Lecture 6: Recursive Algorithms: Finding the Closest Pair of Points

1 Finding the Closest Pair of Points in the Plane

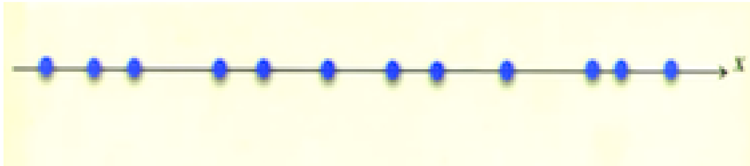
- Given n points $P = \{p_1, p_2, \cdots p_n\}$. find the **closest** pair of points.
- We will investigate the case where the points are in the plane.
  - The point set is 2-dimensional.
  - Specifically, for each $1 \leq i \leq n$, let $P_i = (x_i . y_i)$.
- How quickly can we do?

2 Exhaustive Search

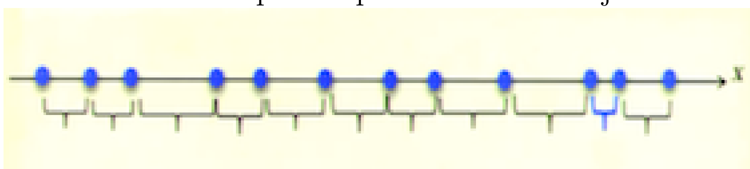
- The simplest algorithm to try is **exhaustive search**:
  - Calculate the distance between <u>every</u> pair of points.
  - Output the pair with the *shortest pairwise distance.*
- What is the running time?
  - There are n points so there are $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ pairs of points.
  - Runtime $= O(n^2)$
- So we have an efficient (dumb) algorithm! Is there a faster algorithm?

3 The 1-Dimensional Case

- It will be informative to first study the problem in 1-D.



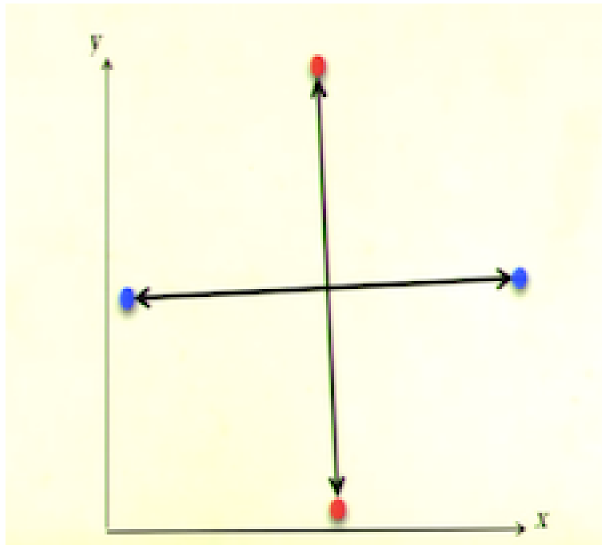
- In 1-D the closest pair of points must be adjacent in the x-ordering.



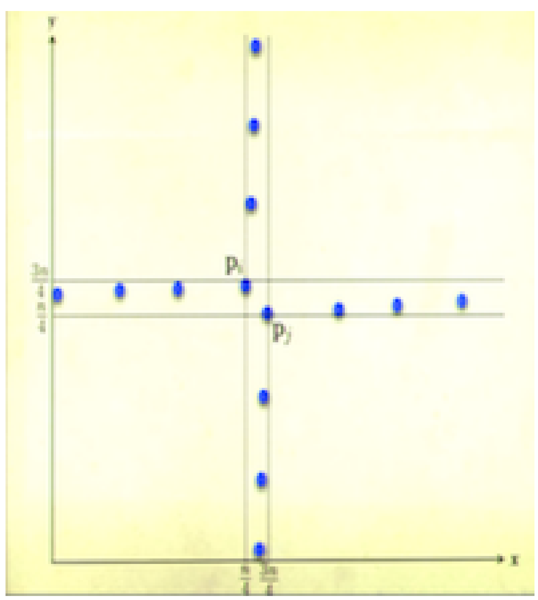
  - Thus we need to calculate only n-1 pairwise distances.
  - Runtime (If the points are already sorted in x-order) $= O(n^2)$
  - Runtime (If we have to sort the points first) $= O(n \cdot \log n)$

4 The 2-Dimensional Case

- Let mimic this idea in 2-D on the point set $P = \{p_1, p_2, \cdots p_n\}$
  - Order the points by their x-coordinate; we call this x-ordering.
  - Order the points by their y-coordinate; we call this y-ordering.
  - Find the *pair of points closest* in their x-coordinate.
  - Find the *pair of points closest* in their y-coordinate.
- Does this algorithm work?

