# Module 5

## Computational Geometry

→ Computational geometry is a branch of computer science that studies algorithms for solving geometric problems.
→ Applications of computational geometry can be found in robotics, VLSI design, computer vision (3-D construction), computer-aided engineering and geographic information systems (GIS).

→ The input to a computational geometry problem is typically a description of a set of geometric objects, such as a set of points, a set of line segments, or the vertices of a polygon.
→ The output is often a response to a query about the objects, such as whether any of the lines intersect, or perhaps a new geometric object.

Example 1:-
→ Video games, where the scenes have to be displayed containing the environment, as the player who moves around.
→ To take care of what a particular user is seeing in what direction, how will he move around, when an action happens and a lot of other things are taken care of using computational geometry.
→ A data structure known as a binary space partition is commonly used for this purpose

Examples:-

→ Consider the following situation:-
You own a art gallery and want to install security cameras to guard the artwork. But we are under a tight budget. So we we want to use as few cameras as possible. How many cameras do we need? This is commonly called as the Art Gallery problem.

→ When we translate this to the computational geometric notation, the "floor plan" of the gallery is just a simple polygon. And we can prove that n/3 cameras are always sufficient for a polygon of n vertices.

* Line segment and line segment properties →

Point:-
A point (or vector) is defined by its co-ordinates x, y, z. In case of 2-D problems, z=0.

Line:- A line is represented using two distinct points $P_0$ and $P_1$.

Line segment:-
→ It is a part of a line that is bounded by two distinct end points
→ It contains every point on the line between its end points.

→ A closed line segment includes both endpoints, while an open line segment excludes both endpoints.
→ A half-open line segment includes exactly one of the end points.

Vector:-
→ Vector is used to deal with the geometry problems.
→ A vector has a direction and a magnitude.
→ In case of 2-dimensions, a vector is represented as a pair of numbers ie. x and y, which gives both direction and a magnitude.
→ Vectors are commonly used to represent directed line segments.
→ In 3-dimensions,

$$V = V_x i + V_y j + V_z k.$$

There are many mathematical operations that can be performed on vectors:-

1) Addition:-
→ It is the simplest operation.
→ Adding two vectors results into a new vector.
→ If $(x_1, y_1)$ and $(x_2, y_2)$ are two vectors, then the sum of these two vectors is simply $(x_1 + x_2, y_1 + y_2)$.
→ The order doesn't matter, just like regular addition.

2) Dot product:-

→ The dot product is a quantity which is the length of one vector times the length of another vector parallel to that vector.

→ Let u and v be the two vectors, then the dot product is $|u||v|\cos(\theta)$,
where, $\theta$ is the angle between the two vectors u and v.

→ If the two vectors are perpendicular, then the dot product is zero.

→ The dot product is greatest when the lines are parallel.

3) Cross product:-

→ The cross product of two 2-D vectors is $(x_1 y_2 - y_1 x_2)$

→ The cross product takes two vectors and produces a vector that is perpendicular to both vectors.

$$A \times B = |A||B|\sin(\theta)$$

where, $\theta$ is the angle between the 2 vectors.

→ $\theta$ is negative or positive based on the right-hand rule
ie.
$\theta$ is positive, if $A < 180^\circ$ clockwise from B and the cross ~~and~~ product of two parallel vectors is zero.

→ The cross product is also not commutative
ie $u \times v = -v \times u$.

Smita Attarde

4) CCW (Counter clockwise function) →
→ It is the most important operation essential to many geometric solutions.
→ Given the three coordinates of the three points, this function-
→ returns 1 if the points are in counter-clockwise
→ returns -1, if they are in clockwise order.
→ returns 0, if they are collinear.

5) Determining whether consecutive segments turn left or right →

→ It states whether two consecutive line segments $\overline{P_0P_1}$ and $\overline{P_1P_2}$ turn left or right at point $P_1$.
→ For this, we need to determine which way a given angle $\angle P_0P_1P_2$ turns.
→ Cross product does this, without computing the angle.

