© Gara Pruesse

$O(n \lg n)$ Algorithm for Closest Pair

(Computational Geometry).

Given: n points on a plane

Find: the pair of points that are closest to one another

i.e. minimize

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where $p_1 = (x_1, y_1)$

$p_2 = (x_2, y_2)$

are some pair of distinct points in the given set.

$\{7.6, 3.2, 11.9, 2.0, 6.6\}$

$\Downarrow$

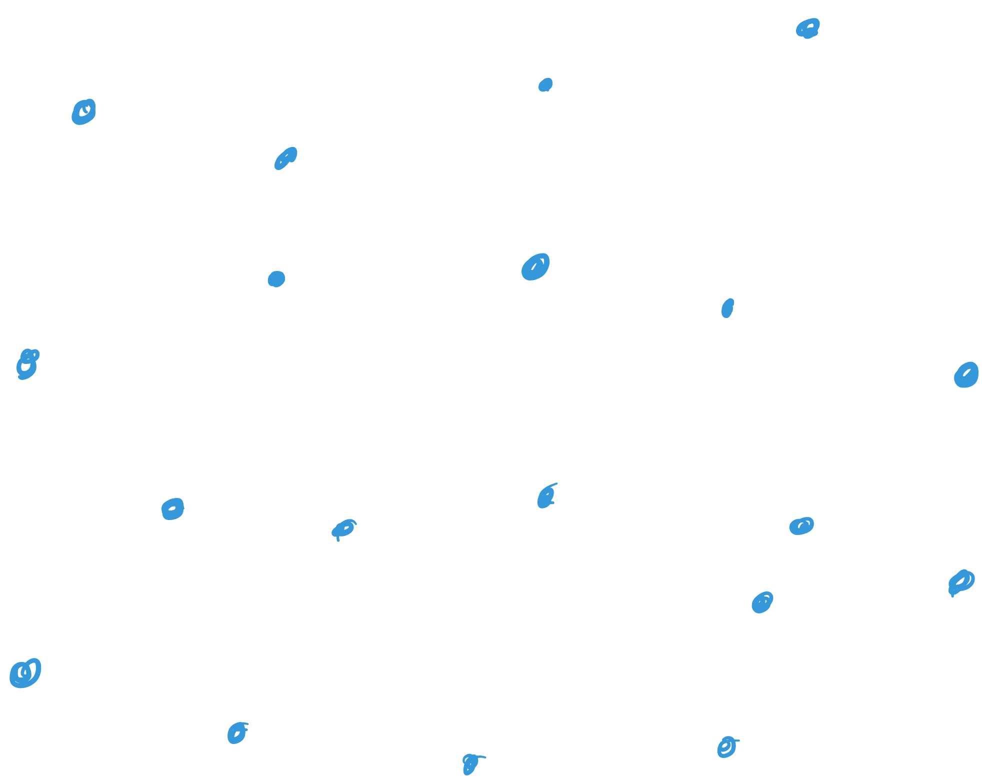
2.0

What if 1-dimensional?

1D_Closest_Pair algorithm sketch

- Sort the points
- for each point, calculate distance from previous point in sorted order
    - keep track of minimum value calculated and the point that yielded it.
- return the overall minimum.

2D Closest Pair ?



<u>Want</u> to find closest pair in time $O(n \lg n)$

<u>Can</u> sort the points in the x-dimension in $O(n \log n)$

" " " y-dimension in $O(n \log n)$

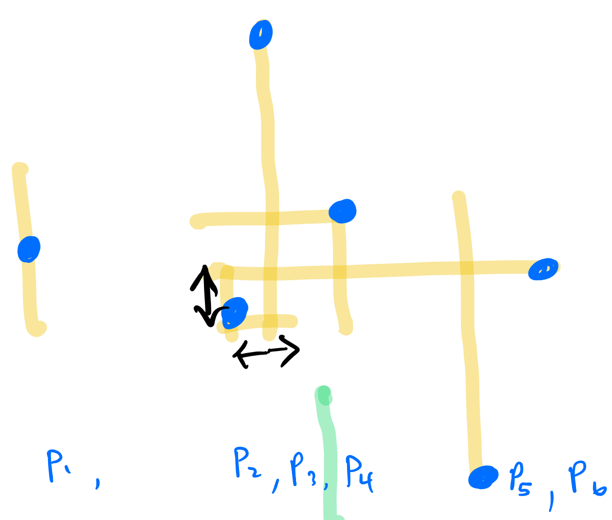
We can do each of these steps as part of our preprocessing "for free" if it helps us.

