Lecture 30

CSE 331

Nov 10, 2017

Mini project video due Mon

note

stop following 127 views

Video submission now open on Autolab

Sorry, forgot to do this earlier: you can now submit your video (note still PDF with the link in it) on Autolab.

YOU WILL NEED TO FORM YOUR GROUP ON AUTOLAB AGAIN BEFORE SUBMITTING.

See the mini project page for the details:

http://www-student.cse.buffalo.edu/~atri/cse331/fall17/mini-project/index.html

#pin

mini_project

edit good note | 0

Updated 2 days ago by Atri Rudra

Homework 8

Homework 8

Due by **11:00am, Friday, November 17, 2017**.

Make sure you follow all the homework policies.

All submissions should be done via Autolab.

Question 1 (Programming Assignment) [40 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

In this problem, we will explore minimum spanning trees.

We are given a undirected, connected graph represented by its **adjacency matrix** representation. Our goal it to find a minimum spanning tree of that graph

Input

The input file is given as an $n \times n$ matrix where each entry (u, v) represents the weight of the edge between nodes $u \in \{0, 1, \ldots, n-1\}$ and $v \in \{0, 1, \ldots, n-1\}$. If there is no edge then the weight is `-1`. Edge weights will be `0 <= w <= 50`.

Solutions to Homework 7

At the END of the lecture

Counting Inversions

Input: n distinct numbers $a_1,a_2,\ldots,a_n$

Inversion: (i,j) with $i < j$ s.t. $a_i > a_j$

Output: Number of inversions

Divide and Conquer

Divide up the problem into at least two sub-problems

Recursively solve the sub-problems

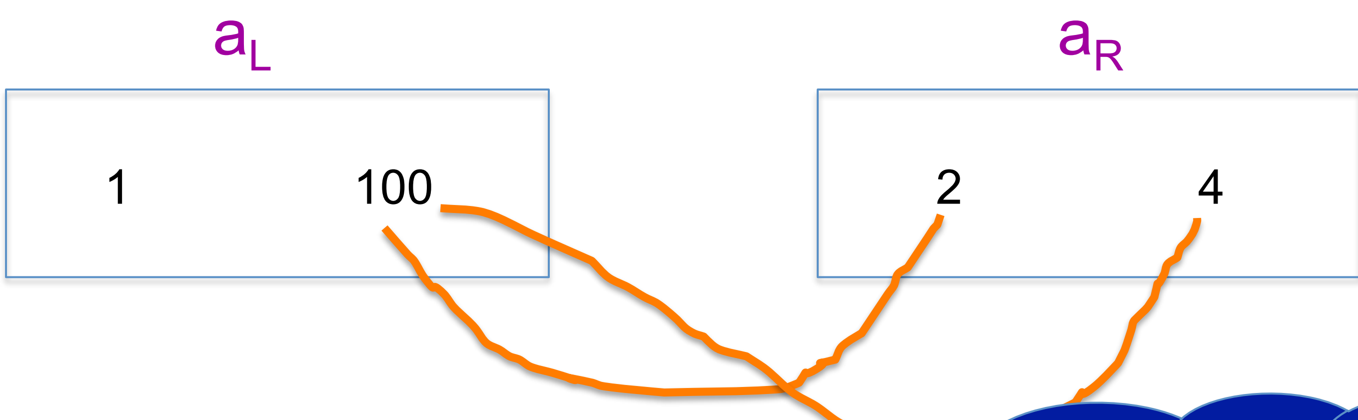
Solve all sub-problems: Mergesort

Solve some sub-problems: Multiplication

Solve stronger sub-problems: Inversions

"Patch up" the solutions to the sub-problems for the final solution

Handling crossing inversions



Why should a_L and a_R be sorted?

http://www.dovecoteidea.com/

Sort a_L and a_R recursively!

