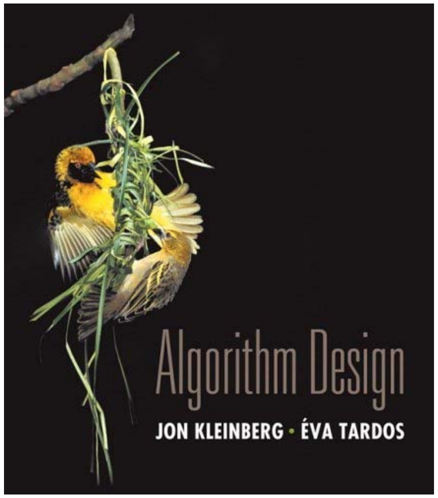
Divide and Conquer: Closest Pair of Points

From the book by Kleinberg and Tardos



Adapted from Kleinberg's notes

Problem: Closest Pair of Points

Problem Definition:

Given n points in the plane, find a pair with smallest Euclidean distance between them.

A fundamental computational geometry problem with many applications

- Graphics
- computer vision
- geographic information systems
- molecular modeling
- air traffic control.
- …

Preliminary thoughts

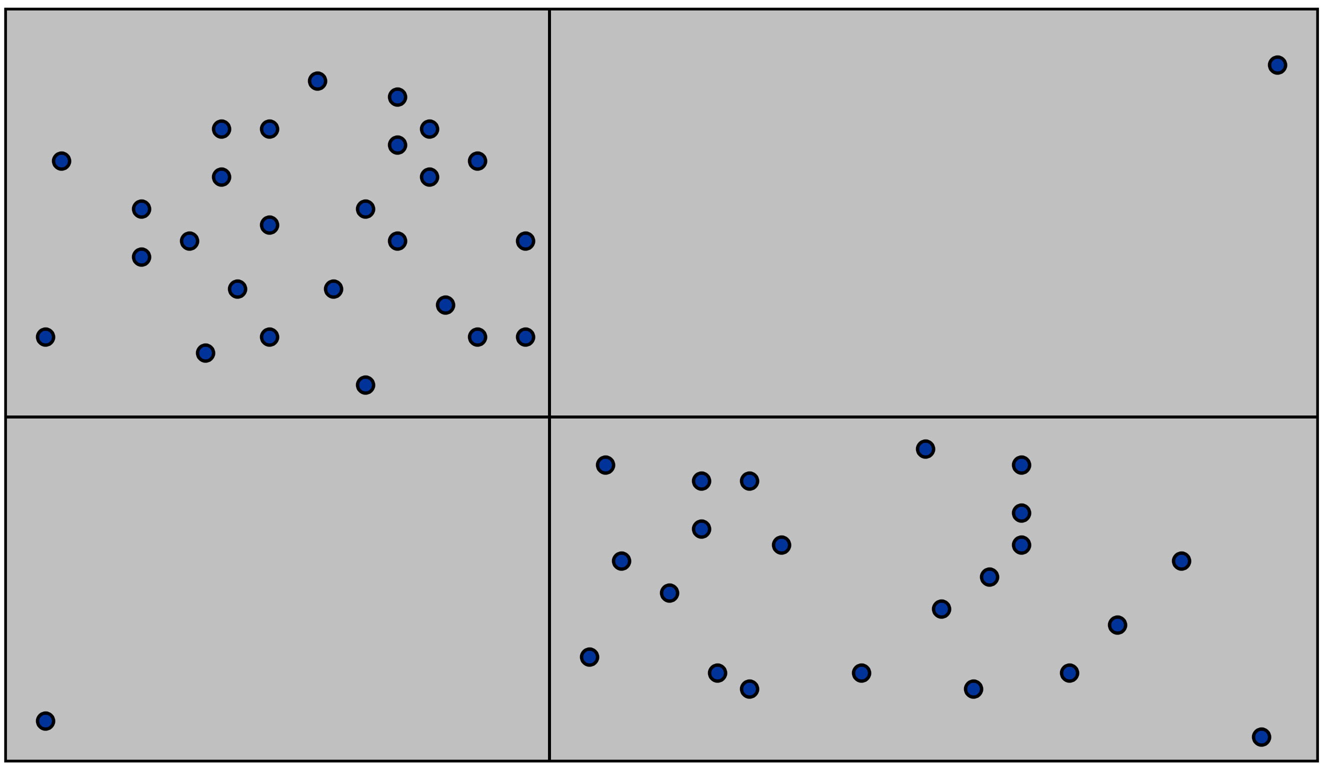
Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

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

Assumption. No two points have same x coordinate.

