e-PG PATHSHALA- Computer Science
Design and Analysis of Algorithms
Module 12
Component-I (A) - Personal Details

Role	Name	Designation
Principal Investigator	Dr.T.V.Geetha	Senior Professor, Department of Computer Science &Engineering, Anna University, Chennai
Content Writer (CW)	Dr.S.Sridhar	Associate Professor Department of Information Science and Technology, Anna University, Chennai
Content Reviewer (CR)	Dr.K.S.Easwarakumar	Professor Department of Computer Science &Engineering, Anna University, Chennai
Language Editor (LE)		

e-PG PATHSHALA- Computer Science
Design and Analysis of Algorithms
Module 12

Component-I (B) Description of Module

Items	Description of Module
Subject Name	Computer Science
Paper Name	Design and Analysis of Algorithms
Module Name/Title	Closest pair and Convex Hull Problems using Divide and Conquer
Module Id	CS/DAA/12
Pre-requisites	None
Objectives	To understand closest pair and convex hull finding algorithms that use Divide and Conquer design

	paradigm.
Keywords	Algorithms, Computational Geometry, closest Pair, Convex Hull Problem , Divide and conquer

Module 12: Closest pair and Convex Hull Problems using Divide and Conquer

This module 12 focuses on two important algorithms. One is to find closest pair of points and another is to find convex hull for set of points. The Learning objectives of this module are as follows:

- To understand the concept of closest pair problem
- To implement closest pair problem using divide and conquer strategy
- To know about Convex Hull
- To understand Quick Hull algorithm
- To understand Merge Hull algorithm

What is divide and Conquer design paradigm?

Divide and conquer is a design paradigm. It involves the following three components:

Step 1: (Divide) The problem is divided into subproblems. It must be noted that the subproblems are similar to the original problem but smaller in size.

Step 2: (Conquer) after division of the original problem into subproblems, the sub-problems are solved iteratively or recursively. If the problems are small enough, then they are solved in a straightforward manner.

Step 3: (Combine) Then, the solutions of the subproblems are combined to create a solution to the original problem

Closest Pair problem

What is a closest pair problem? The problem can be stated as follows: Given a set of points in the plane, find closest pair of points. The points can be a city, transistors on a circuit board, or computers in a network.

So the aim of the closest pair problem is to find the closest pair among *n* points in 2-dimensional space. This requires finding the distance between each pair of points and identifies the pair that gives the shortest distance.

Formally stated, the problem is given a set P of 'N' points, find points p and q, such that the d(p,q), distance between points p and q , is minimum.

Brute force method:

The simplest brute force algorithm approach is to find distances between all points and finding the pair where the distance is minimum. What is a distance? A distance is a measure of closeness. There are many types of distances. A Euclidean distance between two points $p(x, y)$ and q(s,t) is given as follows:

$$D_e(p,q) = [(x - s)^2 + (y - t)^2]^{1/2}$$

The distance measure should satisfy the following criteria to qualify as a metric. They are listed as follows:

1. D(A,B) = D(B,A)

   This property is called s*ymmetry* property

2. D(A,A) = 0

   This property is called Constancy of Self-Similarity

3. D(A,B) >= 0

   This property is called positivity

4. D(A,B) ≤ D(A,C) + D(B,C)

This property is called Triangular Inequality

Informal Algorithm

The informal brute force algorithm [1] is given as follows:

For each point i∈S and

another point j∈S , {points i and j are distinct}

compute distance of i, j

if distance of i, j < minum distance then update

min_dist = distance i, j

return min_dist

Complexity analysis:

The computational complexity of brute force approach is given based on [2,3] as

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2\sum_{i=1}^{n-1} (n-i)$$

$$= 2((n-1)+(n-2)+\cdots+1] = 2 \times \frac{n(n-1)}{2}$$

$$= n(n-1) \in \Theta(n^2)$$

Therefore, the complexity of the algorithm is $\theta(n^2)$. Now, the objective is to solve the same problem with reduced number of computations using the divide-and-conquer strategy.

