Design and Analysis of Algorithms

Unit-3

Divide and Conquer

Divide and Conquer Approach

- The divide-and-conquer paradigm involves three steps at each level of the recursion:
- **Divide** the problem into a number of sub-problems that are smaller instances of the same problem.
- **Conquer** the sub-problems by solving them recursively. If the sub-problem sizes are small enough, however, just solve the sub problems in a straightforward manner.
- **Combine** the solutions to the sub-problems into the solution for the original problem.

Divide and Conquer Approach

We will solve the following problems using divide and conquer approach :-

- Sorting
  - Merge sort
  - Quick sort
- Searching
  - Binary search
- Matrix Multiplication
- Convex Hull

Binary search

- **Binary search** is the most popular Search algorithm. It is efficient and also one of the most commonly used techniques that is used to solve problems.

- Binary search works only on a sorted set of elements. To use binary search on a collection, the collection must first be sorted.

- When binary search is used to perform operations on a sorted set, the number of iterations can always be reduced on the basis of the value that is being searched.

Binary search process

Binary search algorithm

```
Binary-search(A, n, x)
l = 1
r = n
while l ≤ r
do
        m = ⌊(l + r) / 2⌋
        if A[m] < x then
                l = m + 1
        else if A[m] > x then
                r = m − 1
        else
                return m
return unsuccessful
```

Time complexity **$T(n) = O(\lg n)$**

Matrix Multiplication (Divide and Conquer Method)

To multiply two matrices A and B of order nxn using **Divide and Conquer approach**, we use to multiply two matrices of order 2x2.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Where, A_{ij} and B_{ij} are $\frac{n}{2} \text{x} \frac{n}{2}$ matrices for i,j = 1,2.

Resultant matrix C will be

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Where,

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{21} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{21} + A_{22}B_{22}$

Matrix Multiplication

Clearly, computation of c_{ij} consists of two multiplications of two $\frac{n}{2}$ x $\frac{n}{2}$ matrices and one addition of two $\frac{n}{2}$ x $\frac{n}{2}$ matrices. Therefore, the algorithms for this is the following:-

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Matrix Multiplication

Time complexity of this algorithm is computed as following:-

$$T(n) = \begin{cases} \theta(1) & \text{if } n=1 \\ 8T(n/2) + \theta(n^2) & \text{if } n > 1 \end{cases}$$

After solving this recurrence relation, we get

$$\mathbf{T(n) = \theta(n^3)}$$

Strassen's Matrix Multiplication Algorithm

Strassen's algorithm has three steps:

1) Divide the input matrices A and B into $\frac{n}{2} \text{x} \frac{n}{2}$ sub-matrices.
2) Using the sub-matrices created from the step above, recursively compute seven matrix products P_1, P_2, … P_7. Each matrix P_i is of size $\frac{n}{2} \text{x} \frac{n}{2}$.

   $P1 = A_{11}(B_{12}-B_{22})$ $\quad$ $P2 = (A_{11}+A_{12})B_{22}$

   $P3 = (A_{21}+A_{22})B_{11}$ $\quad$ $P4 = A_{22}(B_{21}-B_{11})$

   $P5 = (A_{11}+A_{22})(B_{11}+B_{22})$ $\quad$ $P6 = (A_{12}-A_{22})(B_{21}+B_{22})$

   $P7 = (A_{11}-A_{21})(B_{11}+B_{12})$
3) Get the desired sub-matrices C_{11}, C_{12}, C_{21}, and C_{22} of the result matrix C by adding and subtracting various combinations of the P_i sub-matrices.

   $C_{11} = P_5+P_4-P_2+P_6$ $\quad$ $C_{12} = P_1+P_2$

   $C_{21} = P_3+P_4$ $\quad$ $C_{22} = P_1+P_5-P_3-P_7$

Strassen's Matrix Multiplication Algorithm

Strassen_Matrix_Multiplication(A, B, n)

If n=1 then return AxB.

