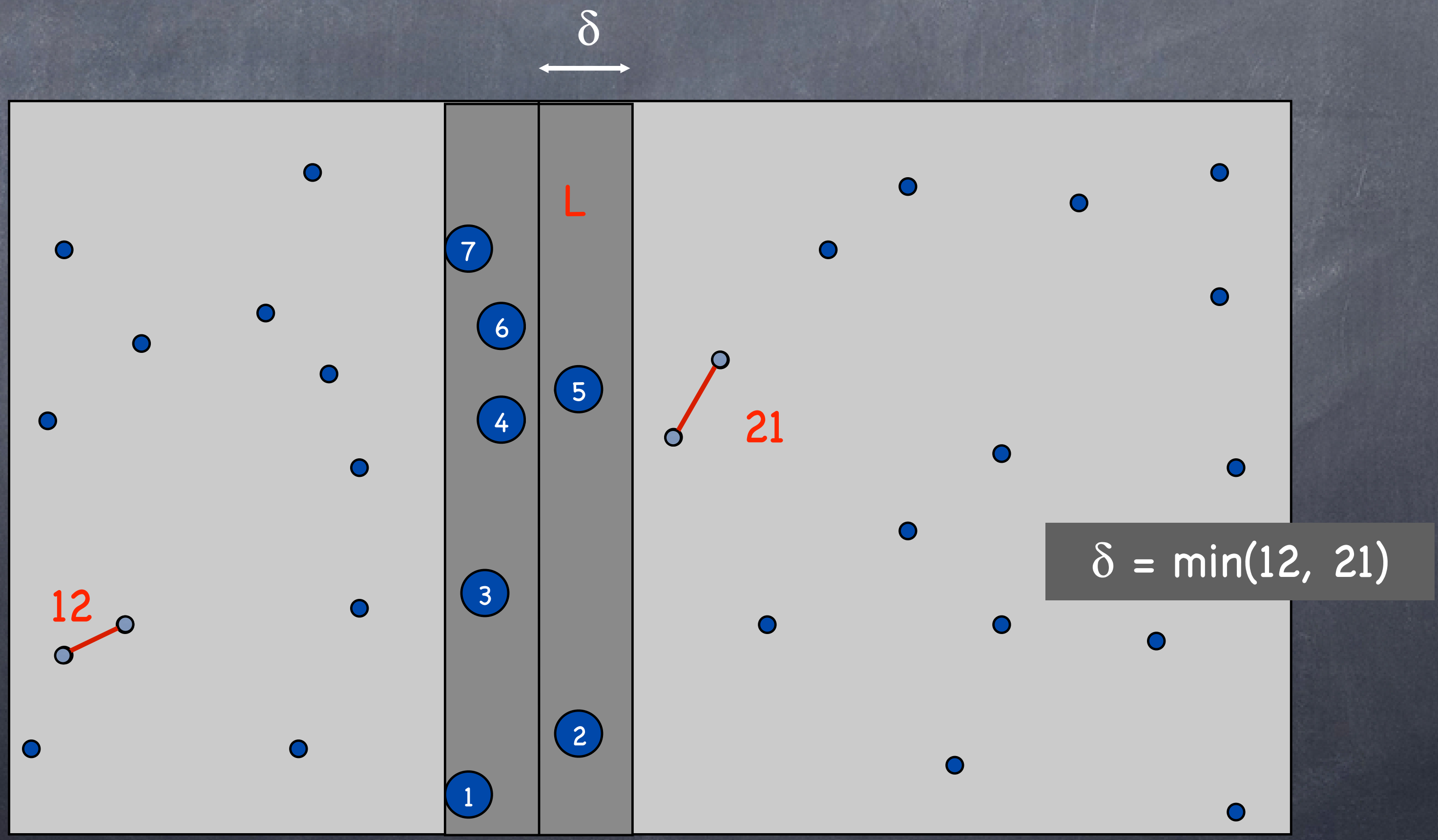
Closest Pair of Points

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