Mergesort-Count algorithm

Input: $a_1, a_2, \ldots, a_n$

Output: Numbers in sorted order+ #inversion

MergeSortCount(a, n)

If $n = 1$ **return** $(0, a_1)$

If $n = 2$ **return** ($a1 > a2$, $\min(a_1,a_2)$; $\max(a_1,a_2)$)

$a_L = a_1,\ldots, a_{n/2}$ $a_R = a_{n/2+1},\ldots, a_n$

(c_L, a_L) = MergeSortCount(a_L, $n/2$)

(c_R, a_R) = MergeSortCount(a_R, $n/2$)

(c, a) = MERGE-COUNT(a_L,a_R)

return $(c+c_L+c_R, a)$

$T(2) = c$

$T(n) = 2T(n/2) + cn$

O(n log n) time

O(n)

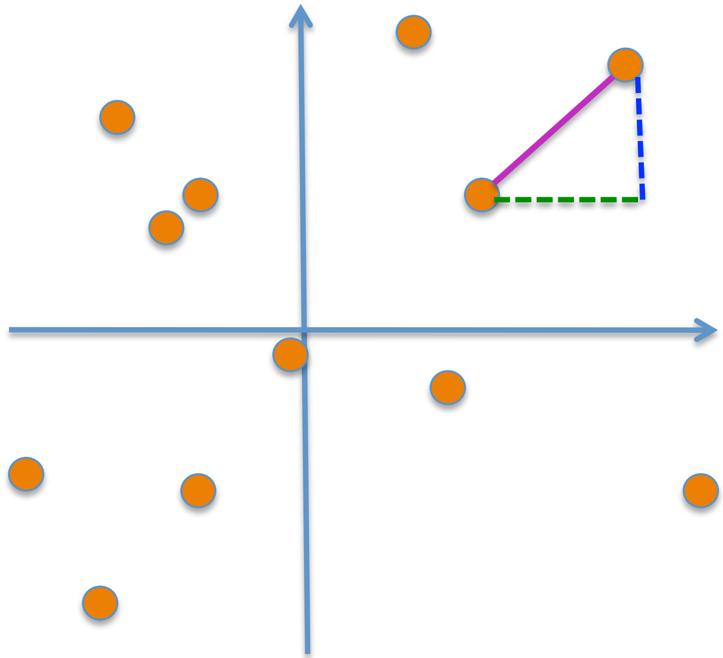
Counts #crossing-inversions+ MERGE

Closest pairs of points

Input: n 2-D points $P = \{p_1,\ldots,p_n\}$; $p_i=(x_i,y_i)$

$$d(p_i,p_j) = ((x_i-x_j)^2+(y_i-y_j)^2)^{1/2}$$

Output: Points p and q that are closest



Group Talk time

$O(n^2)$ time algorithm?

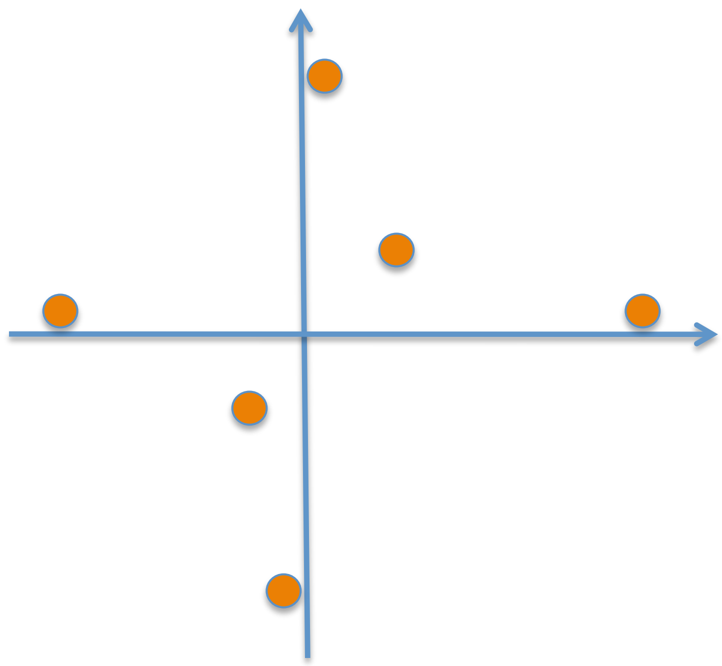
1-D problem in time $O(n \log n)$?

Sorting to rescue in 2-D?