Else

1. Compute A_{11}, B_{11}, ………………., A_{22}, B_{22}.
2. $P_1 \leftarrow$ Strassen_Matrix_Multiplication(A_{11}, B_{12}-B_{22}, n/2)
3. $P_2 \leftarrow$ Strassen_Matrix_Multiplication(A_{11}+A_{12}, B_{22}, n/2)
4. $P_3 \leftarrow$ Strassen_Matrix_Multiplication(A_{21}+A_{22}, B_{11}, n/2)
5. $P_4 \leftarrow$ Strassen_Matrix_Multiplication(A_{22}, B_{21}-B_{11}, n/2)
6. $P_5 \leftarrow$ Strassen_Matrix_Multiplication(A_{11}+A_{22}, B_{11}+B_{22}, n/2)
7. $P_6 \leftarrow$ Strassen_Matrix_Multiplication(A_{12}-A_{22}, B_{21}+B_{22}, n/2)
8. $P_7 \leftarrow$ Strassen_Matrix_Multiplication(A_{11}-A_{21}, B_{11}+B_{12}, n/2)
9. $C_{11} = P_5+P_4-P_2+P_6$
10. $C_{12} = P_1+P_2$
11. $C_{21} = P_3+P_4$
12. $C_{22} = P_1+P_5-P_3-P_7$
13. return C