First Attempt

Divide. Sub-divide region into 4 quadrants.
Obstacle. Difficult to ensure n/4 points in each piece.



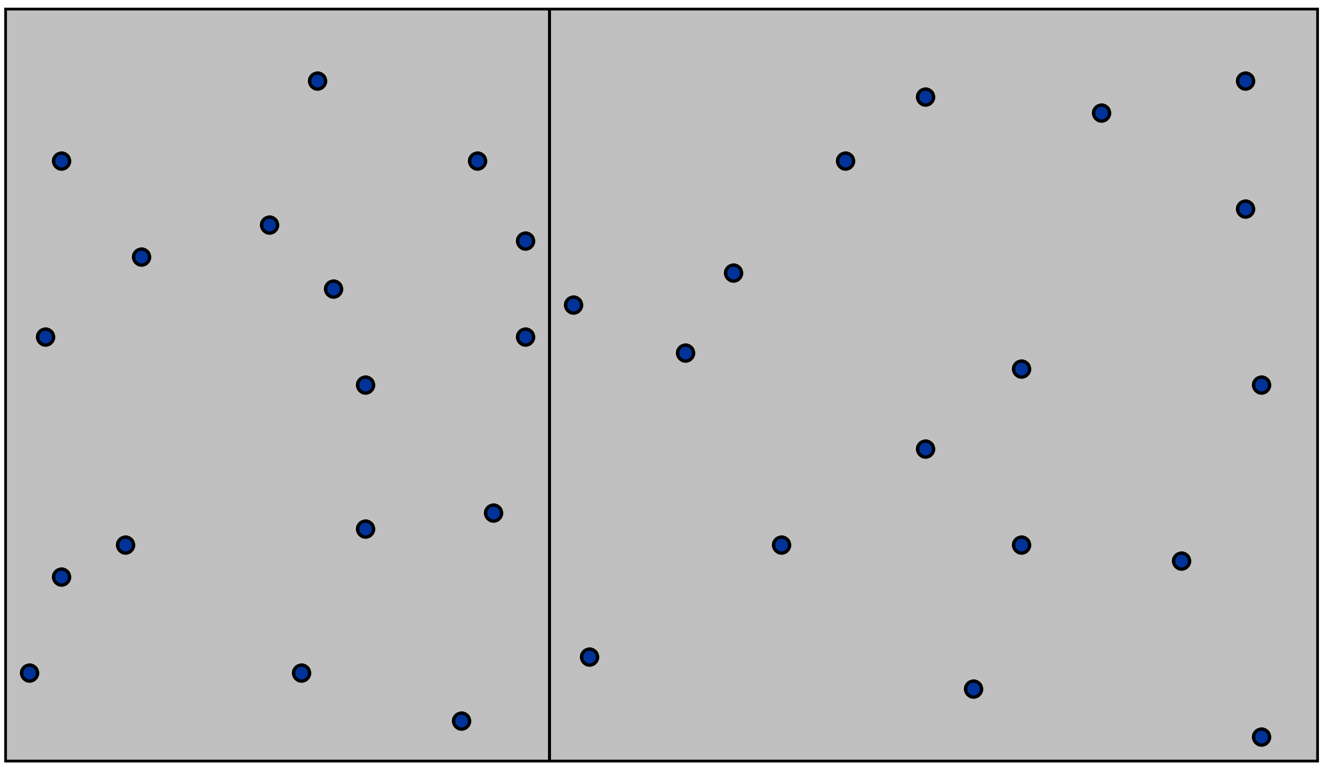
Algorithm: high level idea

Divide: draw vertical line L s.t. ~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.



