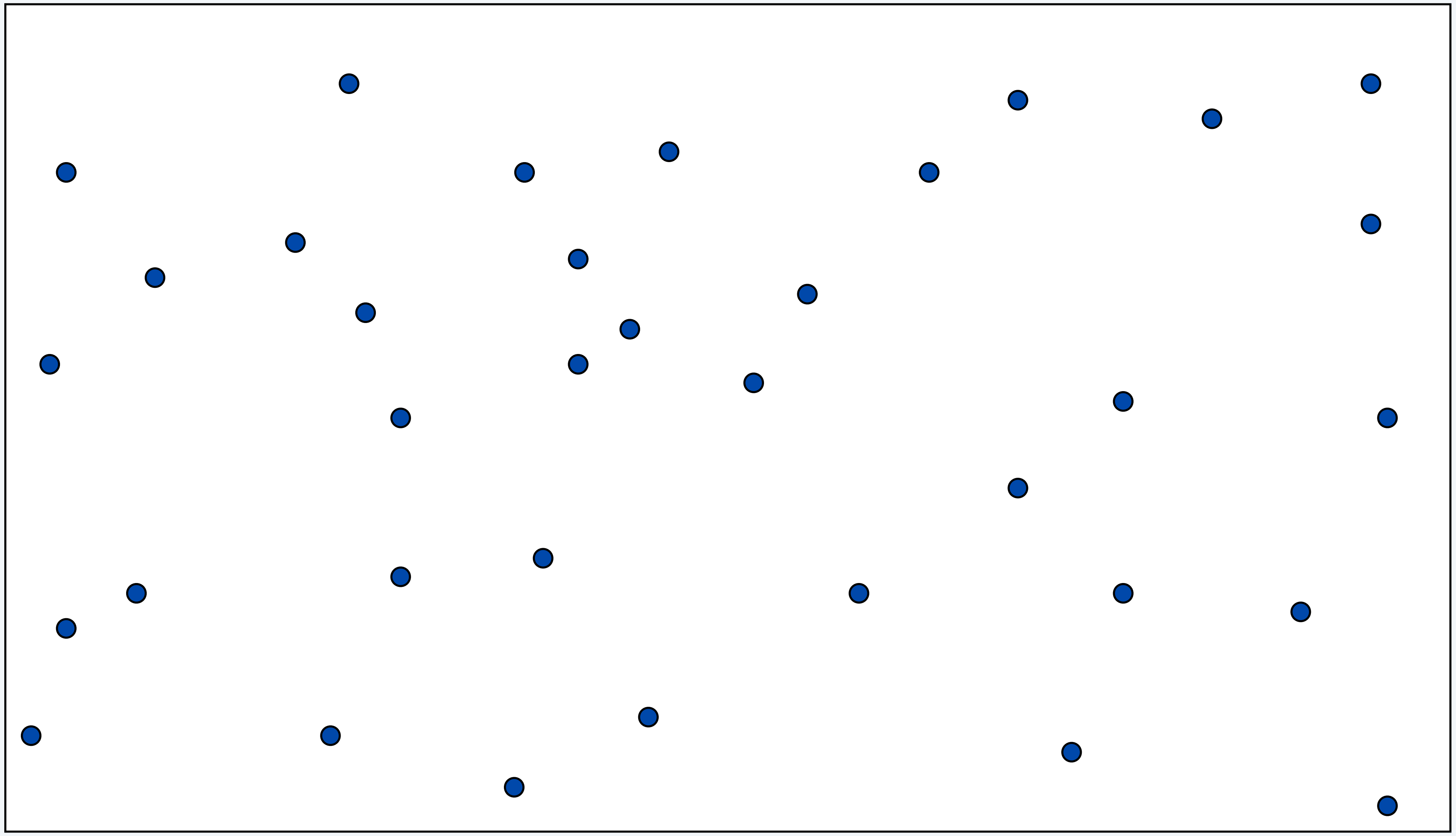
Plan for today

finish challenge problem 1: 2D Closest Points

intro challenge problem 2: Segmented Least Squares

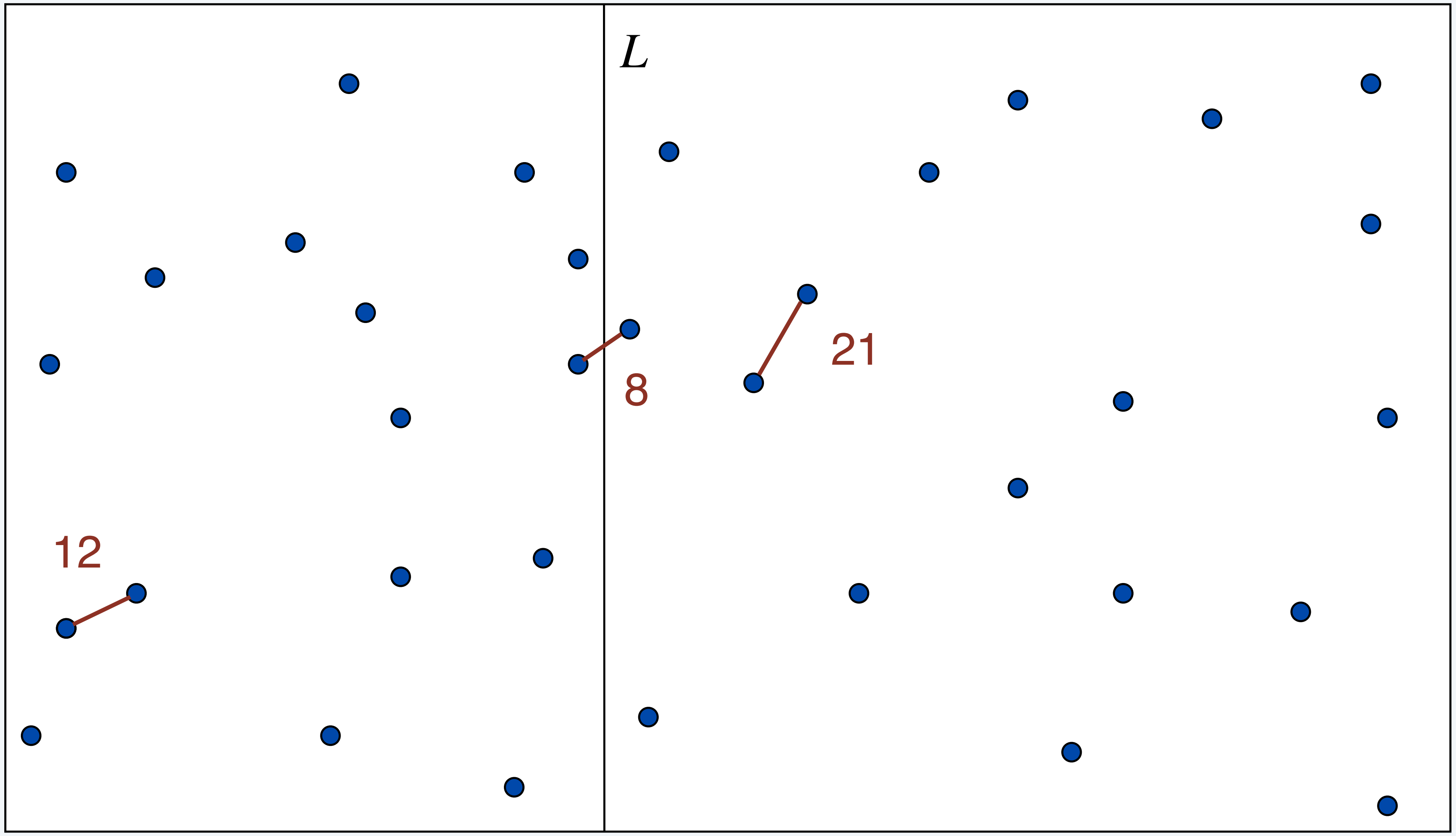