End if

Strassen’s Matrix Multiplication Algorithm

Time complexity of this algorithm is computed as following:-

$$T(n) = \theta(1) \qquad \text{if } n=1$$

$$= 7T(n/2) + \theta(n^2) \qquad \text{if } n > 1$$

After solving this recurrence relation, we get

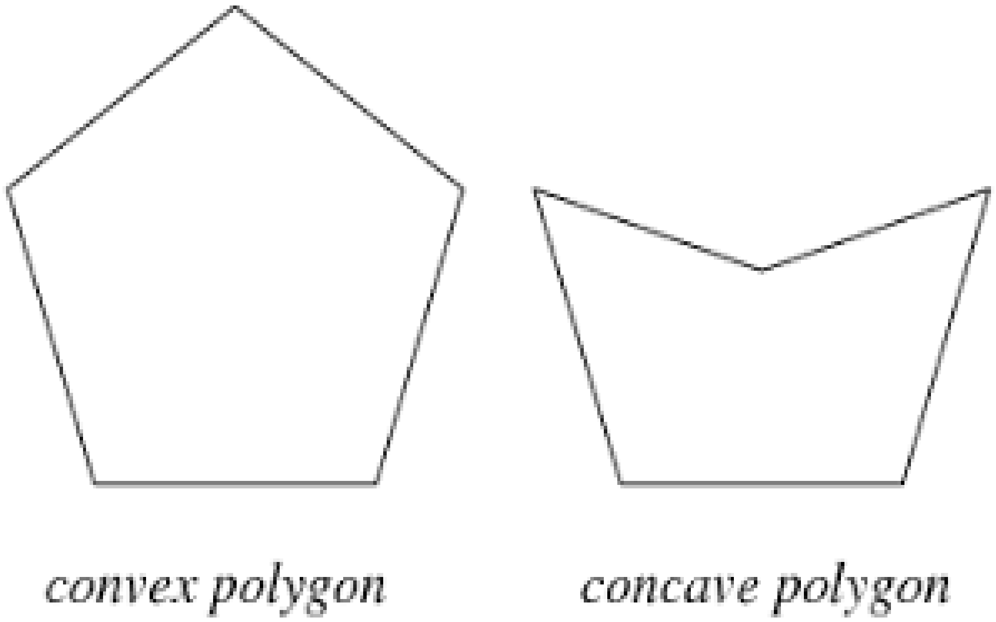
$$\mathbf{T(n) = \theta(n^{2.8})}$$

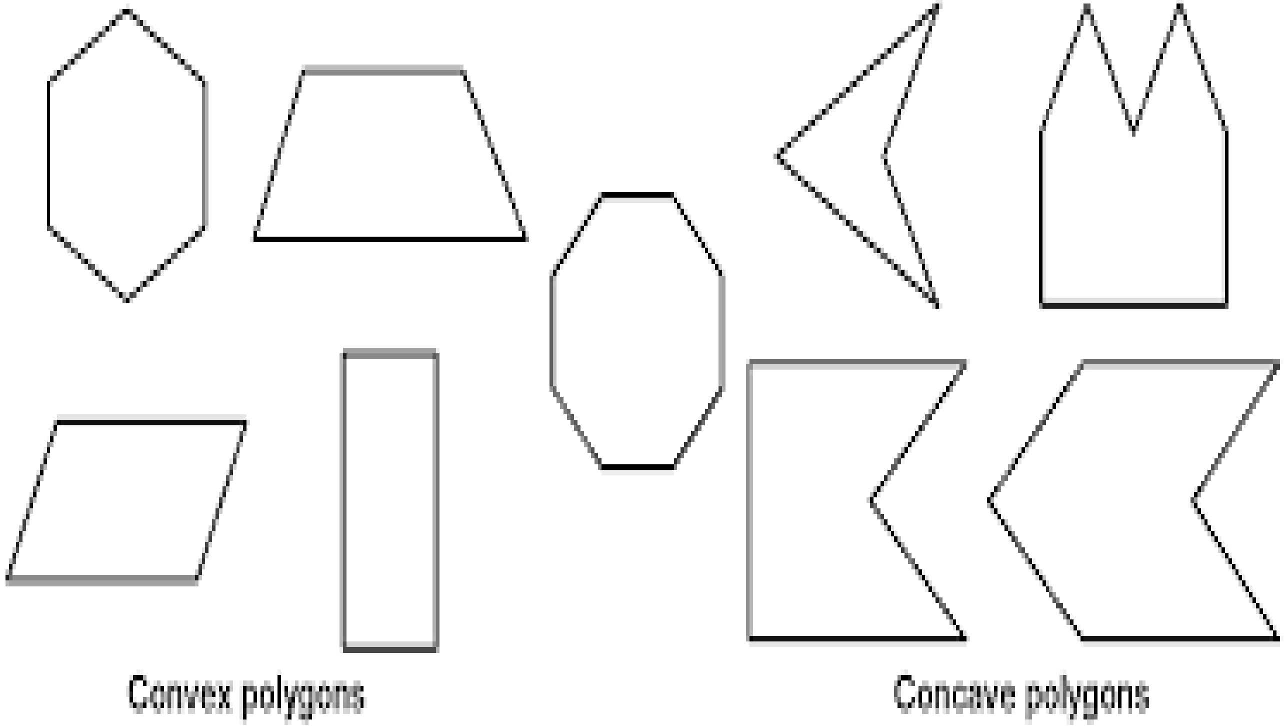
Convex Hull

Convex Polygon:

A polygon P is said to be convex polygon if for any two points p and q on the boundary of P, segment pq lies entirely inside P.

Example:

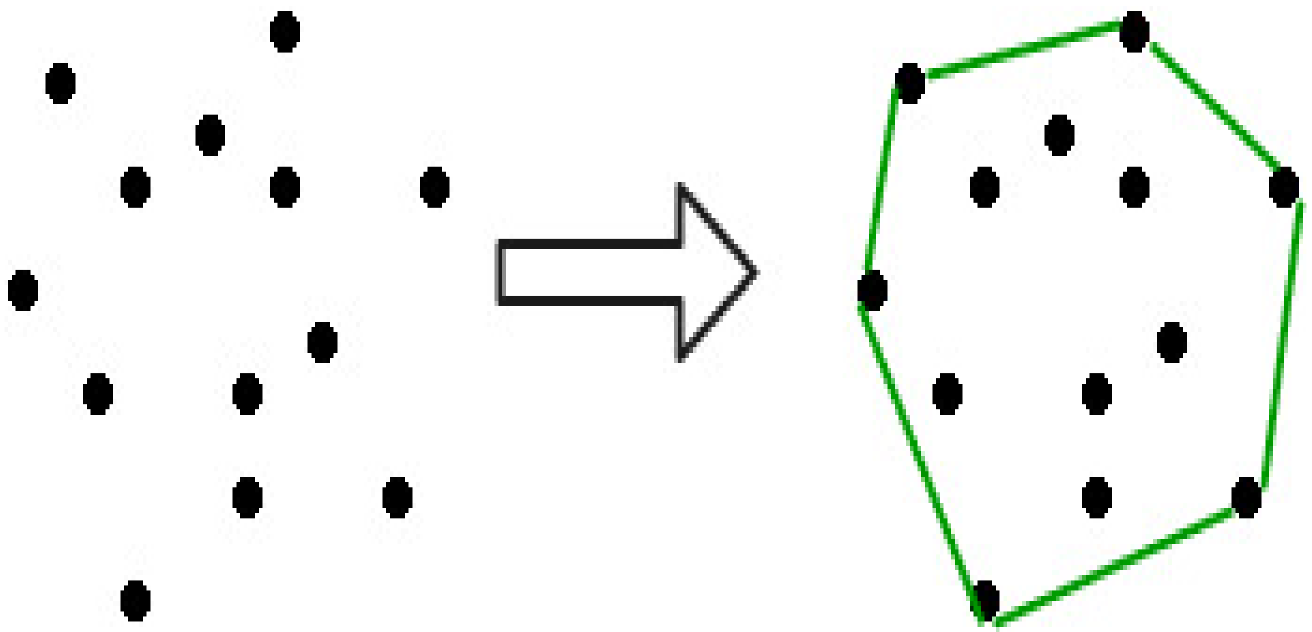




Convex Hull

Convex hull: Convex hull of a set Q of points is the smallest convex polygon P for which each points in Q is either on the boundary of P or in its interior.

Example:



Convex hull

Convex Hull

There are many algorithms for computing convex hull. But here, we shall study only two algorithms.

1. QuickHull
2. Divide and Conquer

QuickHull Algorithm

- To compute the convex hull of a set X of n points in the plane, an algorithm is designed. This algorithm, called QuickHull, first identifies the two points (call them p_1 and p_2) of X with the smallest and largest x-coordinate values. Assume now that there are no ties. Both p_1 and p_2 are extreme points and part of the convex hull.

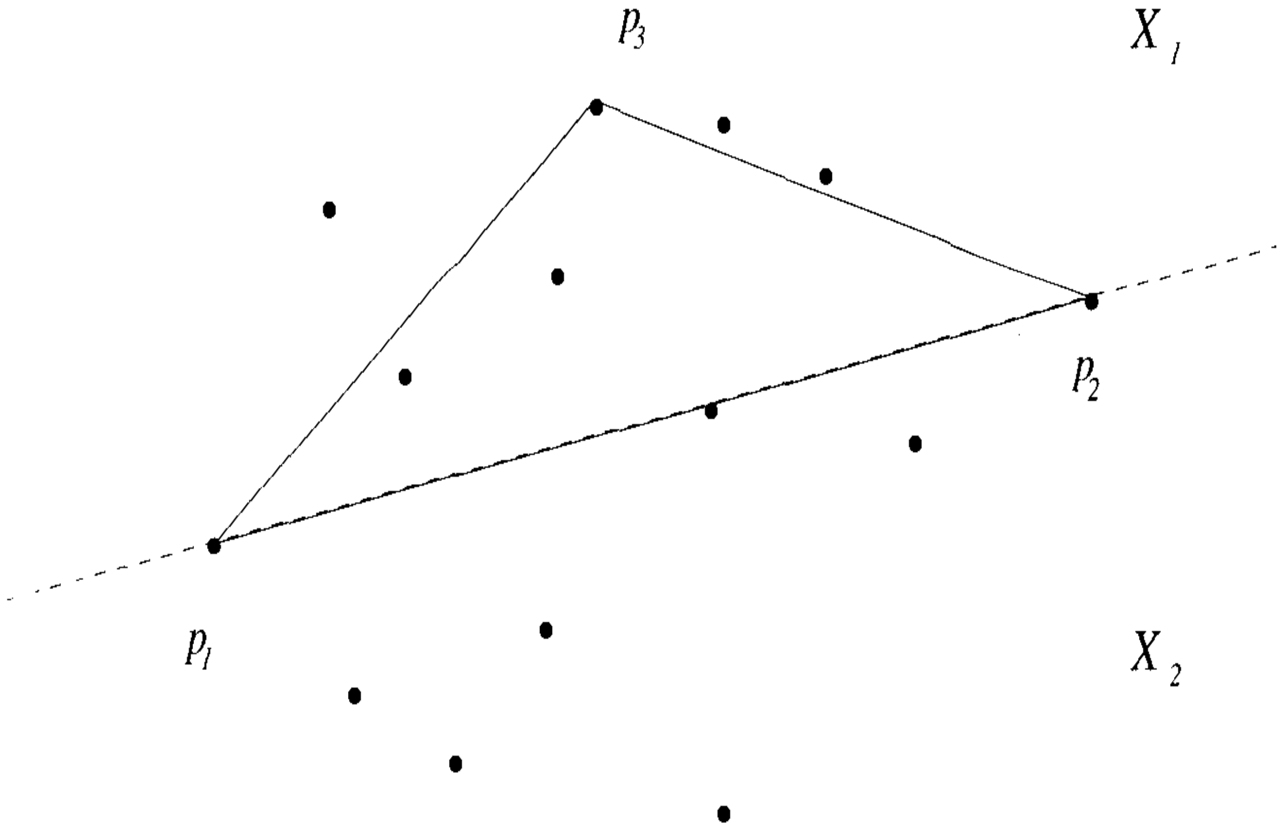
- The set X is divided into X_1 and X_2 so that X_1 has all the points to the left of the line segment (p_1, p_2) and X_2 has all the points to the right of (p_1,p_2). Both X_1 and X_2 include the two points p_1 and p_2. Then, the convex hulls of X_1 and X_2 (called the upper hull and lower hull, respectively) are computed using a divide-and-conquer algorithm called QuickHull. The union of these two convex hulls is the overall convex hull.