Finding the closest pair across

Naïve approach for this will consider every cross pair (one point on each side of L) --- $\left(\frac{n}{2}\right)^2$

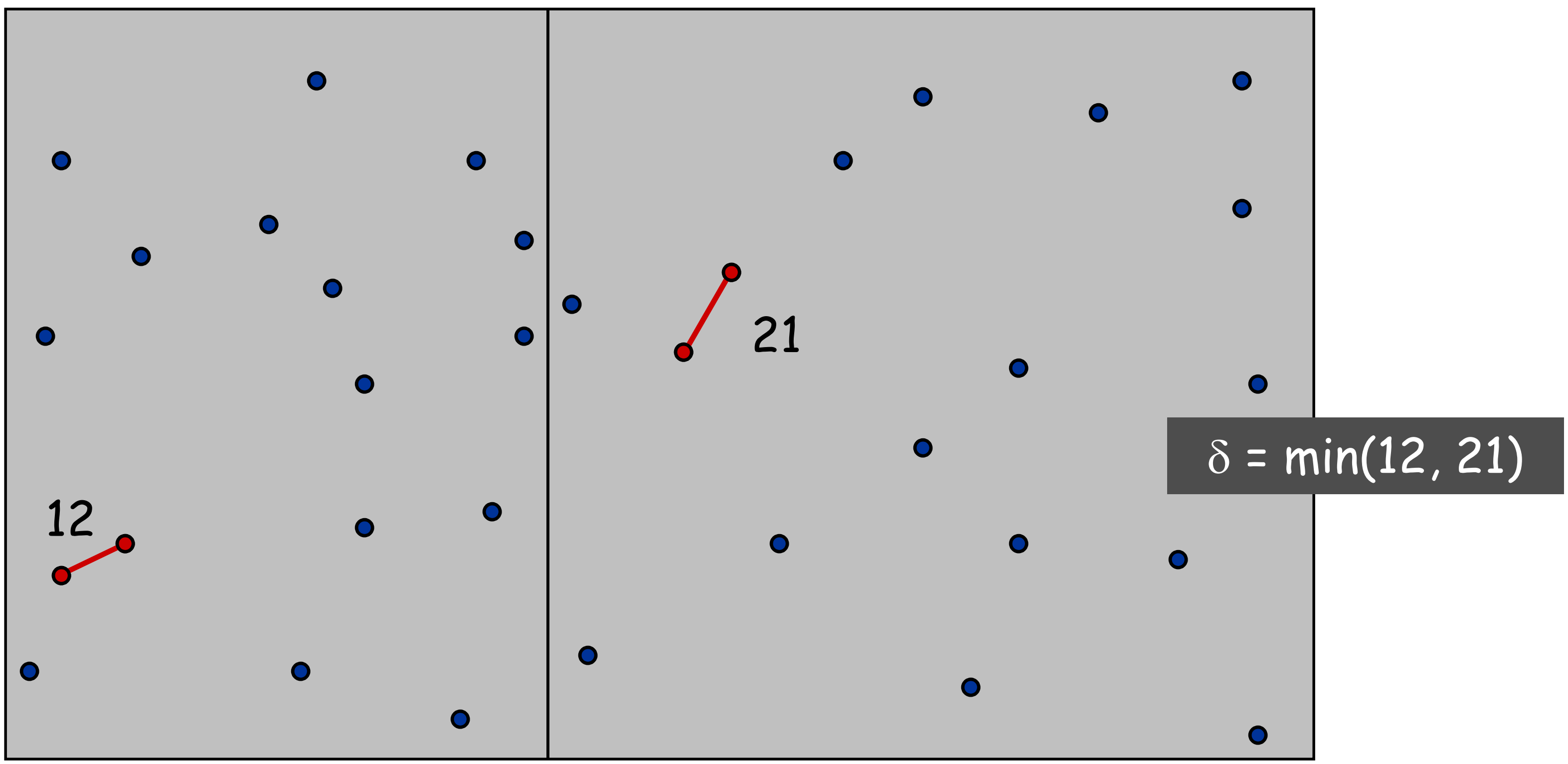
$$T(n) = 2T\left(\frac{n}{2}\right) + cn^2$$

