Design Paradigm: Divide and Conquer

- Finding Rank - Merge Sort
- Karatsuba Algorithm for Integers Multiplication
- Counting Inversions
- Finding Closest Pair in Plane

Imdad ullah Khan

Closest Pair of Points Problem

Given n points in a plane, find a pair of points with minimum Euclidean distance between them

For $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$

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

$\triangleright$ can be computed in $O(1)$

Applications: Computer graphics, computer vision, geographic information systems, molecular modeling, air traffic control

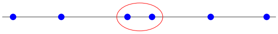
Closest Pair of Points Problem

Input: $P = \{p_1, p_2, \ldots, p_n\}$: a set of n distinct points in $\mathbb{R}^2$

Output: A pair of points in P that minimizes $d(p, q)$

1-dimensional space:

1. Sort points ▷ $O(n \log n)$
2. Find closest adjacent points ▷ $O(n)$



2-dimensional space:

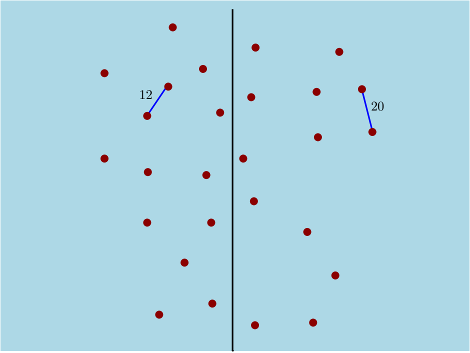
Brute force Algorithm:

FINDMIN among all $\binom{n}{2}$ pairwise distances ▷ $O(n^2)$ comparisons

Goal: $O(n \log n)$ time algorithm for 2-D version

Closest Pair: Divide & Conquer

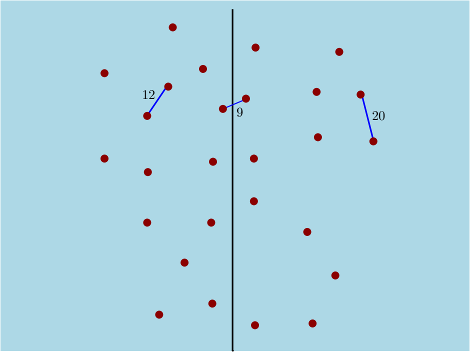
- Divide point set into two halves
- Find closest pair in each part recursively ▷ return closest of the two



Will it find closest pair?

Closest Pair: Divide & Conquer

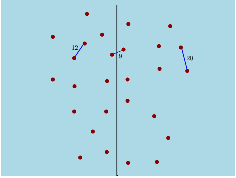
- Divide point set into two halves
- Find closest pair in each part recursively
- Find crossing closest pair ▷ return closest of the three



This will find the overall closest pair

Closest Pair: Divide & Conquer

1. Divide point set into two halves
2. Find closest pair in each part recursively
3. Find closest crossing pair
4. Return the closest of the 3 pairs



Algorithm Divide & Conquer based Closest pair: returns distance

function CLOSEST-PAIR(P)

SPLIT P into left and right halves, P_L and P_R

$\delta_1 \leftarrow$ CLOSEST-PAIR(P_L)

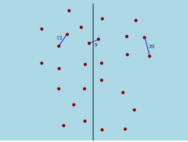
$\delta_2 \leftarrow$ CLOSEST-PAIR(P_R)

$\delta_3 \leftarrow$ FINDMIN distance over all pairs in $P_L \times P_R$

return MIN$\{\delta_1, \delta_2, \delta_3\}$

Closest Pair: Divide & Conquer

1. Divide point set into two halves
2. Find closest pair in each part recursively
3. Find closest crossing pair
4. Return the closest of the 3 pairs



Algorithm Divide & Conquer based Closest pair: returns distance

function CLOSEST-PAIR(P) ▷ $T(n)$

  SPLIT P into left and right halves, P_L and P_R ▷ "$O(n)$"

  $\delta_1 \leftarrow$ CLOSEST-PAIR(P_L) ▷ $T(n/2)$

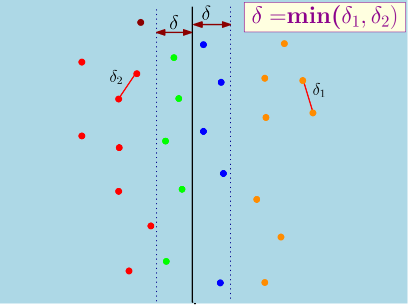
  $\delta_2 \leftarrow$ CLOSEST-PAIR(P_R) ▷ $T(n/2)$

