DIVIDE + CONQUER: CLOSEST POINTS

Learning Goals

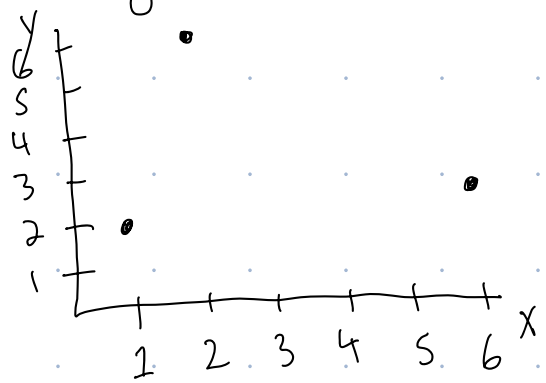
- Analyze runtime of D&C alg [D&C 1]
- Create a D&C alg [D&C 2]
- Benchmark algorithm with brute force / easier problem
- Build Intuition with easier problems
- Analyze ethics of alg using ethical matrix [Eth 1]
- Prove correctness of divide + conquer using strong induction [DC 3]

Closest Points Problem

Input: Array of 2-D points:

$P = \boxed{(1,2) \mid (2,6) \mid (6,3) \mid \cdots}$

$|P| = n$



(2,2)

no 2 pts have same x-coordinate

Output: Distance b/t 2 closest points

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

Applications:
- air traffic control
- robotics
- stereo → 3D

~~(2,6)~~
~~(2,20)~~

Algorithms + Ethics

Algorithm is essentially a mathematical object.

But once it gets implemented for a particular task, has ethical implications

ClosePts(P) (P list of 2-D pts, $|P| \geq 2$)

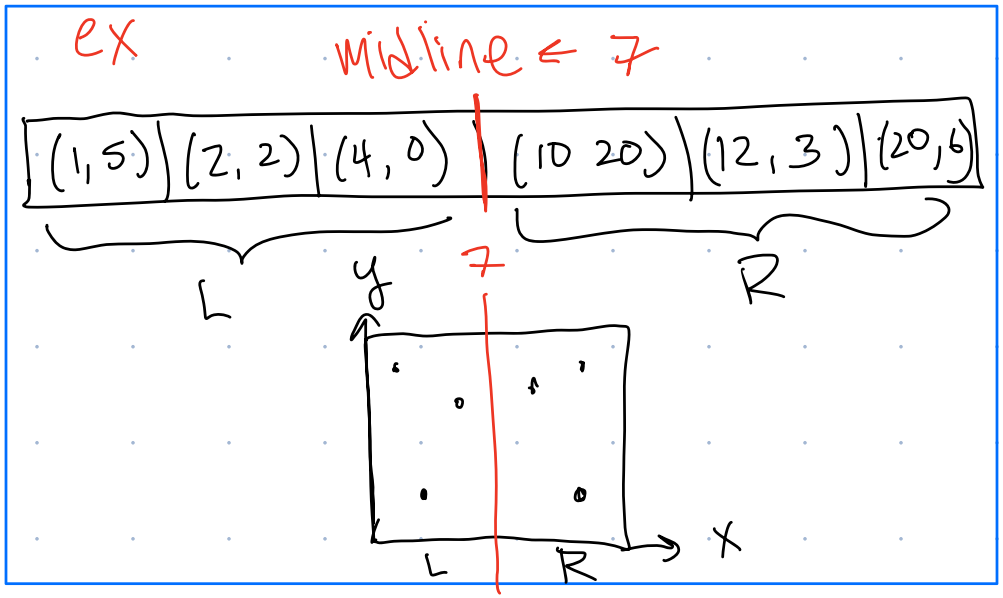
// Base Case

1. If $|P| \leq 3$:
2. | Check distance of each pair + return smallest found

// Divide

3. Sort P by x-coordinate
4. midline ← x-value between L + R halves

ex



We will prove using strong induction that ClosePts(P) correctly returns the distance between the closest points all $n \geq 2$ where $n = |P|$.

(Fill in blanks for other divide + conq.)

Base Case: When $n =$ 2 or 3, the base case of the algorithm correctly finds the smallest distance via brute force in lines 1 + 2.

largest b.c.

