8 March 2021 — Closest Pair of Points (§5.4)

Announcements

1. Optional recitation Thurs 3/11 4pm EST
   Vaishnavi Gupta, see Ed post for Zoom link.
   Topic: dynamic programming
2. Prelim will be Thurs 3/18, online, open-book, open-notes.
   No homework that week.
   Review sheet will be posted.
   Some TFs will offer review sessions.
   Practice problems will be released.
   If you need to take a make-up prelim start an Ed post (private to instructors).
3. No lecture this Weds, 3/10. (Wellness Day).

Closest Pair of Points:

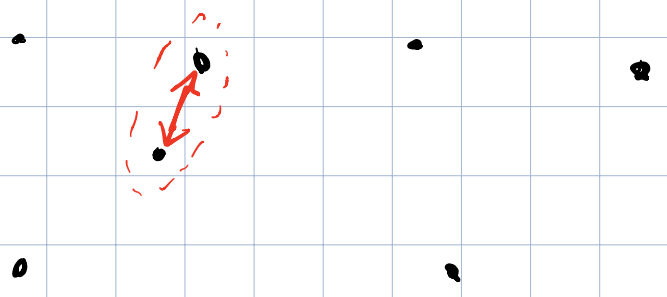
Given n points in the Euclidean plane

(x_1, y_1) (x_2, y_2) ... (x_n, y_n)

define distance in usual way:

$$\| \vec{x} - \vec{x}' \|_2 = \sqrt{(x-x')^2 + (y-y')^2}$$

Find a pair of points whose distance is the minimum among all pairs.



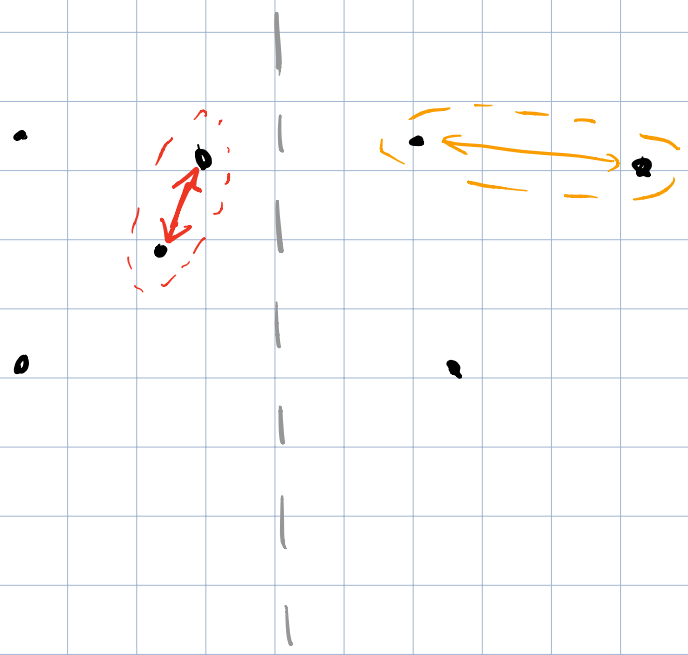
Naive algorithm computes $O(n^2)$ distances, looping over all pairs of points, hence takes $O(n^2)$ time.

We'll see an algorithm running in time $O(n \log n)$.

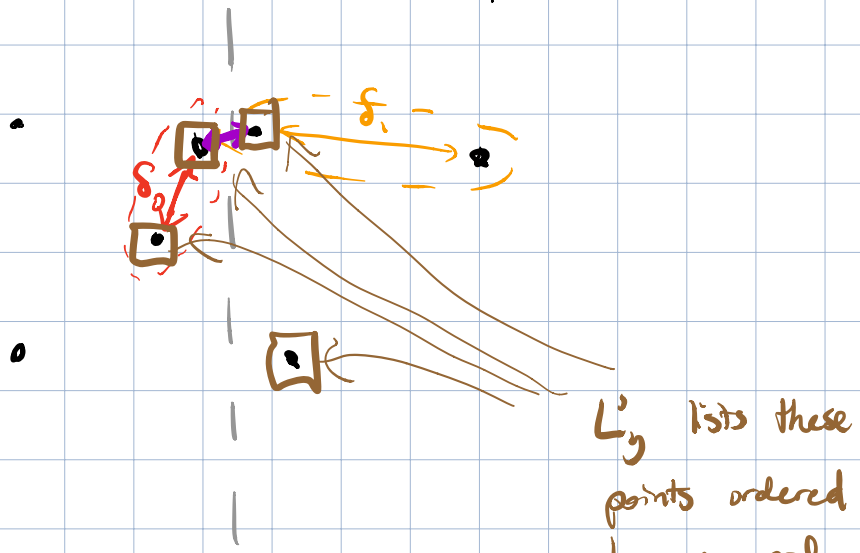
Step 1. Sort the points by x-coord and by y-coord, producing two sorted lists L_x and L_y.

