NPTEL MOOC,JAN-FEB 2015
Week 5, Module 7

DESIGN AND ANALYSIS OF ALGORITHMS

Divide and conquer: Closest pair of points

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Example: Video game

* Several objects on screen
* Basic step: find closest pair of objects
* Given n objects, naïve algorithm is $O(n^2)$
    * For each pair of objects, compute their distance
    * Report minimum distance over all such pairs
* There is a clever algorithm based on divide and conquer that takes time O(n log n)

Formally

- A point p is given by xy coordinates (x_p,y_p)
- Distance between $p_1 = (x_1,y_1)$ and $p_2 = (x_2,y_2)$ is the usual
  - $d(p_1,p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- Given n points $(p_1,p_2,\ldots,p_n)$, find the closest pair
  - Assume that no two points have same x or y coordinate
- Brute force
  - Try every pair (p_i,p_j) and report minimum
  - $O(n^2)$

In 1 dimension

- A point p is given by x coordinate x_p
  - $d(p_i,p_j) = |p_j - p_i|$
- Given n points $(p_1,p_2,\ldots,p_n)$
  - Sort the points — $O(n \log n)$
  - Compute minimum separation between adjacent points after sorting — $O(n)$

2 dimensions, divide and conquer

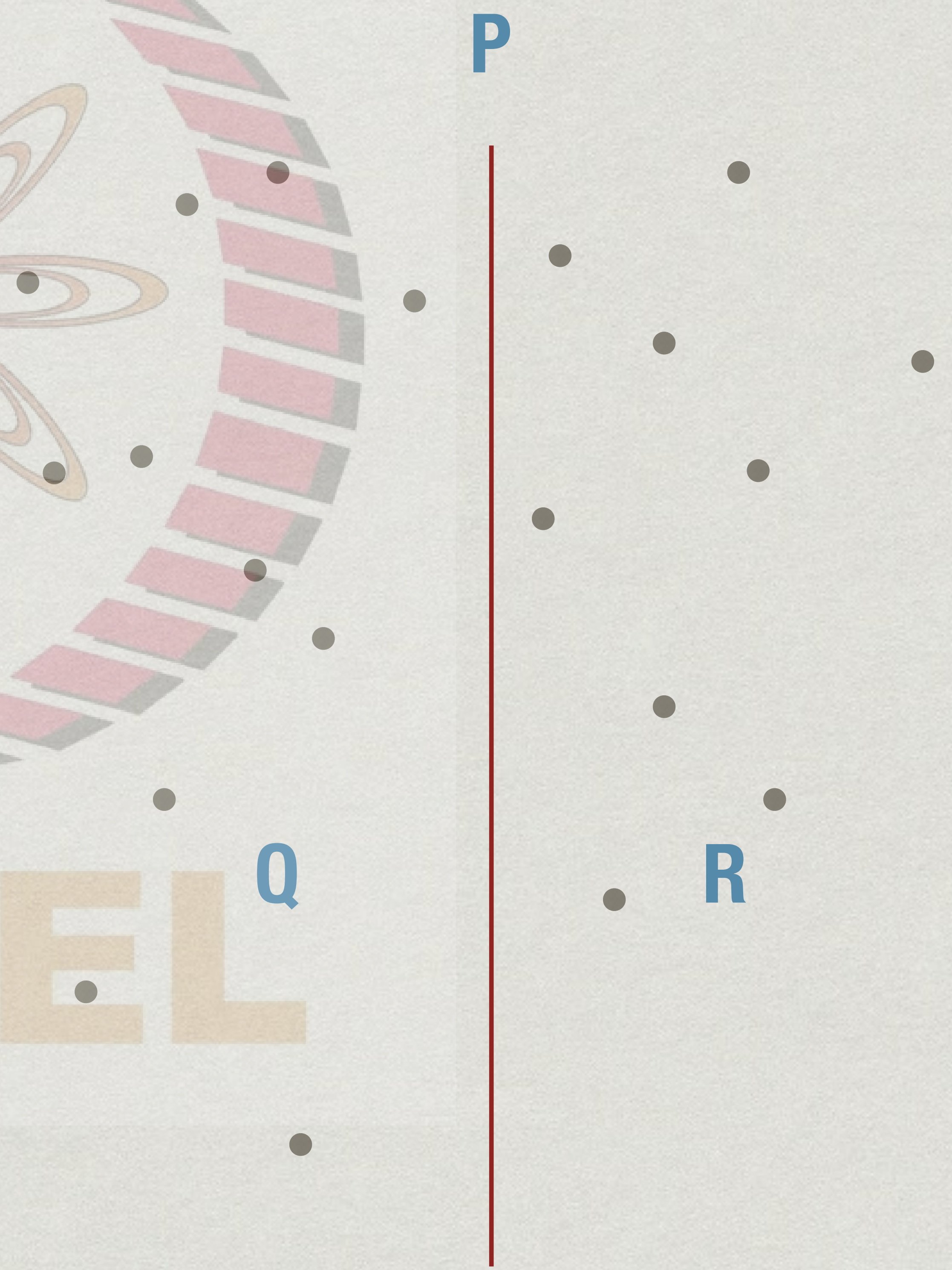
- Split set of points into two halves by vertical line
- Recursively compute closest pair in left and right half
- Need to then compute closest pairs across separating line
- How can we do this efficiently?

