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]

Warm-Up

go/cs302 → hand written notes → review proof so far. closest pts

(can use computer)

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

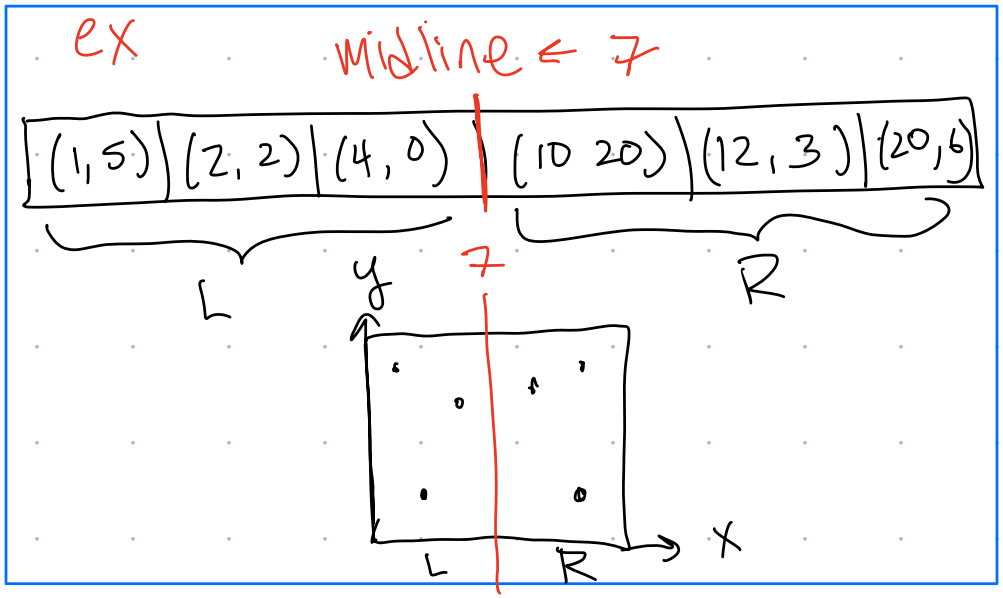
// Base Case

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

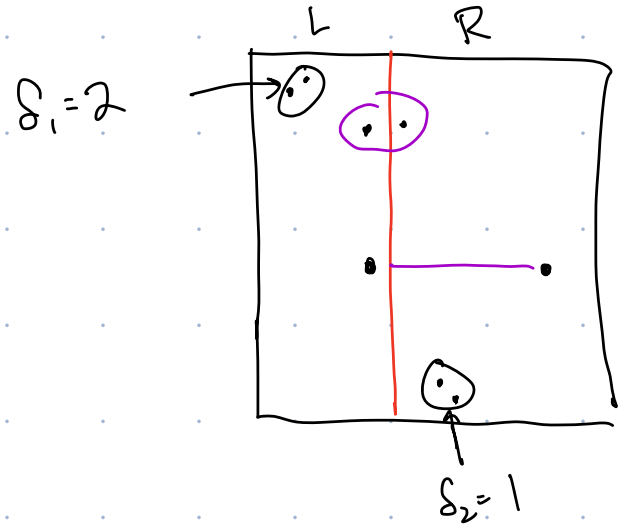
Since $k+1 \geq 4$, we skip the base case and go to the divide step. (line 3.)