Divide and Conquer strategy:

The divide and conquer can be given informally as follows:

- If trivial (small), solve it ñbrute forceò

ï Else

Å **1.divide** into a number of sub-problems

Å **2.solve** each sub-problem recursively

Å **3.combine** solutions to sub-problems

Based on divide and conquer, closest pair problem can be solved. The informal algorithm is given as follows:

Informal Algorithm

Å When n is small, use simple solution.

Å When n is large

- Divide the point set into two roughly equal parts A and B.
- Determine the closest pair of points in A.
- Determine the closest pair of points in B.
- Determine the closest pair of points such that one point is in A and the other in B.
- From the three closest pairs computed, select the one with least distance.

The division of the points is given as follows:

Initially, n points of a set S are sorted based on x-coordinate. Then the set S is divided into two subsets, S_{left} and S_{right}, using a vertical line L. This is shown based on [2] in Fig. 1.

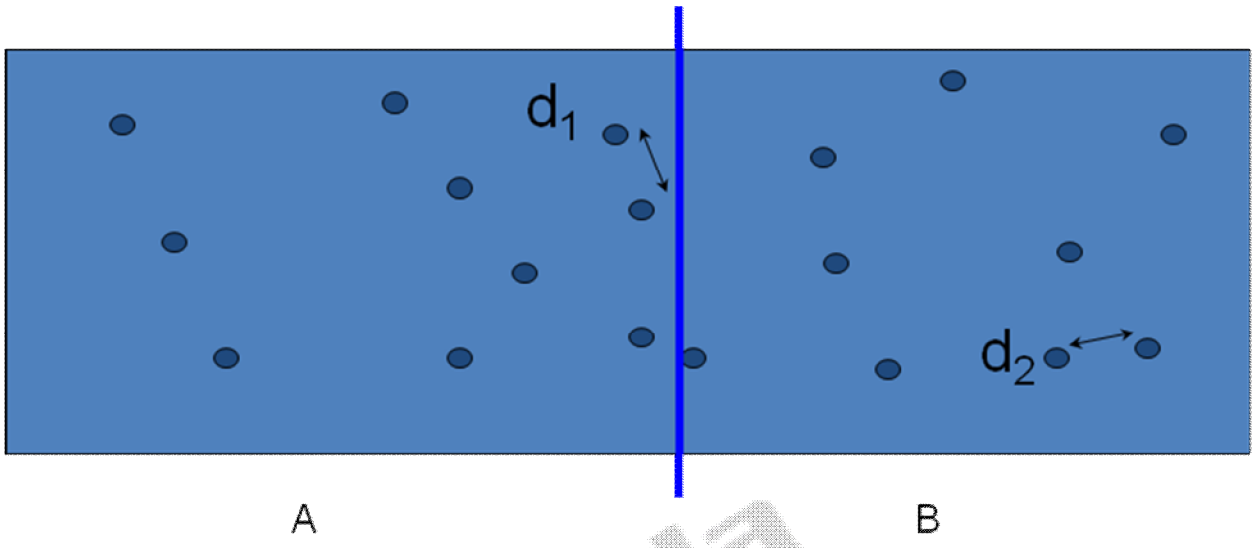

Fig 1: Division of points

It can be verified that the two subsets S_{left} and S_{right} are $\left\lfloor |S|/2 \right\rfloor$ and $\left\lceil |S|/2 \right\rceil$, respectively. Recursively, the closest points of S_{left} and S_{right} are computed. Let *d1* and d2 be the distances of the closest points of the sets S_{left} and S_{right}, respectively. Then the minimum distance d_{min} is calculated as follows:

Å $d = \min\{d_1, d_2\}$.

Therefore, the closest distance may be either d_l or d_r. The only problem here is that the closest-pair points may spread across S_{left} and S_{right}, that is, one point may belong to S_{left} and another to S_{right}. This needs to be computed. This would take $\theta(n^2)$. However, fortunately, one does not require to perform these comparisons as the main objective is to find only the closest pair. Therefore, we examine only a strip of d_{min} from the vertical line based on [2,3], as shown in Fig. 2.

