CSE 347 Lecture 5

Divide-and-conquer II - the closest pair problem

1. Problem: Finding the closest pair of 2-d points
2. Geometric structure of the combining problem
3. An algorithm for the closest pair problem

Divide-and-conquer algorithms can be much more sophisticated than we've seen so far. (Although Strassen's Algorithm is quite ingenious. once one has the formulas, it is relatively straightforward.) Today, we'll consider the following problem: given n 2-d points, find the closest pair under the standard Euclidean distance, $d((x,y),(x',y')) = \sqrt{(x-x')^2 + (y-y')^2}$.

It is straightforward to solve this problem in $O(n^2)$ time - simply scan all of the $\binom{n}{2} = O(n^2)$ pairs of points, and keep a running minimum. As with matrix multiplication, it was natural to conjecture that this simple algorithm is optimal, but a clever divide-and-conquer algorithm can improve it.

For inspiration, let's consider — how can we solve the closest pair problem in 1-d? Sort the points, and in a second pass, compute the difference between consecutive points in the list. Since it's clear that the closest pair must be next to each other in the sorted list, the consecutive pair with the smallest distance is indeed

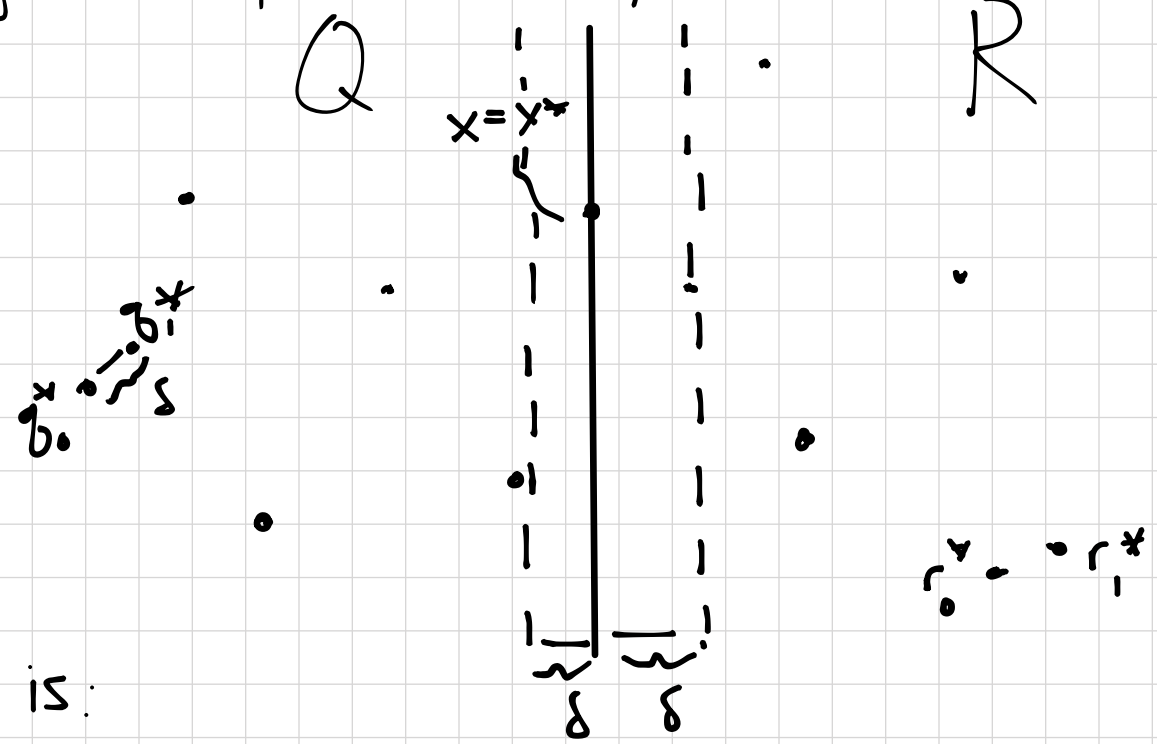
the closest pair. The overall time is $O(n \log n) + O(n) = O(n \log n)$

In 2-d, this approach clearly fails since we need to consider both coordinates. Still, sorting the points is a useful first step, since it allows us to split the "left half" of the points from the "right half". Supposing we have recursively solved the closest pair problem (somehow), the problem now reduces to determining whether a point in the left half is closer to a point in the right half than either of these pairs. Naively, there are still $\frac{n}{2} \cdot \frac{n}{2} = \frac{1}{4}n^2$ such pairs, so this still seems to be quadratic time. This is where the structure of the problem comes to our rescue.

Geometric structure of the combining problem

To be a little more precise, let's assume that all of the points have distinct x and y coorclinates. This is easily ensured as we did with the minimum spanning tree problem. Let's suppose that we have lists P_x and P_y for our points P sorted, respectively, by x- and y-coordinates; let's suppose that in these lists, each point is annotated with its position in the other list. Let Q be the set of points in the first $\frac{n}{2}$ positions of P_x — the "left half" — and R be the set of points in the final $\frac{n}{2}$ positions of P_x, the "right half". In a single $O(n)$-time scan of P,

we can create the lists Q_x and Q_y, containing the points in Q sorted by x- and y-coordinates, and the lists R_x and R_y for R, annotated by their positions in the other list. (This sets up the next level of the recursion.) Now, we suppose that we have recursively found the closest pair q_0^*, q_1^* and r_0^*, r_1^* in Q and R respectively. Let δ be the minimum of the distances of these two pairs. Now, if x^* is the x-coordinate of the rightmost point of Q, the line $x = x^*$ separates Q from R:



So, the key question:
is there a point $q \in Q$ that is distance $< \delta$ from a point $r \in R$?

