Divide and Conquer: Closest Pair of Points on the Plane[1]

1 Closest Pair of Points on the Plane

We look at a simple geometric problem: given n points on a plane, find the pair which is closest to each other. More precisely, the n points are described as their (x, y) coordinates; point p_i will have coordinates (x_i, y_i). The distance between two points p_i and p_j is defined as

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

One could also look at other distances such as $d(p_i, p_j) = \max(|x_i - x_j|, |y_i - y_j|)$ and $d(p_i, p_j) = |x_i - x_j| + |y_i - y_j|$. What we describe below works for both these as well.

<u>CLOSEST PAIR OF POINTS ON THE PLANE</u>
Input: n points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$.
Output: The pair p_i, p_j with smallest $d(p_i, p_j)$.
Size: The number of points, n.

Once again, as many of the examples before, there is a trivial $O(n^2)$ time algorithm: simply try all pairs and return the closest pair. This is the naive benchmark which we will try to beat using Divide-and-Conquer.

How should we divide this set of points into two halves? To do so, let us think whether there is a natural ordering of these points? A moment's thought leads us to two natural orderings: one sorted using their x-coordinates, and one using their y-coordinates. Let us use $P_x[1 : n]$ to denote the permutation of the n points such that[2] $\mathsf{xcoor}(P_x[i]) < \mathsf{xcoor}(P_x[j])$ for $i < j$. Similarly we define $P_y[1 : n]$. Getting these permutations from the input takes $O(n \log n)$ time.

Before moving further, we point out something which we will use later. Let $S \subseteq P$ be an arbitrary set of points of size s. Suppose we want the arrays $S_x[1 : s]$ and $S_y[1 : s]$ which are permutations of S ordered according to their xcoor's and ycoor's, respectively. If S is given as a "bit-array" with a 1 in position i if point $p_i \in S$, then to obtain S_x and S_y we don't need to sort again, but can obtain these from P_x and P_y. This is obtained by "masking" S with P_x; we traverse P_x from left-to-right and pick the point $p = P_x[i]$ if and only if $S[p]$ evaluates to 1. Note this is a $O(n)$ time procedure. This "dynamic sorting" was something we encountered in the Counting Inversions problem and is an useful thing to know. For more details, see UGP2, Problem 1(c). Let us now get back to our problem.

Given P_x, we can divide the set of points P into two halves as follows. Let $m = \lfloor n/2 \rfloor$ and $x^* := \mathsf{xcoor}(P_x[m])$ be the median of P_x. Define $Q_x := P_x[1 : m]$ and $R_x := P_x[m + 1 : n]$, and let us use Q and R to denote the set of these point. Figure 1 illustrates this.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

[2] Just for simplicity we assume no two points share xcoor or ycoor coordinates. Not really necessary, but let's assume anyway.

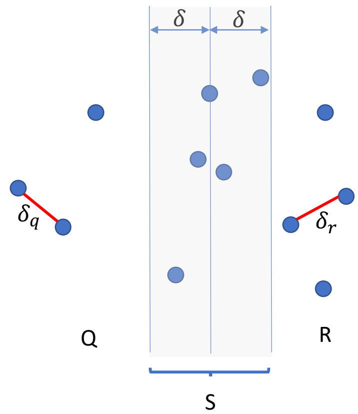

Figure 1: Closest pair in a plane

We recursively call the algorithm on the sets Q and R. Let (q_i, q_j) and (r_i, r_j) be the pairs returned. We will use[3] $\delta_q := d(q_i, q_j)$ and $\delta_r := d(r_i, r_j)$. Clearly these are candidate points for closest pair of points among P.

The other candidate pairs of P are precisely the *cross pairs*: (q_i, r_j) for $q_i \in Q$ and $r_j \in R$. Therefore, to conquer we need to find the nearest cross pair. Can we do this in time much better than $O(n^2)$? If you think for a little bit, this doesn't seem any easier at all – can we still get a win? Indeed we will, but we need to exploit the **geometry** of the problem. And this will form the bulk of the remainder of this lecture.

First let us note that we don't need to consider all pairs in $Q \times R$. Define $\delta := \min(\delta_q, \delta_r)$. Since we are looking for the closest pair of points, we don't need to look at cross-pairs which are more than δ apart.

Claim 1. Consider any point $q_i \in Q$ with $\mathsf{xcoor}(q_i) < x^* - \delta$. We don't need to consider any (q_i, r_j) point for $r_j \in R$ as a candidate. Similarly, for any point $r_j \in R$ with $\mathsf{xcoor}(r_j) > x^* + \delta$, we don't need to consider any (q_i, r_j) point for $q_i \in Q$ as a candidate.