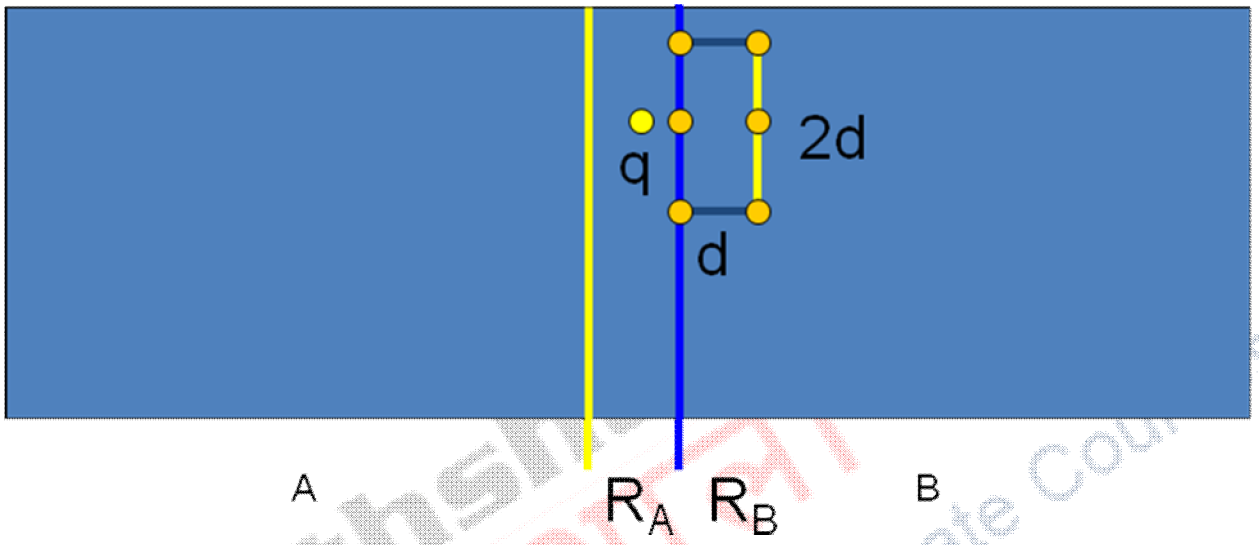

Fig. 2: Grid Formation

It can be observed that the zone around the strip L cannot be larger than d_{min}. Hence, the comparison between these points $p \in S_{\text{left}}$ and $q \in S_{\text{right}}$ is now reduced to this strip only. It can be observed that each point in this strip needs to be compared with at most six points. To find these six points, the points on this strip can be sorted based on y-coordinate. Then every point can be compared with at most six points. Next, the minimum distance (denoted as d_{across}) can be computed. It represents the closest distance between the points in the strip.

Informal Algorithm:

The algorithm based on [1] is given as follows:

Step 1: Sort points in S according to their y-values and x-values.

Step 2: If S contains only two points, return infinity as their distance.

Step 3: Find a median line L perpendicular to the X-axis to divide S into two subsets, with equal sizes, S_L and S_R.

Step 4: Recursively apply Step 2 and Step 3 to solve the closest pair problems of S_L and S_R. Let d_1, d_2 denote the distance between the closest pair in S_L (S_R). Let $d = \min(d_1, d_2)$.

