Lecture 31

CSE 331

Nov 10, 2014

Reports have been graded

Will hand them out at the end of the lecture

note ☆ stop following 1 views Actions

Graded Report

I have graded your mini project report and I will hand them back in class tomorrow (with comments marked on a printout: one per group). Here are the stats (out of a 100):

- Mean: 88.03
- Median: 88
- Std Dev: 6.52

I have uploaded your scores to UBLearns. (Recall that everyone in the group gets the same points.)

Below are the most common mistakes (the actual points deducted depended on the actual report):

1. Problem Statement (out of 25). Not having a clearly defined problem statement. Barring one or two exceptions for your chosen algorithm, you should have been able to explicitly specify the input to and output of your algorithm. Depending on how bad this was you could lose up to 15 points.
2. Algorithm (out of 25). Most of you had this down fine. However, depending on your submission you could lose up to 15 points.
3. Impact (out of 50). The most common mistake was to present applications of the problem your algorithm was solving instead of the impact of your algorithm. Another common mistake was to highlight a potential application of your algorithm instead of an actual verifiable impact that can be backed by citations. Also claiming an impact without presenting specific citation to back your claim. Depending on your answer you could lose up to 15 points.

Important: Please make sure you take my comments into account for your presentation. At the very least make sure you do not get points docked off for the same mistake again.