Worse than the brute force approach

Goal: combine in O(n) time

Key idea: no need to consider pairs more than δ apart

Observation: we only need to consider points that are within δ of L



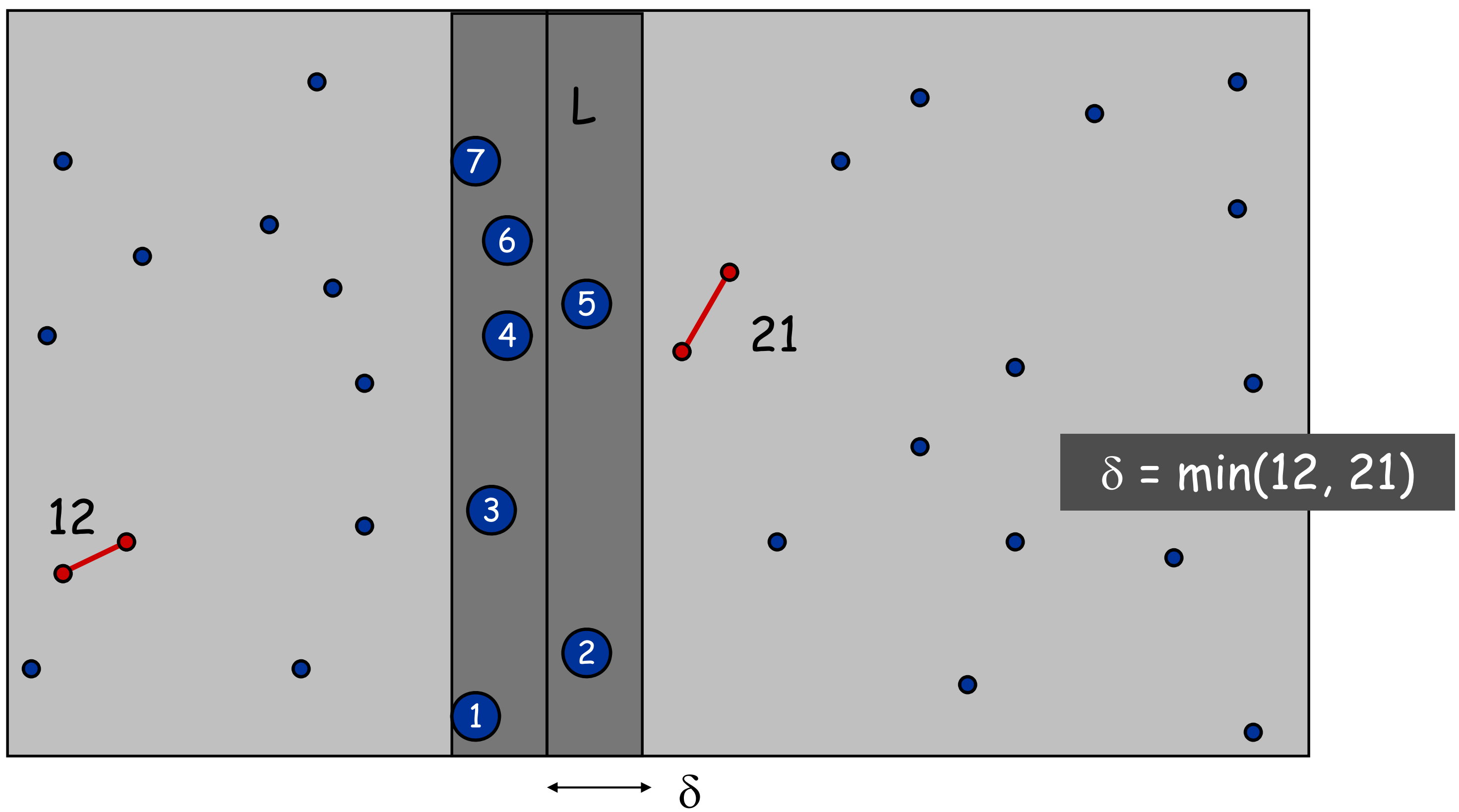
Focusing on the middle strip

In the worst case, O(n) points will be in the middle strip.

Further pruning is needed!