Step 5: For a point P in the half-slab bounded by L-d and L, let its y-value by denoted as y_P

For each such P, find all points in the half-slab bounded by L and L+d whose y-value fall within y_P +d and y_P -d. If the distance d′ between P and a point in the other half-slab is less than d, let d=d′ . The final value of d is the answer. Formally, the algorithm can be stated [1] as follows:

Algorithm closestpair *A*[*A*[1 … *n*]]

%% Input: A set of *n* points

%% Output: Two closest points and distance

Begin

If n < threshold, then solve the problem by conventional algorithm

Else

mid = $\left\lfloor \frac{(i+j)}{2} \right\rfloor$

d_l = closestpair(*A*[1 ... mid])

d_r = closestpair(*A*[mid + 1 … *n*])

$d = \min(d_l, d_r)$

End if

for index = 1 to n do %% collect all points around *L*

if (*A*[index] >= *A*(mid).*x* − *d*) or *A*[index] <= *A*(mid + 1).*x* + *d*) then

Append *A*[*i*] to array *V*

End if

End for

Sort list *V* based on *y*-coordinates

%% Find closes points of the strip and distance

Let d_{across} = minimum of distance among six points of array *V*

return (min(d_{min}, d_{across})) %% Send minimum distance

End

Complexity Analysis

The division of S into S_{left} and S_{right} takes $\theta(1)$ time. Combining the two would take $\theta(n \log n)$ time. The important task here is to sort the points. This would take $\theta(n \log n)$ time using a quicksort algorithm. Therefore, the recurrence equation would be as follows:

$$T(n) = \begin{cases} n & \text{if } n \le 3 \\ 2T(n/2) + n \log n & \text{otherwise} \end{cases}$$

To reduce the complexity of the merging process further, one can presort the points based on *y*-coordinate. This improves the performance as in the combining step, instead of sorting; one needs to extract only the elements in $\theta(n)$ time. Total running time: $O(n \log^2 n)$.

Convex Hull

What is a convex Hull?

Let S be a set of points in the plane. Imagine the points of S as being pegs; the convex hull of S is the shape of a rubber-band stretched around the pegs. Formally stated, the convex hull of S is the smallest convex polygon that contains all the points of S. Convex hull is useful in many applications such as *collision detection in Robotics and Games design.*

Brute force Method:

Extreme points of the convex polygon form the vertex of convex hull. If all the points in the polygon as a set, then *extreme point* is a point of the set that is not a middle point of any line segment with end points in the set. A line segment connecting two points P_i and P_j of a set of n points is a part of its convex hullôs boundary if and only if all the other points of the set lies on the same side of the straight line through these two points.

Informal Algorithm:

The informal algorithm based on [1] is given as follows:

Determine extreme edges

for each pair of points $p,q \in P$ do

if all other points lie on one side of line passing thru *p* and *q* then keep edge (*p, q)*

Convex hull can be solved effectively using divide and conquer approach. Quickhull and Mergehull are two such algorithms for constructing convex hull.

QuickHull

Two algorithms, namely, quickhull and merge hull, are available for constructing a convex hull. The common approach for both these algorithms is given informally as follows:

Step 1: Divide the *n* points into two halves.

Step 2: Construct hulls for the two halves.

Step 3: Combine the two hulls to form a convex hull.

Quickhull is an algorithm that is designed to construct a convex hull; it is called quickhull as its logic is closer to that of finding the pivot element in a quicksort algorithm. This algorithm is dubbed the ñQuickhullò algorithm by Preparata and Shamos (1985) because of similarity to QuickSort and quickhull uses the divide-and-conquer strategy to divide the *n* points of a set *S* in the plane.

This approach is as follows:

- Identify extreme points a and b (part of hull)
- Compute upper hull
- find point c that is farthest away from line a,b
  - Connect the points ac
  - Connect the points cb

This is shown based on [2] in Fig. 3.

