CS 341: Algorithms
Lec 05: Divide and Conquer (part 3)

Armin Jamshidpey   Collin Roberts

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2025

Closest Pairs

Goal: given n points (x_i, y_i) in the plane, find a pair (i, j) that minimizes the distance

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Equivalent to minimize

$$d_{i,j}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

Assumption: all x_i's are pairwise distinct

Setup

1. **Brute Force:** $T(n) \in \Theta(n^2)$.
2. **Goal:** $T(n) \in \Theta(n \log n)$.
3. Q denotes the entire set of points, given when the algorithm kicks off.
4. Each recursive invocation takes as input
   1. a subset $P \subseteq Q$,
   2. an array X, which contains all the points of P, sorted by non-decreasing x-co-ordinate, and
   3. an array Y, which contains all the points of P, sorted by non-decreasing y-co-ordinate.
5. **Note,** we cannot afford to sort in each recursive call: this would make our recurrence $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$, leading to run time $T(n) \in O(n \log^2 n)$ (See the end of this slide deck).
6. Later, we will show how to use "pre-sorting" once and for all to avoid having to sort in each recursive call.

When Recursion Stops

1. Check whether $|P| \leq 3$.
2. If $|P| \leq 3$, then use the brute force approach: Try all $\binom{|P|}{2}$ possible pairs, and return the closest pair.
3. Otherwise, carry out the Divide-and-Conquer algorithm described below.

Divide-and-conquer: Divide

1. Find a vertical line, ℓ that bisects the point set P into two sets P_L, P_R, such that
   1. $$\begin{aligned} |P_L| &= \left\lceil \frac{|P|}{2} \right\rceil \\ |P_R| &= \left\lfloor \frac{|P|}{2} \right\rfloor, \end{aligned}$$
   2. all points in P_L are to the left of ℓ, and
   3. all points in P_R are on or to the right of ℓ.
2. Divide X into arrays X_L, X_R, containing the points of P_L, R_R, respectively, sorted by non-decreasing x-co-ordinate.
3. Divide Y into arrays Y_L, Y_R, containing the points of P_L, R_R, respectively, sorted by non-decreasing y-co-ordinate.

Divide-and-conquer: Conquer

1. Make two recursive calls:
   1. one with arguments (P_L, X_L, Y_L), to find the closest pair of points in P_L, at distance δ_L from one another, and
   2. the other with arguments (P_R, X_R, Y_R), to find the closest pair of points in P_R, at distance δ_R from one another.
2. Let $\delta = \min(\delta_L, \delta_R)$.

Divide-and-conquer: Combine

1. The closest pair of points is either the pair at distance δ from one another found by one of the above recursive calls, or it is a pair of points with one point in P_L and the other in P_R (i.e. a **transverse** pair).
2. The rest of the Combine step is to determine whether there exists a transverse pair whose points are at a distance $< \delta$ from one another.

Divide-and-conquer: Combine - Find Transverse Pair Having Distance $< \delta$, If One Exists

1. Observe that, if such a transverse pair exists, then both points of the pair must be at distance $\leq \delta$ from the line ℓ.
2. This is where the vertical strip of width 2δ comes from.
3. Find such a pair, if it exists, as follows:
    1. Create array Y', from Y, by removing all points from Y which do not lie in the 2δ-wide vertical strip. Preserve the sorting by y-co-ordinate from Y as we create Y' from it.
    2. For each point $p \in Y'$, find all points in Y' that are at distance $< \delta$ from p. We claim that we only need to check the 7 points following p in Y' (proof below). Compute the distance from p to each of these 7 following points. Keep track of the closest pair distance, δ', over all such pairs of points in Y'.
    3. If $\delta' < \delta$, then return the (transverse) pair having distance δ'; otherwise return the closest pair and distance δ found by one of the recursive calls above.

Divide-and-conquer: Combine - Why Only The 7 Following Points Need to be Checked.

1. Suppose that, at some level of the recursion, the closest pair of points is $p_L \in P_L$, and $p_R \in P_R$. I.e. suppose that the distance between these points is $\delta' < \delta$.
2. Then p_L is to the left of ℓ, at a distance of $\leq \delta$ from ℓ; similarly p_R is on or to the right of ℓ, at a distance of $\leq \delta$ from ℓ. (This is the vertical band of width 2δ from earlier.)
3. Moreover, p_L, p_R are at a distance of $\leq \delta$ from each other vertically. (This gives us the rectangle from earlier, disregarding the orderings of the points for the moment; there may be other points within this rectangle as well.)