2D closest points problem



Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

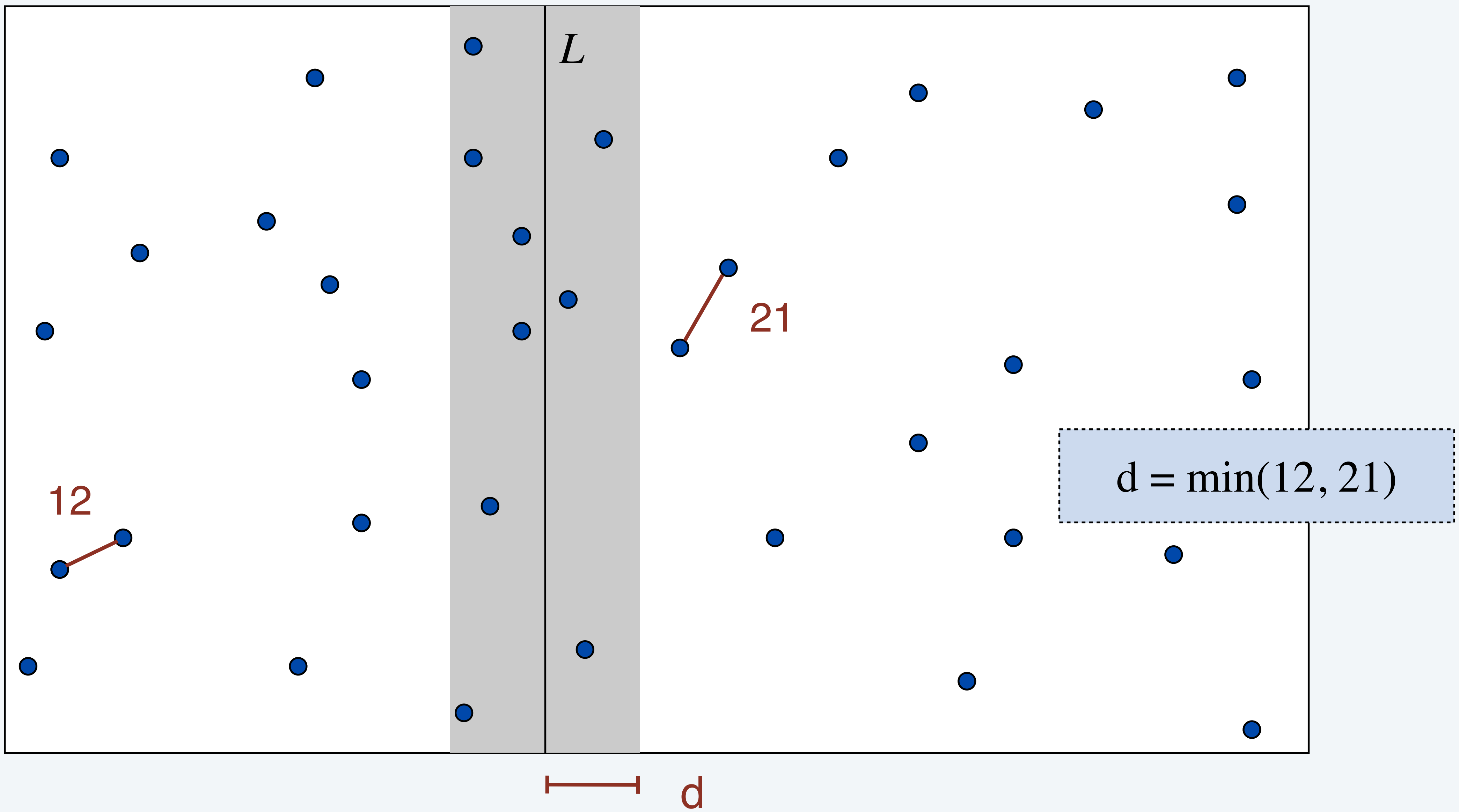
seems like $\Theta(n^2)$



How to find closest pair with one point in each side?

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

- Observation: suffices to consider only those points within d of line L.