Sorting points by x and y

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x coordinate
  - P_y, P sorted by y coordinate
- Divide P by vertical line into equal size sets Q and R
- Need to efficiently compute Q_x, Q_y, R_x, R_y

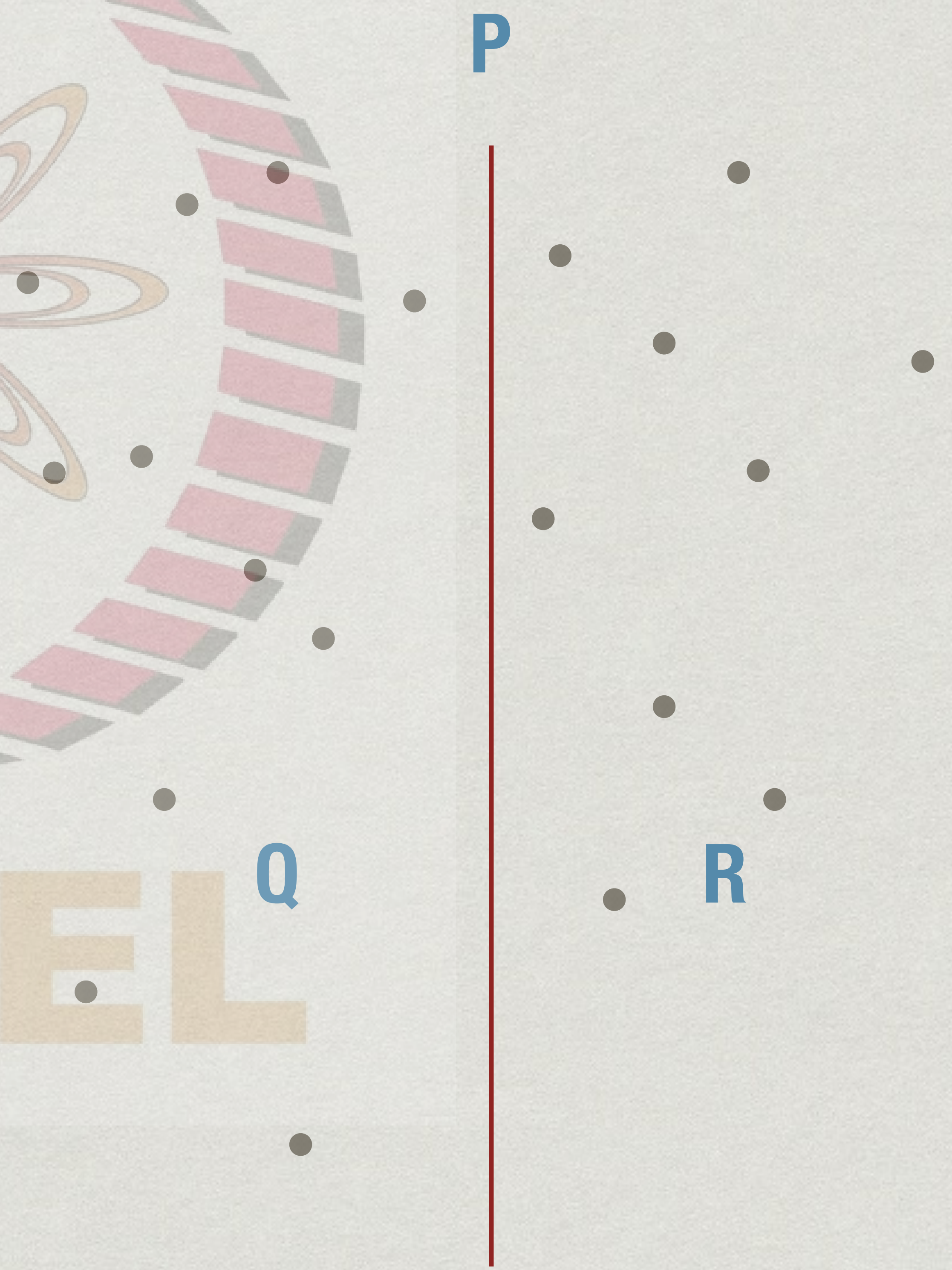
Sorting points by x and y

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x coordinate
  - P_y, P sorted by y coordinate
- Divide P by vertical line into equal size sets Q and R
- Need to efficiently compute Q_x, Q_y, R_x, R_y



Sorting points by x and y

- Need to efficiently compute Q_x, Q_y, R_x, R_y
  - Q_x is first half of P_x, R_x is second half of P_x
  - When splitting P_x, note the largest x coordinate in Q, x_Q
  - Separate P_y as Q_y, R_y by checking x coordinate with x_Q
- All O(n)