Pick pairs of points closest in x co-ordinate

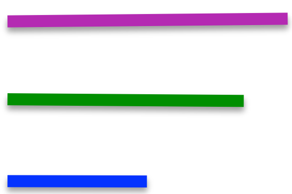
Pick pairs of points closest in y co-ordinate

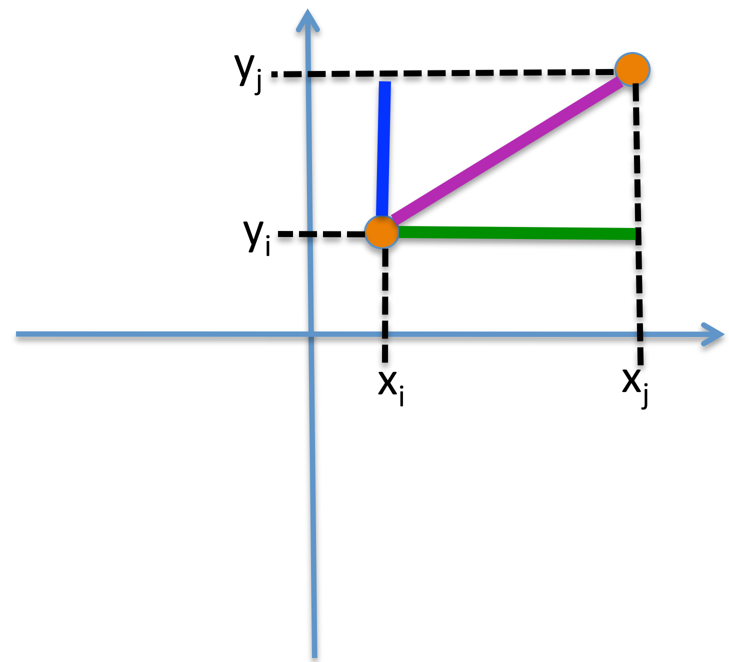
Choose the better of the two