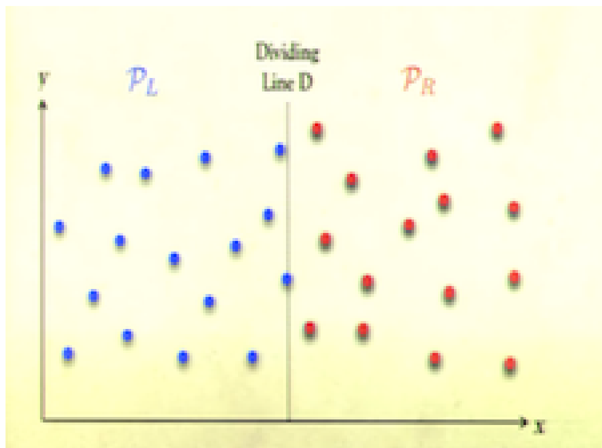


- NO!
- This algorithm does not work: points that are close in their x-coordinate (or y-coordinate) may be very far apart.
- In fact, the closest point P_i and P_j can be seperated by many places in both the x-ordering and the y-ordering.

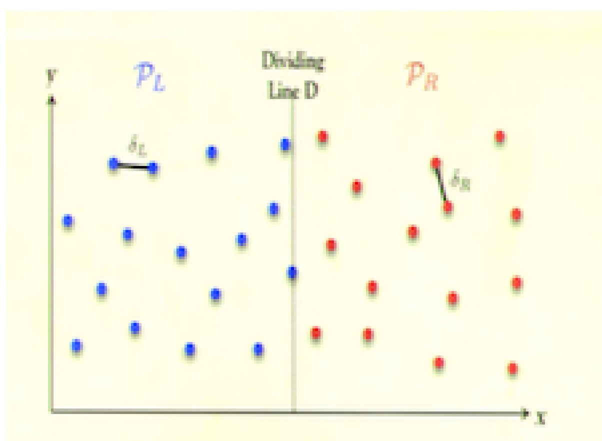


5 A Second Attempt

- Instead let's try a **divide and conquer** approach.
- Partition the points into *two groups* of cardinality $\frac{n}{2}$.
  - One way to do this is via the x-ordering using a **dividing line** D.
  - We select D to pass through the point with the median x-coordinate.
    - → We have a linear time algorithm to find the median.



- Three Possibilities:
  - We can now recursively search for the closest pairs in P_L and P_R.



- But there is a third possibility:
  - The closest pair could have one point in P_L and the other in P_R.
  - So after calculating δ_L and δ_R, we must check there is no better solution between a point in P_L and a point in P_R.

6 The Recursive Algorithm

- We can now recursively search for the closest pair in P_L and P_R.
  1. Find the point q with the **median** x-coordinate.
  2. Partition P into P_L and P_R using point q.
  3. Recursively find the *closest pairs* of points in P_L and P_R.
  4. Find the *closest pair* with one point in P_L and one point in P_R.
  5. Amongst the three pairs found, output the **closest** pair.
- How long does this algorithm take?
  - The recursive formula for the running time is:

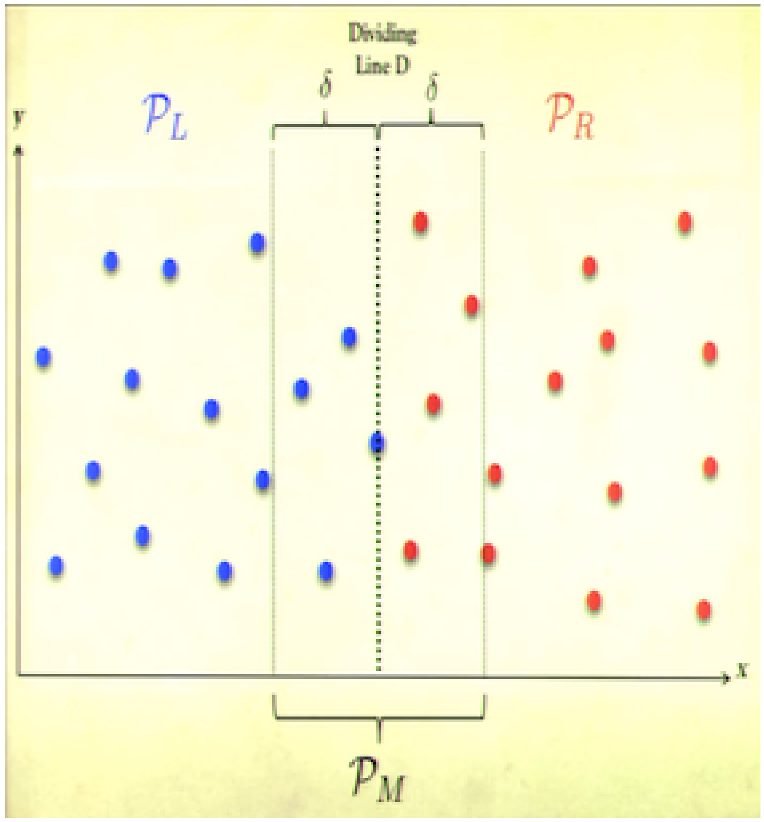
$$T(n) = 2 \cdot T(\frac{n}{2}) + O(n^2)$$