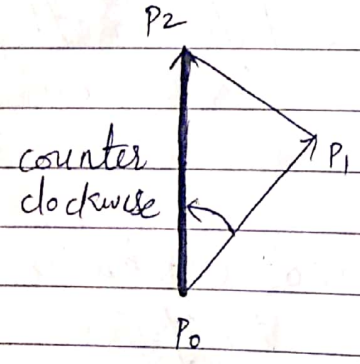



Figure (a)

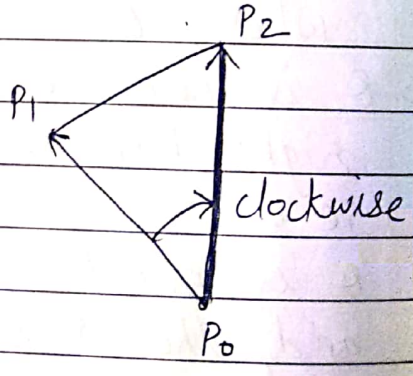



Figure (b)

→ We simply check whether directed segment $\overrightarrow{P_0P_2}$ is clockwise or counter-clockwise relative to the directed segment $\overrightarrow{P_0P_1}$. For this, we compute the cross product as-

$$(P_2 - P_0) \times (P_1 - P_0).$$

→ If the sign of this cross product is negative, then $\overrightarrow{P_0P_2}$ is counter-clockwise with respect to $\overrightarrow{P_0P_1}$, and thus we make a left turn at $P_1$.

→ A positive cross product indicates a clockwise orientation and a right turn.

→ A cross product of zero means that point $P_0$, $P_1$ and $P_2$ are colinear.

* Determining whether any pair of segments intersects →

Problem statement:-
We are given n line segments and we need to find out if any two line segments intersect or not.

Naive algorithm:-
→ A naive solution to solve this problem is to check every pair of lines and check if the pair intersects or not. To check two line segments, we require O(1) time. Therefore, this approach takes $O(n^2)$ time for n line segments.

~~Sweep line algorithm~~ →

* Mathematically, to find intersection point →

Let the given lines be -

$a_1x + b_1y = c_1 \rightarrow (1)$

$a_2x + b_2y = c_2 \rightarrow (2)$

Solving these 2 equations
i.e. multiplying eqn (1) by $b_2$ and eqn (2) by $b_1$,

$a_1b_2x + b_1b_2y = c_1b_2 \rightarrow (3)$

$a_2b_1x + b_2b_1y = c_2b_1 \rightarrow (4)$

subtracting eqn (3) and (2), we get,

$$(a_1 b_2 - a_2 b_1) x = c_1 b_2 - c_2 b_1$$

This gives value of x.
Similarly value of y is calculated.

* Sweep line algorithm →
→ The algorithm first sorts the end points along the x-axis from left to right, then it passes a vertical line through all points from left to right and checks for intersections.

Steps for sweep line algorithm →
1) Let the number of given lines be n. There must be 2n end points to represent these n lines. Sort all the points according to x-coordinates. While sorting, maintain a flag to indicate whether this point is left point of its line or right point.

2) Start from the leftmost point.
Do the following for every point:-
a) If the current point is a left point of its line segment, check for intersection of its line segment with the segments just above and below it.
→ Add its line to active line segments (line segments for which left end point is seen, but right end point is not seen yet).

→ we consider only those neighbours which are still active.

b) If the current point is a right point, remove its line segment from active list and check whether its two active neighbours (ie. points just above and below) intersect with eachother.

→ The step 2 is like passing a vertical line from all points starting from the leftmost point to the rightmost point. That's why this algorithm is called as sweep line algorithm.

* Data structures used:-

→ In step 2, we need to store all active line segments. The following operations need to be performed:-