A property of Euclidean distance

$$d(p_i,p_j) = ((x_i-x_j)^2+(y_i-y_j)^2)^{1/2}$$



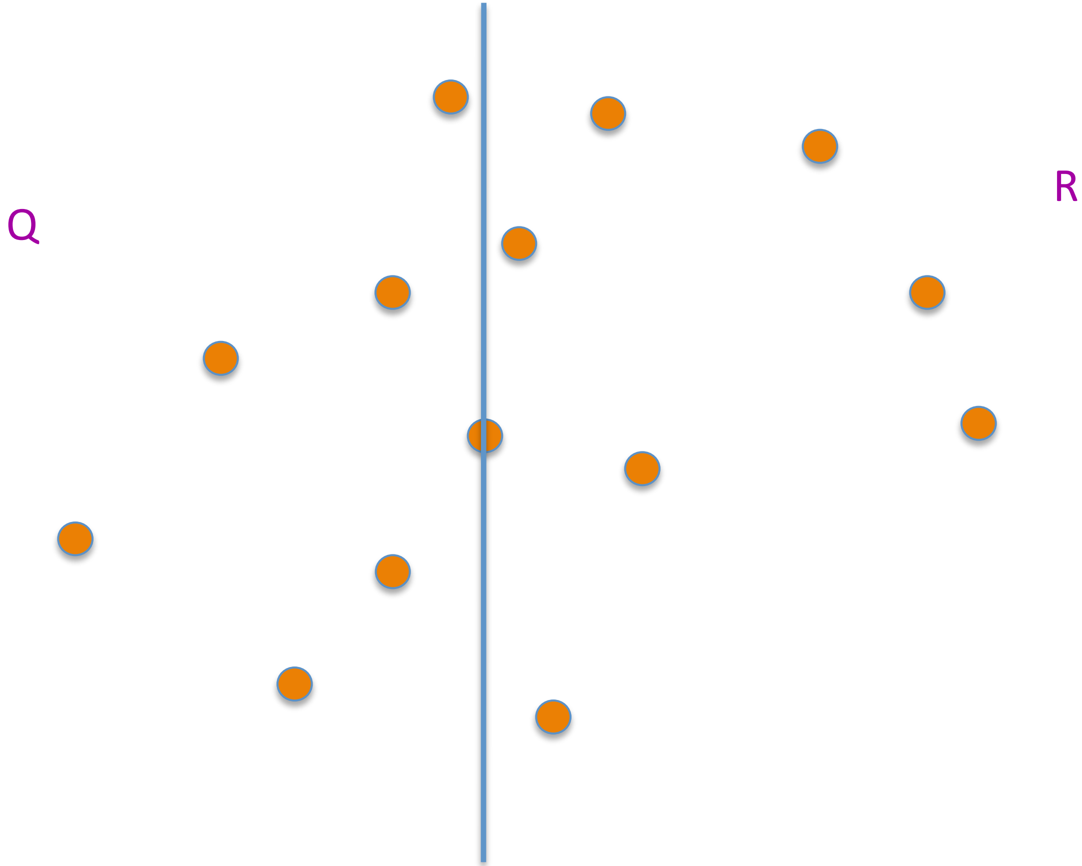


The distance is larger than the **x** or **y**-coord difference

Rest of Today's agenda