    → Thus we have a = 2, b = 2, and d = 2.
    → This is Case 1 of the Master Theorem.
    → Runtime = $O(n^d) = O(n^2)$
  - The bottleneck operation is Step 4.
    → So it takes far too long to measure the distance between every point in P_L and every point in P_R.
    → But, by solving the subproblems of P_L and P_R, we know the **minimum pairwise distance** is at most:

$$\delta = min\{\delta_L . \delta_R\}$$

    → The trick is to use the value of δ (which we have calculated by recursion) to reduce the number of distances we measure between P_L and P_R.

→ The key observation is that to find a better solution than δ the two points in P_L and P_R must be very close to the **dividing line** D...

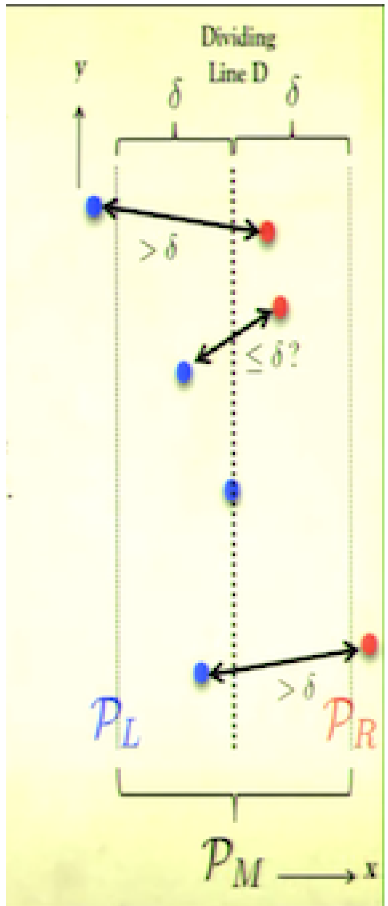


- **Lemma.** Take $p_i \in P_L$ and $p_j \in P_R$. If $d(p_i, p_j) \leq \delta$ then $\{p_i, p_j\} \subseteq P_M$

  Proof.

  - Without lost of generality, assume $p_i \notin P_M$
  - Then $d(p_i, p_j) > \delta$

- So if the closest-pair is not in one of the two sub-problems then it is within P_M.



7 A modified Recursive Algorithm

- This induces the following fine-tuned divide and conquer algorithm.
  1. Find the point q with the **median** x-coordinate.
  2. Partition P into P_L and P_R using point q.
  3. Recursively find the *closest pairs* of points in P_L and P_R.
  4. Find the *closest pair* of points in P_M. (Modified step!)
  5. Amongst the three pairs found, output the **closest** pair.
- Problem?
  - But P_M may contain **all** (or most) of the points!

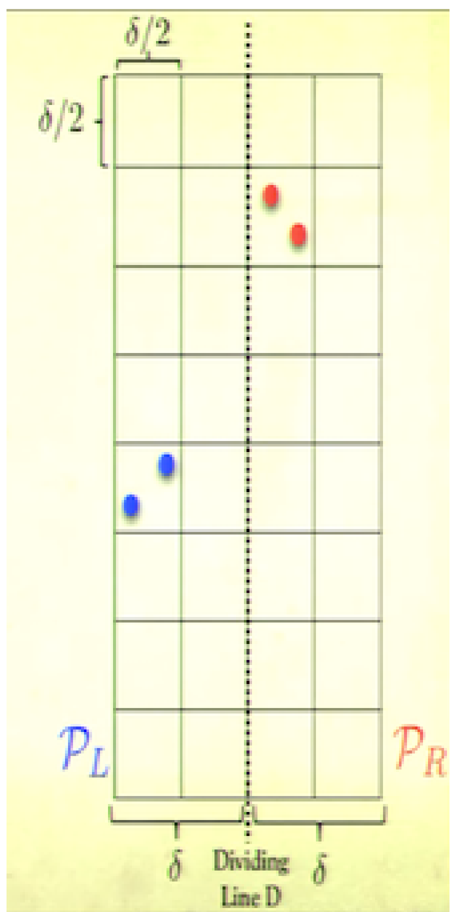
→ It may still take time $\Omega(n^2)$ to measure the distances between the points in $P_L \cap P_M$ and the points in $P_R \cap P_M$

- But the points in P_M cover a narrow band of the x-axis. As we have a **target** distance of δ, this helps a lot...
  - Consider the area of the plane within distance of δ of the line D.
  - Divide this area up into small squares of width $\frac{\delta}{2}$ as shown.

