Closest Pair of Points KT 5.4

Frank

September 30, 2021

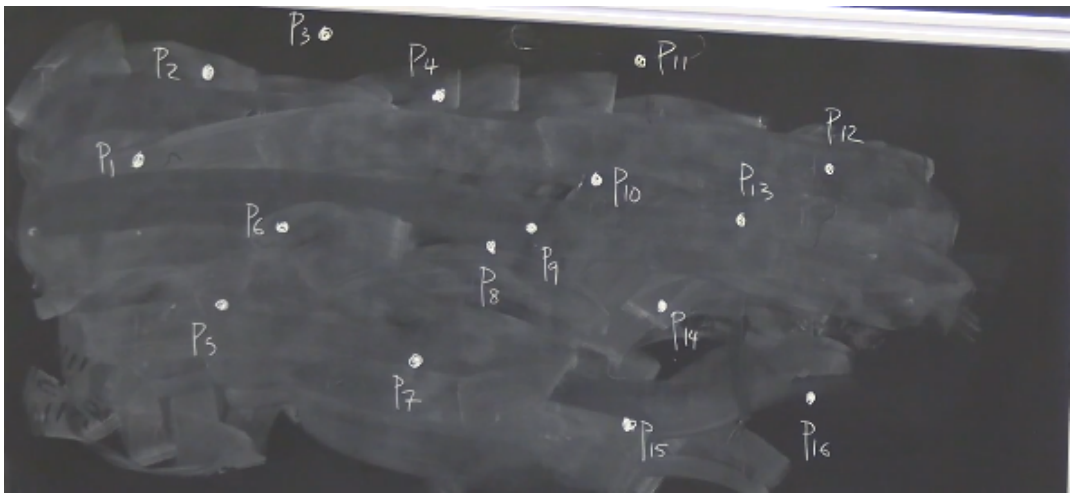
1 Problem

1.1 Input

$P =$ set of points on the plane
$p \in P$, where $p = (x, y)$
Distance between $p = (x, y), p' = (x', y')$ is

$$d(p, p') = \sqrt{(x - x')^2 + (y - y')^2}$$

n is the number of points in the set P



1.2 Output

$p \neq q \in P$ s.t. $\forall p' \neq q' \in P,\ d(p, q) \leq d(p', q')$. In other words, closest pair of points

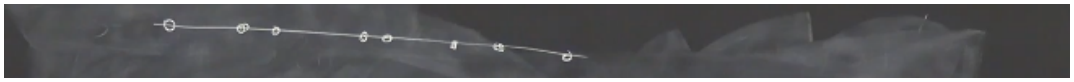
2 Solutions

2.1 Naive

Since we can look at every pair of points, our naive implementation would have time complexity of $O(n^2)$

2.2 DC attempt 1

- Look at a simpler version of the problem.



In this case, simply sort end points (1), and go down the line, looking at consecutive ones and keep track of the closest pair (2). We can do this in $O(n \log n) + O(n) = O(n \log n)$

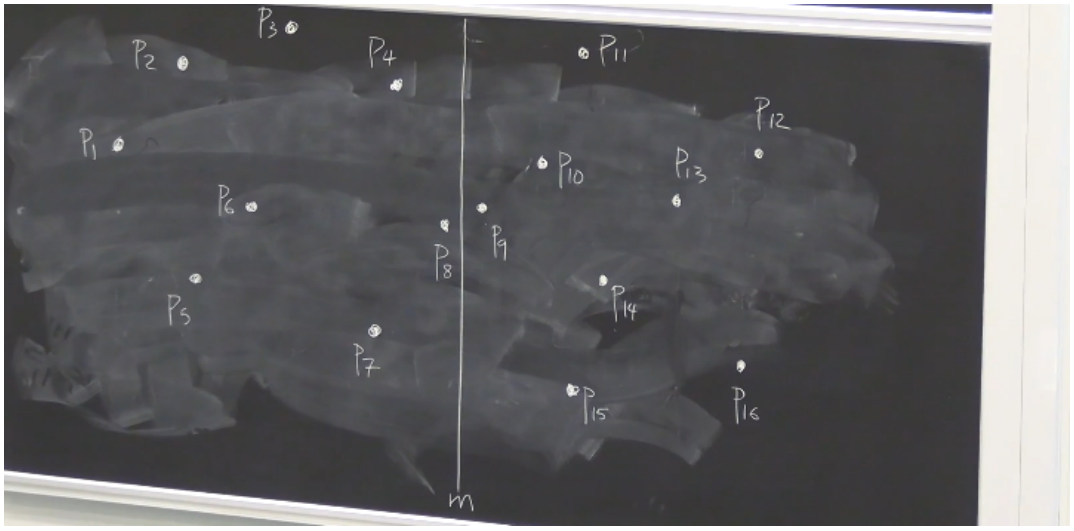
However, we can use DC to do (2) in a different way. Consider now that we split the array in half and find the closest pair in both halves, then all we have to do is compare the two halfs and the two points in the middle, which takes a constant amount of work.