Divide-and-conquer: Combine - Why Only The 7 Following Points Need to be Checked.

1. The argument from the earlier slide deck, that at most 8 points can lie inside this rectangle, is clear enough.
2. We still need to show that it suffices to check only the 7 points following each point in the array Y'.
3. WLOG, suppose that p_L occurs before p_R in the array Y' (if not, simply swap them).
4. Because p_L, p_R both lie in the rectangle, with p_L before p_R in the y-co-ordinate ordering, it suffices to consider points p_R having y-co-ordinates in the range $y_{p_L} \leq y \leq y_{p_L} + \delta$ (as in the earlier slide deck).
5. Even if p_L occurs as early as possible in Y', and p_R occurs as late as possible in Y', they can be no more than 8 positions apart from one another (because at most 8 points can lie in the rectangle, as observed earlier).
6. Hence p_R lies no more than 7 positions after p_L, in Y'.

Divide-and-conquer: Combine - Why Only The 7 Following Points Need to be Checked.

1. This completes the argument of the correctness of the provided algorithm.

Divide-and-conquer: Preserving Sorting by x- and y-Co-ordinates

How to split the sorted arrays for the recursive calls:

1. A particular invocation is given a subset P and the arrays X, sorted by x-co-ordinate and Y, sorted by y-co-ordinate.
2. Consider forming Y_L, Y_R from Y. The same approach will then work for forming X_L, X_R from X.
3. Having partitioned P into P_L, P_R, we must form arrays Y_L, Y_R, sorted by y-co-ordinate (in linear time).
4. Think of this as the opposite of MERGE: split a sorted array into two sorted arrays.
5. Examine the points in Y in order.
6. If a point $Y[i]$ is in P_L, then append it to Y_L; otherwise append it to Y_R.

Counting Inversions

Analysis of $T(n) = 2T\left(\frac{n}{2}\right) + dn\log n$

1. We guess that $T(n) \in \Theta\left(n\log^2 n\right)$.
2. These proofs follow the technique of **substitution**, demonstrated in §4.1 of CLRS.
3. As in CLRS, we ignore boundary conditions, since they do not affect the order of the growth.

Counting Inversions

Proof that $T(n) \in O(n \log^2 n)$

1. Assume that n is even, and that the bound holds for $\frac{n}{2}$.
2. Suppose that there exists a constant $c > 0$ and an n_0 such that, for all $n \geq n_0$,

$$T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right) \log^2\left(\frac{n}{2}\right).$$

Counting Inversions

Then substituting into the recurrence yields

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + dn\log n \\
&\leq 2\left[c\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right)\right] + dn\log n \\
&= cn\left(\log n - \log 2\right)^2 + dn\log n \\
&= cn\left(\log n - 1\right)^2 + dn\log n \\
&= cn\left(\log^2 n - 2\log n + 1\right) + dn\log n \\
&= cn\log^2 n + cn(1 - 2\log n) + dn\log n \\
&\leq cn\log^2 n,
\end{aligned}$$

provided $cn(1 - 2\log n) + dn\log n \leq 0$, which will hold provided

$$c \geq \frac{d\log n}{2\log n - 1} \geq \frac{d\log n}{2\log n} = \frac{d}{2}.$$

Counting Inversions

Proof that $T(n) \in \Omega(n \log^2 n)$

Suppose that there exists a constant $c > 0$ and an n_0 such that, for all $n \geq n_0$,

$$c\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right) \leq T\left(\frac{n}{2}\right).$$

Counting Inversions

Then substituting into the recurrence yields

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + dn\log n \\ &\geq 2\left[c\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right)\right] + dn\log n \\ &= cn\left(\log n - \log 2\right)^2 + dn\log n \\ &= cn\left(\log n - 1\right)^2 + dn\log n \\ &= cn\left(\log^2 n - 2\log n + 1\right) + dn\log n \\ &= cn\log^2 n + cn(1 - 2\log n) + dn\log n \\ &\geq cn\log^2 n, \end{aligned}$$

provided $cn(1 - 2\log n) + dn\log n \geq 0$, which will hold provided

$$c \leq \frac{d\log n}{2\log n - 1} \leq \frac{d\log n}{2\log n - \log n} = d.$$