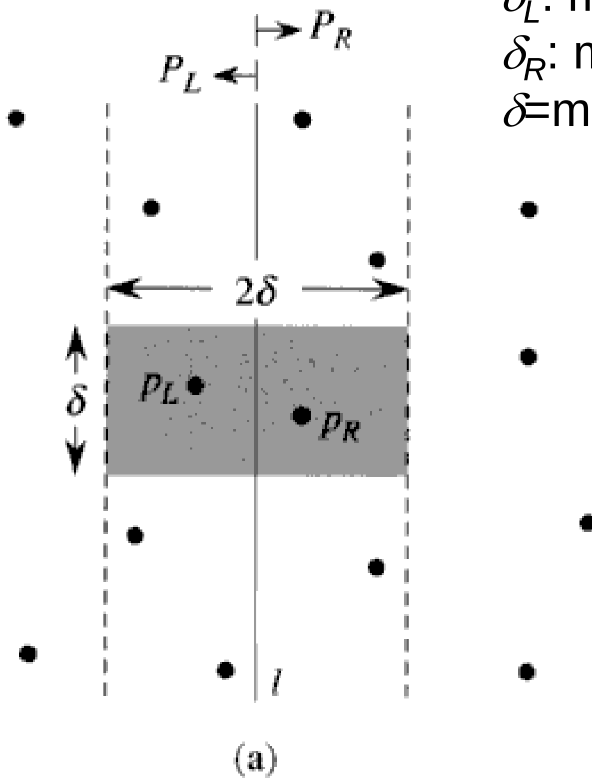
Divide-and-conquer: closest pair (Chap.33)

- Given a set of points, find the closest pair (measured in Euclidean distance)
- Brute-force method: $O(n^2)$.
- Divide-and-conquer method:
  - Want to be lower than $O(n^2)$, so expect $O(n\lg n)$.
  - Need $T(n)=2T(n/2)+O(n)$.
- How?
  - Divide : into two subsets (according to x-coordinate)
  - Conquer: recursively on each half.
  - Combine: select closer pair of the above.

what left? One point from the left half and the other from the right may have closer distance.

δ_L: minimum distance of P_L
δ_R: minimum distance of P_R
δ=min{δ_L, δ_R}.



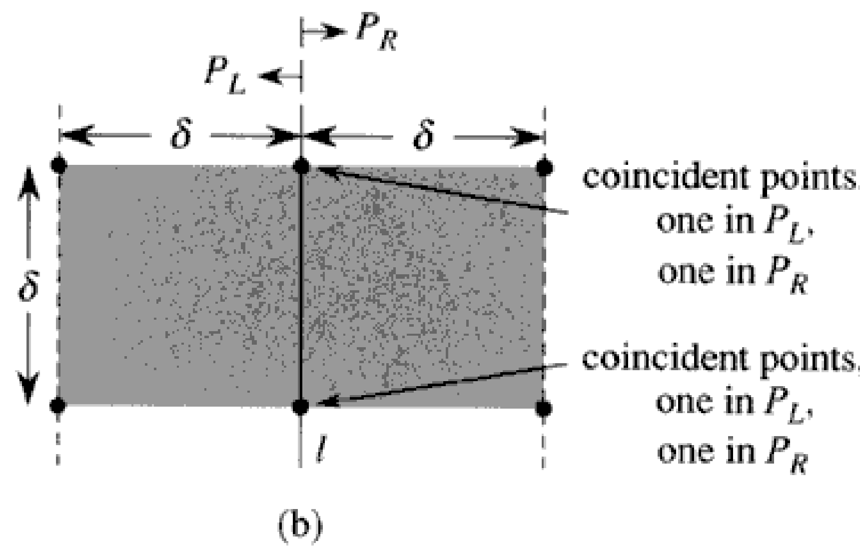


Figure 33.11 Key concepts in the proof that the closest-pair algorithm needs to check only 7 points following each point in the array Y'. **(a)** If $p_L \in P_L$ and $p_R \in P_R$ are less than δ units apart, they must reside within a $\delta \times 2\delta$ rectangle centered at line l. **(b)** How 4 points that are pairwise at least δ units apart can all reside within a $\delta \times \delta$ square. On the left are 4 points in P_L, and on the right are 4 points in P_R. There can be 8 points in the $\delta \times 2\delta$ rectangle if the points shown on line l are actually pairs of coincident points with one point in P_L and one in P_R.

Divide and conquer

- Divide : into two subsets (according to x-coordinate) : $P_L <= l <= P_R$ (O(n))
- Conquer: recursively on each half.
  - Get δ_L, δ_R
  - 2T(n/2).
- Combine:
  - select closer pair of the above. $\delta = \min\{\delta_L, \delta_R)$, O(1)
  - Find the smaller pairs, one $\in P_L$ and the other $\in P_R$
    - Creat an array Y' of points within 2δ vertical strip, sorted by y-coor. O($n$lg$n$) or O(n).
    - for each point in Y', compare it with its following 7 points. O(7n).
      - *If a new smaller δ'* is found, then it is new δ and the new pair is found.

Sort points according to coordinates

- Sort points by x-coordinates will simplify the division.
- Sorting by y-coordinates will simplify the computation of distances of cross points.
- sorting in recursive function will not work
  - $O(n\lg n)$, so total cost: $O(n \lg^2 n)$
- Instead, pre-sort all the points, then the half division will keep the points sorted
  - The opposite of merge sort.

CLOSEST_PAIR(P, X, Y)

- P: set of points, X: sorted by x-coordinate, Y: sorted by y-coordinate
- Divide P into P_L and P_R, X into X_L and X_R, Y into Y_L and Y_R,
  - δ_1=CLOSET-PAIR(P_L,X_L,Y_L) and //T(n/2)
  - δ_2=CLOSET-PAIR(P_R,X_R,Y_R) //T(n/2)
- Combine:
  - $\delta=\min(\delta_1, \delta_2)$
  - compute the minimum distance between the points $p_l \in P_L$ and $p_r \in P_R$. // $O(n)$.
    - Form Y′, which is the points of Y within 2δ-wide vertical strip.
    - For each point p in Y′, 7 following points for comparison.

CLOSEST-PAIR

- Sort X (or X_L, X_R) and Y (or Y_L,Y_R):
  - Pre-sort first,
  - Cut X to get X_L and X_R, naturally sorted.
    - $P_L = X_L$, $P_R = X_R$
  - From Y, get sorted Y_L and Y_R by the algorithm: (p.961)
    - Length[Y_L]=length[Y_R]=0
    - For i=1 to length[Y]
      - If(Y[i]$\in P_L$)then
        - » Length[Y_L]++, Y_L[Length[Y_L]]=Y[i]
      - Else
        - » Length[Y_R]++, Y_R[Length[Y_R]]=Y[i]

Question: Y[i]$\in P_L$? P_L is defined by line=x, so Y[i].x$\leq$x .

Any further question?

In summary

- $T(n) = $ - O(1), if $n <= 3$.
    - $2T(n/2)+O(n\lg n) \rightarrow O(n\lg^2 n)$
- $T'(n)=O(n\lg n)+T(n)$
    - $T(n)=2T(n/2)+O(n)$
    - So $O(n\lg n)+O(n\lg n)=O(n\lg n)$.

Improvements: Comparing 5 not 7?
Does not pre-sort Y?
Different distance definition?
Three dimensions?