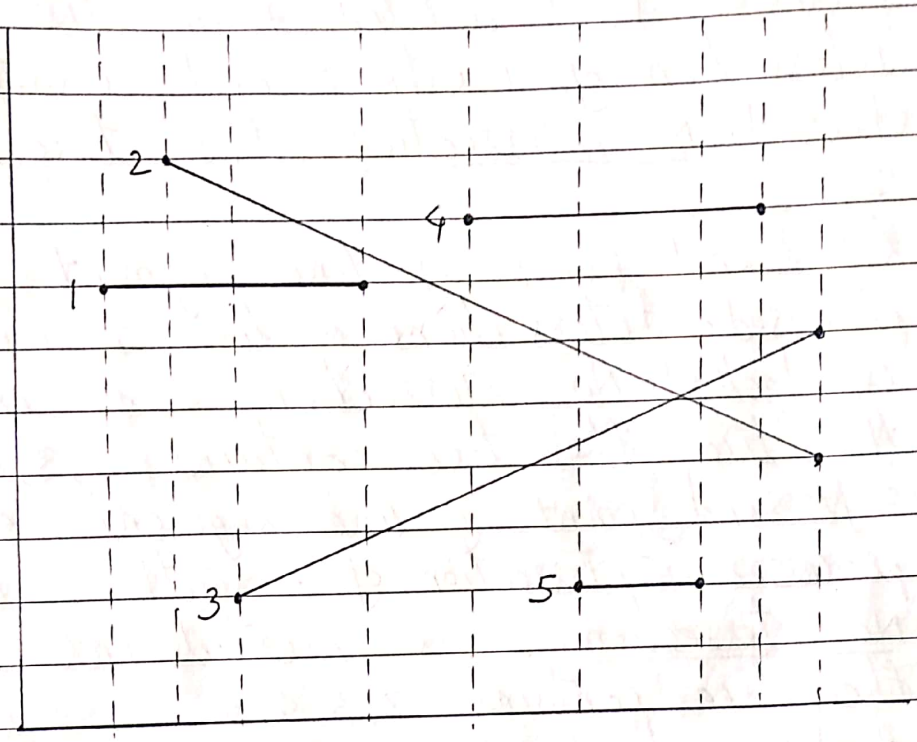
a) Insert a new line segment.
b) Delete a line segment.
c) Find predecessor and successor according to y-coordinate values.

→ All the above operations can be done using a self-balancing binary search tree like AVL tree or red-black tree.

Example:-

→ Consider the following example, which consists of 5 line segments 1, 2, 3, 4 and 5.

→ The dotted lines indicates the sweep lines.




Steps of the algorithm →

→ All the points from left to right are processed one by one. We maintain a self-balancing binary search tree.

1) Left end point of line segment 1 is processed. ie. 1 is inserted into the tree. The tree contains 1. No intersection.

2) Left end point of line segment 2 is processed. Intersection of 1 with 2 is checked. 2 is inserted into the tree. No intersection. The tree contains 1, 2.

Sneha Attarde

3) Left end point of line segment 3 is processed. Intersection of 3 with 1 is checked. No intersection 3 is inserted into the tree. The tree contains 1, 2, 3.

4) Right end point of line segment 1 is processed. 1 is deleted from the tree. Intersection of 1 with 2 and 1 with 3 is checked. No intersection. The tree contains 2, 3.

5) Left end point of line segment 4 is processed. Intersections of line 4 with line 2 is checked. No intersection. 4 is inserted into the tree. The tree contains 2, 3, 4.

6) Left end point of line segment 5 is processed. Intersection of 5 with 3 is checked. No intersection. 5 is inserted into the tree. The tree contains 2, 3, 4, 5.

7) Right end point of line segment 5 is processed. 5 is deleted from the tree. Intersection of 5 with 2 is checked. No intersection. The tree contains 2, 3, 4.

8) Right end point of line segment 4 is processed. 4 is deleted from the tree. The tree contains 2, 3. Intersection of 4 with 2 is checked. No intersection. The tree contains 2, 3.

9) Right end point of line segments 2 and 3 are processed. Both are deleted from the tree and the tree becomes empty. Intersection of 2 and 2 is checked. Intersection of 2 and 3 is reported.

* Time Complexity →
→ The first step is sorting which takes O(n log n) time.
→ The second step processes 2n points and for processing every point, it takes O(log n) time
→ Thus, the overall time complexity is O(n log n).

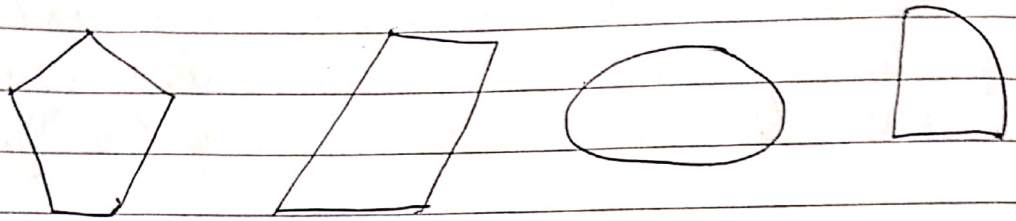
* Finding the Convex Hull →

polygon:-
→ A polygon is just a collection of line segments, forming a cycle, and not crossing eachother. We can represent it as a sequence of points, each of which is just a pair of coordinates
→ For eg, the points (0,0), (0,1), (1,1), (1,0) forms a square.
→ All the sequences of points does not form a polygon. For example, the points (0,-1), (0,1), (1,0), (0,-1) would result in two segments that cross eachother.