QuickHull Algorithm

Computation of the convex hull of X_1

- We determine a point of X_1 that belongs to the convex hull of X_1 and use it to partition the problem into two independent subproblems.
- Such a point is obtained by computing the area formed by p_1 , p , and p_2 for each p in X_1 and picking the one with the largest (absolute) area.
- Ties are broken by picking the point p for which the angle pp_1p_2 is maximum. Let p_3 be that point.
- Now X_1 is divided into two parts; the first part contains all the points of X_1 that are to the left of (p_1,p_3) (including p_1 and p_3), and the second part contains all the points of X_1 that are to the left of (p_3, p_2) (including p_3 and p_2).
- All the other points are interior points and can be dropped from future consideration.
- The convex hull of each part is computed recursively, and the two convex hulls are merged easily by placing one next to the other in the right order.

QuickHull Algorithm



Identifying a point on the convex hull of X_1

Running time of algorithm

$T(n) = O(n\log n)$

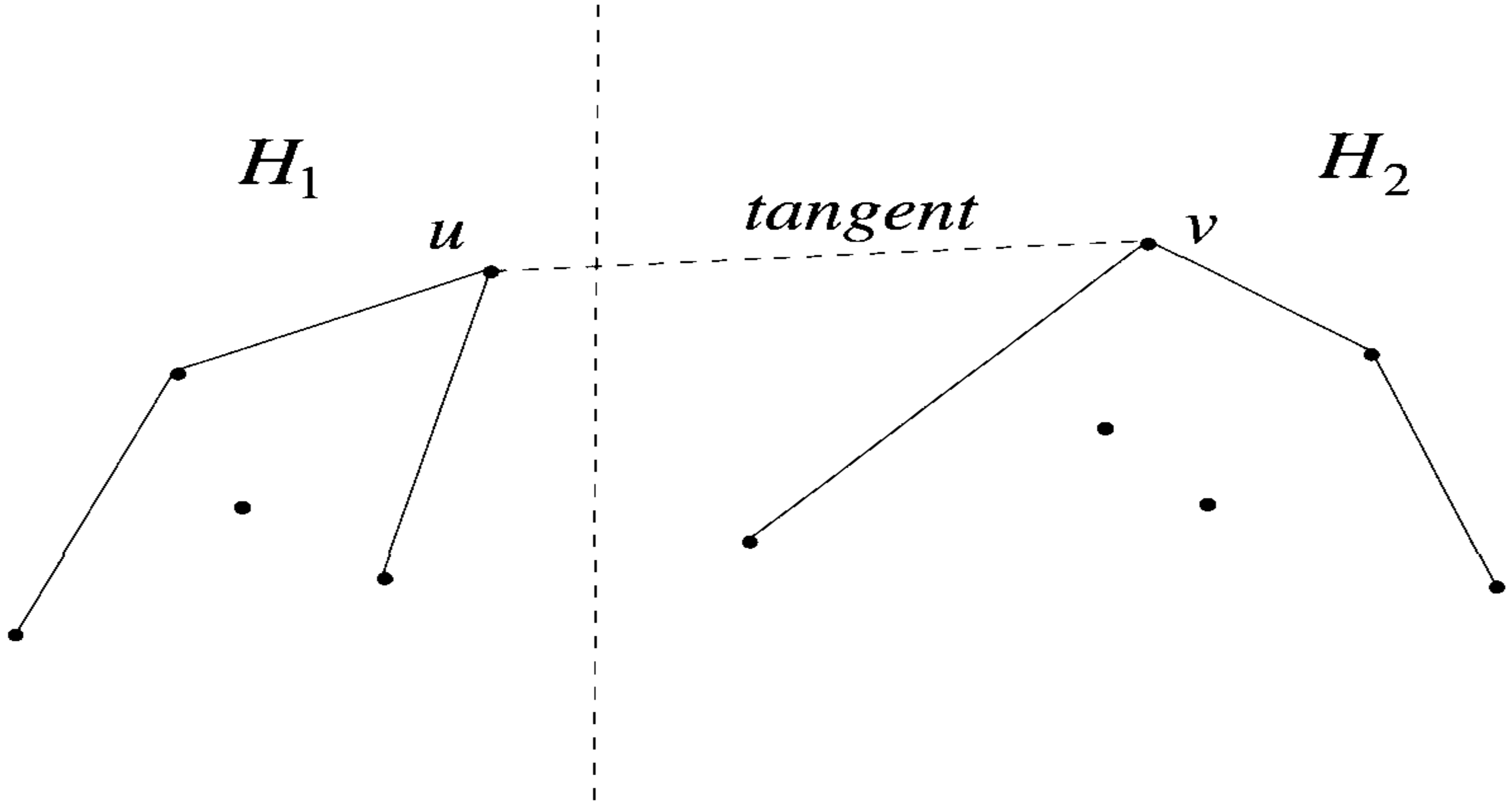
Where, n is the number of points in the set.