[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

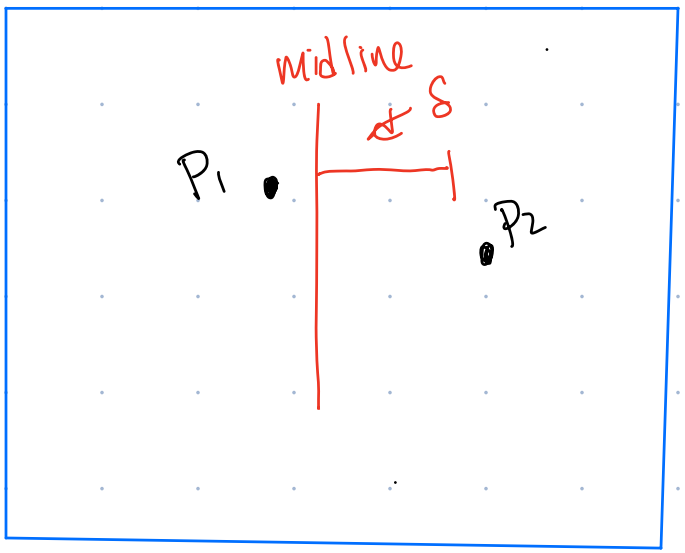
B) $\sqrt{\delta}$ of midline

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

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

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

$$= x_2 - x_1$$

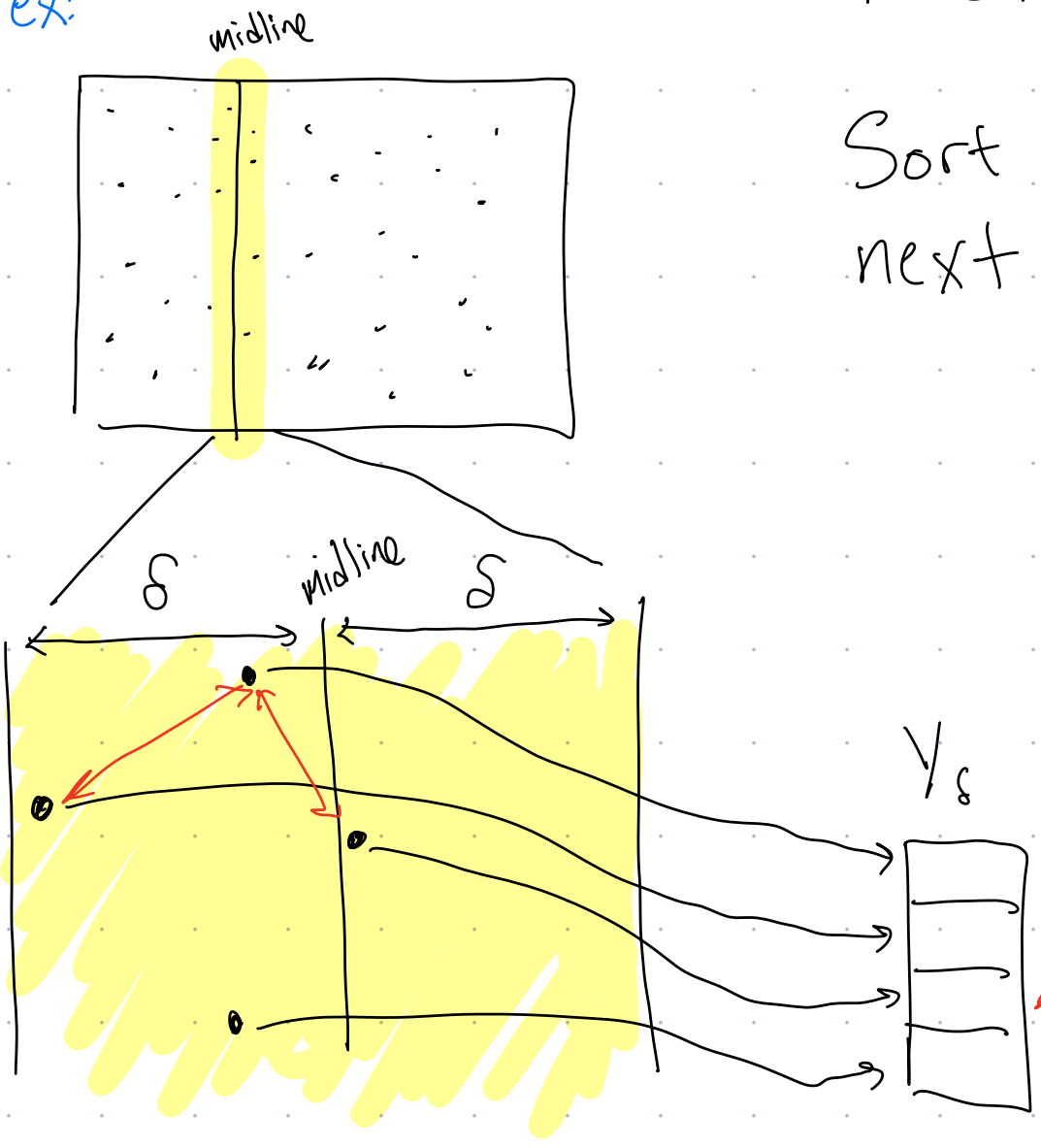
$$> \text{~~d~~ } \delta$$

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 ∈ Y_S:
10   • check distance from p to next 7 pts in Y_S
11   • δ_LR ← smallest distance found
12 return min{δ, δ_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 7 pts of p in Y_S. To see this, imagine placing a grid of $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes in the region within δ of the midline, starting at pt p, as in Fig 4. We will argue by contradiction that at most 1 pt can be in each box.

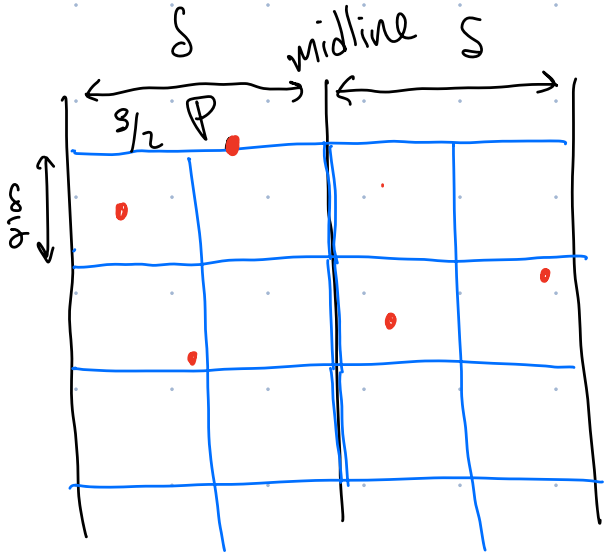


Fig 1: Imagined grid overlaying Y's starting at p.

Suppose for contradiction there are 2 pts in a box. Then the distance between pts is

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

$$\leq \sqrt{\left(\frac{\delta}{2}\right)^2+\left(\frac{\delta}{2}\right)^2}$$

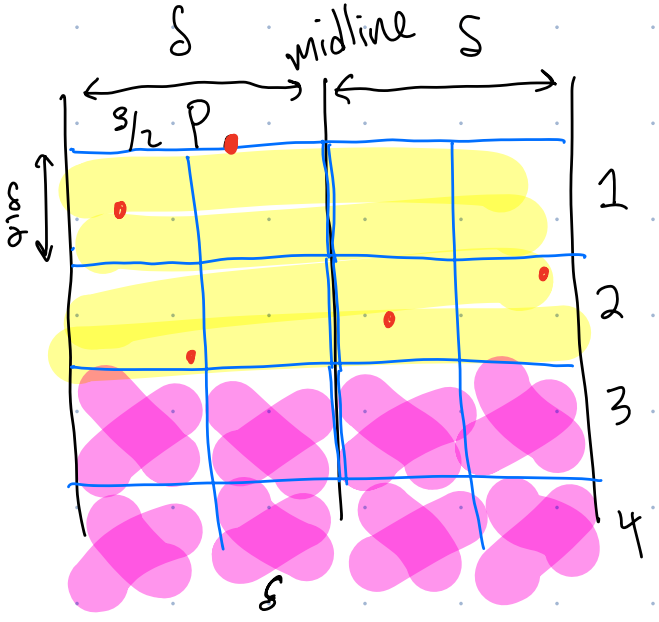
$$= \sqrt{\frac{\delta^2}{4}+\frac{\delta^2}{4}}$$

$$= \sqrt{\frac{\delta^2}{2}}$$

$$= \frac{\delta}{\sqrt{2}}$$

$$< \delta$$

But each box is in L or R, so their distance must be at least δ, a contradiction.