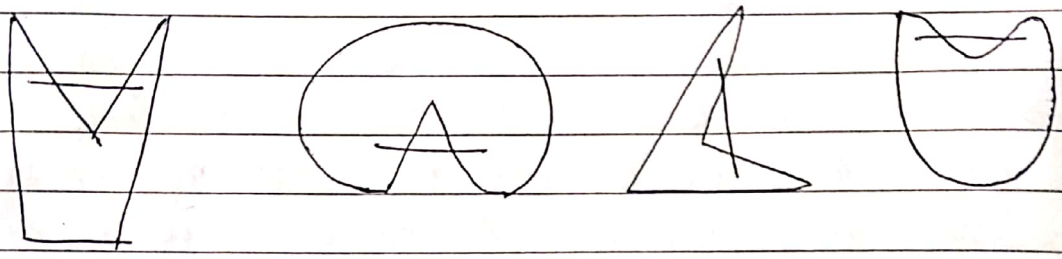
Convex polygon →
→ A figure is convex if every line segment drawn between any 2 points inside the figure lies entirely inside the figure.
→ A figure that is not convex is called a concave figure.

Example:-

Conven figures:-



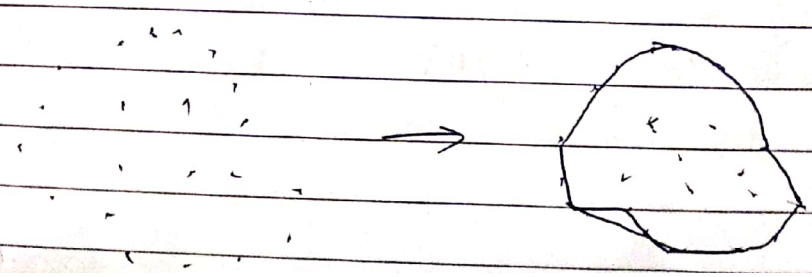
Concave figures:-



→ A conven polygon can also be defined formally as having the property that any two points inside the polygon can be connected by a line segment that doesn't cross the polygon.

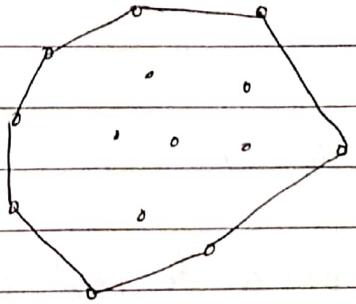
Conven Hull:-

→ Given a set of points in the plane, the conven hull of the set is the smallest conven polygon that contains all the points of it.



Let S be the set of points in the plane.

Intuition :- Imagine each point of S as being a nail sticking out from a board. The convex hull of S is then the shape formed by a tight rubber-band that surrounds all the nails.



Formal definition :- The conven hull of S is the smallest conven polygon that contains all the points of S.

## Graham's scan algorithm to find convex hull

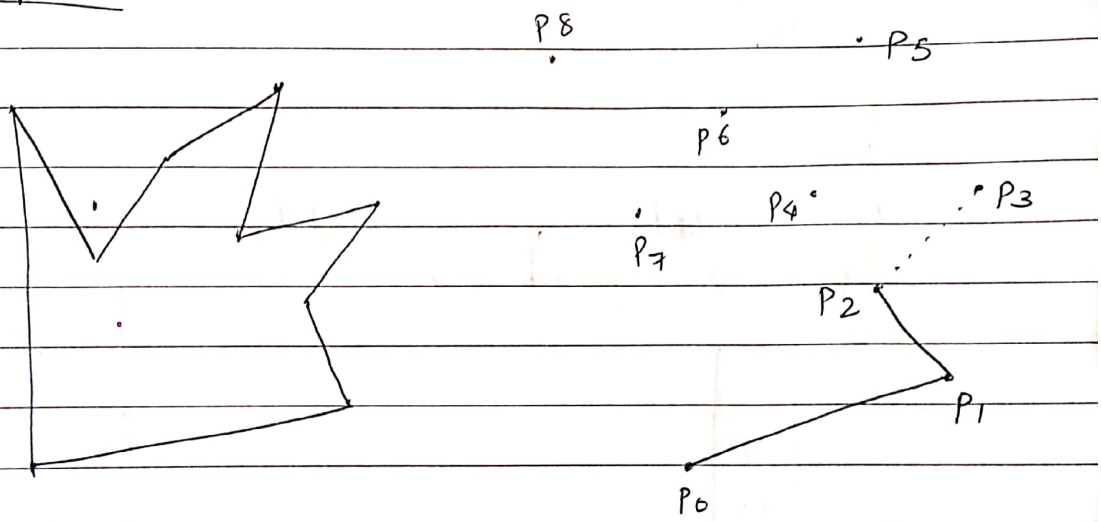
1) Find $p_0$ ie. the point with the minimum y-coordinate. This point is called the pivot.
2) Sort all the remaining points in order of their polar angle from $p_0$.
3) Initialize a stack, S.
4) push (S, $p_0$)
5) push (S, $p_1$)
6) push (S, $p_2$)
7) for i = 3 to n
   {
   while (angle formed by top next(S), top(S) and $p_i$ makes a right turn)
   {
   pop(S)
   }
   push (S, $p_i$)
   }
8) Return S.

