Algorithm to find the closest pair of points

Vassos Hadzilacos and Sam Toueg

Let \( P \) be the set of coordinates of \( n \) points on the plane. For any pair of points \( p = (x, y) \) and \( q = (x', y') \), let \( d(p, q) = \sqrt{(x - x')^2 + (y - y')^2} \); i.e., \( d(p, q) \) is the Euclidean distance between \( p \) and \( q \).

\[
\text{ClosestPair}(P) \\
\text{▷ } P \text{ is a set of at least two points on the plane, given by their } x- \text{ and } y-\text{coordinates} \\
P_x := \text{the list of points in } P \text{ sorted by } x-\text{coordinate} \\
P_y := \text{the list of points in } P \text{ sorted by } y-\text{coordinate} \\
\text{return RCP}(P_x, P_y) \\
\]

\[
\text{RCP}(P_x, P_y) \\
\text{▷ } P_x \text{ and } P_y \text{ are lists of the same set of (at least two) points sorted by } x- \text{ and } y-\text{coordinate, respectively} \\
1 \text{ if } |P_x| \leq 3 \text{ then} \\
2 \quad \text{calculate all pairwise distances and return the closest pair} \\
3 \text{ else} \\
4 \quad L_x := \text{first half of } P_x; R_x := \text{second half of } P_x \\
5 \quad m := (\max x-\text{coordinate in } L_x + \min x-\text{coordinate in } R_x)/2 \\
6 \quad L_y := \text{sublist of } P_y \text{ of points in } L_x; R_y := \text{sublist of } P_y \text{ of points in } R_x \\
7 \quad (p_L, q_L) := \text{RCP}(L_x, L_y); (p_R, q_R) := \text{RCP}(R_x, R_y) \\
8 \quad \delta := \min(d(p_L, q_L), d(p_R, q_R)) \\
9 \quad \text{if } d(p_L, q_L) = \delta \text{ then } (p^*, q^*) := (p_L, q_L) \text{ else } (p^*, q^*) := (p_R, q_R) \\
10 \quad B := \text{sublist of } P_y \text{ of points whose } x-\text{coordinates are within } \delta \text{ of } m \\
11 \quad \text{for each } p \in B \text{ in order of appearance on } B \text{ do} \\
12 \quad \quad \text{for each } q \text{ of the (up to) next seven points after } p \text{ on } B \text{ do} \\
13 \quad \quad \quad \text{if } d(p, q) < d(p^*, q^*) \text{ then } (p^*, q^*) := (p, q) \\
14 \quad \text{return } (p^*, q^*) \\
\]

For the running time analysis, let \( T(n) \) be the running time of \( \text{RCP}(P_x, P_y) \), where \( n \) is the number of points on the list \( P_x \) (which is also the number of points on the list \( P_y \)). Assume that \( n \) is a power of 2.

We claim that \( T(n) \) satisfies the recurrence \( T(n) = 2T(n/2) + cn \), for some constant \( c \). This is because the algorithm calls itself twice on instances of half the size (see line 7), and requires \( \Theta(n) \) time to divide up the input and to combine the results of the two smaller instances into the result of the original instance. To see the latter point (i.e., that the algorithm requires only \( \Theta(n) \) time for the divide and combine steps), note that lines 4, 6, and 10 can be easily implemented in \( \Theta(n) \) time each. The loop in lines 11–13 takes \( \Theta(n) \) time since there are \( \Theta(n) \) points in \( B \) and each iteration takes \( \Theta(1) \) time (since the inner for loop is executed only seven times for each point \( p \in B \)). All other lines only require \( \Theta(1) \) time.

From the Master Theorem, we conclude that \( T(n) = \Theta(n \log n) \).

If the given set \( P \) contains \( n \) points, the main procedure \( \text{ClosestPair}(P) \) requires \( \Theta(n \log n) \) time to sort the points in \( P \) by \( x \)-coordinate and by \( y \)-coordinate in order to produce the lists \( P_x \) and \( P_y \), and an additional \( \Theta(n \log n) \) time for the call \( \text{RCP}(P_x, P_y) \). So the entire algorithm runs in \( \Theta(n \log n) \) time.