Proof. Any candidate (q_i, r_j) we need to consider better have $d(q_i, r_j) \leq \delta$. But

$$d(q_i, r_j) \geq |\mathsf{xcoor}(q_i) - \mathsf{xcoor}(r_j)|$$

Therefore, if $\mathsf{xcoor}(q_i) < x^* - \delta$, and since $\mathsf{xcoor}(r_j) \geq x^*$ for all $r_j \in R$, we get $|\mathsf{xcoor}(q_i) - \mathsf{xcoor}(r_j)| > \delta$. Thus, we can rule out (q_i, r_j) for all $r_j \in R$. The other statement follows analogously. $\square$

Motivated by the above, let us define $Q' := \{q_i \in Q : \mathsf{xcoor}(q_i) \geq x^* - \delta\}$ and $R' := \{r_j \in R : \mathsf{xcoor}(r_j) \leq x^* + \delta\}$. That is $S := Q' \cup R'$ lies in the band illustrated in Figure 1. To summarize, we only need to look for cross-pairs[4] in $S \times S$.

Have we made progress? Note that all of Q could be sitting in Q' and all of R could be sitting in R', and it may feel we haven't moved much. But note, if that is the case, then all points are in a "narrow band". We will soon see why that is important.

[3] We haven't discussed the base case: if $n = 2$, then we return that pair; if $n = 1$, then we actually return $\perp$ and the corresponding $\delta = \infty$.

[4] Actually, we can restrict to $Q' \times R'$, but searching more widely doesn't hurt and makes exposition easier.

Let us start with a "naive" way of going over all cross-pairs in $S \times S$. Start with a point $q \in S$. Go over all *other* points $r \in S$ evaluating $d(q, r)$ as we go and store the minimum. Then repeat this for all $q \in S$ and take the smallest of all these minimums. Again, to make sure we are on the same page, given that in the worst case $S = P$, as stated this naive algorithm is still $O(n^2)$.

Once again, we want to use the observation that pairs which are $> \delta$ far needn't be considered. In particular, if the *y-coordinates* of two points are more than δ, we don't need to consider that pair. So, for any fixed $q \in S$, we could restrict our search only on the points $r \in S$ with $|\mathsf{ycoor}(r) - \mathsf{ycoor}(q)| \leq \delta$. We can do this restriction easily using the *sorted array* S_y.

To formalize this, first note that, as mentioned before, we can use P_y (the sorted array of the original points) to find the array S_y which is the points in S sorted according to the ycoor's. To find the closest cross-pair, we consider the points in the increasing ycoor order; for a point $q \in S$ we look at the other points $r \in S$ subsequent to it in S_y having $\mathsf{ycoor}(r) \leq \mathsf{ycoor}(q) + \delta$, store the distances $d(q, r)$, and return the minimum. The following piece of pseudocode formalizes this.

procedure ClosestCrossPairs(S, δ):
    ▷ *Returns cross pair $(q, r) \in S \times S$ with $d(q, r) < \delta$ and smallest among them.*
    ▷ *If no $d(q, r) < \delta$, then returns $\perp$.*
    Use P_y to compute S_y i.e. S sorted according to ycoor. ▷ *Can be done in $O(n)$ time.*
    $t \leftarrow \perp$ ▷ *t is a tuple which will contain the closest cross pair*
    $\mathsf{dmin} \leftarrow \delta$ ▷ *dmin is the current min init to δ*
    for $1 \leq i \leq |S|$ **do**:
        $\mathsf{p_{cur}} \leftarrow S_y[i]$.
        ▷ *Next, check if there is a point $\mathsf{q_{cur}}$ such that its distance to $\mathsf{p_{cur}}$ is $< \mathsf{dmin}$.*
        ▷ *If so, then we define this pair to be t and define this distance to be the new dmin.*
        ▷ ***Crucially**, we don't need to check points which are $\geq \delta$ away in the y-coordinate.*
        $j \leftarrow 1$; $\mathsf{q_{cur}} \leftarrow S_y[i + j]$.
        while $\mathsf{ycoor}(\mathsf{q_{cur}}) < \mathsf{ycoor}(\mathsf{p_{cur}}) + \delta$ **do**:
            if $d(\mathsf{p_{cur}}, \mathsf{q_{cur}}) < \mathsf{dmin}$ **then**:▷ *Modify dmin and t.*
                $\mathsf{dmin} \leftarrow d(\mathsf{p_{cur}}, \mathsf{q_{cur}})$;
                $t \leftarrow (\mathsf{p_{cur}}, \mathsf{q_{cur}})$
            $j \leftarrow j + 1$; $\mathsf{q_{cur}} \leftarrow S_y[i + j]$. ▷ *Move to the next point in S_y.*
    return t ▷ *Could be $\perp$ as well.*