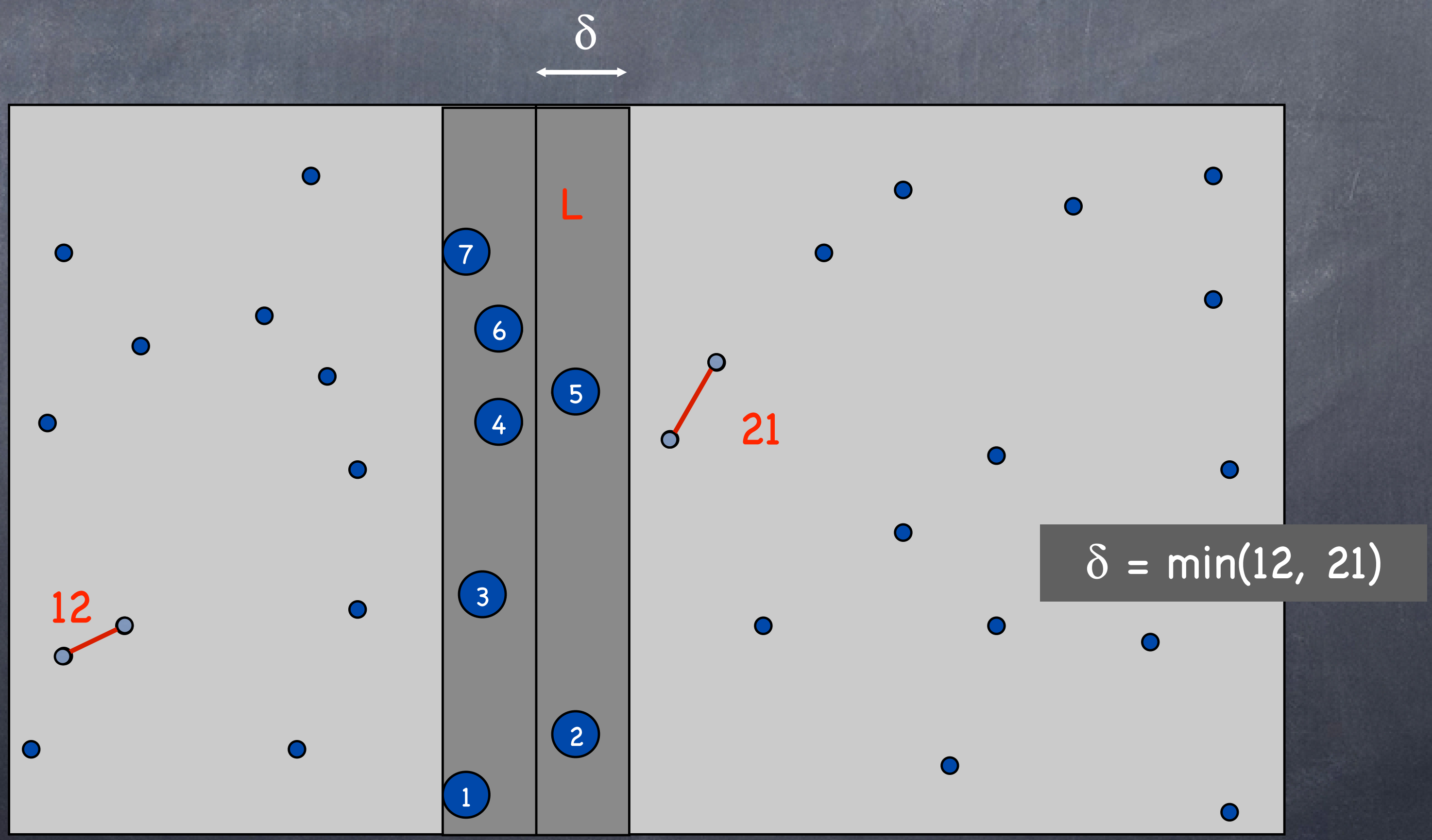
Closest Pair of Points

Sort points in 2δ-strip by their y coordinate.