Delay in HW 6 pickup

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

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 , a1)
    If n = 2 return  ( a1 > a2, min(a1,a2); max(a1,a2))
    aL = a1,..., an/2      aR = an/2+1,..., an
    (cL, aL) = MergeSortCount(aL, n/2)
    (cR, aR) = MergeSortCount(aR, n/2)
    (c, a) = MERGE-COUNT(aL,aR)
    return (c+cL+cR,a)
```

$T(2) = c$

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

O(n log n) time

O(n)

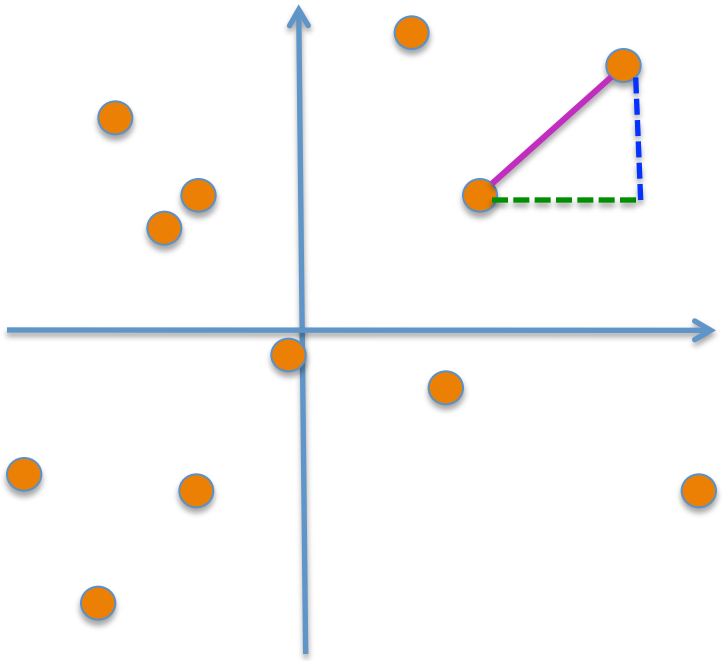
Counts #crossing-inversions+ MERGE

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?

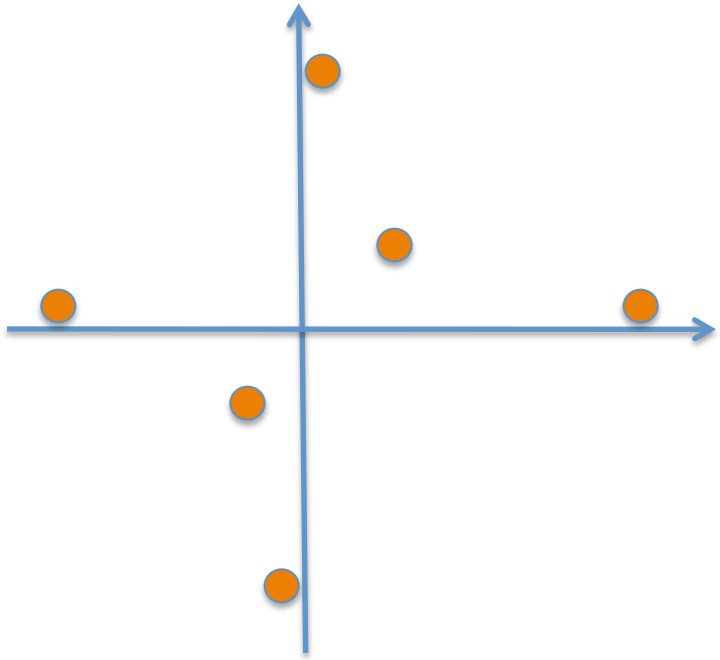
1-D problem in time $O(n \log n)$?

Sorting to rescue in 2-D?