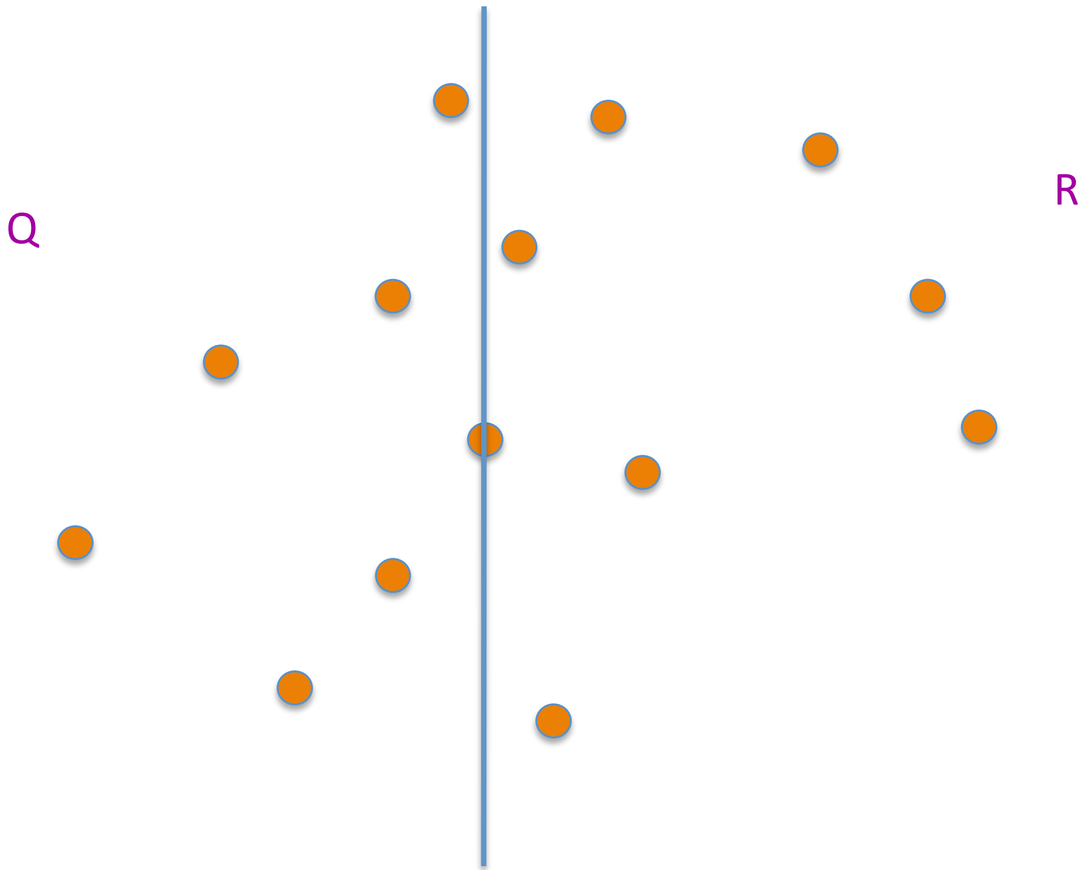
Divide and Conquer based algorithm

Dividing up P



First n/2 points according to the x-coord

Recursively find closest pairs



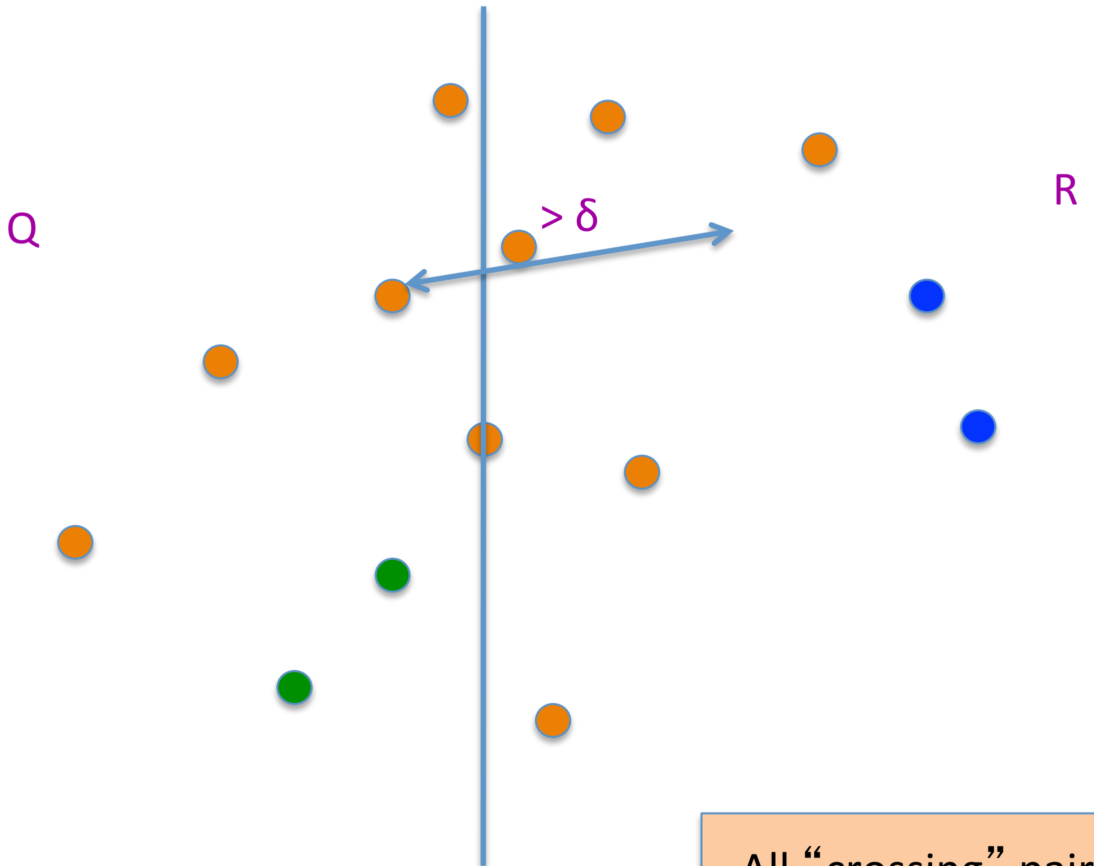
δ = min (**blue**, **green**)

