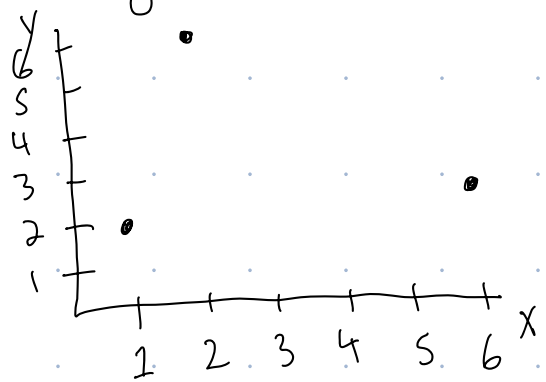
DIVIDE + CONQUER: CLOSEST POINTS

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



Output: Distance b/t closest 2 pts

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

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

Algorithms + Ethics

Algorithm is essentially a mathematical object.
But once it gets implemented for a particular task, has ethical implications

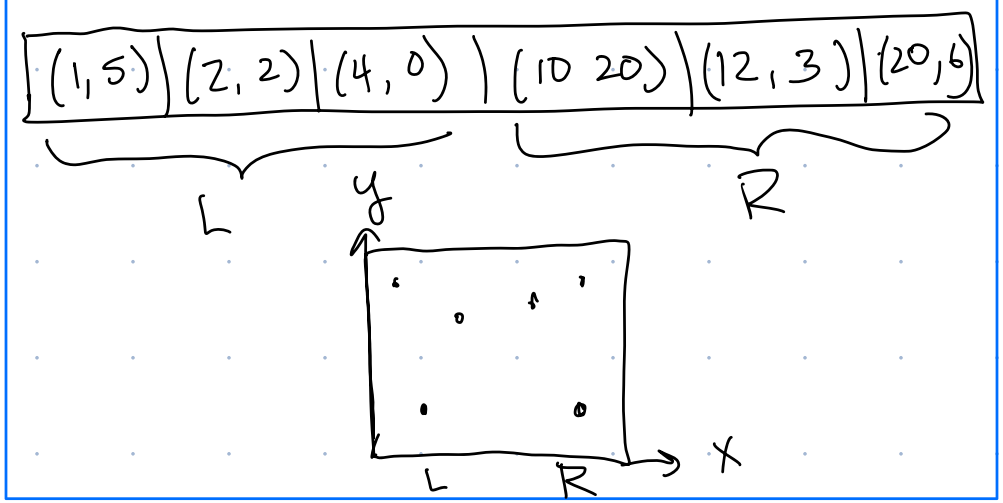
ClosePts(P) (P list of 2-D pts, $|P| \geq 2$)

// Base Case

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

// Divide

ex



We will prove using strong induction that _____ correctly returns _____

_____

all $n \geq$ _ where _____

(Fill in blanks for other divide + conq.)

Base Case: When $n =$ _____, the base case of the algorithm correctly _____ _____ via brute force.

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

// Conquer

// Combine

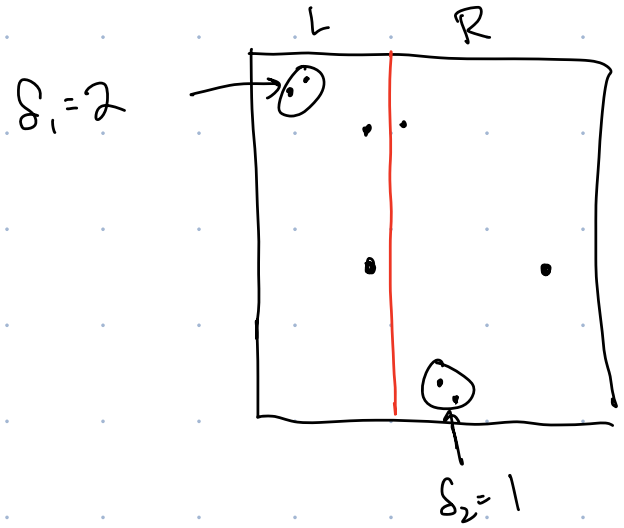
Since $k+1 \geq$ _, we skip the base case and go to the divide step.

