Divide and Conquer

Find closest pair of points given a set of points

- The problem statement: Given n points in the plane, find the pair that is closest together.
- Let us denote the set of points by P = {p1,..., pn}, where pi has coordinates (xi, yi); and for two points pi, pj ∈ P, we use d(pi, pj) to denote the standard Euclidean distance between them.
- Our goal is to find a pair of points pi, pj that minimizes d(pi, pj).

Brute force method:

Try every pair (pi,pj) and report minimum

Complexity: $O(n^2)$

Studying One Dimensional problem first..

- A point p is given by x coordinate xp only
- d(pi,pj) = |pj - pi|
- Given n points (p1,p2,…,pn), how to find closest pair
- Sort the points (Complexity — O(n log n))
- Compute minimum separation between adjacent points after sorting
- We can walk through the sorted list, computing the distance from each point to the one that comes after it. We can pick the minimum out of these.
- Complexity of finding minimum distance after sorting is :O(n)
- Overall complexity: O(nlogn)

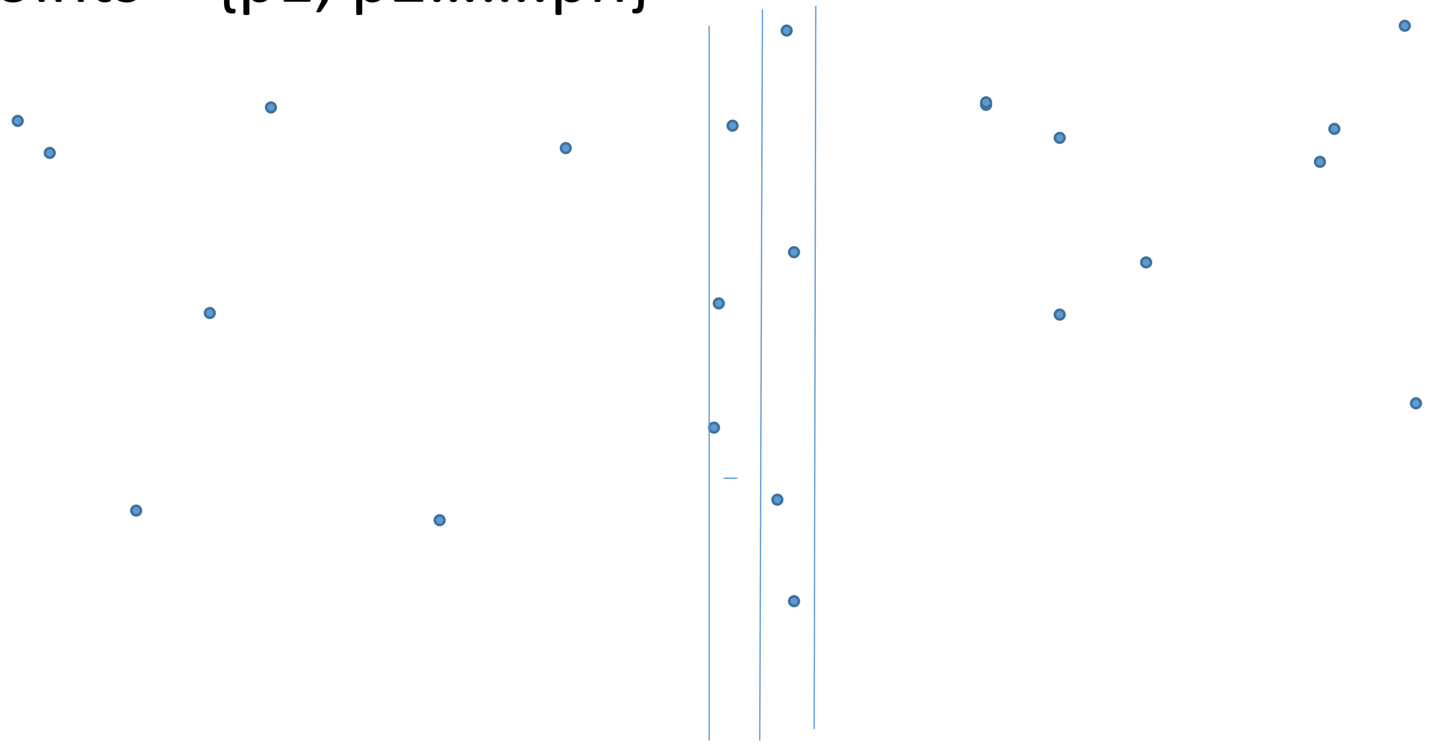
2 dimensional, Using Divide and conquer

Divide and Conquer algorithm:

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

Cont….

Set of points = {p1, p2…….pn}



How do we split?

