Closest Pair

- Problem definition:
  - We are given n points $p_i=(x_i, y_i)$ $i = 1, 2,\ldots, n$
  - Which two points are closest?
    - We wish to find the i and j values that minimize the Euclidean distance:

$$\sqrt{\left(x_i - x_j\right)^2 + (y_i - y_j)^2}$$

  - Brute force: compute distances for all $n(n-1)/2$ pairs and find the minimum.
    - This algorithm runs in time $Q(n^2)$.

Closest Pair by Divide and Conquer

- Divide:
  - Sort halves by x-coordinate.
  - Find vertical line splitting points in half.
- Conquer:
  - Recursively find closest pairs in each half.
- Combine:
  - Check vertices near the border to see if any pair straddling the border is closer together than the minimum seen so far.
- Our goal:
  - Get the overhead down to $Q(n)$ so that the total run time is $Q(n \log n)$.

Closest Pair Implementation Details

- Input to the algorithm will be a subset of points *P* sorted with respect to the *x* coordinate.
- Initially, *P* = all points, and we pay *Q(n* log *n)* to sort them before making the first call to the recursive subroutine.
- Given the sorted points, it is easy to find the dividing line.
  - Let P_L be points to the left of the dividing line.
  - Let P_R be points to the right of the dividing line.

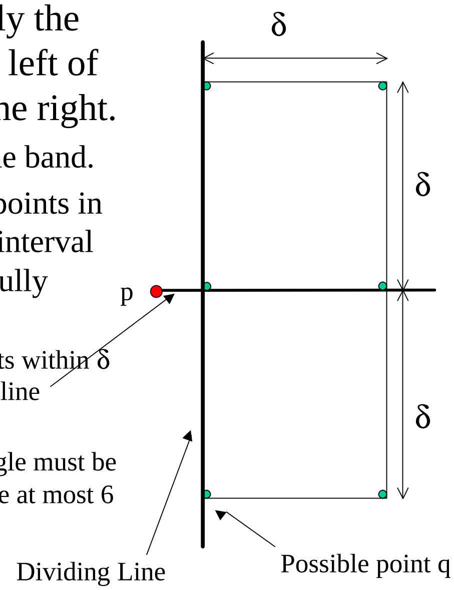
Closest Pair Implementation Details (Cont.)

- Recursively:
  - Find closest pair in P_L: Let d_L be their separation distance
  - Similarly find closest pair in P_R, separation distance d_R.

  - Clever observation: If the closest pair straddles the dividing line, then each point of the pair must be within $d = min\{d_L, d_R\}$ of the dividing line.
    - This will usually specify a fairly narrow band for our “straddling” search.

Closest Pair Implementation Details (Cont.)

– Suppose p and q are possibly the closest points, with p to the left of the dividing line and q on the right.
  - q cannot be on the right of the band.
  - Also, if p = (x, y) then only points in P_R with y coordinates in the interval [y - δ, y + δ] can be successfully paired with p.
    – So, we need only look at points within δ above and below a horizontal line through p.
    – Since the points in this rectangle must be separated by at least δ we have at most 6 points to investigate.



Closest Pair Pseudocode (page 1)

- Let P be a global array storing all the points with P_R and P_L defined as described earlier.
- Let `QL` be the subset of points from P_L which are at most δ (`delta` in the code) to the left of the dividing line.
- Let `QR` be the subset of points from P_R which are at most δ to the right of the dividing line.

```
//-------- main ---------
// P contains all the points
sort P by x-coordinate;
return closest_pair(1, n);
```

Closest Pair Pseudocode (page 2)

```
function delta_m(QL,QR,delta)
   // Are there two points p in QL, q in QR such that
   // d(p,q)<=delta? Return closest such pair.
   // Assume QL and QR are sorted by the y coordinate.
   j := 1;   dm := delta;
   for i := 1 to size(QL) do
       p := QL[i];
       // find the bottom-most candidate from QR
       while (j <= n and QR[j].y < p.y-delta) do
              j := j+1;
       // check all candidates from QR starting with j
       k := j;
       while (k <= n and QR[k].y <= p.y + delta) do
              dm := min(dm, d(p, QR[k]));
              k := k+1;
   return dm;
```

Closest Pair Pseudocode (page 3)

```
function select_candidates(l,r,delta,midx)
   // From P[l..r] select all points which are
   // at most delta from midx line
   create empty array Q;
   for i := l to r do
       if (abs(P[i].x - midx) <= delta)
              add P[i] to Q;
   return Q;
```

Closest Pair Pseudocode (page 4)

```
function closest_pair(l,r)
  // Find the closest pair in P[l..r] (sorted by x-coordinate)
  if size(P) < 2 then return infinity;
  mid := (l + r)/2; midx := P[mid].x;
  dl := closest_pair(l, mid);
  dr := closest_pair(mid + 1, r);
  // Side effect: P[l..mid] and P[mid+1..r] are now wrt the y-coordinate
  delta := min(dl, dr);
  QL := select_candidates(l, mid, delta, midx);
  QR := select_candidates(mid + 1, r, delta, midx);
  dm := delta_m(QL, QR, delta);
  // use merge to make P[l..r] sorted by y-coordinate
  merge(l, mid, r);
  return min(dm, dl, dr);
```

Closest Pair Analysis

– Let $T(n)$ be the time required to solve the problem for n points:
  - Divide: *Q*(*1*)
  - Conquer: *2T* (*n/2*)
  - Combine: *Q*(*n*)

– The precise form of the recurrence is:

$$T(n) = T\left(\lceil n/2 \rceil\right) + T\left(\lfloor n/2 \rfloor\right) + \Theta(n)$$

which we can approximate by:

$$T(n) = 2T(n/2) + \Theta(n)$$

– Solution: *Q(n* log *n).*