[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, ClosPts(L) and ClosPts(R) each correctly return the closest distance among their respective points. Thus S 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 the closest distance between pts where one is in L and one is 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

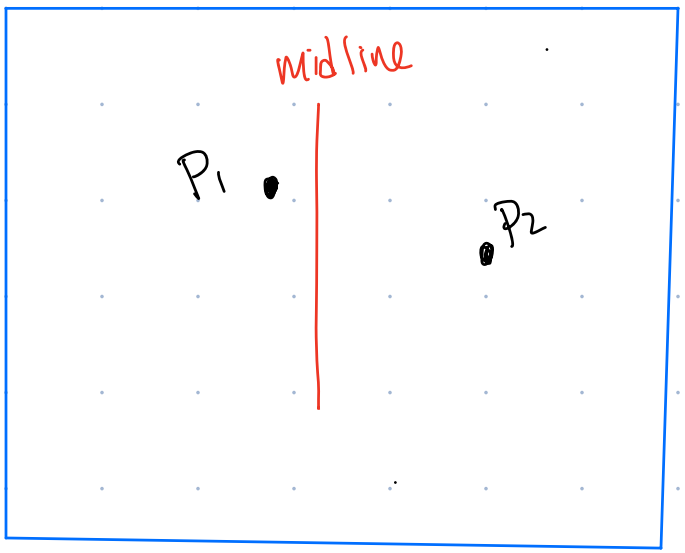
B) $\sqrt{\delta}$ of midline

C) δ of midline

D) 2δ of midline

// Combine

8. Y ← Pts within δ of midline...



However, we only need concern ourselves with pts whose X-coor. are within δ of the midline. This is b/c any pair where p_1 in in L (so $x_1 <$ midline), and p_2 is in R with X-coor $x_2 >$ midline $+ \delta$, has

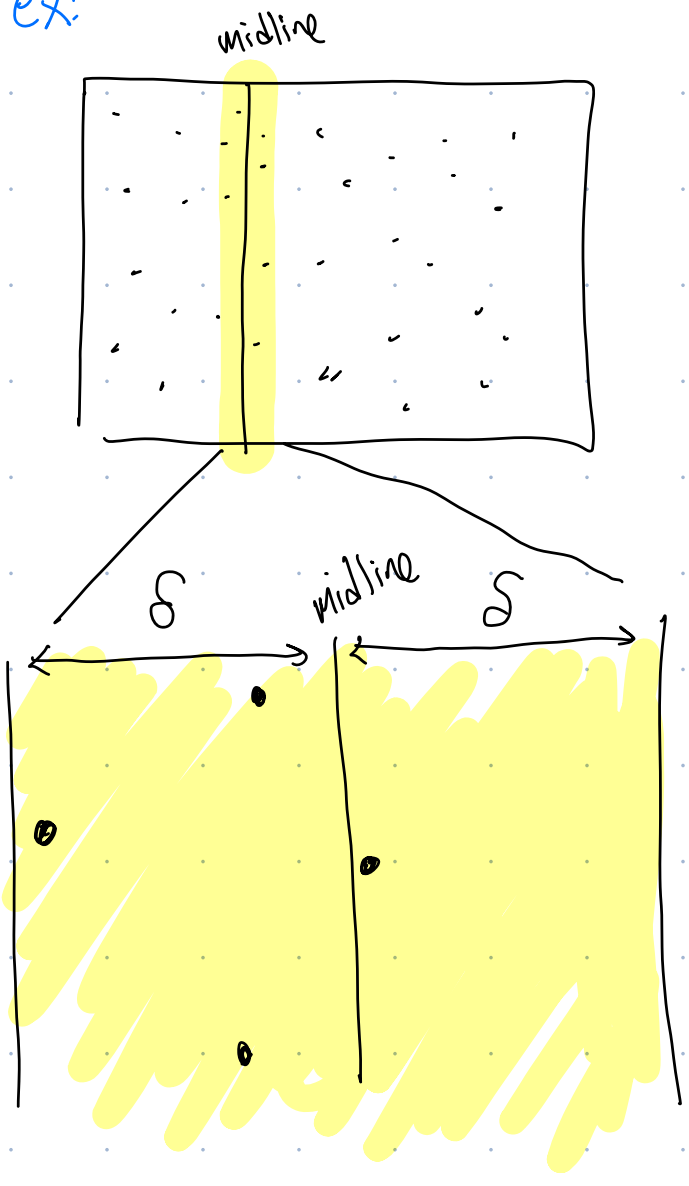
$d(p_1, p_2) =$

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.)

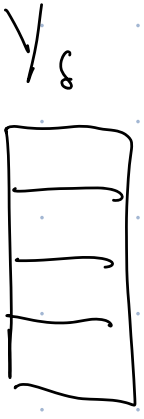
Therefore, we only need consider pts in y_S.

ex:



Looks almost 1-D?

Sort by y and check next pt?



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 ___ pts of p in Y_S.