Pick pairs of points closest in x co-ordinate

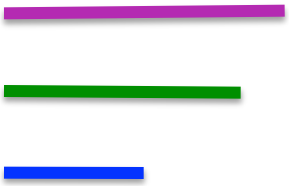
Pick pairs of points closest in y co-ordinate

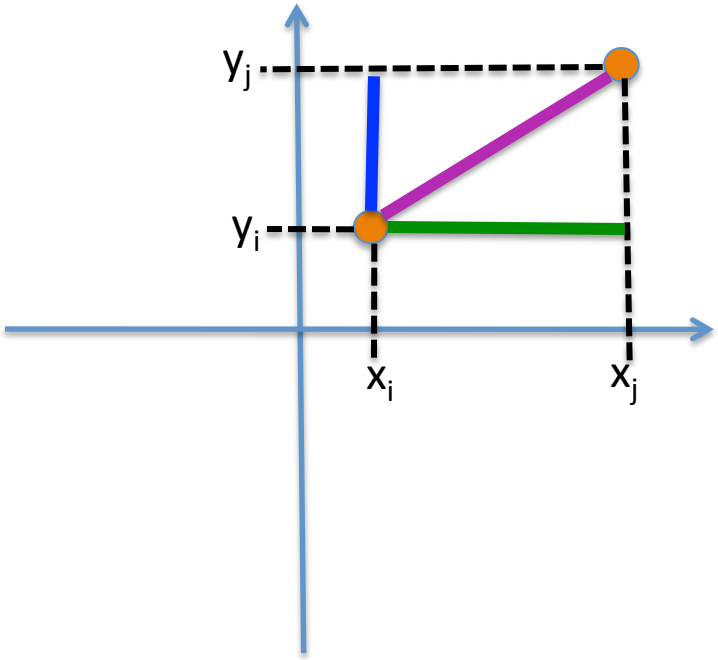
Choose the better of the two



A property of Euclidean distance

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



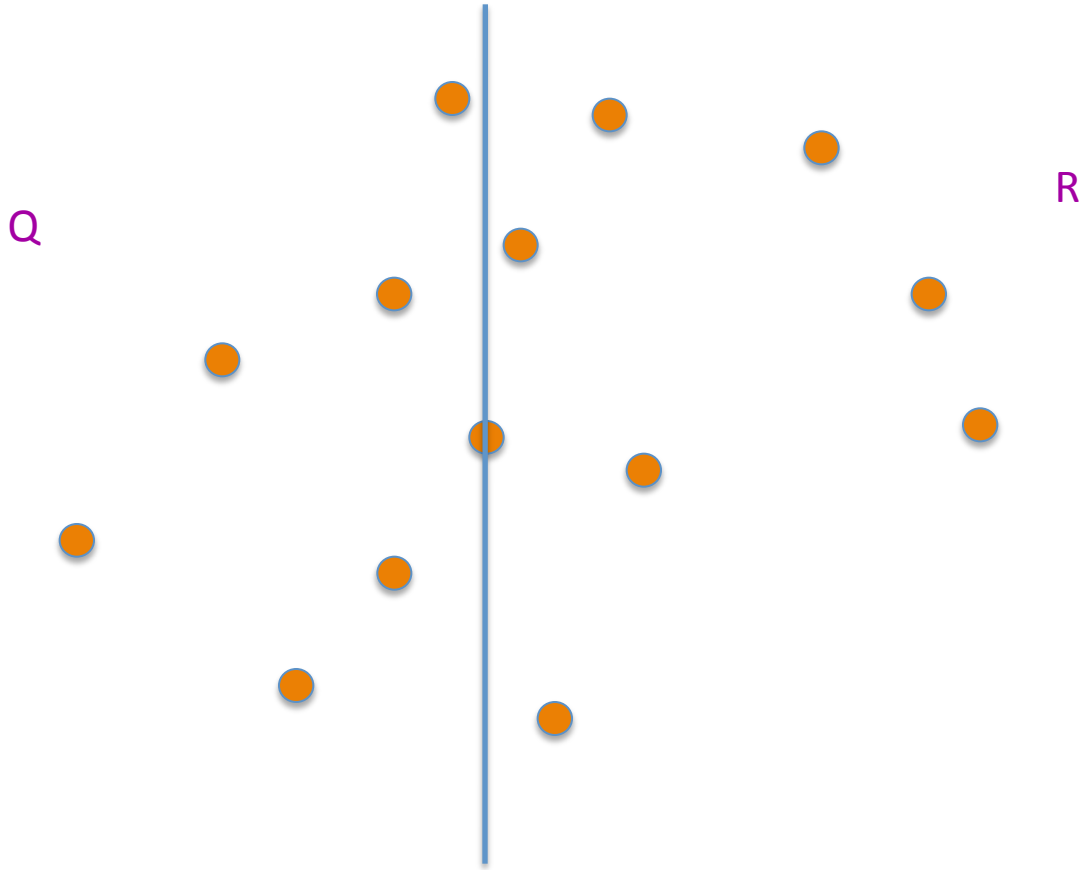