Why do we consider it "free" to do this work?

- if we have $O(n \log n)$ as our <u>goal</u> then a constant number of calls to $O(n \log n)$-time subroutines does not increase the <u>asymptotic</u> running time

But....

the closest pair may <u>not</u> be closest in x-direction nor closest in y-direction



Brute Force approach:

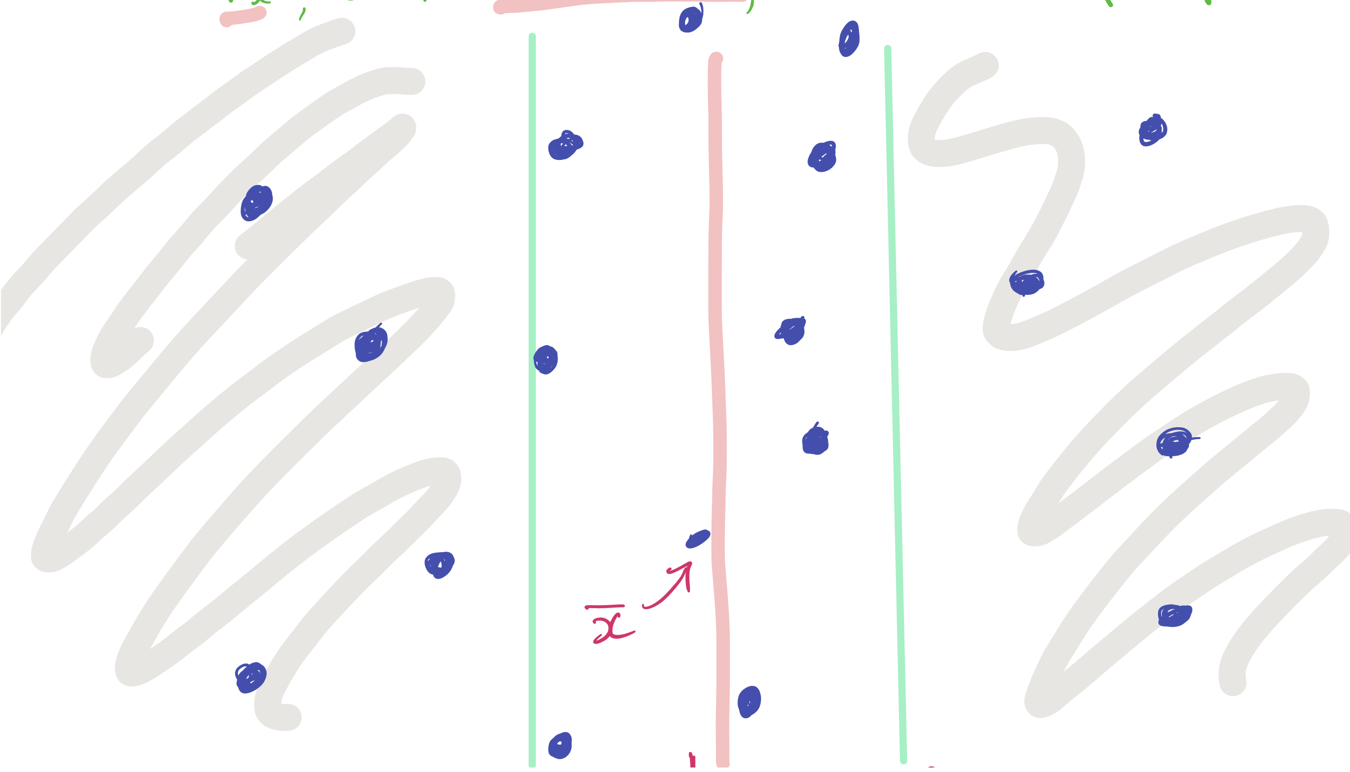
- n^2 pairs of points: compute the distance in each case and note the minimum.