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

Observation: for any point p, we only need to consider points whose y value is within δ of p



Algorithm: Closest-cross-pairs(M_y, δ)

$M_y = \{p_1, p_2, \ldots, p_m\}$: the list of points in the middle strip sorted in increasing order of y

```
d_m = δ
for i = 1 to m − 1
    j = i + 1
    while p_j(y) − p_i(y) ≤ δ and j ≤ m
            d = D(p_i, p_j)
            d_m = min{d, d_m}
            j = j + 1
    end while
end for
Return d_m
```

Correctness of Closest-cross-pairs

Claim: any pair (p,q) whose $D(p,q) \leq \delta$ will be considered and compared

Justification: Assume the pair is ordered by their y value. D(p,q) will be computed and compared when the while loop visits p

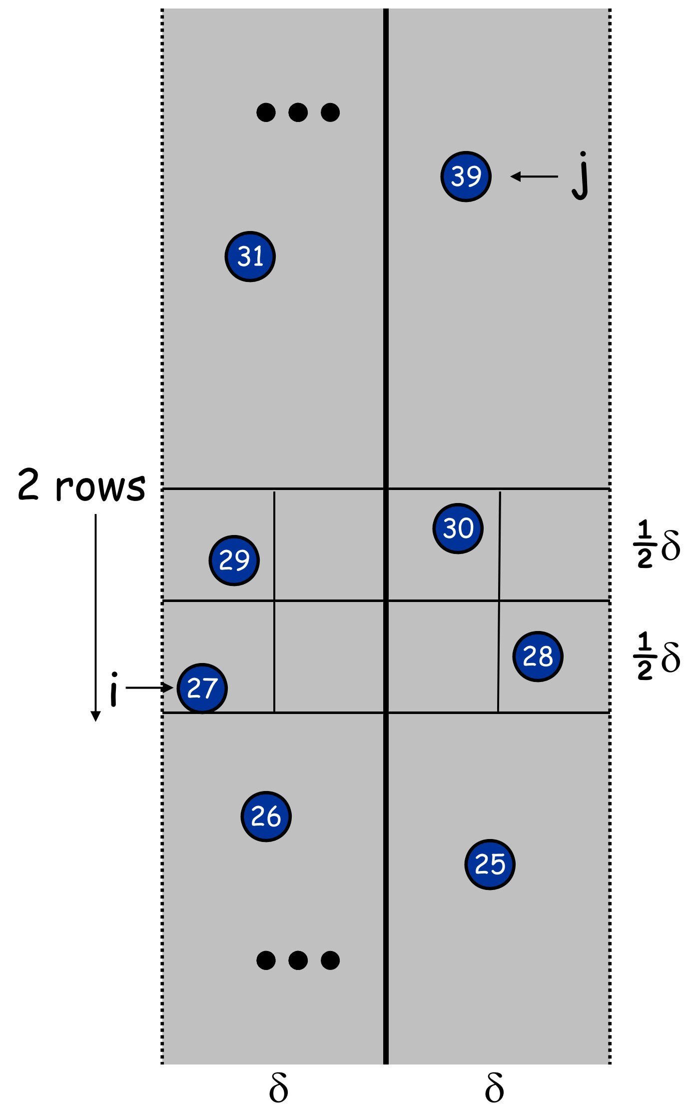
Run time of Closest-cross-pairs?

Q: How many times will the while loop be executed?

Claim. For any point p_i, the while loop (4-8) will execute at most 7 times

Proof:
Consider all points above p_i that have y value within δ of $p_i(y)$
Together with p_i they must lie in side the rectangle of height δ and width 2δ

The rectangle can be divided into 8 cells. There are at most 1 point inside each cell.



Closest Pair Algorithm

Closest-Pair(p_1, …, p_n) {

1. if $n \leq 3$
2. compute and return the min distance
3. else
4. Compute separation line L $O(n \log n)$
5. δ_1 = Closest-Pair(left half) $2T(n / 2)$
6. δ_2 = Closest-Pair(right half)
7. δ = min(δ_1, δ_2)
8. Identify all points within δ from L $O(n)$
9. Sort them by y-coordinate into M_y $O(n \log n)$
10. d_m = closest-cross-pair (M_y, δ) $O(n)$
11. return d_m.

}

Closest Pair of Points: Analysis

Run time

$$T(n) = 2T\left(\frac{n}{2}\right) + cn\log n$$
$$T(n) = O(n\log^2 n)$$

Q. Can we achieve O(n log n)?

A. Yes. Don't sort points in strip from scratch

- Pre-sort all points based on x and y coordinates
- For Line 8-9 just scan the master list pre-sorted based on y to create M_y by excluding points δ away from L in x-coordinate

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \Rightarrow T(n) = O(n\log n)$$