The distance is larger than the **x** or **y**-coord difference

Rest of Today's agenda

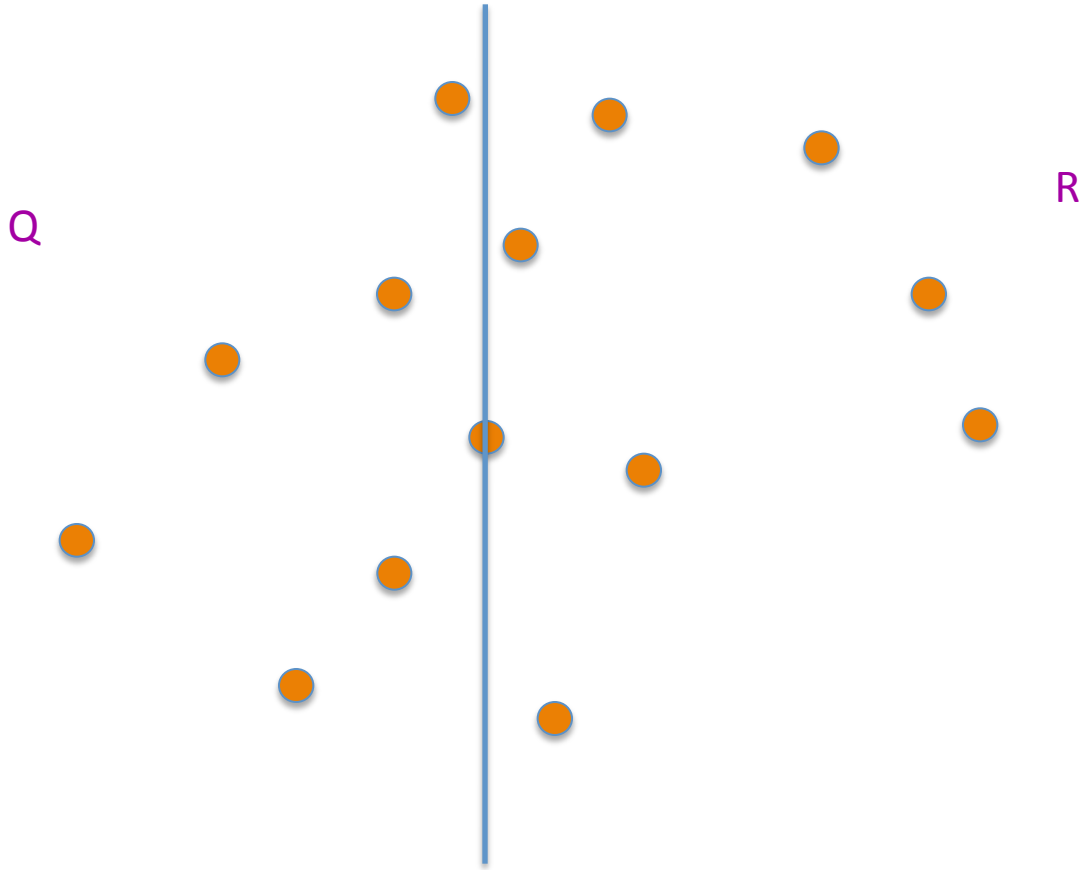
Divide and Conquer based algorithm

Dividing up P



First n/2 points according to the x-coord

Recursively find closest pairs



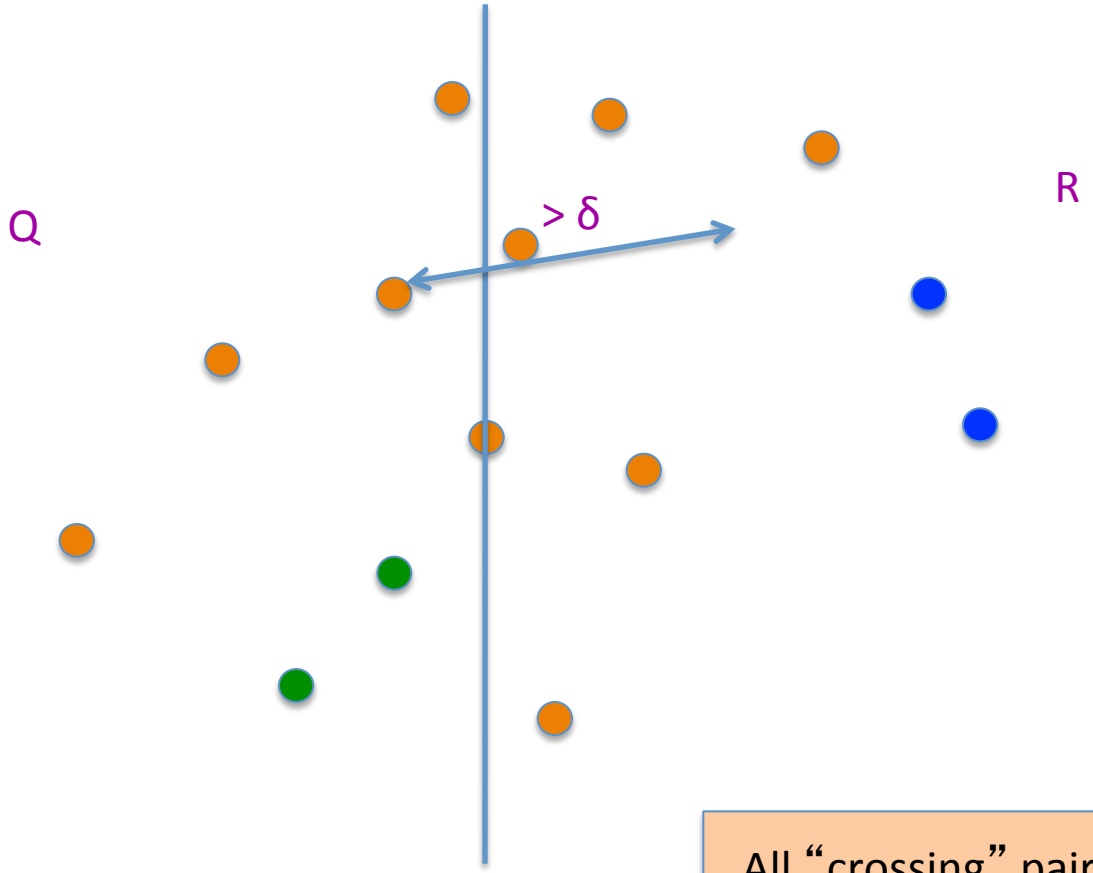
δ = min (**blue**, **green**)

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

