Closest Pair of Points

Input: \( n \) points \((x_i, y_i)\) on the \((2-D)\) plane.
Output: (the distance between) the closest pair of points.

\(O(n^2)\) algorithm:
\[
\text{ans} = +\infty \\
\text{for } i = 1 \text{ to } n \\
\quad \text{for } j = 1 \text{ to } n \\
\quad \quad \text{dis}(i, j) = \sqrt{(x_i-x_j)^2 + (y_i-y_j)^2} \\
\quad \quad \text{ans} = \min(\text{ans}, \text{dis}(i, j)) \\
\quad \text{end for} \\
\text{end for} \\
\text{return } \text{ans}.
\]

\(O(n \log n)\) algorithm via divide and conquer.

1. Split the points into two sets \(L\) and \(R\) of same size.
   Using a line \(x = l\).

2. Find the closest pair in \(L\) recursively.
   Find the closest pair in \(R\) recursively.
   Let \(s\) denote the minimum distance so far.

3. Consider a narrow band near the boundary with width \(2 \cdot s\).
   \(l-s \leq x \leq l+s\)
   Sort all points in this band by \(y\)-axis.
   For every point \(p\) in this sorted list,
   compute the distance between \(p\) and each of
   the next 11 point \(q\).
   \(s = \min(s, \text{dis}(p, q))\).

4. Return \(s\).
Correctness: If \( |L| < 10 \), then brute-force. \( O(10^2) = O(1) \).

We can assume the subproblems in \( L \) and \( R \) are solved correctly.

In the merge step, for two points \( p \in L \) and \( q \in R \) to be closer than \( s \) to each other, they must both be in the narrow band.

In fact, we can partition this band into \( \frac{s}{2} \) by \( \frac{s}{2} \) squares.

1. \( p \) and \( q \) cannot be more than \( 2 \) squares away from each other either vertically or horizontally.
   
   o.w. \( \text{dis}(p, q) \geq s \)

2. There can be at most one point in each "\( \square \"")
   
   o.w. \( \min \left\{ \text{dis}(i, j) \right\} \leq \frac{1}{2} \frac{s}{2} < s \).

Therefore, it is sufficient for \( p \) to consider the next 11 points (sorted by \( y \)-coordinate) in this band.

Runtime: To avoid repeated sorting, \( P = \) all input points

\[ O(n \log n) \]

Time: \( P_x = \) points in \( P \) sorted by \( x \) (we only sort once)

\( (\text{w.l.o.g.}, \text{we can assume all } x \text{ and } y \text{ coordinates are unique}) \)

\[ O(n) \]

1. Find the median \( \ell \) in \( P_x \) s.t.

\[ L = \{ i \in P : x_i \leq \ell \} \quad \text{and} \quad R = \{ i \in P : x_i > \ell \} \]

have roughly the same size.

2. Construct \( L_x \) (points in \( L \) sorted by \( x \)), \( R_x \).

\( L_y \) (points in \( L \) sorted by \( y \)), \( R_y \).

\[ O(n) \]

\( L_y \) can be obtained by iterating over points in \( P_y \) in order and select those in \( L \).

\[ 2 \cdot T \left( \frac{n}{2} \right) \]

3. \( S = \min (\text{closest-pair}(L_x, L_y), \text{closest-pair}(R_x, R_y)) \).

\( \uparrow \)

\( \text{Same set of points, but ordered differently.} \)

4. \( S_y = \) points in \( S \) sorted by \( y \)-coordinate.

\[ O(n) \]

\( S_y \) can be computed by iterating over \( P_y \) in order.

\[ O(n) \]

5. For every \( p \in S_y \), consider the next 11 points.

Overall: \( O(n \log n) + T(n) \)

\[ \Rightarrow T(n) = O(n \log n) \]
An alternative solution: Delaunay Triangulation (DT)

A triangulation that maximizes the minimum angle.

There are various other equivalent definitions.

E.g. there is no point inside any circumcircle of any triangle.

The following statements are true:

(source: Wikipedia) (the proof in beyond the scope of this course).

1. DT has $O(n)$ edges.
2. the edge between the closest pair of points is in DT.
3. DT can be computed in $O(n \log n)$ time.

$1 + 2 + 3$ gives an $O(n \log n)$ time for closest pair of points.

Planar MST:

Input: $n$ 2D points $(x_i, y_i)$

Output: An MST of the complete graph $G$

on these points where $C_e = \text{distance}(u,v)$, $e=(u,v)$

$G$ has $\Theta(n^2)$ edges.

If we run Kruskal or Prim directly $\Theta(E \log V) = \Theta(n^2 \log n)$.

Lemma: Planar MST $\leq$ DT.

(First compute DT) + (Prim or Kruskal on DT) gives an $O(n \log n)$ algorithm for planar MST.