Inductive Step: Let $K \geq 3$ and assume ClosePts is correct for all input sizes j for $2 \leq j \leq K$. Consider an input of size $K+1$.

smallest b.c.

// Conquer

$\delta_1 \leftarrow$ ClosePts(L)

$\delta_2 \leftarrow$ ClosePts(R)

$\delta \leftarrow \min\{\delta_1, \delta_2\}$

// Combine

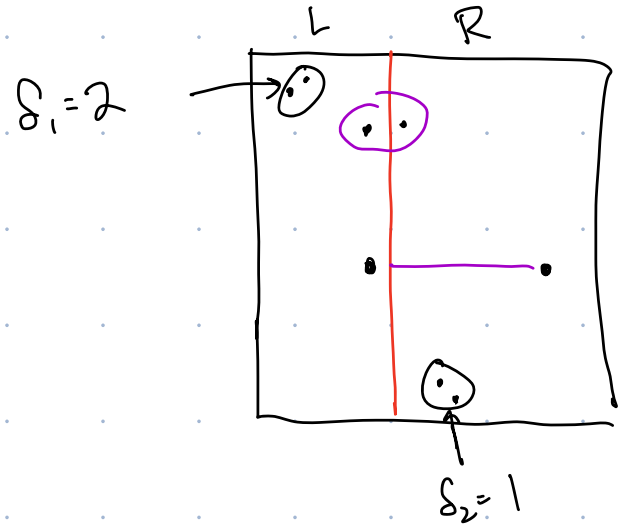
Since $k+1 \geq 4$, we skip the base case and go to the divide step. (line 3.)

[Since $k+1 \geq 4$, when we divide into L + R, L, R each have at least 2 but less than $k+1$ pts, so by inductive assumption, ClosePts(L) and ClosePts(R) each correctly return the closest distance among their respective points. Thus δ contains the smallest distance between points where both pts are to the left of the midline or both are to the right.

Next we need to find if there is a closer pair with one pt in L and one in R.

Combine:

Let's think about this:



What points do we need to worry about in combine step?

Those within

A) $\delta/2$ of midline

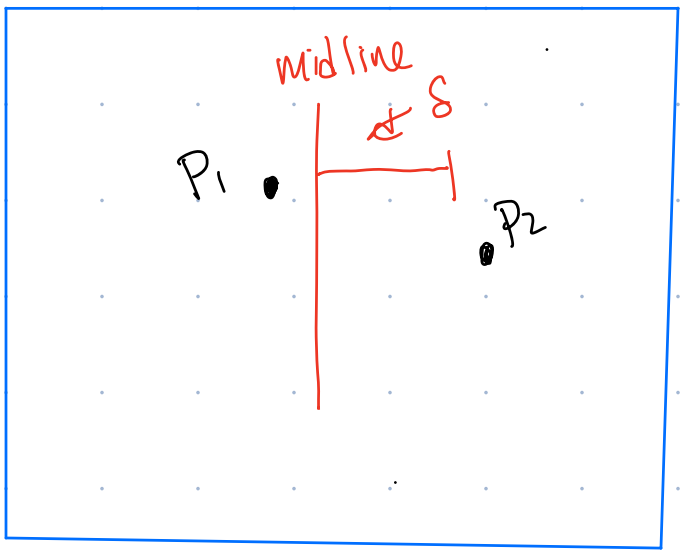
B) $\sqrt{\delta}$ of midline

C) δ of midline

D) 2δ of midline

// Combine

8. $Y_S \leftarrow$ Pts within δ of midline ... sorted by y



However, we only need concern ourselves with pts that are within ~~d~~ δ (t.b.d) of the midline. Too see this, consider p_1 in L, and p_2 in R, but at least ~~d~~ δ away from the midline. Then