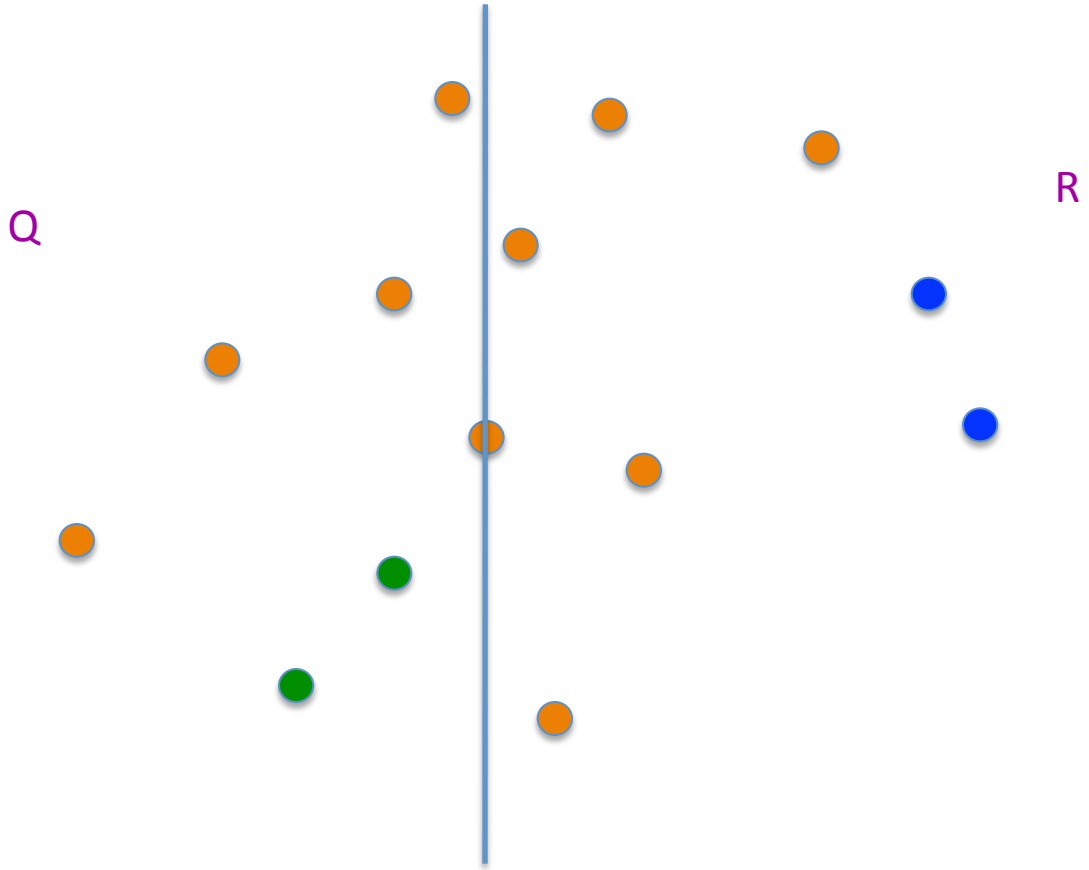
An easy case



All "crossing" pairs have distance > δ

δ = min (**blue**, **green**)

Life is not so easy though



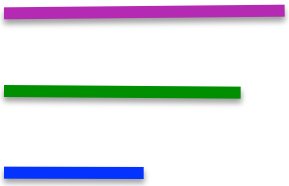
δ = min (**blue**, **green**)

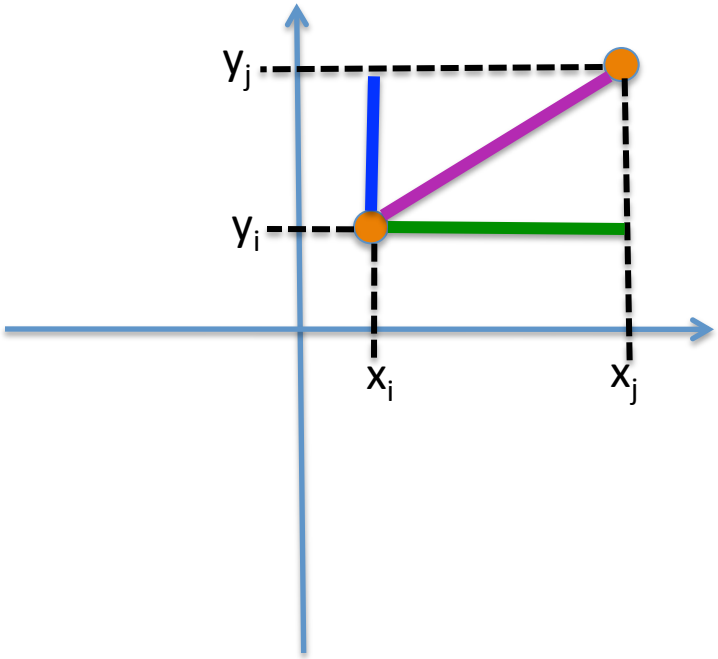
Rest of Today's agenda

Divide and Conquer based algorithm

Euclid to the rescue (?)