Divide and Conquer Algorithm

This algorithm is called DCHull. This algorithm also takes O(nlogn) time. It computes the convex hull in clockwise order.

- Given a set of n points, the problem is reduced to finding the upper hull and the lower hull separately and then putting them together. Since the computations of the upper and lower hulls are very similar, we restrict our discussion to computing the upper hull.
- The divide-and-conquer algorithm for computing the upper hull partitions X into two nearly equal halves. Partitioning is done according to the x-coordinate values of points using the median x-coordinate as the splitter.
- Upper hulls are recursively computed for the two halves. These two hulls are then merged by finding the line of tangent (i.e., a straight line connecting a point each from the two halves, such that all the points of X are on one side of the line).

Divide and Conquer Algorithm



Divide and Conquer Algorithm

- To begin with, the points p_1 and p_2 are identified [where p_1 (p_2) is the point with the least (largest) x-coordinate value].
- All the points that are to the left of the line segment (p_1, p_2) are separated from those that are to the right.
- Sort the input points according to their x-coordinate values.
- Let $q_1, q_2, \ldots \ldots, q_N$ be the sorted order of these points. Now partition the input into two equal halves with $q_1, q_2, \ldots \ldots, q_{N/2}$ in the first half and $q_{N/2+1}, q_{N/2+2}, \ldots \ldots, q_N$ in the second half.
- The upper hull of each half is computed recursively. Let H_1 and H_2 be the upper hulls. Upper hulls are maintained as linked lists in clockwise order.

Divide and Conquer Algorithm

- ❖The line of tangent is then found in $0(\log^2 N)$ time. If (u, v) is the line of tangent, then all the points of H_1 that are to the right of u are dropped.
- ❖Similarly, all the points that are to the left of v in H_2 are dropped. The
- ❖Remaining part of H_1, the line of tangent, and the remaining part of H_2 form the upper hull of the given input set.

- ❖If T(N) is the run time of the above recursive algorithm for the upper hull on an input of N points, then we have

$$T(N) = 2T(N/2) + \log^2 N$$

The solution of this recurrence relation is $T(N) = O(N)$.

- ❖But, the sorting of points takes O(N logN) time, therefore the running time of whole algorithm is O(N logN).

AKTU Examination Questions

1. Describe convex hull problem.
2. What do you mean by convex hull? Describe an algorithm that solves the convex hull problem. Find the time complexity of the algorithm.
3. Given an integer x and a positive number n, use divide & conquer approach to write a function that computes x^n with time complexity O (log n).

Thank You.