This leads to the following recurrence

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c$$

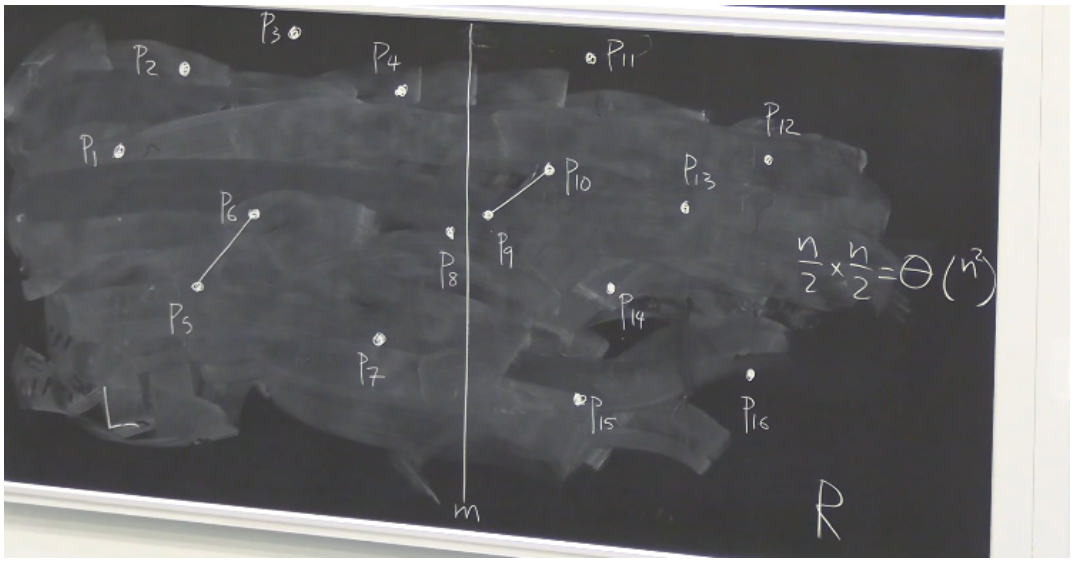
By Master Theorem, $T(n) = O(n^1) = O(n)$. Though this is no improvement, but we will use the same idea to find the closest pair on a plane.

- For our main problem, draw a line $x = m$ (we are using x-coordinate) to split the coordinates on the plane.



Additionally, assume that all points have distinct x, y coordinates.

We can find the closest pair on each side, and then compare every point on plane L wi h every point on plane R, but the problem is then that it will still have a complexity of $O(n^2)$, only for the cross line comparison part!