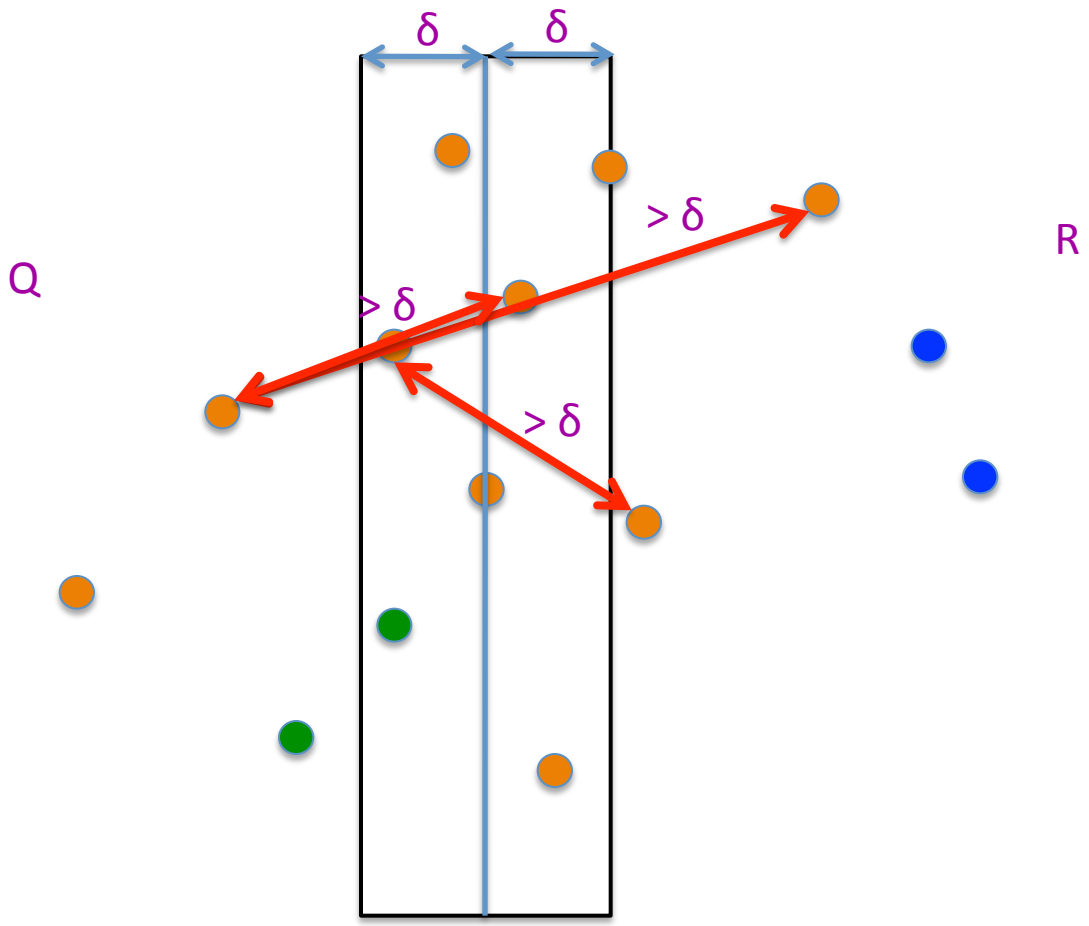
$$d(p_i,p_j) = ((x_i-x_j)^2+(y_i-y_j)^2)^{1/2}$$





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

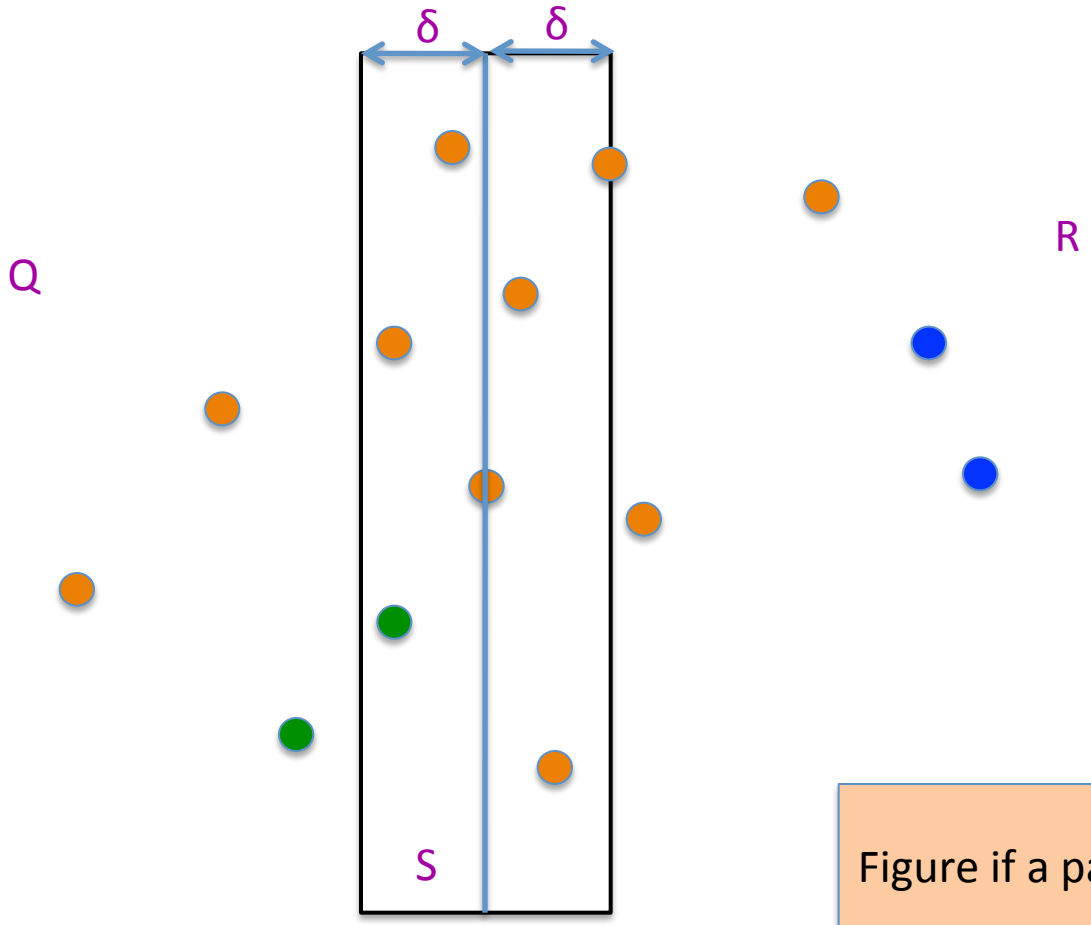


Figure if a pair in S has distance < δ

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)

δ = 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