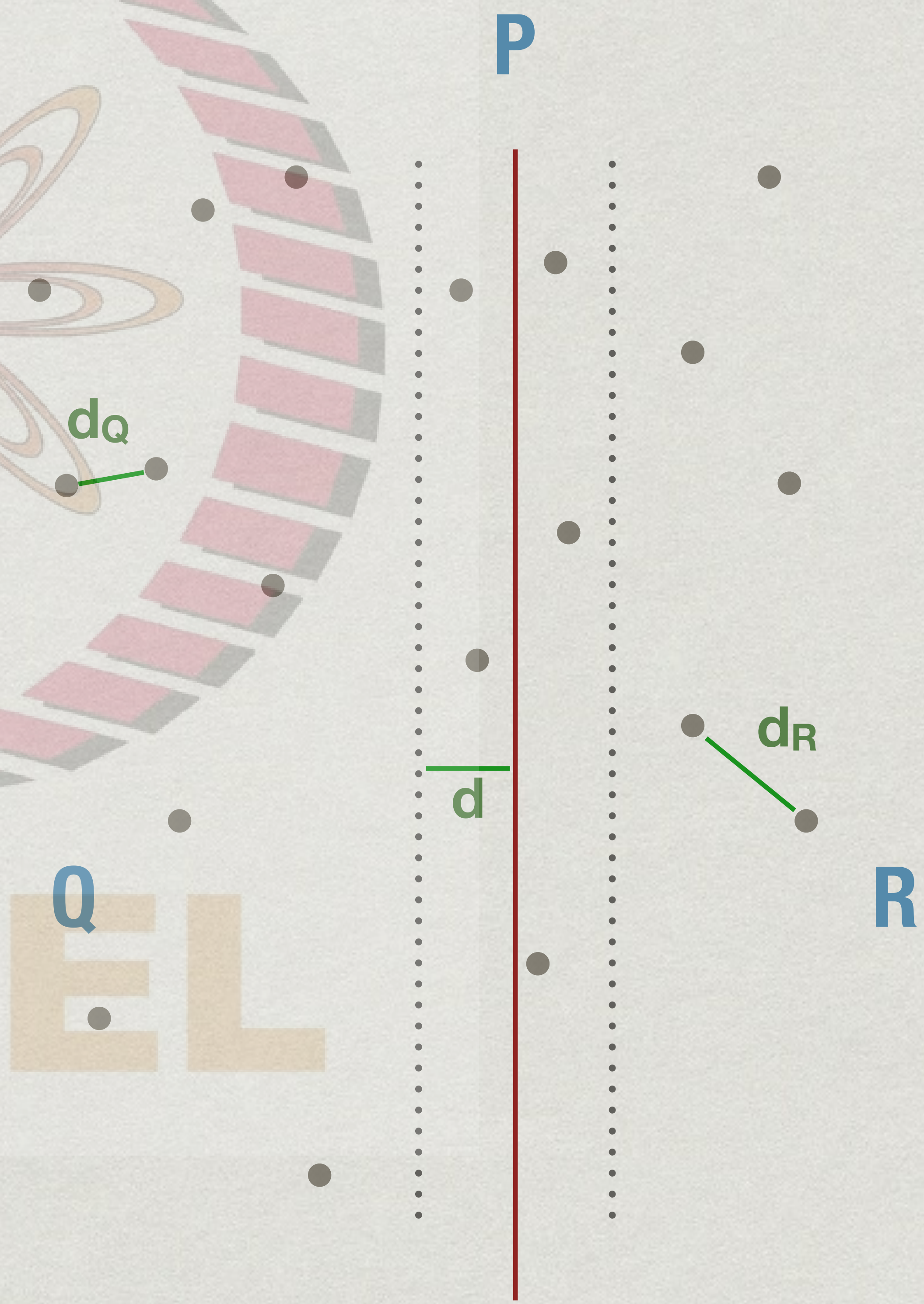


2 dimensions, divide and conquer

- Basic recursive call is ClosestPair(P_x,P_y)
- Set up recursive calls ClosestPair(Q_x,Q_y) and ClosestPair(R_x,R_y) for left and right half of P in time O(n)
- How to combine these recursive solutions?

Combining solutions

- Let d_Q be closest distance in Q and d_R be closest distance in R
- Let d be min(d_Q,d_R)
- Only need to consider points across the separator at most distance d from separator
  - Any pair outside this band cannot be closest pair overall



Combining solutions

- Divide the distance d zone into boxes of side d/2
  - Cannot have two points in same box
  - Diagonal is $\sqrt{2}d/2$
- Any point within distance d must lie in a neighbourhood of 4x4 boxes
  - Need to check each point against 15 others

Combining solutions

- From Q_y, R_y, extract S_y, points in d-band sorted by y coordinate

- Scan S_y from bottom to top, comparing each point against next 15 points in S_y

- Linear scan

Algorithm

```
function ClosestPair(Px,Py)

if (|Px| <= 3)
  compute pairwise distances and
  return closest pair and distance

Construct (Qx,Qy,Rx,Ry)

(dQ,q1,q2) = ClosestPair(Qx,Qy)

(dR,r1,r2) = ClosestPair(Rx,Ry)

Construct Sy and scan to find (dS,s1,s2)

Return (dQ,q1,q2), (dR,r1,r2), (dS,s1,S2) depending
on which among (dQ,dR,dS) is minimum
```

Analysis

- Computing (P_x,P_y) from P takes O(n log n)
- Recursive algorithm
  - Setting up (Q_x,Q_y,R_x,R_y) from (Px,Py) is O(n)
  - Setting up S_y from Q_y,R_y is O(n)
  - Scanning S_y is O(n)
  - Recurrence is same as merge sort
- Overall T(n) = O(n log n)