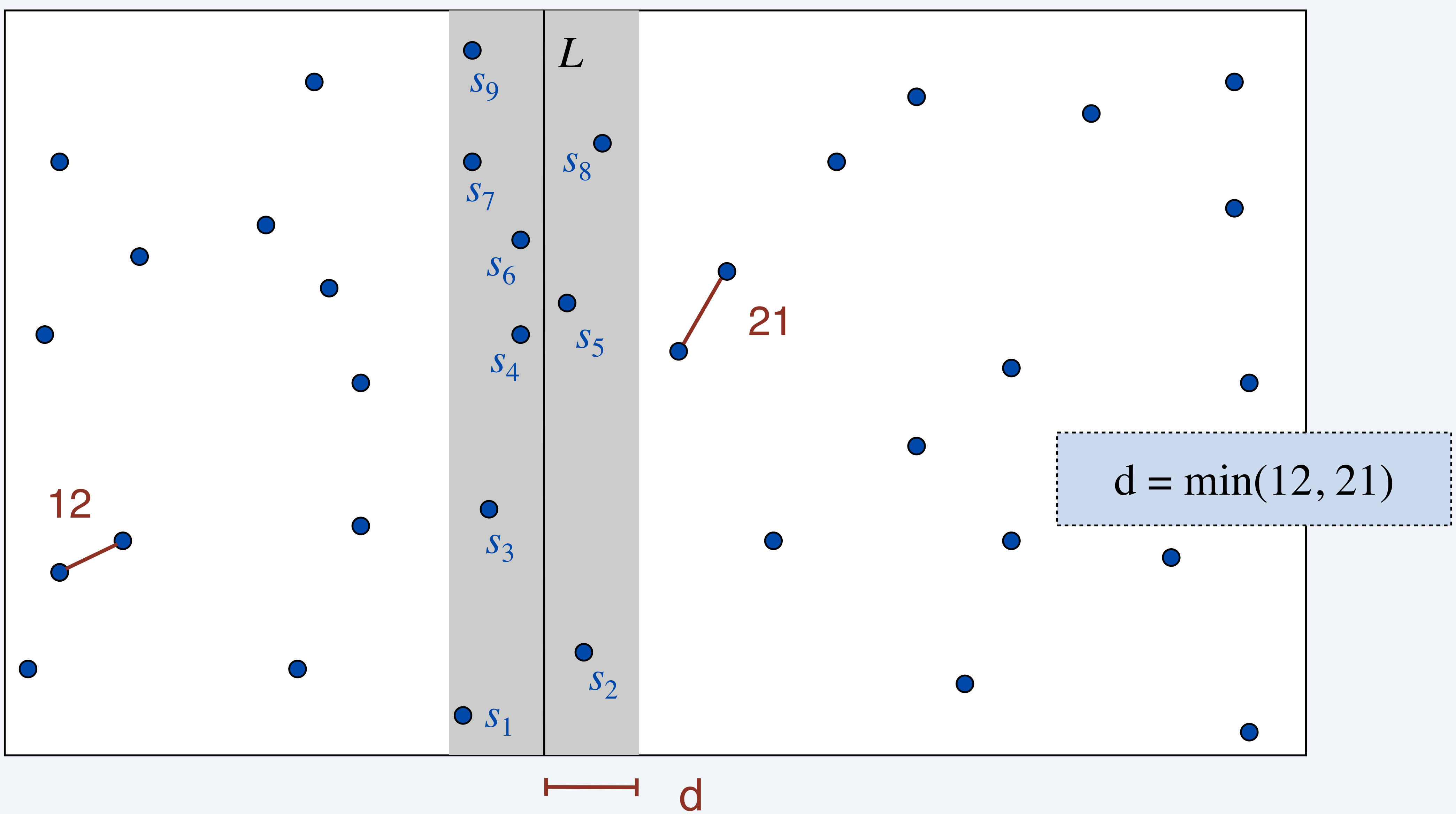


How to find closest pair with one point in each side?

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

- Observation: suffices to consider only those points within d of line L.
- Sort points in $2\,\mathrm{d}$-strip by their y-coordinate.
- Check distances of only those points within 7 positions in sorted list!

why?



How to find closest pair with one point in each side?