Given n points P = {p1,..., pn}, we compute:

- P_x , P sorted by x co-ordinate
- P_y , P sorted by y co-ordinate
- Divide P by vertical line into two equal sets Q and R
- Vertical line will be at the mid point of P_x, so we take median of x-coordinates and that divides points into half
- So we have Q and R separately each of size n/2

Building up the recursion…

- For recursive calls, we have Q and R and we have to have Qx and Qy, Rx and Ry
- Finding Qx and Rx is easy but Qy and Ry is not so straight forward
- Qx is first half of Px and Rx is second half of Px
- To find Qy, we will scan R and all the points having x<xq are put in Qy and others in Ry.
- ***Note: Px is sorted on x axis and Py is sorted on y-axis***
- This can be done in linear time

Related Theorems

Th1: If there exists q Є Q and r Є R for which dist(q,r) <d then each of q and r lies within a distance of d of L (separating line of Q and R)

Th2: If s and s' Є S and have the property that dist(s,s') < d, then s and s' are within 15 positions of each other in the sorted list Sy.

S: Set of points in the band 'd' of separating line of Q and R

Sy: The list consisting of the points in S sorted by increasing y-coordinate

Cont…

- Suppose d= min(d1, d2) where d1 = minimum distance in left partition and d2 = minimum distance in right partition
- Now we have to examine the points across the separating line
- What are the candidate points (one from left and one from right) which can have distance less than d?
- All the points have to be within the distance d from the dividing line
- But do we have to consider all the points in that distance? (in the worst case all n/2 points may lie there)

Claim…

- Suppose we sort according to y-coordinate for all those points which are in the band of separating lines at distance d then

If points are arranged in sorted order of y-coordinate like …

Sorted list L = P'_1, P'_2, P'_3, …….. P'_{30} then

For every point P'_i, we need to calculate its distance with the next 7 consecutive points in list L and if there exist any two points having distance less than d, it will be found within this window only.

Hence: the complexity of finding two points in the band of separating line is 7* O(n) or we can say O(n)

Explanation..

- Actually for a point p in the band of separating line of left half, we can draw two squares of depth d, one below the point p along y axis and another above the point p along the y-axis
- Let us examine it for one point in the left band of separating line
- Suppose point is p_{left} and now with how many points in right band we need to calculate the distance for this point such that there are chances that the distance is less than d
- Let us draw a square of length d first
- Now in this square how many relevant points can be there…

Cont…

- Let us see how many points can be within each square:
- If we divide a square into 4 parts than the length of diagonal of each will be d/Sqrt(2) which is less than d
- So there can be atmost four points which will have distance more than d/2 or equal to.
- Pigeonhole principle can be applied here. If there is any 5th point in this square than it will lie in one of the smaller square of size n/2 only and then the distance between two points will be less than d which is contradiction to our assumption.

Cont..

- Hence, if we take any point in the left band of depth d, we have to take points bottom to top from that point.
- For this we have to maintain a sliding window and check points in that window
- So if we sort the points in that band by y-axis then we can look at each point in the direction of window and then correspondingly move the window up
- This takes linear time
- Also we have to check maximum 8 points for distance

Cont..

- Hence recurrence relation will look like this:
- $T(n) = 2T(n/2) + n \log n$
- This comes out to be of order: $O(n(\log n)^{2)}$

- This can be reduced to $O(n\log n)$ by making the second term of recurrence relation to $O(n)$.
- This can be done by pre-sorting the pairs with respect to x and then sorting them with respect to y so that we do not have to sort the points in the band with respect to y axis
- So recurrence relation will look like this:
- $T(n) = 2T(n/2) + O(n)$ giving overall complexity $O(n\log n)$

Algorithm…(assume all points are distinct, no two points can lie on same line)

- Split X into XL and XR
- Points in XL have x co-ordinates less than x_med and XR contains other points
- Similarly split Y into YL and YR
- But YL contains same points as XL and YR contains same points as XR
- Input is divided into XL,YL and XR, YR
- Recurse on both
- Find dL and dR and compute minimum d

Cont…

- Now we have to check the band of width d on both sides
- XL’: points from XL whose x co-ordinates are >=xmed-d
- YL’: sorted on y-coordinates but same points as XL’
- XR’: points from XL whose x co-ordinates are <=xmed+d
- YL’: sorted on y-coordinates but same points as XR’
- For each point in YL’ we will find points whose y-coordinate are in the range y’-d to y’+d
- Compute distances and take the minimum say m
- Return min(d,m)

Recursive calls

- Basic recursive call is Closestpair(Px,Py)
- Set up recursive calls Closestpair(Qx,Qy) and Closestpair(Rx,Ry)

(recursive call for left half and right half of P)

(complexity =O(n))

- How to combine these