Closest Pair

input: 2D points $P = (p_1, p_2, \ldots, p_n)$ where each $p_i = (x_i, y_i) \in \mathbb{R}^2$

output: the minimum Euclidean distance $\delta = \min\{d(p_i, p_j) : 1 \le i < j \le n\}$ between any two of the points. Here $d((x_i, y_i), (x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. For $n \le 1$, the answer is ∞.

The obvious algorithm (compare all points) takes time $O(n^2)$. We want time $O(n \log n)$.

We'll assume the points are sorted by x-coordinate. If not, we can sort them in $O(n \log n)$ time in a pre-processing step. For simplicity, for now, we also assume that the x-coordinates are distinct. We use a divide-and-conquer approach, something like this:

closest-pair-1$(P = (p_1, p_2, \ldots, p_n))$ *($p_i = (x_i, y_i) \in \mathbb{R}^2$, with $x_1 < x_2 < \cdots < x_n$)*

1. if $n \le 1$: return ∞
2. if $n = 2$: return $d(p_1, p_2)$
3. let $m = \lfloor n/2 \rfloor$
4. let $L = (p_1, p_2, \ldots, p_m)$; let $R = (p_{m+1}, \ldots, p_n)$ *(partition P into left and right halves by x-coordinate)*
5. let $\delta_L =$ closest-pair-1(L); let $\delta_R =$ closest-pair-1(R)
6. let $\delta = \min(\delta_L, \delta_R)$
7. return δ **(WRONG! See below)**

The running time of this algorithm satisfies $T(n) = 2T(n/2) + O(n)$, so is $O(n \log n)$, as desired. But the algorithm is wrong, because it doesn't consider the pairs (p_i, p_j) such that $i \le m < j$. That is, $(p_i, p_j) \in L \times R$. To fix it, we need to consider those pairs. But among those pairs, it is enough to consider only those with $d(p_i, p_j) < \delta$ (as δ is defined in Step 6). (Why?)

Let $X = x_m$ be the median x-coordinate (separating L from R) and let $M = \{p_i : |x_i - X| \le \delta\}$ contain the points lying in a vertical strip of width 2δ centered at X. Any $p_i \in L \setminus M$ has distance at least δ from any point in R. Likewise any $p_i \in R \setminus M$ has distance at least δ from any point in L. So we only need to consider pairs $(p_i, p_j) \in M \times M$.

Further, within M, it is enough to compare each point p_i in M only to those points p_j above p_i in M such that $y_i \le y_j < y_i + \delta$. (Any other points above p_i in M has distance at least δ to p_i.) And there are at most eight such points:

Lemma 1. *Assume δ_L and δ_R are calculated correctly. For any point $p_i \in M$, there are at most 8 points p_j above p_i in M such that $y_i \le y_j \le y_i + \delta$.*

Proof. We use that *any $\delta \times \delta$ square contains at most 4 points in L and at most 4 points in R.* To prove this observation, imagine partitioning the square into its four $(\delta/2) \times (\delta/2)$ quadrants. Within each quadrant, every pair of points has distance at most $\sqrt{(\delta/2)^2 + (\delta/2)^2} = \delta/\sqrt{2} < \delta$. So R has at most one point in each quadrant (otherwise R would have a pair with distance less than $\delta \le \delta_R$, contradicting the correctness of δ_R). Likewise for L.

Now, consider any $p_i \in M$. Consider the two $\delta \times \delta$ squares $S_L = [X - \delta, X] \times [y_i, y_i + \delta]$ and $S_R = [X, X + \delta] \times [y_i, y_i + \delta]$. Any point p_j in M with $y_i \le y_j \le y_i + \delta$ must lie in either S_L or S_R. S_L contains no points from R, and (by the observation above) contains at most four points from L. So S_L contains at most four points from P. Likewise S_R contains at most four points from P. The lemma follows. □

Summarizing, it is enough to compare each point p_i in M with the points p_j in M such that $y_i \le y_j \le y_i + \delta$, and there are at most eight such points (per p_i). To fix the algorithm, we have it sort the points in M by y-coordinate, then have it compare each point p_i in M to the points p_j in M such that $y_i \le y_j \le y_i + \delta$:

closest-pair$(P = (p_1, p_2, \ldots, p_n))$ *($p_i = (x_i, y_i) \in \mathbb{R}^2$, with $x_1 < x_2 < \cdots < x_n$)*

1. if $n \le 1$: return ∞
2. if $n = 2$: return $d(p_1, p_2)$
3. let $m = \lfloor n/2 \rfloor$
4. let $L = (p_1, p_2, \ldots, p_m)$; let $R = (p_{m+1}, \ldots, p_n)$ *(partition P into left and right halves by x-coordinate)*
5. let $\delta_L =$ closest-pair(L); let $\delta_R =$ closest-pair(R)
6. let $\delta = \min(\delta_L, \delta_R)$
7. let $M = \{p_i : |x_i - x_m| \le \delta\}$ *(compute this in $O(n)$ time)*
8. let $\delta_M = \min\{d(p_i, p_j) : p_i, p_j \in M,\, y_i \le y_j \le y_j + \delta\}$. *(compute this as described below)*
9. return $\min(\delta, \delta_M)$

Compute δ_M efficiently as follows. Sort the points in M by y-coordinate, then enumerate the points p_i in M in order of increasing y-coordinate. For each such p_i, consider the points p_j in the list after p_i, such that $y_j \le y_i + \delta$. There will be at most eight of these for each p_i, so the total time (after sorting) is $O(n)$.

As described, this would give time $T(n) = 2T(n/2) + \Theta(n \log n)$, which gives $T(n) = \Theta(n \log^2 n)$.

To save a $\log n$ factor, instead of sorting the points in M by y-coordinate within each recursive call, do the following. Along with P, pass in a second list P' of the points in P, this one sorted by y-coordinate. From P', the sorted list of points in M can be computed in $O(n)$ time. For the recursive calls on L and R, let L' and R' denote the lists L and R sorted by y-coordinate. Compute each of these (from P' again) in time $O(n)$, then pass L' with L, and pass R' with R, to each recursive call.

The time to preprocess all the points to sort them by x and y coordinate (before any recursion) is $O(n \log n)$. The remaining time then satisfies $T(n) = 2T(n/2) + O(n)$, which gives $T(n) = O(n \log n)$.

External resources on Closest Pair

- CLRS Chapter 9.2; Dasgupta et al. Problem 2.32 (draft). Kleinberg & Tardos Chapter 5.4.