Divide & Conquer Approach

P_x = list of points, sorted by x-coord

P_y = list of points, sorted by y-coord

The closest pair is either in the first half of P_x, or the second half, or is a "split pair"



$\underbrace{\hspace{6em}}_{\text{recurse - find distance } \delta_1}$ $\bar{x} - \delta$ $\bar{x} + \delta$ $\underbrace{\hspace{6em}}_{\text{recurse - find distance } \delta_2}$

recurse - find distance δ_1

recurse - find distance δ_2

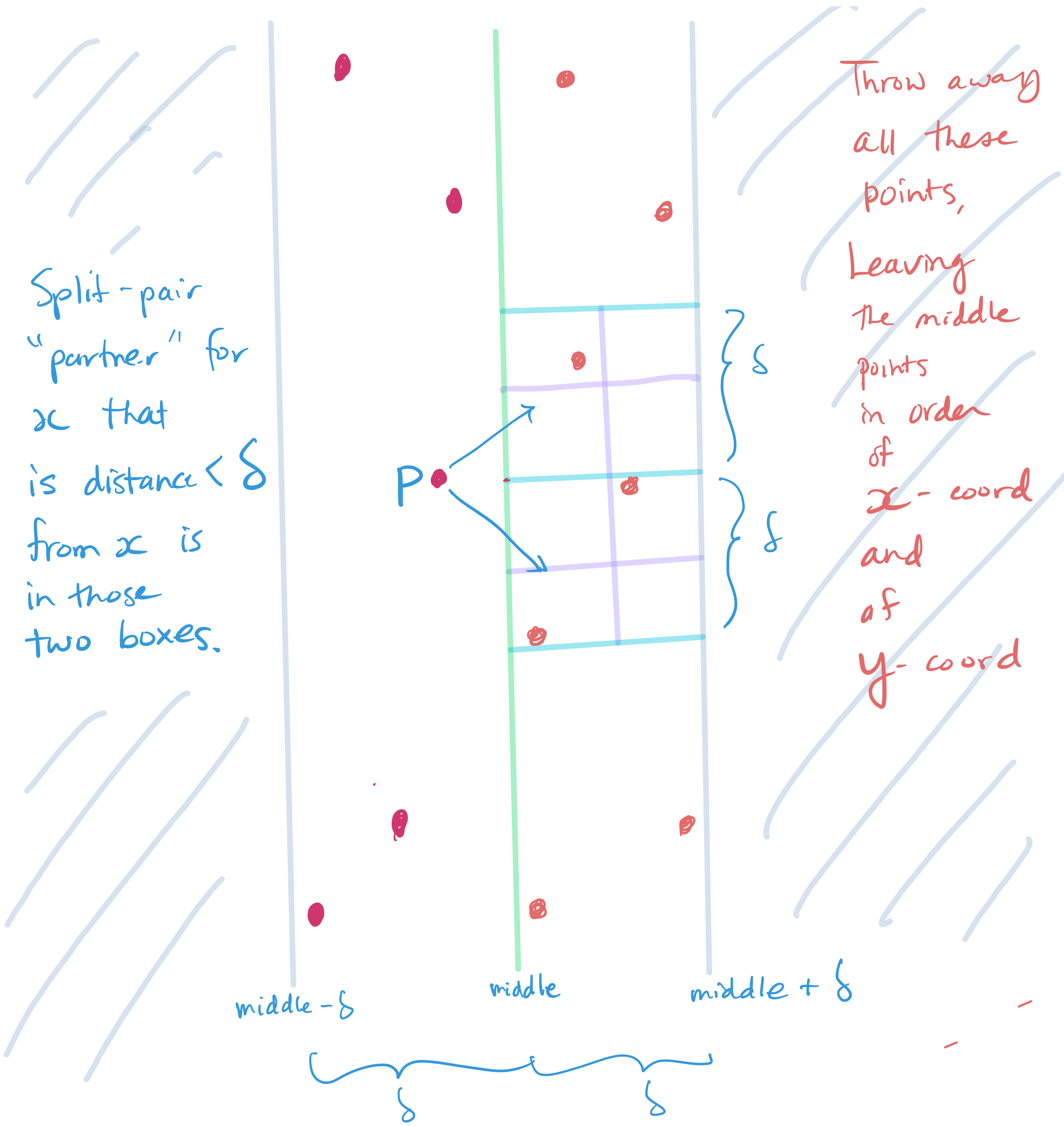
$\delta = \min(\delta_1, \delta_2)$.

Now find min dist <u>split</u> pair in linear time

- must be within δ of $\bar{x}$ in x-dimension

- for each pt p in that range, check the 8 points that come after it in the y-dimension (ie in P_y).

Why is it sufficient to only look at 8? (in the y-direction)?