An aside: maintain sorted lists

P_x and P_y are P sorted by x-coord and y-coord

Q_x, Q_y, R_x, R_y can be computed from P_x and P_y in $O(n)$ time

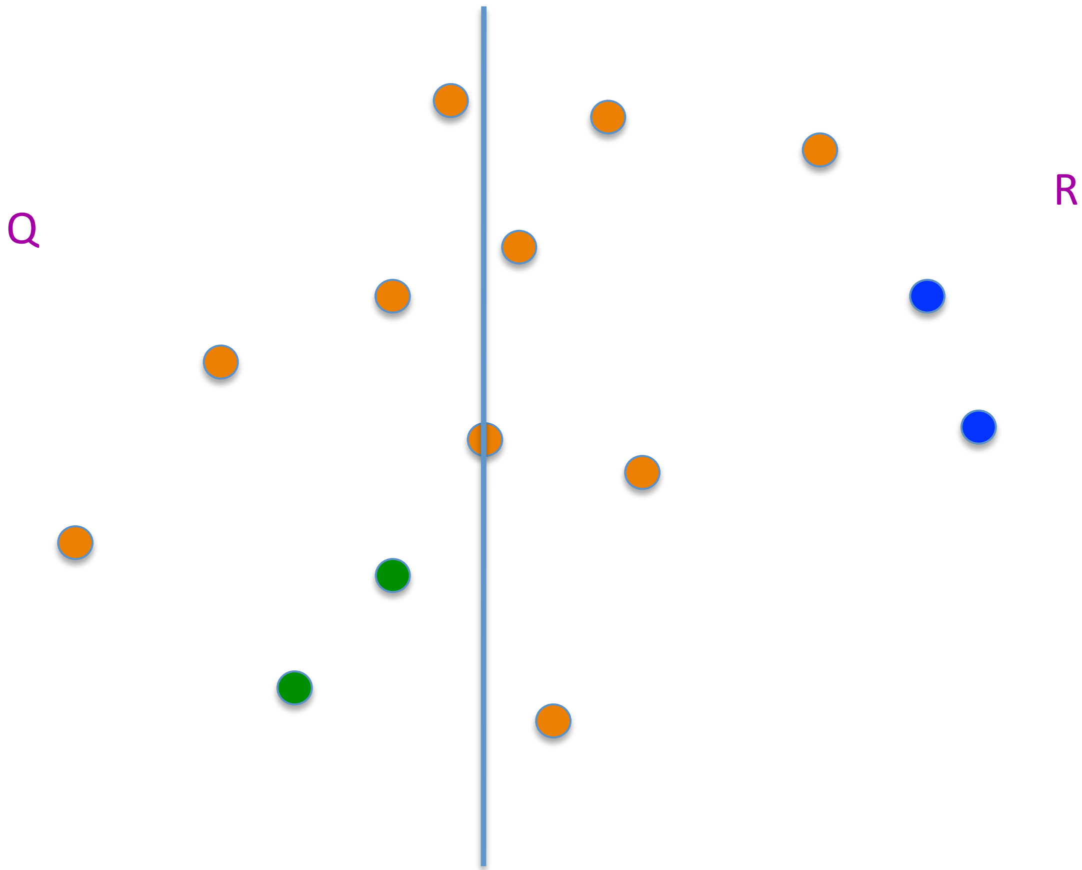
An easy case



All “crossing” pairs have distance > δ

δ = min (**blue**, **green**)

Life is not so easy though



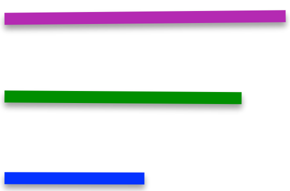
δ = min (**blue**, **green**)

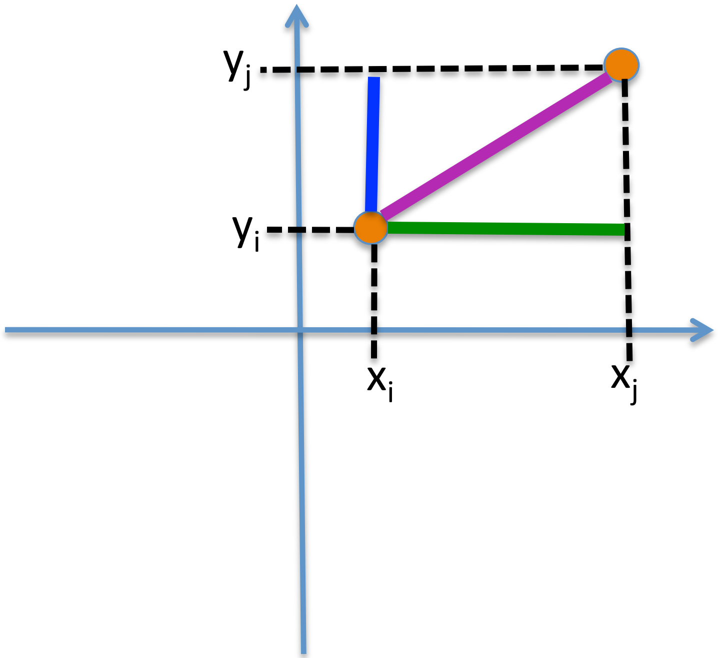
Rest of Today’s agenda

Divide and Conquer based algorithm

Euclid to the rescue (?)