Time complexity:-

→ Line 1 requires $O(n)$ time to find $P_0$
→ Line 2 requires $O(n \log n)$ time to sort using heap sort or merge sort.
→ Lines 3, 4, 5, 6 takes $O(1)$ time each.
→ The amortised cost of each ~~co~~ iteration of the while loop takes $O(1)$ time. So, the for loop takes $O(n)$ time.
→ Hence, the total time is-

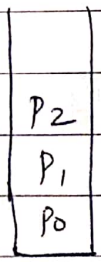
$$T(n) = O(n) + O(n \log n) + O(1) + O(n)$$
$$= O(n \log n)$$

Example:-



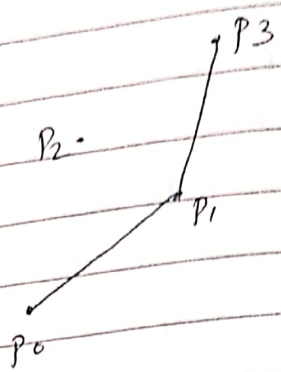

Concave polygon

$P_0, P_1, P_2$ are in stack



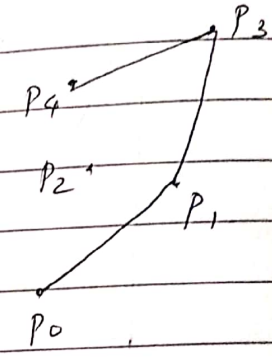

Now, topnext (S) = $P_1$ , top (S) = $P_2$
∴ $\angle P_1 P_2 P_3$ is right turn, so pop $P_2$

| |
|---|
| $P_1$ |
| $P_0$ |



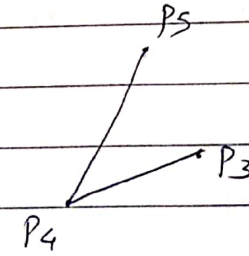

Now, $\angle P_0 P_1 P_3$, is a left turn, so push $P_3$

| |
|---|
| $P_3$ |
| $P_1$ |
| $P_0$ |



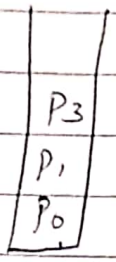

Now, $\angle P_1 P_3 P_4$, is a left turn, so push $P_4$

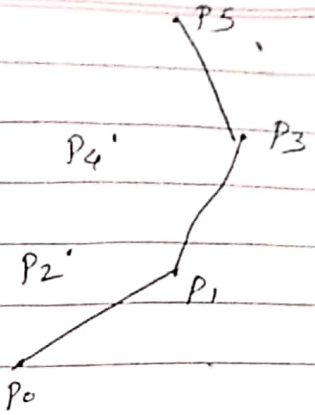
| |
|---|
| $P_4$ |
| $P_3$ |
| $P_1$ |
| $P_0$ |