  Claim. No two points of P lie in the same square.

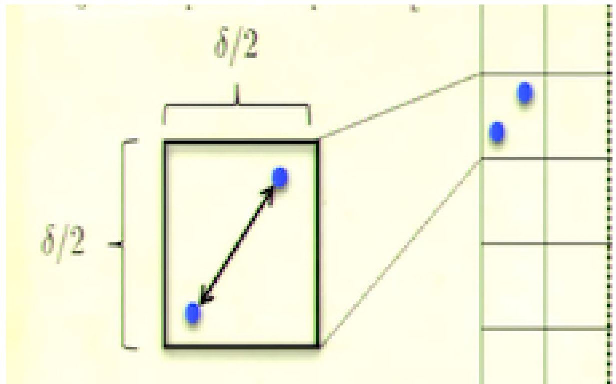
  - To prove this claim we will use the following observation.

  Observation. Each square lies entirely on one side of D.



Claim. No two points of P lie in the same square.
Proof.

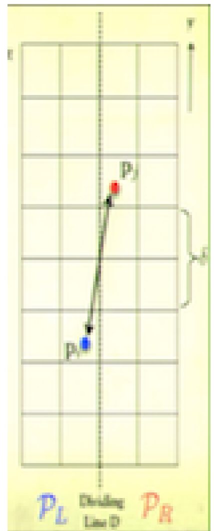
$\rightarrow$ *wlog* Take two points in a square in P_L.



$\rightarrow$ Their pairwise distance is: $\leq \sqrt{(\frac{\delta}{2})^2 + \frac{\delta}{2})^2} = \frac{\delta}{\sqrt{2}} < \delta$

$\rightarrow$ This contradicts the fact that the smallest pairwise distance in P_1 is at least δ.

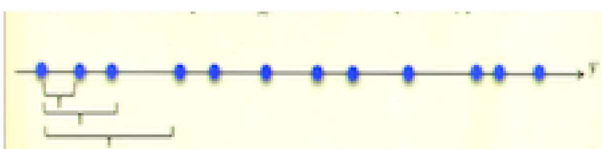
- **Theorem.** Take $p_i \in P_L$ and $p_j \in P_R$. If $d(p_i, p_j) \leq \delta$ then p_i and p_j have at most 10 points between them in the y-order of P_M.
  Proof.
  - *wlog* p_i is below p_j in the y-order.
  - Then p_j is in the same row of squares as p_i or in the next two higher rows.
  - If not, $d(p_i, p_j) > \delta$, a contradiction.



- ◦ But, by the claim, there is at most one point in each square.
  - → There are at most 10 points between p_i and p_j in the y-order of P_M.

8 Back to the 1-D Case

- This means the basic idea for the 1-D algorithm will work here!
- To find the **closest pair** in P_M, we first sort the points by y-coordinate.



- Then, rather than finding the distances between points that are 1-apart in the y-order, we find the distances for all pairs up to 11 places apart.
- Thus we calculate less than 11n pairwise distances. After doing so:
  - ◦ Either we find a pair of points at distance less than δ.
  - ◦ Or we conclude that **no** such pair of points exists.
- So give $\delta = min\{\delta_L, \delta_R\}$, we can check if there is a pair of points between P_L and P_R that are closer than δ in time $O(n)$.

9 An Enhanced Modified Recursive Algorithm

- Finally, this will give us very fast divide and conquer algorithm.
  1. Find the point q with the **median** x-coordinate.
  2. Partition P into P_L and P_R using point q.
  3. Recursively find the *closest pairs* of points in P_L and P_R.
  4. Find the *closest pair* of points in P_M using the **enhanced** 1-D algorithm. (Modified step!)
  5. Amongst the three pairs found, output the **closest** pair.
- The recursive formula for the running time is:

$$T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$$

  - Find median wrt x-coordinates.
  - Partition into P_L and P_R.
  - Find P_M.
  - Apply 1-D algorithm on P_M.
- Thus we have a = 2, b = 2, and d = 1.
- This is Case 2 of the Master Theorem.
- Runtime = $O(n^d \cdot \log n) = O(n \cdot \log n)$
- Validity of the algorithm:
  - As usual, the correctness of the algorithm follows by *strong induction.*
  - For the **bases cases**, we can find the closest pair in constance time by exhaustive search when there are a constant number of points left.
  - That the **inductive step** is correct follows by our discussion in the lecture.
    → **Theorem.** The recursive algorithm finds the minimum distance pair of points.

10 Computational Geometry

- The closest pair of points problem was a foundation question in the field of **computational geometry**.
- Computational has numerous applications:
  - Computer Graphics.
  - Computer Vision.
  - Geometric Information Systems.
  - ...