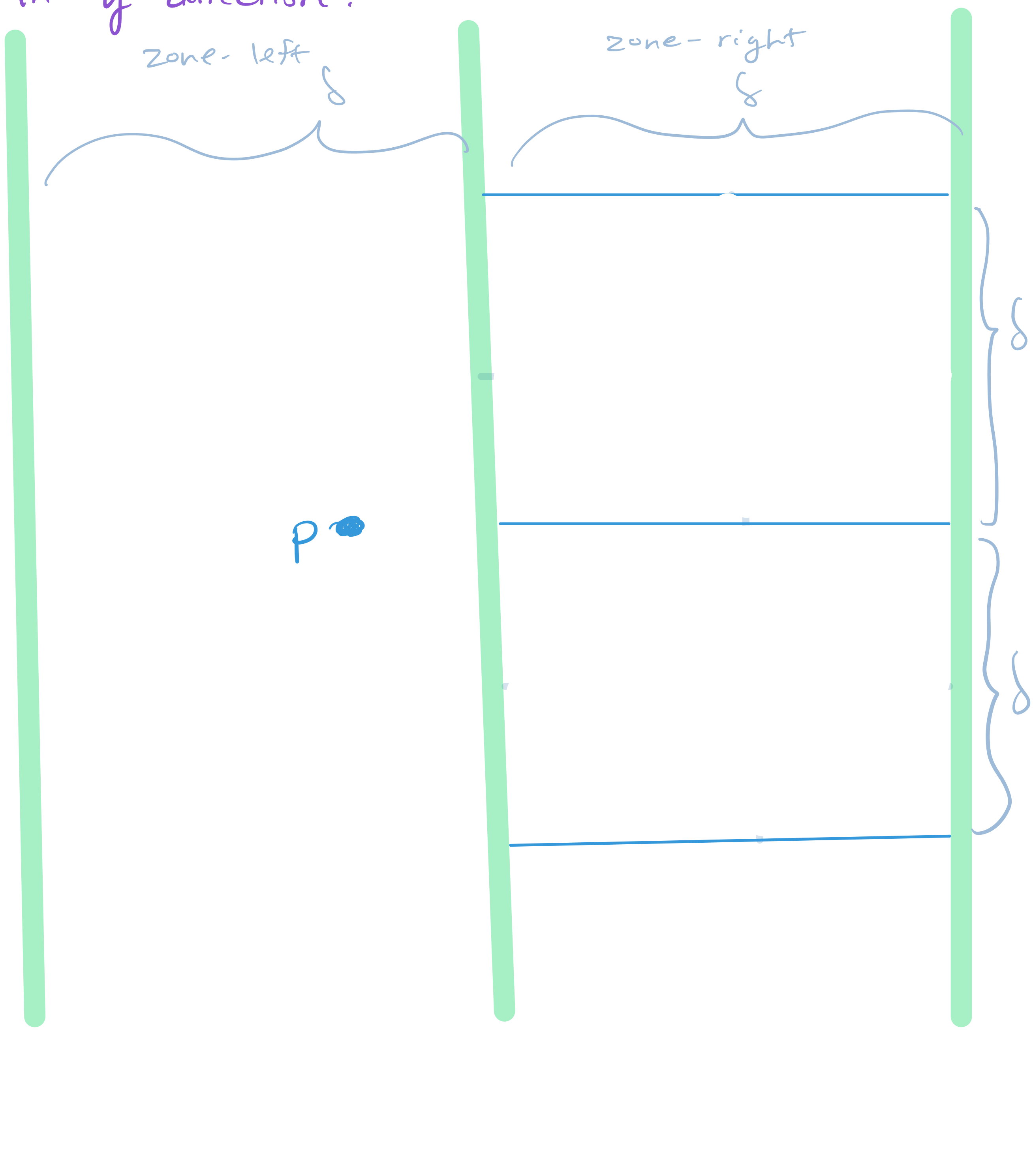


S_y = List of points, sorted by y-co-ord

Filter S_y of any points not in the

$\bar{x}-\delta \ldots \bar{x}+\delta$ zone $\qquad O(n)$

For a given point p in (say) the left part of the zone, how many points can be in Right side of zone, and within δ in y-direction?



You need only check the 8 points around p in Zone-filtered S_y

This can be done in linear time!

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

$n \in \Theta(n^1)$ $\therefore$ by MT case 2,

$$T(n) \in O(n \lg n) \quad \blacksquare$$