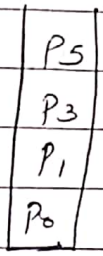

$\angle P_3 P_4 P_5$, is a right turn, so pop $P_4$

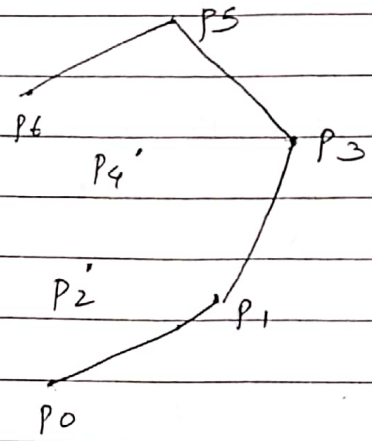
Smita Attarde



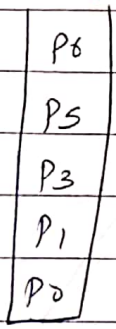



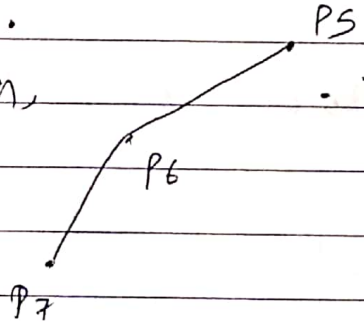

$\angle P_1 P_3 P_5$ is a left turn, so push $P_5$



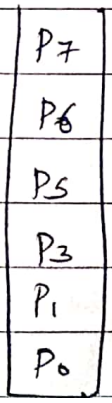




$\angle P_3 P_5 P_6$ is left turn, so push $P_6$

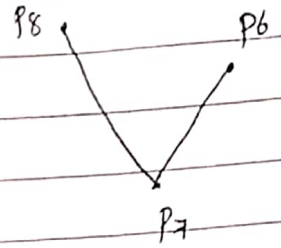



$\angle P_5 P_6 P_7$ is a left turn, so push $P_7$

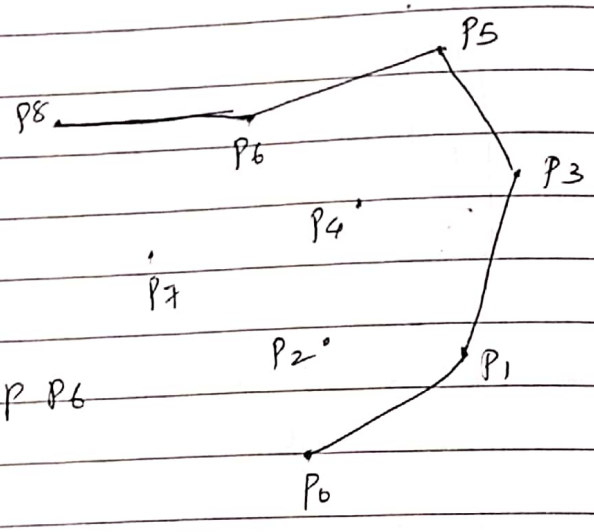


$\angle P_6 P_7 P_8$ is a right turn, so pop $P_7$



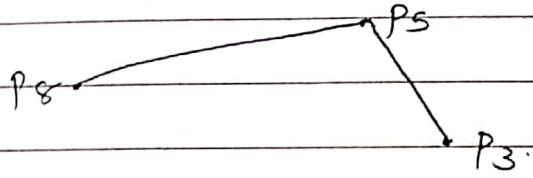

| |
|---|
| P6 |
| P5 |
| P3 |
| P1 |
| P0 |

$\angle P_5 P_6 P_8$ is a right turn, so pop $P_6$



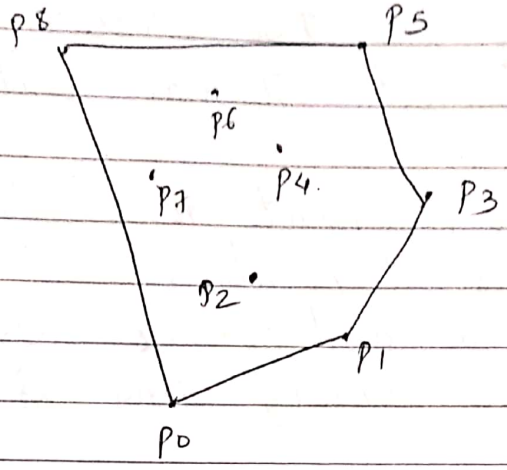

| |
|---|
| P5 |
| P3 |
| P1 |
| P0 |

$\angle P_3 P_5 P_8$ is a left turn, so push $P_8$




| |
|---|
| P8 |
| P5 |
| P3 |
| P1 |
| P0 |

Hence, the final convex hull consists of the points present in the stack.




* Finding the closest pair of points →

problem statement :- Given are n points in the plane, find a pair with smallest Euclidean distance between them.
ie. the distance between this pair of points should be minimum.