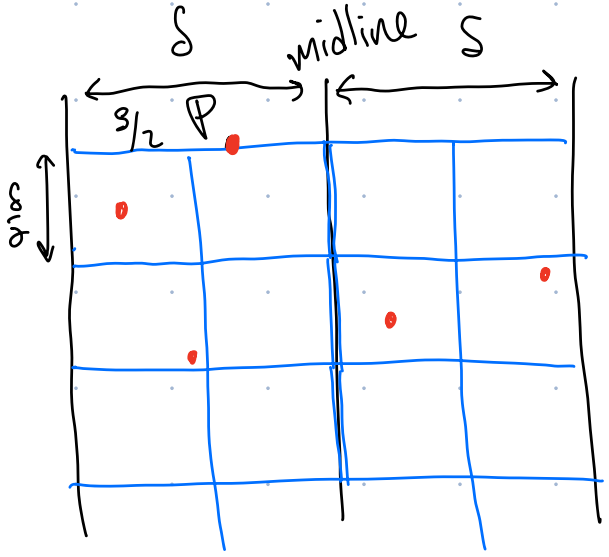
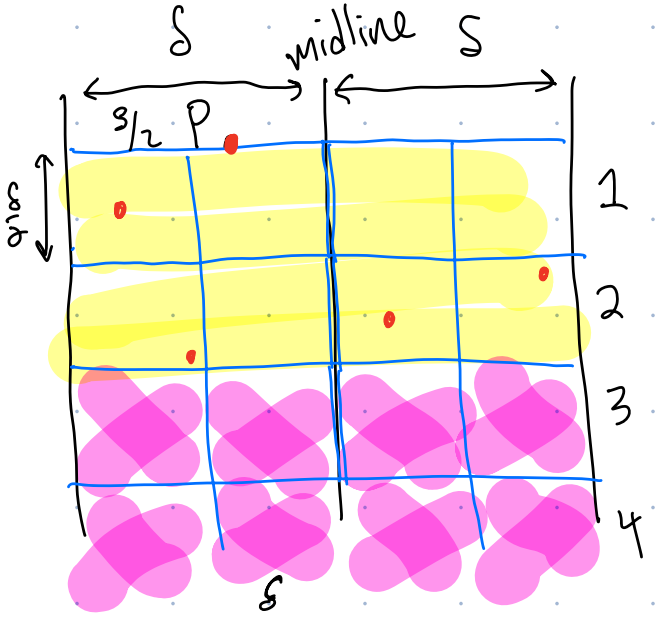


Fig 1: Imagined grid overlaying Ys starting at P.



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 pts we will

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

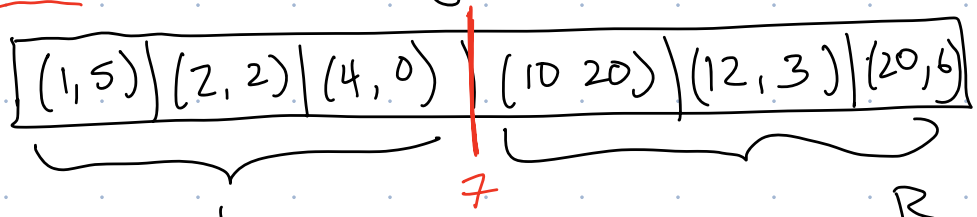
Therefore our combine step will find any pair with distance less than δ in Y_S.

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

ClosePts(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:

δ_1 = ClosePt(L)

δ_2 = ClosePt(R)

δ = min(δ_1, δ_2)

Combine:

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

for $p_i \in y_s$:

  for $j \leftarrow i+1$ to $i+7$:

    if $d(p_i, p_j) < \delta$, then $\delta \leftarrow d(p_i, p_j)$

return δ

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

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

ClosestPts(X, Y)

1 If $|X| \leq 3$, do Brute Force

2. Divide into X_L, Y_L, X_R, Y_R

3. $\delta = \min\{\text{ClosestPts}(X_L, Y_L), \text{ClosestPts}(X_R, Y_R)\}$

4 Create Y_S

5. For $p_i \in Y_S$:

   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.

Dividing Arrays Efficiently

Step 2.

X

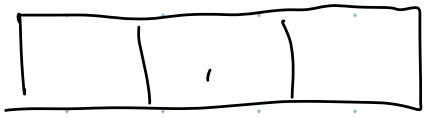
(1,5)	(2,2)	(4,0)	(10 20)	(12,3)	(20,6)

X_L

X_R

Y

(4,0)	(2,2)	(12,3)	(1,5)	(20,6)	(10,20)



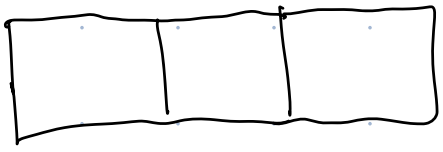
Y_L

Y_R

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

Y

(4,0)	(2,2)	(12,3)	(1,5)	(20,6)	(10,20)



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!