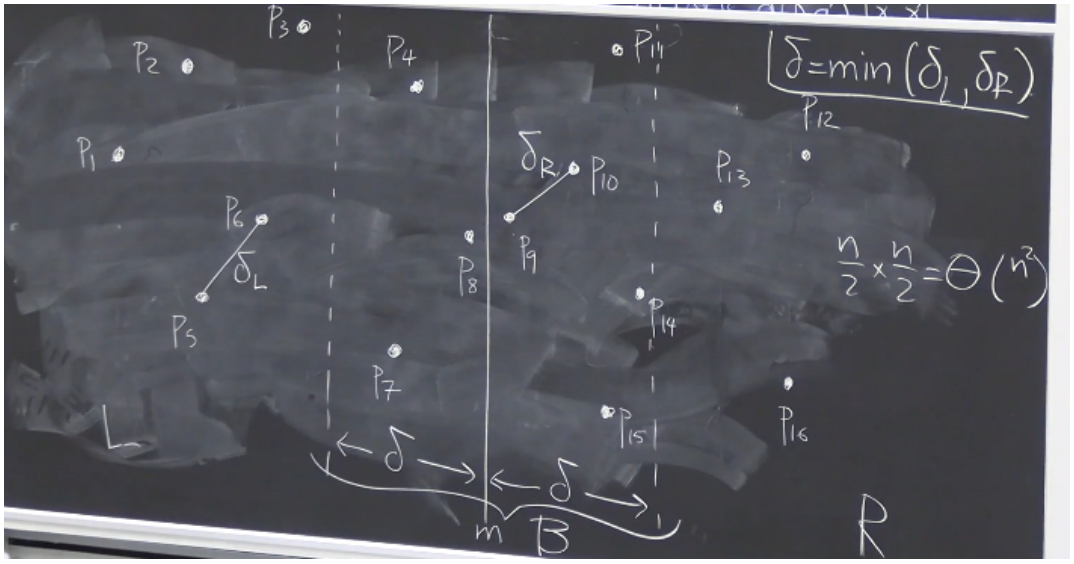
This algorithm then have recurrence

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn^2 \implies T(n) = O(n^2) \text{ (master theorem)}$$

2.3 DC attempt 2

To improve upon the previous attempt, maybe we can eliminate points that are too far away from each other.

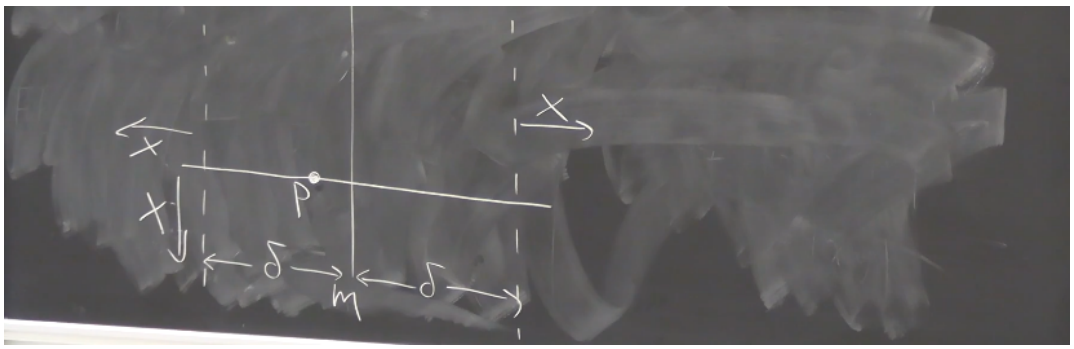
Note that we have found the closest points in L, R by our IH, we name them δ_L and δ_R, and let $\delta = min(\delta_L, \delta_R)$, then it's enough to focus on points within δ distance from the line $x = m$. Call this bound B.



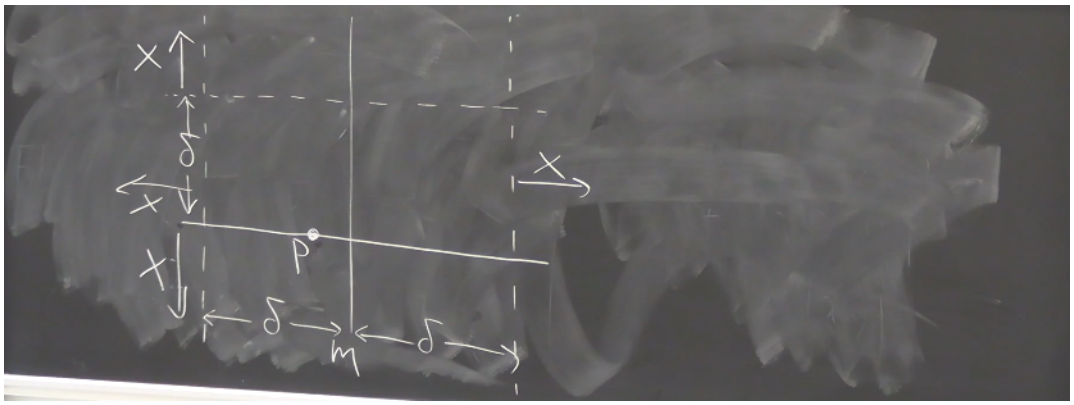
Now, how many points could lie in B? In fact, all of them. So we have to reduce the number of points we have to consider even more.