$x_1 <$ midline $\quad x_2 >$ midline + ~~d~~ δ,

and so

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

$$\geq \sqrt{(x_1-x_2)^2} = |x_1-x_2|$$

$$= x_2 - x_1$$

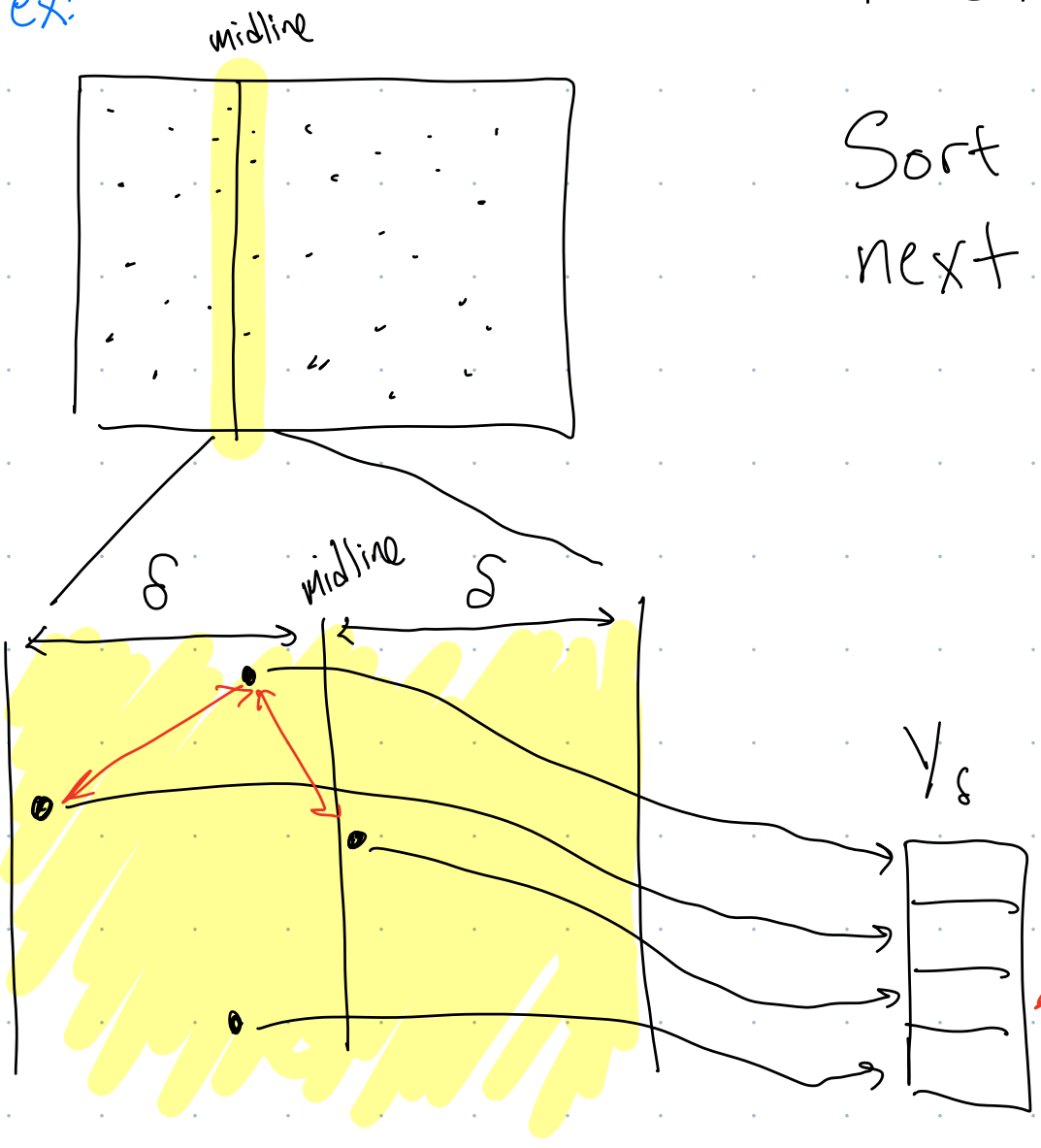
$$> \text{~~d~~ } \delta$$

Thus p_1 and p_2 have distance larger than δ, so can not be the closest pair. (A similar

argument holds for pts to the left of the midline.)

Therefore, we only need consider pts in Y_S as in line 8.

ex:



Looks almost 1-D?

Sort by y and check next pt?

closest pts not adjacent

9 for $p \in Y_S$:

10 • check distance from p to next ___ pts in Y_S

11 • $\delta_{LR} \leftarrow$ smallest distance found

12 return $\min\{\delta, \delta_{LR}\}$

What goes in blank?

Show can't cram too many pts near p because closest pts in L, R have distance δ

With Y_S sorted by y-coordinate, given a pt p in Y_S, we will show any pt closer than δ to p must be within $\square$ pts of p in Y_S.