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

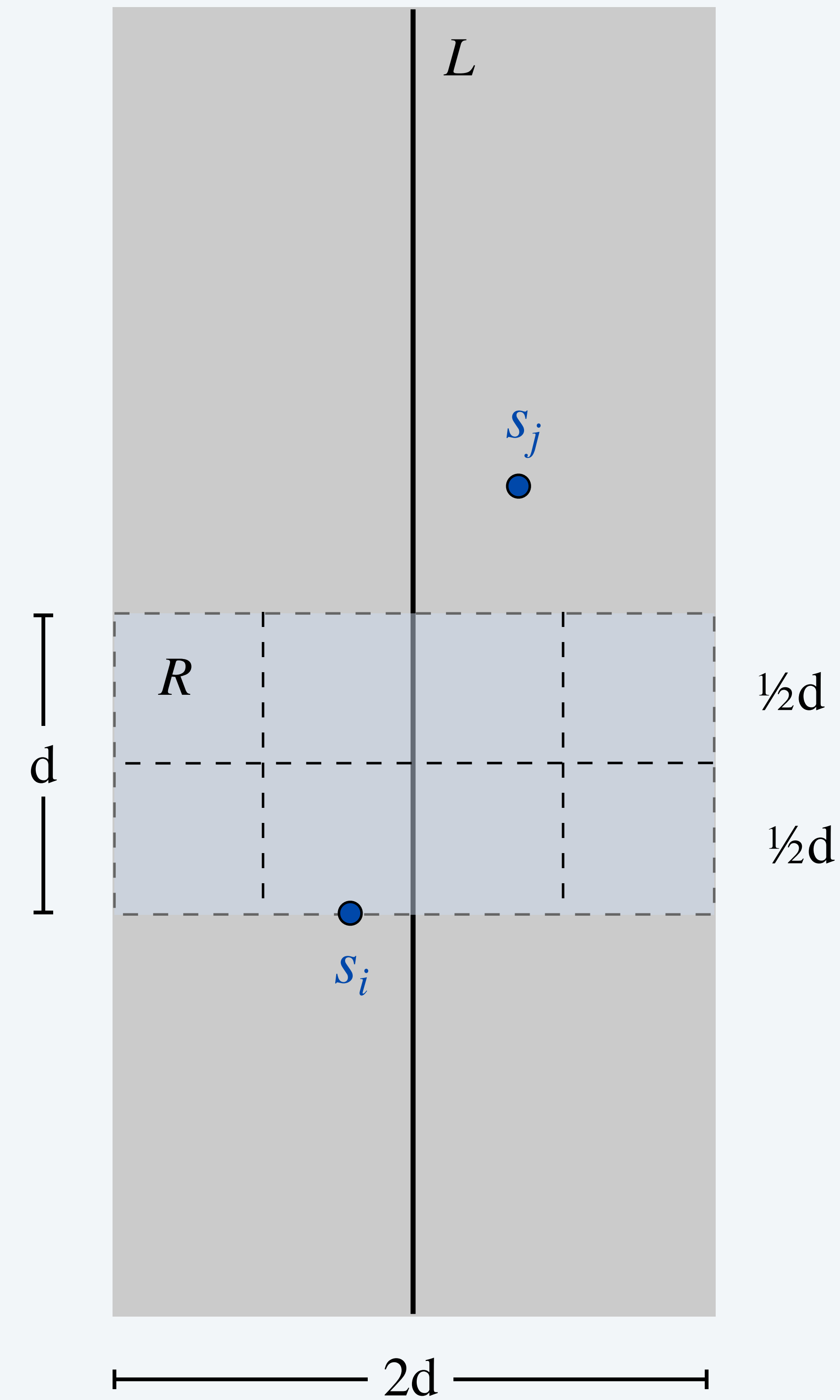
Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least d.

Pf.

- Consider the $2d$-by-d rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
- Distance between s_i and any point s_j above R is $\geq d$.
- Subdivide R into 8 squares.
- At most 1 point per square.
- At most 7 points other than s_i can be in R. ▪

diameter of square is $d/\sqrt{2} < d$

constant can be improved with more refined geometric packing argument



Closest pair of points: divide-and-conquer algorithm

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

Compute vertical line L such that half the points are on each side of the line.	← $O(n)$
$d_1 \leftarrow$ CLOSEST-PAIR(points in left half).	← $T(\lfloor n / 2 \rfloor)$
$d_2 \leftarrow$ CLOSEST-PAIR(points in right half).	← $T(\lceil n / 2 \rceil)$
$d \leftarrow \min \{ d_1, d_2 \}$.	
$A \leftarrow$ list of all points closer than d to line L.	← $O(n)$
Sort points in A by y-coordinate.	← $O(n \log n)$
Scan points in A in y-order and compare distance between each point and next 7 neighbors. If any of these distances is less than d, update d.	← $O(n)$
RETURN d.	

with table: what's the base case?

Think by yourself for a while...

What is the solution to the following recurrence? (make a guess)

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

A. $T(n) = \Theta(n)$.

B. $T(n) = \Theta(n \log n)$.

C. $T(n) = \Theta(n \log^2 n)$.

D. $T(n) = \Theta(n^2)$.

Refined version of closest-pair algorithm

Q. How to improve to $O(n \log n)$?

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

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

Trace through

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

Compute vertical line L such that half the points are on each side of the line.

$d_1 \leftarrow$ CLOSEST-PAIR(points in left half).

$d_2 \leftarrow$ CLOSEST-PAIR(points in right half).

$d \leftarrow \min \{ d_1, d_2 \}$.