Remark: *One may wonder that we are not returning cross-pairs as we could return q, r both in Q'. However, for any pair (q, r) returned, we have $d(q, r) < \delta$; since $\delta = \min(\delta_q, \delta_r)$, this pair can't lie on the same side.*

Armed with the above "conquering" step, we can state the full algorithm.

procedure ClosestPair(P):
  ▷ *We assume $n = |P|$.*
  ▷ *We assume arrays $P_x[1:n]$ and $P_y[1:n]$ which are* xcoor *and* ycoor*-sorted P.*
  if $n \in \{1, 2\}$ **then**:
    If $n = 1$ **return** $\perp$; else **return** P.
  $m \leftarrow \lfloor n/2 \rfloor$
  Q be the points in $P_x[1:m]$
  R be the points in $P_x[m+1:n]$
  $(q_1, q_2) \leftarrow$ ClosestPair(Q); $\delta_q \leftarrow d(q_1, q_2)$.
  $(r_1, r_2) \leftarrow$ ClosestPair(R); $\delta_r \leftarrow d(r_1, r_2)$.
  $\delta \leftarrow \min(\delta_q, \delta_r)$
  $x^* \leftarrow \mathsf{xcoor}(P_x[m])$.
  Compute $S \leftarrow \{p_i : x^* - \delta \leq \mathsf{xcoor}(p_i) \leq x^* + \delta\}$. ▷ *Store as indicator bit-array*
  ▷ *All cross-pairs worthy of consideration lie in S*
  $(s_1, s_2) \leftarrow$ ClosestCrossPair(S, δ)
  return Best of (q_1, q_2), (r_1, r_2) and (s_1, s_2).

How long does the above algorithm take? It really depends on how long ClosestCrossPair(S) takes. We now focus on the running time of this algorithm. Note $|S|$ could be as large as $\Theta(n)$. The inner while loop, a priori, can take $O(|S|)$ time, and thus along with the for-loop, the above seems to take $O(n^2)$ time. Doesn't seem we have gained anything. Next comes the real geometric help.

Lemma 1. Fix any point $p \in S$. Then there are at most 8 points $q \in S$ such that $\mathsf{ycoor}(p) \leq \mathsf{ycoor}(q) < \mathsf{ycoor}(p) + \delta$.

Proof. Suppose not. Suppose there are at least 9 such points. Concretely, define $S_p := \{q \in S : \mathsf{ycoor}(p) \leq \mathsf{ycoor}(q) < \mathsf{ycoor}(q) + \delta\}$, and suppose for the sake of contradiction $|S_p| \geq 9$. Since these points of S_p either lie in Q or R, we are *guaranteed* at least 5 points in one side. Without loss of generality, suppose $|S_p \cap R| \geq 5$. See Figure 2 for an illustration where the points marked are the points in S_p.

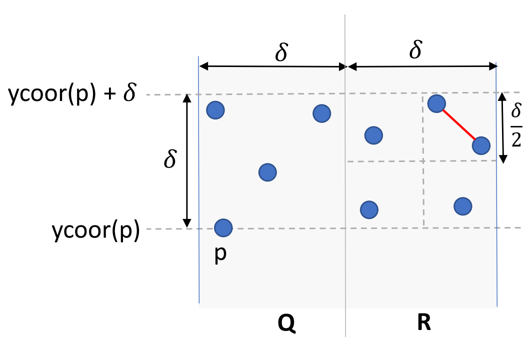

Figure 2: Illustration of the S_p set and how they are all cooped up in a $\delta \times 2\delta$ rectangle. And if there are more than 8, then two of them must be a contradicting pair. In this picture, the red-pair is one such.

Here is the key point: *every* pair of points in Q is at least $\delta_r \geq \delta$-apart. This is because δ_r was the distance of the closest pair in R. And yet, the 5 points in $S_p \cap R$ are all constrained in an $\delta \times \delta$ square. This is just not possible. To see this, divide this $\delta \times \delta$ squares into four $\frac{\delta}{2} \times \frac{\delta}{2}$-squares. At least one of these four must contain two points from $S_p \cap R$. However, the farthest two points in any square are the diagonal, and they are $\sqrt{\frac{\delta^2}{4} + \frac{\delta^2}{4}} < \delta$-apart. Thus, we obtain our contradiction. $\square$

Remark: *As you may see, the number* 8 *is not probably the best. How small can you make it?*

As a corollary, we get

Corollary 1. The inner while loop of ClosestCrossPair(S, δ) takes $O(1)$ time.

If $T(n)$ is the worst case running time of ClosestPair when run on point set of n points, we get the recurrence inequality which I hope we all have learned to love:

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$$

This evaluates to $T(n) = O(n \log n)$.

Theorem 1. The closest pair of points among n points in a plane can be found by ClosestPair in $O(n \log n)$ time.