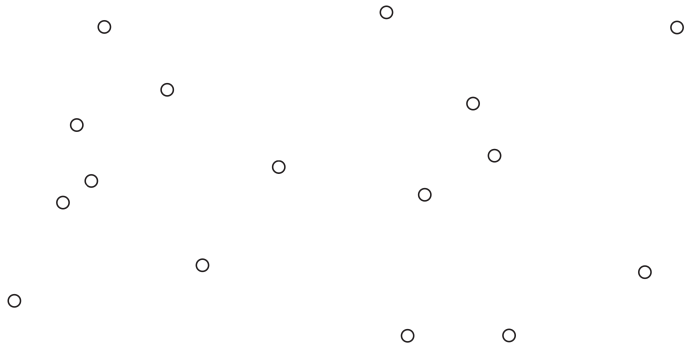
DS 320 Algorithms for Data Science **Lecture #11 Worksheet**
Spring 2025 Prof. Kira Goldner and Jeffrey Considine

Divide & Conquer II: Closest Pair of Points

The Problem

Your input for the *closest pair of points* problem is a set P of n points in $\mathbb{R}^2$. Assume that all of the x and y coordinates are distinct. The goal is to output a pair of points p_1 and p_2 minimizing the Euclidean L_2 distance $d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.



This problem extremely common: in graphics, computer vision, robotics, scientific simulation, etc.

As we mentioned last time, Divide & Conquer is usually the way to come up with a *more efficient* algorithm for a problem which already has a polynomial brute force solution. What's the **naïve algorithm** here and what is its **running time**?

Step 1: Define your recursive subproblem.

Hint: An idea similar to mergesort works here.

Step 2: Combine the solutions to your subproblems.

Given the solutions (closest pair) from your subproblems (make sure the parameters make sense), how do you combine or compare these solutions to get the solution to our problem of size n?

Computing the Closest Pair Across Halves

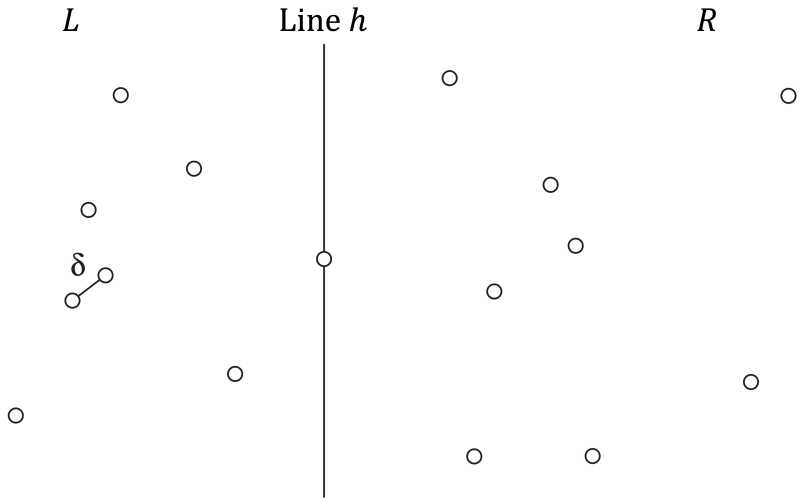
Idea: Narrow down the set of points we need to search.

Lemma 1. *For subproblem solutions of closest pairs $\ell_0^*, \ell_1^* \in L$ and $r_0^*, r_1^* \in R$, let*

$$\delta = \min\{ \quad d(\ell_0^*, \ell_1^*) \quad , \quad d(r_0^*, r_1^*) \quad \}.$$

If $\ell \in L$ and $r \in R$ and $d(\ell, r) \leq \delta$, then ℓ and r are within δ of h.

Proof.



Definition 1. Let $S = \{p \in P : d(p, h) < \delta\}$ be the set of points within distance δ of the line h.

Note: We can compute S in linear time.

Lemma 1 implies that we only need to search S. Problem: Brute-force over S is $O(n^2)$ still.

Idea: Brute force more intelligently.

Lemma 2. *Let S_y be a list of the points in S sorted by y-coordinate and let (s, s') be closest pair of points in P. If $s, s' \in S$, then they cannot lie more than 15 positions apart in S_y.*

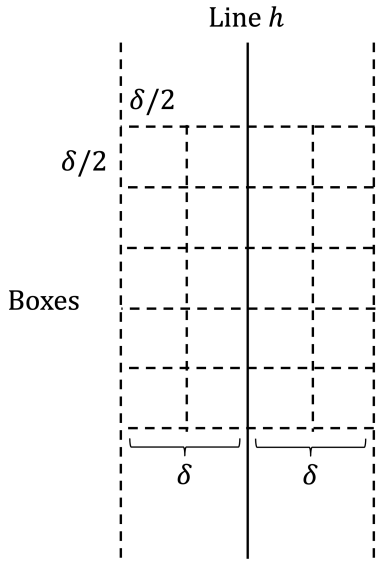
(There are less than $15n$ such pairs, hence linear time to brute force!)

Proof.

a. Divide S into boxes of side-length $\delta/2$.

b. **Show:** No two points can share a box.

c. **Show:** If s and s' are at least 16 positions apart in S_y, then they're not the closest pair of points.



The Algorithm

Preprocessing: Generate P_x and P_y.

Algorithm 1 closest(P_x, P_y).

Input: Array P_x of points in P sorted by x coordinate; array P_y sorted by y coordinate.
if $|P| \leq 3$ **then** // Base Case
  return closest pair by brute force
end if
Construct L_x, L_y, R_x, R_y // Subproblem: Recursive calls on halves
$(\ell_0^*, \ell_1^*) =$ closest(L_x, L_y)
$(r_0^*, r_1^*) =$ closest(R_x, R_y)
Construct S_y
$(s_0^*, s_1^*) =$ closest pair in S_y within 15 spots of each other
return closest of (ℓ_0^*, ℓ_1^*), (r_0^*, r_1^*), (s_0^*, s_1^*)

Runtime

$T(n) = a\,T(n/b) + O(f(n))$ $\quad a =$ $\quad b =$ $\quad f(n) =$

$\Rightarrow T(n) =$

Proof of correctness