→ Distance between the points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is given by -

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

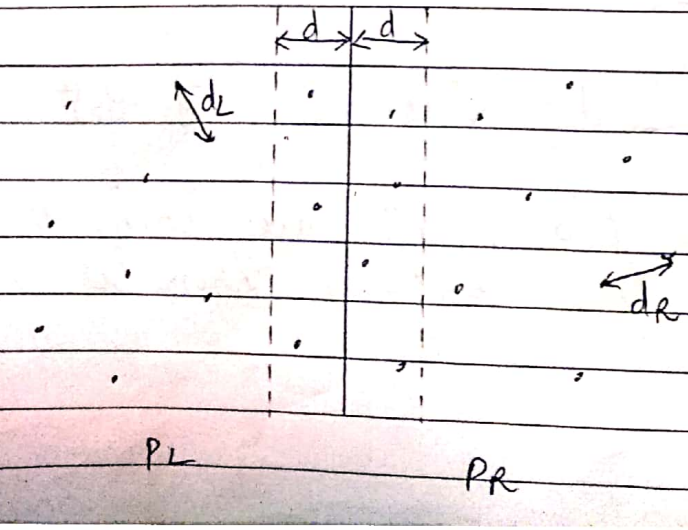
→ If any two points are coincident, then the distance between them is zero.

Smita Attarde

* Applications →
Traffic control system → A system for controlling air or sea traffic might need to identify the two closest vehicles in order to detect potential collisions.

→ Naive algo (ie. Brute force algorithm) requires $O(n^2)$ pairs of points.
→ Hence divide-and-conquer algorithm is used, whose running time is $O(n \log n)$.

* Divide-and-conquer algorithm →
1) P is the input set of all points.
2) All these points are also stored in arrays X and Y.
3) The points in array X are sorted according to their x-coordinates, and points in array Y are sorted according to their y-coordinates.
4) Divide :- Find a vertical line l that bisects the point set P into two sets $P_L$ and $P_R$ such that,
$P_L = P/2$ and $P_R = P/2$.



→ Divide the array X into arrays $X_L$ and $X_R$ which contains the points of $P_L$ and $P_R$ respectively.
→ Similarly, divide the array Y into arrays $Y_L$ and $Y_R$, which contains the points of $P_L$ and $P_R$ respectively.

5) Conquer:-
Two recursive calls are made:-
a) First recursive call -
input is $P_L$, $X_L$, $Y_L$
output is $d_L$ ie. distance between the closest pair of points in $P_L$

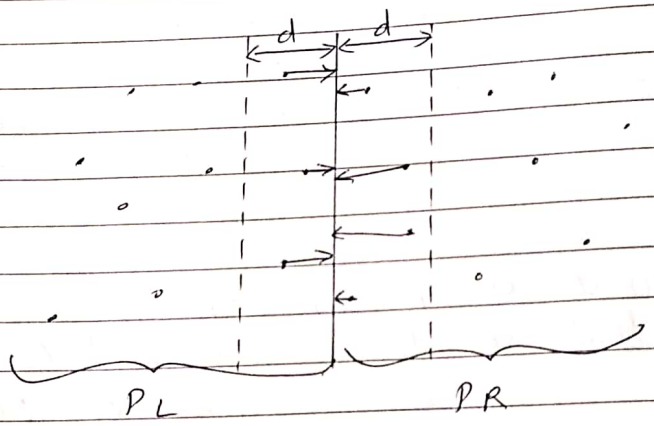
b) Second recursive call -
input is $P_R$, $X_R$, $Y_R$
output is $d_R$ ie. distance between the closest pair of points in $P_R$

$$\boxed{d = \min(d_L, d_R)}$$

6) Combine:-
→ The closest pair is either the pair with distance d found by one of the recursive calls, or it is a pair of points with one point in $P_L$ and other in $P_R$.

#)

7) Now, consider the pairs, such that one point in the pair is from left half ($P_L$) and other is from right half ($P_R$).
→ Consider the vertical line passing through P(~~at~~), and find all points whose x-coordinate is closer than d to this middle vertical line. Build an array strip[] of all such points.




8) Sort this array strip[] according to their y-coordinates.
9) Find the smallest distance in strip[].
10) Finally, return the minimum of d and distance calculated in the above step (step 9)

* Time complexity :-
The overall complexity of the algorithm is O(n log n).

Smita Attarde