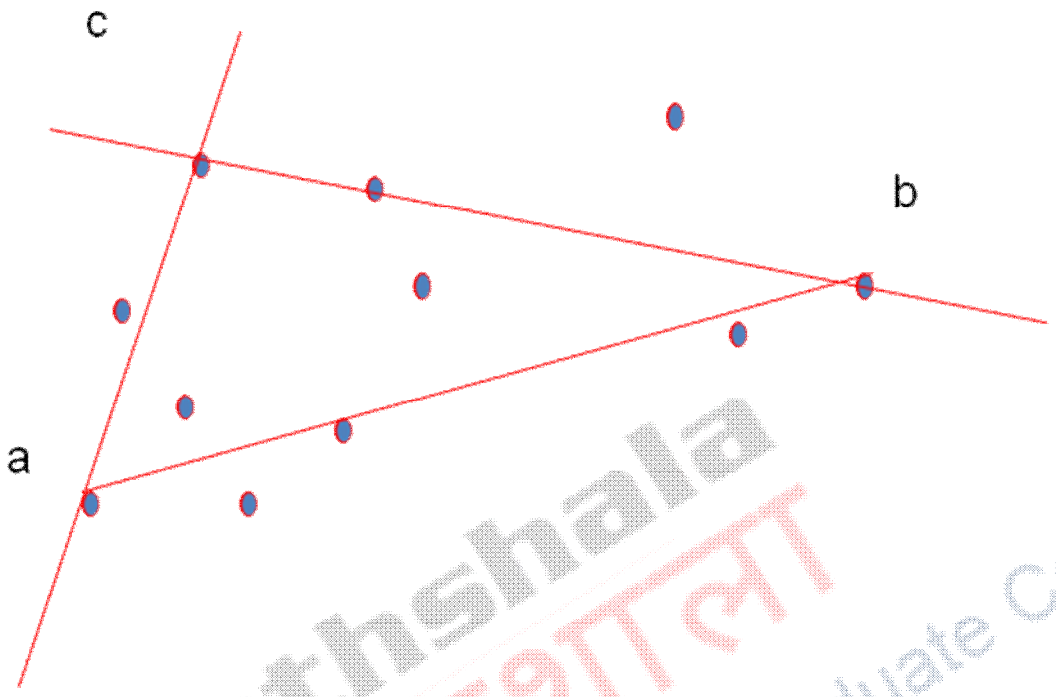

Fig. 3: Quickhull Formation

Thus the idea of quickhull is given [2,3] as follows:

function *QuickHull*(*a*, *b*, *S*)

if *S* = {*p1*, *pn*} then return {p1,pn}

else

pmax ← point furthest from edge (*p1,pn*)

A ← points right of (*p1,pmax*)

B ← points right of (*pmax,p2)*

return *QuickHull*(p1,pmax,*A*) concatenate with QuickHull(*pmax,p2,B)*

Informal Algorithm

Informally, the algorithm [1] for Quickhull can be written as follows:

Step 1: Sort the points based on *x*-coordinates.

Step 2: Identify the first point p_1 and last point p_n.

Step 3: Use $\overline{p_1 p_n}$ to divide the set into S_{left} and S_{right}.

Step 4: For S_{left}, find a point $p_{\max}$ that is far from the line $\overline{p_1 p_n}$. This line divides the set of points of S_{left} into two sets S_{11} and S_{12}.

Step 5: Ignore all the points inside the triangle $p_1\, p_{\max}\, p_n$.

Step 6: Form the left convex hull as $p_1 \cup S_{11} \cup p_{\max}$ and $p_{\max} \cup S_{12} \cup p_n$.

Step 7: Form the right convex hull using the steps similar to those used for the formation of the left convex hull.

Step 8: Combine the left and right convex hulls to get the final convex hull.

Complexity Analysis

Let n be the number of points of a set S, which are evenly divided into sets S_1 and S_2 in a quickhull algorithm. Let the sets consist of points n_1 and n_2, then the recurrence equation for quickhull algorithm is given as follows:

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

The solution of this leads to $O(n \log n)$.

Merge Hull

Merge hull is another algorithm that is based on merge sort for constructing a convex hull [3]. It uses the divide-and-conquer strategy. This algorithm is as effective as quickhull.

Idea of Merge Hull

Sort the points from left to right

Let A be the leftmost $\lceil n/2 \rceil$ points

Let B be the rightmost $\lfloor n/2 \rfloor$ points

Compute convex hulls $H(A)$ and $H(B)$

Compute $H(A \cup B)$ by merging $H(A)$ and $H(B)$

Initially, the points are sorted based on x-coordinates. Then partition is made based on the median of x-axis coordinates. Imagine a line that passes through the median dividing the set of points. This process divides the set of n points into two sets S_1 and S_2. Then convex hulls are constructed recursively from the sets of points S_1 and S_2. The main focus shifts to merging the convex hulls that are constructed.