Step 2. Split L_x into two equal halves with $\frac{n}{2}$ points in each. (If n is odd, $\lfloor \frac{n}{2} \rfloor$ points in one half, $\lceil \frac{n}{2} \rceil$ points in other half.)

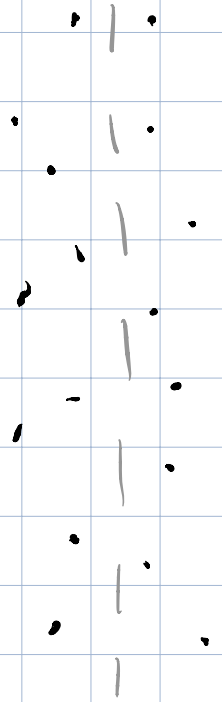
Step 3. Recursively solve the two subproblems.



WATCH OUT! The better of the two solutions on the subproblems may not be optimal for the full point set.



Step 4. Check for pairs that cross the midline.



Testing every pair of points near the midline might still be $O(n^2)$ pairs. How to save work in this step?

Key observation 1. Let δ_0 be dist of closest pair on left, δ_1 be dist of closest pair on right.

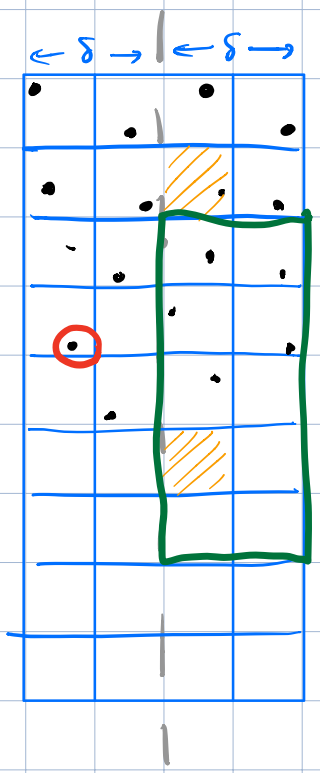
$$\delta = \min\{\delta_0, \delta_1\}.$$

We only need to test dist between a point on left, point on right if both are within δ of the mid-line.

Step 4a. Compute δ_0, δ_1, $\delta = \min\{\delta_0, \delta_1\}$.

Step 4b. Let L'_y = the elements of L_y, filtering out those whose distance from midline exceeds δ, keeping the rest in order of y-coord.

Key observation 2. For each point in L'_y if it has a neighbor at distance $< \delta$ its index in the list L'_y must be near that neighbor's index.



Grid cells have side length $\delta/2$, hence diameter $\sqrt{2} \cdot \frac{\delta}{2} < \delta$.

Each grid cell is on one side of the midline

$\Longrightarrow$ no grid cell contains more than one point.

For each point, we only need to look for neighbors in its grid row and the two preceding and following rows.

≤ 20 grid cells in total.

(10 before, 10 after).

Step 4c. For each element of L'_y calculate distance to the 10 preceding and 10 following elements of L'_y.

* If you were implementing this algorithm when testing for near neighbors of (x, y) you would only look at L'_y elements whose y-coord is in $[y-\delta, y+\delta]$. There are at most 20, but maybe fewer

Step 5. Return min of δ or the smallest distance found in step 4c.

Running time analysis.

building L'_y

$$T(n) = \underbrace{O(n \log n)}_{\text{Initial sorting by } x,y} + 2 \cdot T\left(\frac{n}{2}\right) + \overbrace{O(n)}^{} + \underbrace{O(n)}_{\text{checking pairs in } L'_y}$$

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n \log n)$$

$$\Longrightarrow \quad T(n) = O(n \log^2 n).$$

But actually, we can improve to $O(n \log n)$.

Conserve on time spent sorting by doing it only once, when preprocessing.

RecClosestPair (L_x, L_y):

// L_x and L_y are the same list of points, sorted by x-coord and by y-coord.

Do steps 2-5 of the algorithm written above. When passing the left and right subproblems to the recursive call, form (in order $O(n)$ time)

L_{x0}, L_{y0} = {points left of midline, sorted by x and y}

L_{x1}, L_{y1} = {points right of midline sorted by x and y}

$$T(n) = O(n) + 2T\left(\frac{n}{2}\right)$$

$$\Rightarrow T(n) = O(n \log n)$$