$A \leftarrow$ list of all points closer than d to line L.

Sort points in A by y-coordinate.

Scan points in A in y-order and compare distance between each point and next 7 neighbors.
If any of these distances is less than d, update d.

RETURN d.

how many recursive calls do you make?

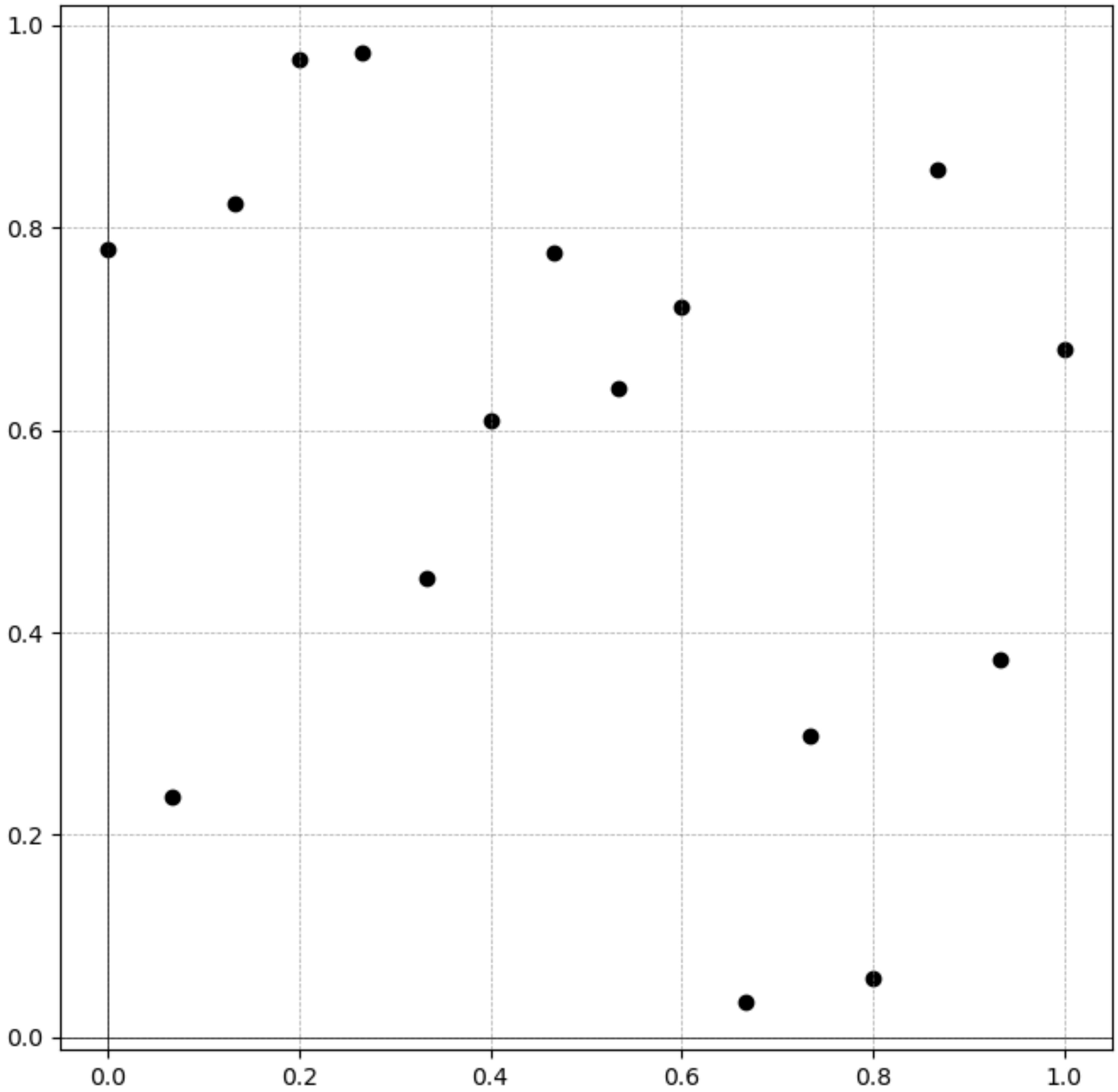
where are the lines L for each call?

which points end up together in base case calls?

which points form d for each call?

which points are in A?

which are the 7 neighbors of each point in A?


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0