

## Fast Closest-Pair Algorithm

This handout gives pseudocode for the  $\Theta(n \log n)$  closest-pair algorithm in the form I presented in class. I'm pretty sure the pseudocode is correct, but I make no iron-clad guarantees – if something looks wrong to you, think before you implement!

Keep in mind that the arrays `ptsByX` and `ptsByY` in the algorithm (a) are assumed to be sorted by  $x$  and  $y$  coordinate respectively and (b) are actually *references* to a common set of points, *not* two distinct point sets.  $n$  is the number of points.

```

CLOSESTPAIR(ptsByX, ptsByY, n)
  if (n = 1) return  $\infty$ 
  if (n = 2) return distance(ptsByX[0], ptsByX[1])

  // Divide into two subproblems
  mid  $\leftarrow$   $\lceil n/2 \rceil - 1$ 
  copy ptsByX[0...mid] into new array XL in x order.
  copy ptsByX[mid+1...n-1] into new array XR in x order.

  copy ptsByY into arrays YL and YR in y order, s.t.
    XL and YL refer to same points, as do XR and YR.

  // Conquer
  distL  $\leftarrow$  CLOSESTPAIR(XL, YL,  $\lceil n/2 \rceil$ )
  distR  $\leftarrow$  CLOSESTPAIR(XR, YR,  $\lceil n/2 \rceil$ )

  // Combine (was a separate procedure in class)
  midPoint  $\leftarrow$  ptsByX[mid]
  lrDist  $\leftarrow$  min(distL, distR)
  Construct array yStrip, in increasing y order,
    of all points p in ptsByY s.t.  $|p.x - \text{midPoint}.x| < \text{lrDist}$ 

  minDist  $\leftarrow$  lrDist
  for ( $j \leftarrow 0$ ;  $j \leq \text{yStrip.length} - 2$ ;  $j++$ ) {
     $k \leftarrow j + 1$ 
    while ( $k \leq \text{yStrip.length} - 1$  and  $\text{yStrip}[k].y - \text{yStrip}[j].y < \text{lrDist}$ ) {
       $d \leftarrow$  distance( $\text{yStrip}[j]$ ,  $\text{yStrip}[k]$ )
      minDist  $\leftarrow$  min(minDist,  $d$ )
       $k++$ 
    }
  }
  return minDist

```