- Consider point $p \in B$, does $\exists q \in B$ s.t. $d(p, q) < \delta$?

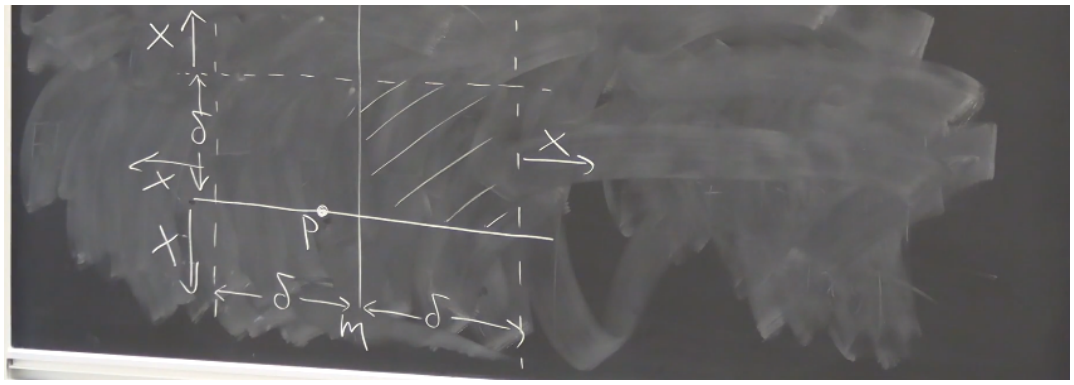
  Assume that p is the lowest point in B (minimum y-coordinate)



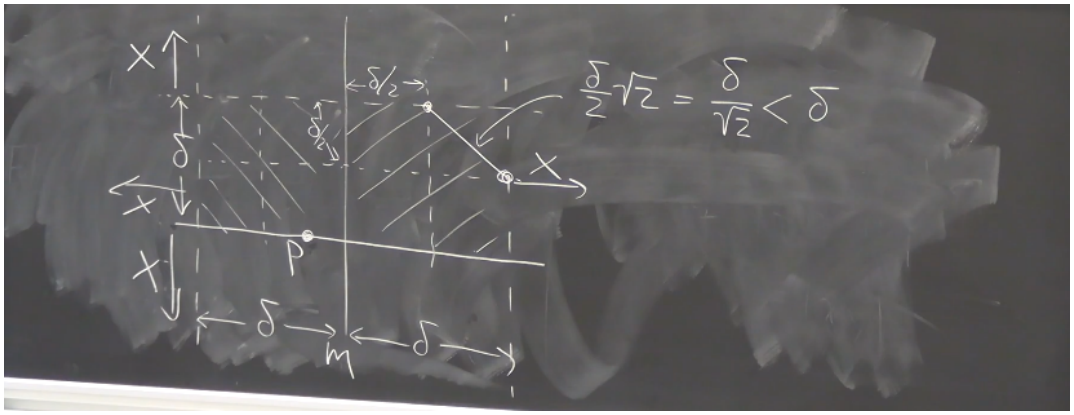
Then note that no points are below p, but also, if we look greater than length δ above p, then there would also be no points, since then it will definitely be greater than δ. Also, q cannot be on the same side of p, because the closest distance between any two points one the same sides is δ (by def of δ)



So then q must be on the other side



There will be at most four points within the shaded side, and 7 in total, because when we subdivide the square with length δ, we get that we can have at most one point within these smaller squares (with the exception of p square, because it contains p).



So we only have to look at 7 points above p in order to guarantee that we scanned the entire area of length δ above p, and if none of these 7 points are within distance δ of p, then no points within B will be within distance δ of p

After we verify that the next 7 points above p are not within distance δ, then we can move on to the next point above p and repeat this process, since the points below are already verified.