The first key property is:

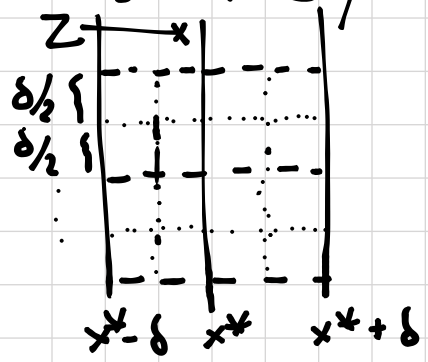
Lemma 1: If some pair $q \in Q$ and $r \in R$ has $\delta(q,r) < \delta$, then $|q_x - x^*| < \delta$ and $|r_x - x^*| < \delta$.

Proof? Observe $q_x \le x^* \le r_x$, so $|q_x - x^*| = x^* - q_x \le r_x - q_x$ which is less than δ, and likewise, $|r_x - x^*| = r_x - x^* \le r_x - q_x < \delta$. ∎

So now, by a single pass of P_y, we can find the list S_y, of points $S \subseteq P$ that have an x-coordinate within δ of x^*, sorted by y-coordinate. (Note, S could contain all of P, so there is still more to do.) The next property is the key:

Lemma 2: If points $s, s' \in S$ have distance less than δ, then they must appear within 15 positions in S_y.

Proof: We start by considering the set Z of points with x-coordinate within δ of x^*. We partition Z into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes, four boxes across. Observe that each box must contain at most one point of S – why? Points in the same box either both lie in Q or both lie in R. But, the boxes have radius $\delta \cdot \frac{\sqrt{2}}{2} < \delta$, so these would be a pair of points on the same side with distance less than δ, contradicting the choice of δ. So, consider any pair of points $s, s' \in S$ of distance less than δ. WLOG, $s_y < s'_y$. Now, observe that since $s'_y - s_y < \delta$, s' lies at most three rows of boxes away from s, as otherwise they have distance at least $\frac{3}{2}\delta > \delta$. Thus, since there are four boxes per row and at most one point per box, there can be at most 16 points in the strip containing s and s' — i.e., s' lies at most 15 points after s in S_y. ▨



An algorithm for the closest pair problem

Lemma 2 implicitly tells us how we can search for a closer pair of points between Q and R in time $O(n)$, which completes the algorithm. In summary: given a set of points P, we first sort P by x- and y-coordinates (in time $O(n \log n)$) and call

```
Closest-Pair-Rec(P_x, P_y)
  If n ≤ 3
    Return the pair of points of smallest distance of the 3
  Put Q_x ← P_x[1 ... n/2], R_x ← P_x[n/2+1, ..., n]; x* ← P_x[n/2]_x
   Q_y ← [(x,y) ∈ P_y : x ≤ x*], R_y ← [(x,y) ∈ P_y : x > x*]
   (q_0*, q_1*) ← Closest-Pair-Rec(Q_x, Q_y)
   (r_0*, r_1*) ← Closest-Pair-Rec(R_x, R_y)
   δ ← min{d(q_0*, q_1*), d(r_0*, r_1*)}
   S_y ← [(x,y) ∈ P_y : |x - x*| ≤ δ].
   For i = 1, ..., |S_y|
     For j = 1, ..., 15
       If d(S_y[i], S_y[i+j]) < δ
         Put δ ← d(S_y[i], S_y[i+j])
         Put s_0 ← S_y[i], s_1 ← S_y[i+j]
   Return the pair among (q_0*, q_1*), (r_0*, r_1*), and (s_0, s_1)
     that is the closest.
```

Theorem Closest-Pair-Rec, given the sorted lists P_x & P_y, returns the closest pair of points in time $O(n \log n)$.

Proof: By induction on n, the size of P:

Base: $n \leq 3$ — immediate.

Induction Step: Since Q + R are necessarily of size $\leq \frac{n}{2}$, we have by IH that (q_0^*, q_1^*) and (r_0^*, r_1^*) are, respectively, the closest pairs of points in Q and R respectively. Now, by Lemma 1, if there is a closer pair of points — which must be $q \in Q$, $r \in R$ — then both $|q_x - x^*| < \delta$, $|r_x - x^*| < \delta$.

Therefore, any such q and r must be in S_y. Moreover, Lemma 2 tells us that all such pairs (if they exist) must appear within 15 positions in S_y. Thus, either the closest pair is one of (q_0^*, q_1^*) or (r_0^*, r_1^*), in which case we correctly return this pair at the end, or else the closest pair is among $(S_y[i], S_y[i+j])$ for $j = 1, \ldots, 15$; in this final case, the algorithm returns the closest such pair s_0, s_1, which then is the closest pair in P.

Finally, the running time of Closest-Pair-Rec is described by the recurrence $T(n) = 2T(\frac{n}{2}) + O(n)$ since each line of the body of the algorithm can be done in time $O(n)$. We know that $T(n) = O(n \log n)$, therefore. (This is the MergeSort recurrence) ▨