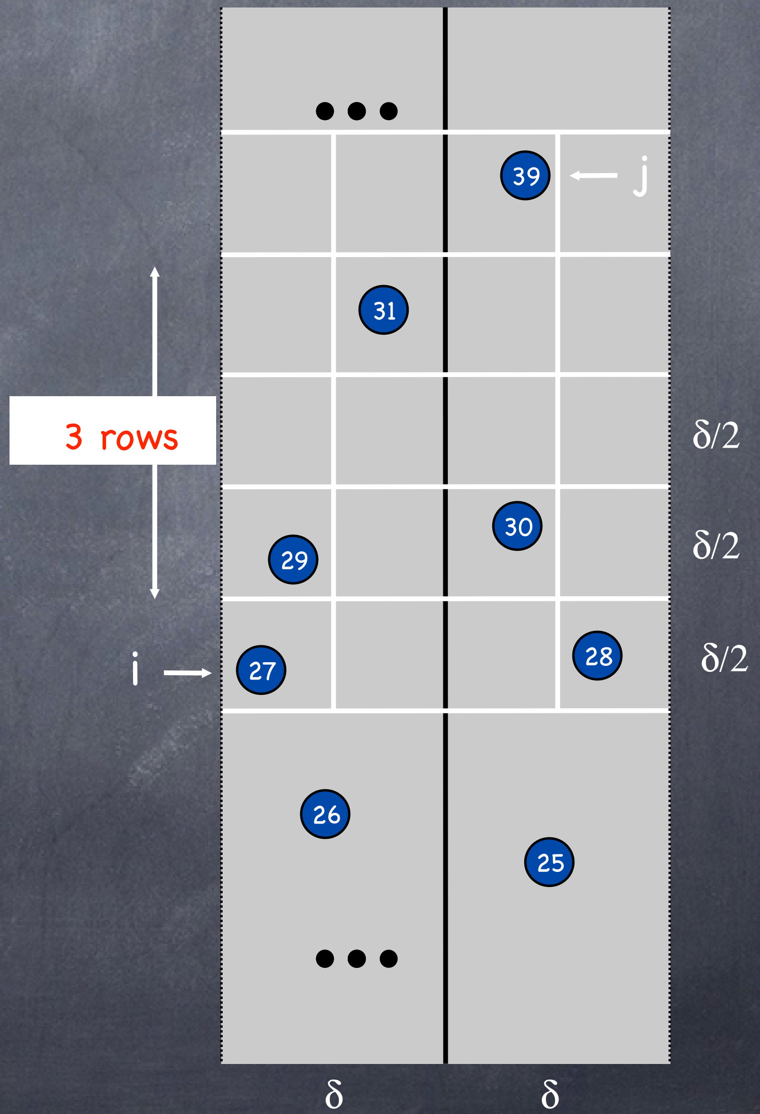
Closest Pair of Points

Unbelievable lemma: only need to check distances of those within 15 positions in sorted



Closest Pair of Points

- Let $s_1, s_2, \ldots, s_k$ be the points in the 2δ-strip sorted by y-coordinate.
- Claim. If $|i - j| > 15$, then the distance between s_i and s_j is at least δ.
- Proof:
  - No two points lie in same $\delta/2$-by-$\delta/2$ box.
  - Two points separated by at least 3 rows have distance $\geq 3\delta/2$.



Closest Pair Algorithm

	Cost
Closest-Pair(p_1, …, p_n) {	
Compute separation line L such that half the points are on one side and half on the other side.	$O(n \log n)$
δ_1 = Closest-Pair(left half) δ_2 = Closest-Pair(right half) $\delta = \min(\delta_1, \delta_2)$	$2T(n / 2)$
Delete all points further than δ 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 δ, update δ.	$O(n)$
return δ.	
}	

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

Closest Pair of Points: Improvement

- Can we achieve O(n log n)?
- Yes: pre-sort all points by x- and y-coordinates, and filter sorted lists to find the points within δ of L.
- See the book for details.