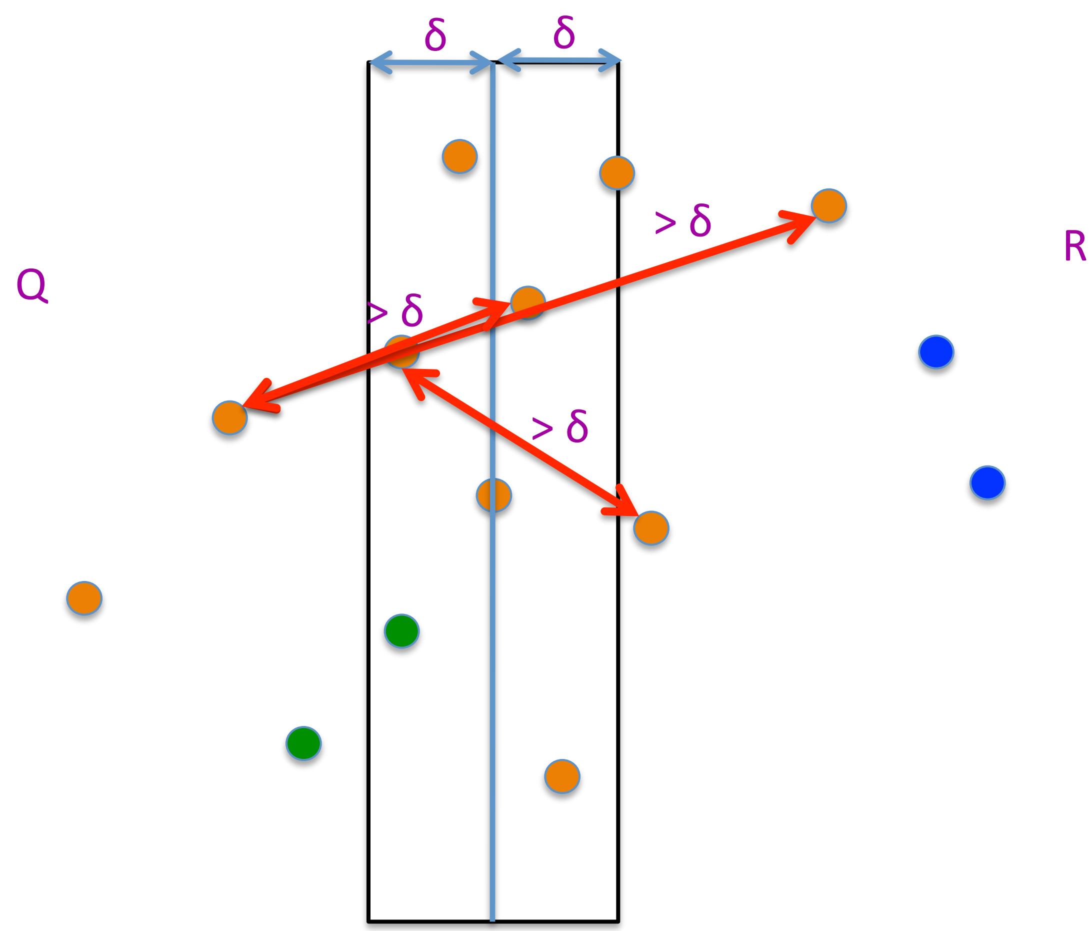
$d(p_i,p_j) = ((x_i-x_j)^2+(y_i-y_j)^2)^{1/2}$





The distance is larger than the **x** or **y**-coord difference

Life is not so easy though



δ = min (**blue**, **green**)