  $\delta_3 \leftarrow$ FINDMIN distance over all pairs in $P_L \times P_R$ ▷ $n/2 \times n/2 = O(n^2)$

  return MIN$\{\delta_1, \delta_2, \delta_3\}$

$$T(n) = 2T\left(n/2\right) + O(n^2) \implies T(n) = O(n^2)$$

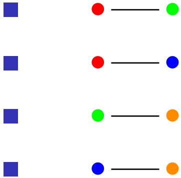
Closest Pair: Divide & Conquer

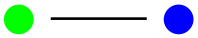


Find closest crossing pair?

Consider points within δ strip of the *'x-bisecting line'*

Closest crossing pair cannot be



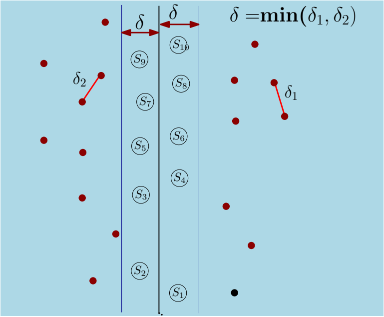
Critical observation: **closest crossing pair must be** 

- it not only (possibly) reduces the search space
- but gives us a very efficient algorithm

Closest Pair: Divide & Conquer

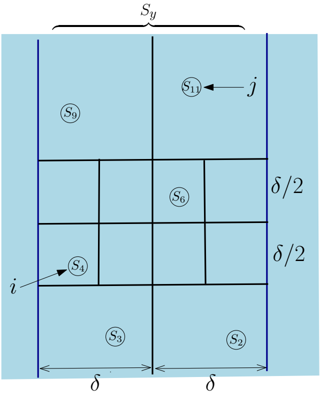
To find closest crossing pair (p_i, p_j) such that $d(p_i, p_j) < \delta$

- Consider points within δ of the bisecting line (in both directions)
- Sort points in 2δ strip by their y-coordinates, $S_y : s_1, s_2, \ldots,$
- Starting from lowest point $s_1 \in S_y$
- For each s_i only check the next 7 points in S_y, $s_{i+1}, s_{i+2}, \ldots, s_{i+7}$



Closest Pair: Grid Scan

- **Defn:** Let s_i be a point in the 2δ-strip with i^{th} smallest y-coordinate
- **Claim:** If $|i\text{-}j| \geq 7$, then $d(s_i, s_j) \geq \delta$
- **Proof:**
  - No two points lie in the same $\delta/2 \times \delta/2$ box
  - Two points, at least 2 rows apart, have distance $\geq 2(\delta/2)$



Closest Pair: Algorithm

Algorithm Divide & Conquer strategy for Closest pair: returns distance

function CLOSEST-PAIR(P)
  Compute bisecting line b_l
  SPLIT P into left and right halves, P_L and P_R
  $\delta_1 \leftarrow$ CLOSEST-PAIR(P_L)
  $\delta_2 \leftarrow$ CLOSEST-PAIR(P_R)
  $\delta = min(\delta_1, \delta_2)$
  Delete all points further than δ from separation line b_l
  SORT remaining points by y-coordinate
  Scan points in y-order and compare distance between each point and its next 7 neighbors. If any of these distances is less than δ, update δ
  return δ

Getting the actual pair realzing the distance δ is easy

Closest Pair: Correctness

Claim: Let p, q be pair having $d(p, q) \leq \delta$

Then:

- p and q are members of S_y
  - Closest crossing pair must be 
- p and q are at most 7 positions apart in S_y
  - Grid Scan is a proof of this

Closest Pair: Runtime Analysis

Running Time:

$$T(n) \; \leq \; 2T\left(\frac{n}{2}\right) + O(n \log n) \implies T(n) \; = \; \underbrace{O(n \log^2 n)}_{\log n \text{ times sorting}}$$

Can we acheive $O(n \log n)$?

- Pre-sort all points by x and y-coordinates
- Filter sorted lists to find the points within δ of b_l (no need to sort in every step to get S_y)

$$T(n) \; \leq \; 2T\left(\frac{n}{2}\right) + O(n) \implies T(n) \; = \; O(n \log n)$$