Note that for each of the n points that can be within B, we only look at 7 other points. Then we have a complexity of $O(n)$

2.4 Algorithm

Let P be a collection of points, and have the following

$$P \overset{sort}{\Longrightarrow} \begin{cases} P_x & = \text{list of points in } P \text{ sorted by x-coordinate} \\ P_y & = \text{list of points in } P \text{ sorted by y-coordinate} \end{cases}$$

Then divide like the following

$$P \overset{div}{\Longrightarrow} \begin{cases} L_x & = \text{first half of } P_x \\ R_x & = \text{second half of } P_x \end{cases}$$

Then define

$$L_y = \text{subsequence of } P_y \text{ of points in } L_x$$
$$L_y = \text{subsequence of } P_y \text{ of points in } R_x$$

```
1: procedure ClosestPair(P)                                   ▷ P is a set of points
2:     P_x as defined by above, sorted                        ▷ O(n log n)
3:     P_y as defined by above, sorted
4:     return RCP(P_x, P_y)
```

```
1: procedure RCP(P_x, P_y)                                    ▷ P_x, P_y are sorted set of points
2:     if (|P_x| ≤ 3) then
3:         calculate all pairwise distances                   ▷ brute force
4:         return closest pair
5:     else
6:         L_x ← first half of P_x, R_x ← second half of P_x  ▷ O(n)
7:         m ← (max x-coordinate in L_x + min x-coordinate in R_x)/2
8:         L_y ← sublist of P_y consisting of points in L_x   ▷ run through P_y and look at x
9:         R_y ← sublist of P_y consisting of points in R_x   ▷ O(n)
10:        (p_L, q_L) ← RCP(L_x, L_y)                         ▷ closest pair on left
11:        (p_R, q_R) ← RCP(R_x, R_y)                         ▷ closest pair on right
12:        δ ← min(d(p_L, q_L), d(p_R, q_R))
13:        if (d(p_L, q_L) = δ) then
14:            (p*, q*) ← (p_L, q_L)                          ▷ closest pair of points
15:        else
16:            (p*, q*) ← (p_R, q_R)
17:        B ← sublist of P_y w/ points whose x-coordinates are within δ of m   ▷ O(n)
18:        for (p ∈ B (in order)) do                          ▷ O(n)
19:            for (each of the next (up to) 7 points q ∈ B after p) do
20:                if (d(p, q) < d(p*, q*)) then
21:                    (p*, q*) ← (p, q)
22:        return (p*, q*)
```

2.5 Running Time

Procedure RCP has recurrence

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn$$

$\implies T(n) = O(n \log n)$

Sorting is $O(n \log n)$

So our entire algorithm is $O(n \log n)$

2.6 Things to think about

- possible identical x or y coordinates for some points
- Generalize this to 3-dimensions

  Consider a plane m, and a rectangular tube for B
- Find the closest 3 points

  Divide into 3 sections centered around an origin

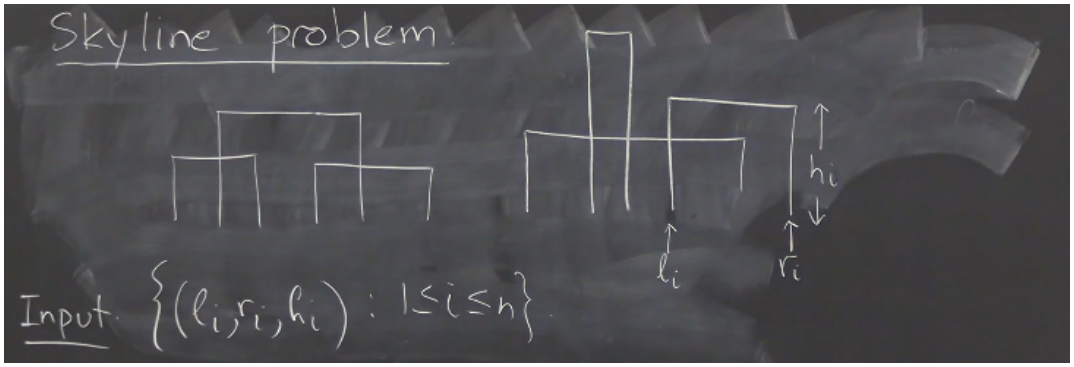
3 Puzzle

City of Toronto has just acquired a piece of land (35m x 35m) and wants to create a little park and plant 50 trees. Trees must be at least 7.5m apart.

Prove that this cannot be done

4 Skyline Problem

Given a description of a series of buildings.



- Output: Skyline of the buildings