All we have to do now

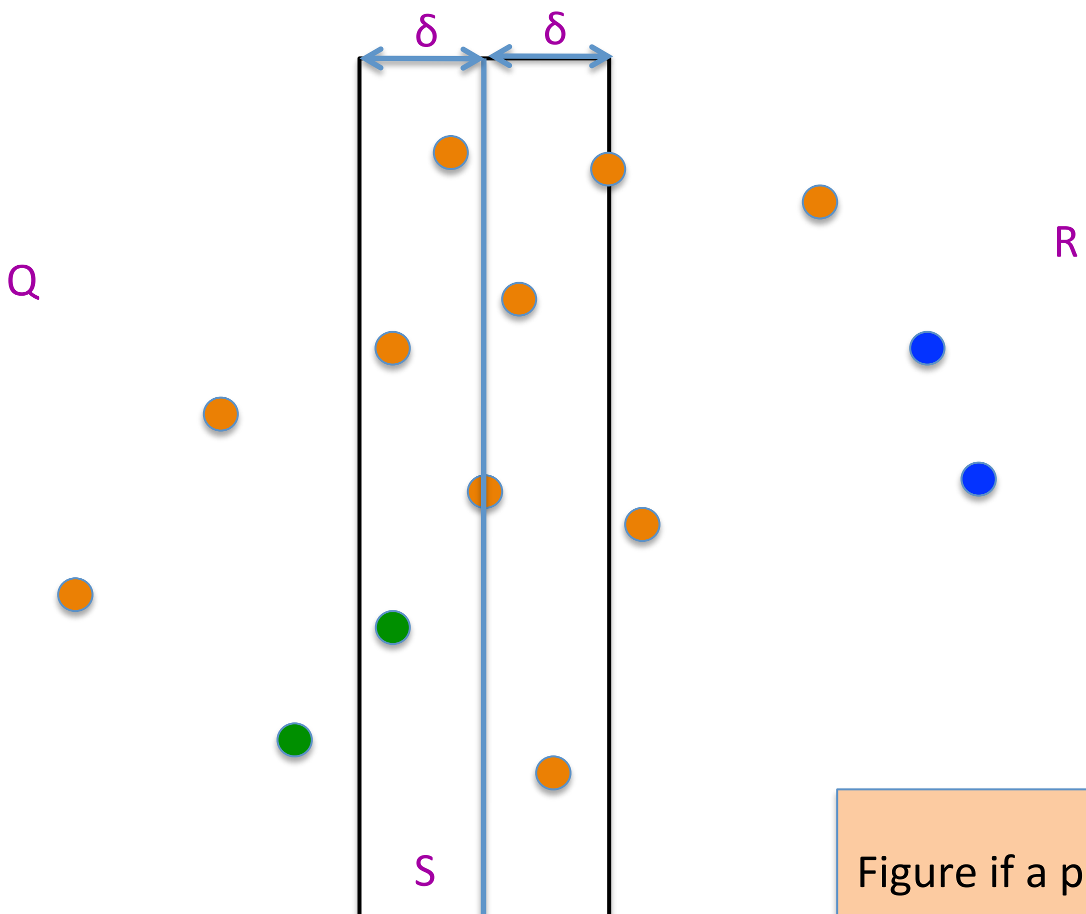


Figure if a pair in S has distance < δ

δ = min (**blue**, **green**)

The algorithm so far…

$O(n \log n) + T(n)$

Input: n 2-D points $P = \{p_1,\ldots,p_n\}$; $p_i=(x_i,y_i)$

Sort P to get P_x and P_y — $O(n \log n)$

Closest-Pair (P_x, P_y)

If $n < 4$ then find closest point by brute-force

Q is first half of P_x and R is the rest — $O(n)$

Compute Q_x, Q_y, R_x and R_y — $O(n)$

(q_0,q_1) = Closest-Pair (Q_x, Q_y)

(r_0,r_1) = Closest-Pair (R_x, R_y)

$\delta = \min(d(q_0,q_1), d(r_0,r_1))$ — $O(n)$

S = points (x,y) in P s.t. $|x - x^*| < \delta$ — $O(n)$

return Closest-in-box (S, (q_0,q_1), (r_0,r_1)) — Assume can be done in $O(n)$

$T(< 4) = c$

$T(n) = 2T(n/2) + cn$

$O(n \log n)$ overall