Further, we note that only boxes in the two rows immediately below p can contain a pt closer than δ to p, because the y-coordinate difference to pts in further rows is at least δ.

There are 8 boxes in the first two rows, the points in the first two rows will appear in Y_S before any later points b/c Y_S is sorted by y, so checking the next 7 pts we will

find a pt closer than δ to p, if it exists.

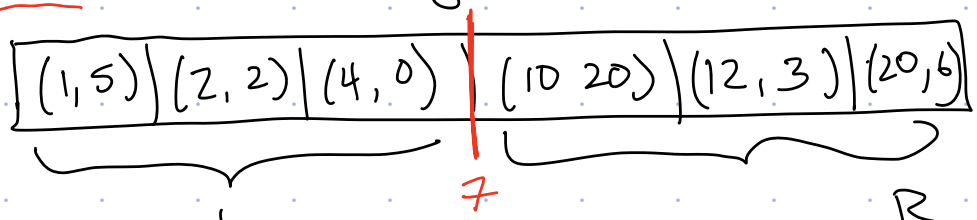
Therefore our combine step will find any pair with distance less than δ in Y_S.

Thus, our algorithm will correctly find the closest distance of any pair

CloPts(P) (Divide + Conquer 2D Closest Pts)

Base Case: If $|P| \leq 3$, then do brute force

Divide: Sort by X



Divide into L, R by midline

Conquer:

$\delta_1 = \text{CloPt}(L)$

$\delta_2 = \text{CloPt}(R)$

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

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 3 \\ 2T\left(\frac{n}{2}\right) + O(n \log n) & \end{cases}$$

$\hookrightarrow O(n \log^2 n)$

- Create recurrence relation for runtime.
- Explain why correct
- Q's about correctness

Combine:

$y_s \leftarrow$ pts w/in δ of midline, sorted by y-coordinate

for $p_i \in y_s$: ← $O(n)$

  for $j \leftarrow i+1$ to $i+7$: ← $O(1)$ } $O(n)$

    if $d(p_i, p_j) < \delta$, then $\delta \leftarrow d(p_i, p_j)$ ← $O(1)$ } $O(1)$

return δ

Problem: Sorting at each recursive step takes too long. Presort!

PreSort(P)

$X \leftarrow$ P sorted by x

$Y \leftarrow$ P sorted by y

return X, Y

$\downarrow$

$O(n \log n)$

ClosePts(X, Y)

1. If $|X| \leq 3$, do Brute Force
2. Divide into X_L, Y_L, X_R, Y_R $O(n)$
3. $\delta = \min\{\text{ClosePts}(X_L, Y_L), \text{ClosePts}(X_R, Y_R)\}$
4. Create Y_S $O(n)$
5. For $p_i \in Y_S$: $O(n)$
   - For $j \leftarrow i+1$ to $i+7$
     - If $\text{dist}(p_i, p_j) < \delta$, $\delta \leftarrow \text{dist}(p_i, p_j)$
6. Return δ.