Two convex hulls are joined by a lower tangent and an upper tangent. A tangent is also known as a bridge. It connects a vertex on the left convex hull with a vertex on the right convex hull. It is obtained by keeping one end fixed and changing another end rapidly to find whether it is a potential tangent. Then the convex hulls are merged using the upper and lower tangents while ignoring all the points between the tangents.

The following is the informal algorithm for merge hull:

Step 1: If the number of points involved is less, say less than 3, solve the problem conventionally using a brute force algorithm.

Step 2: Sort the points based on x-axis coordinates.

Step 3: Partition the set *S* into two sets S_{left} and S_{right} such that

$$Set\ \mathrm{S}_{left} = \{1, 2, 3, \cdots S_{mid}\}$$
$$Set\ \mathrm{S}_{right} = \{S_{mid}, S_{mid+1}, \cdots S_n\}$$

where mid is the median of *x*-axis coordinates. S_{left} now has all the points that are less than the median and S_{right} has all the points that are higher than the median.

Step 4: Recursively construct the convex hull.

Step 5: Find the lower and upper tangents between convex hulls

Step 6: Form the convex hull by merging the lower and upper convex hulls using the lower and upper tangents and ignoring all the points that fall between them.

Finding Tangent Lines

The most important aspect of merge hull is finding the upper and lower tangents based on [1] , which are shown in Fig. 4.

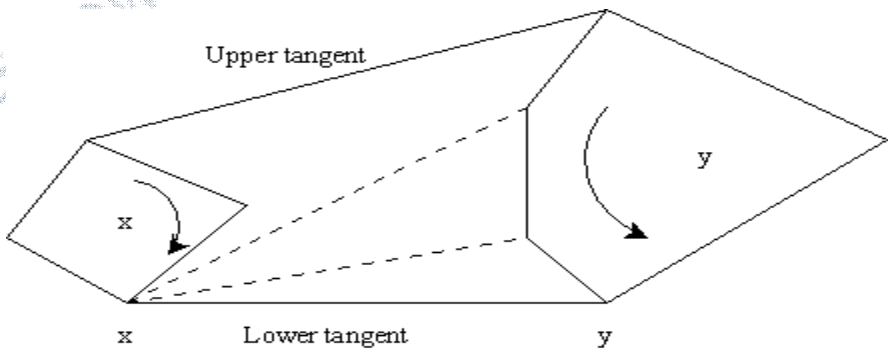

Fig. 4 Lower and upper tangents

Let *x* be the rightmost point of the S_{left} convex hull and *y* the leftmost point of the S_{right} convex hull. Connect *x* and *y*. If *xy* is not a lower tangent of S_{left}, then perform a clockwise rotation and pick the next vertex of the convex hull. Similarly, if *xy* is not a lower tangent for S_{left}, perform a counter-clockwise rotation and pick the next vertex of the convex hull. Using the right and left turns between points, one can decide whether the points lie on the tangent or not. In the same manner, the upper tangent is also formed.

Complexity Analysis

The recurrence equation of merge hull is given as follows:

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

Therefore, the solution of this equation leads to $O(n \log n)$.

Summary

In short, one can conclude as part of this module 12 that

- Å Brute force guarantee solution but inefficient. Divide and conquer is effective.
- Å Closest Pair problem is important problem and can be solved by divide and conquer strategy.
- Å Convex hull algorithm can be solved effectively using divide and conquer strategy.

References:

1. *S.Sridhar , Design and Analysis of Algorithms , Oxford University Press, 2014.*
2. *A.Levitin, Introduction to the Design and Analysis of Algorithms, Pearson Education, New Delhi, 2012.*
3. T.H.Cormen, C.E. Leiserson, and R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA 1992.