$\swarrow$

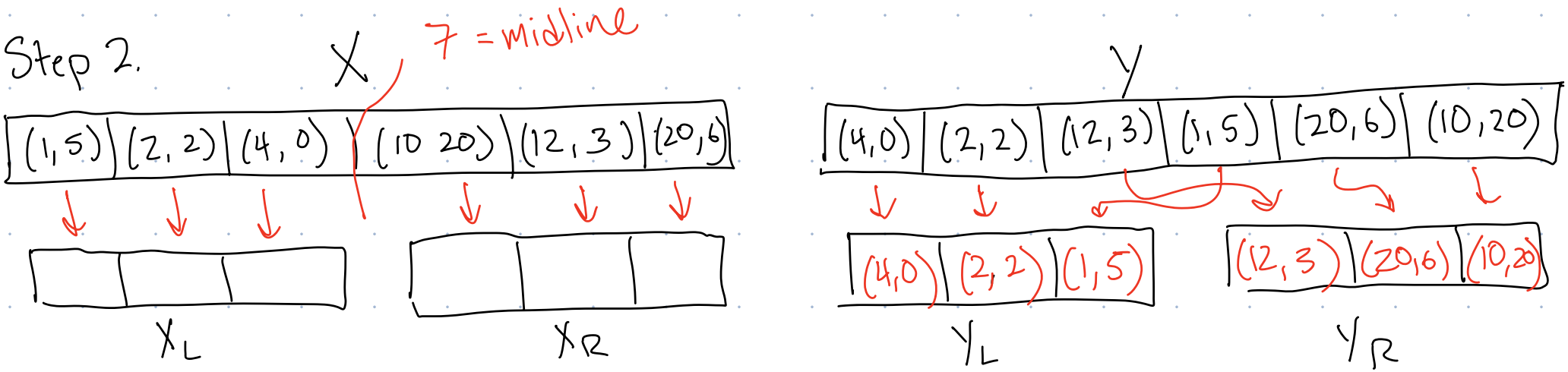
$O(n \log n)$ Yay!

$$T(n) = \begin{cases} O(1) & n \leq 3 \\ 2T\left(\frac{n}{2}\right) + O(n) \end{cases}$$

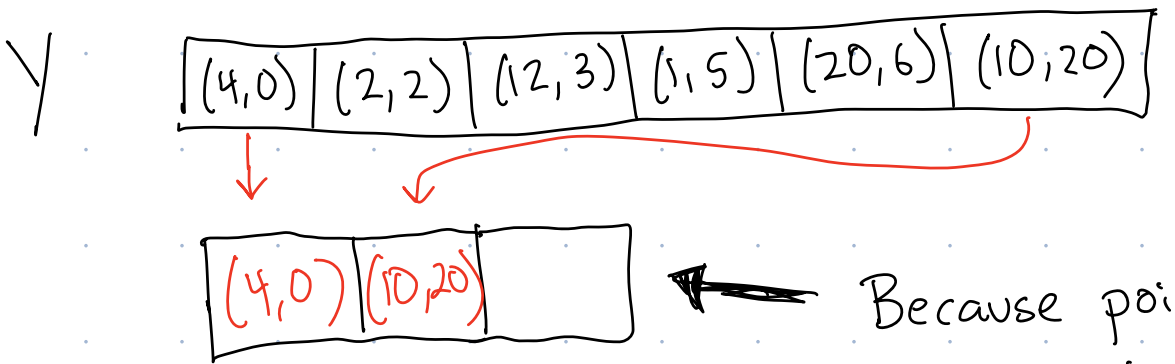
$\hookrightarrow O(n \log n)$

Dividing Arrays Efficiently

Step 2.



Step 4 : $S = 4$ Want x from 3 to 11



Because points in L and R are grouped together in one array, until we look at the pt, we don't know if in L or R

- Where did we use our assumption that x,y points are unique?

Difficult to divide into L and R using O(n) time.

- You can do Prg. Assign. 1!

Honor Code Conversation!

genAI
- use to check code / look for error / write tests
- help with explaining error messages
- generating main function
- Syntax
- Data wrangling / IO
- Optimizing, improving existing code

No-ethics

No - copy/paste output, asking for advice before starting
- use without understanding

Peers

- Work with others if list names (2-4?)
  - → Brainstorming
  - → Pair Code (alternate who's coding)
  - → Code review
  - → conceptual problems with others

Course Assistants + Shelby OK

Readme done on own

Copy? "Write in own words"

AI?

Peers?

Course Assistants?

Other resources?

Me?