COMPUTATIONAL GEOMETRY

Lec 1

Before starting with the theory, lets first look at some scenarios where we need convex hulls. Convex hulls have many applications

(1) Shape Analysis

(2) Collision Detection & Avoidance

In robotics, when we want to move an object from one place to other, we should avoid any collisions on the way. objects encountered may not necessarily be of regular shapes. They can be arbitrary shaped. In order to avoid collision, the object is approximated in terms of convex hull of that object. If the moving object avoids that convex hull, it avoids the original object.

Whenever we have any object or a set of points in an n-dimensional plane, we have to find the smallest bounding box that encloses all of them, it is found using convex hulls.

DEFINITION

Convex object: Take any two points on an object. If all the points on the line segment joining these points lie inside this object and this is true for all possible pairs of points, then the object is called a convex object.

eg: ball, ellipsoid, any convex polygon in 2D

This definition is general, thus valid for all dimensions and all types of objects (open or closed).

eg:- paraboloid (convex but open)

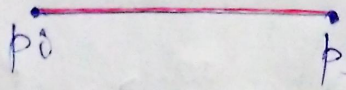
Convex Combination

Let there be two points p_i and p_j

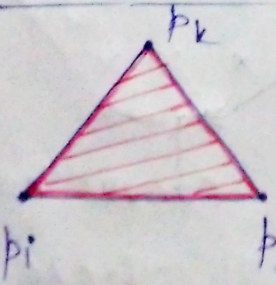
the convex combination of these two points is given by the set of points

$$\{p_{cc} : p_{cc} = \alpha_1 p_i + \alpha_2 p_j,\ \alpha_1 + \alpha_2 = 1,\ \alpha_1 \geq 0,\ \alpha_2 \geq 0\}$$

On taking a closer look, we see that the convex combination of these 2 points is the line segment joining them.



EXTENSION TO 3 POINTS



Convex combination of these 3 points is

$$\{p_{cc} : p_{cc} = \alpha_1 p_i + \alpha_2 p_j + \alpha_3 p_k,\ \alpha_1 + \alpha_2 + \alpha_3 = 1,\ \alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0\}$$

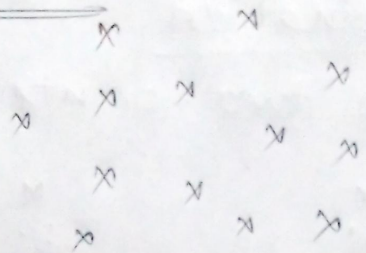
For 3 points, the convex combination is the set of all points lying on the boundary or inside the triangle.

This can be generalised to higher dimensions.

Let us come back to convex hulls, convex hulls become increasingly complicated in higher dimensions. Hence, we will limit our discussion to 2D.

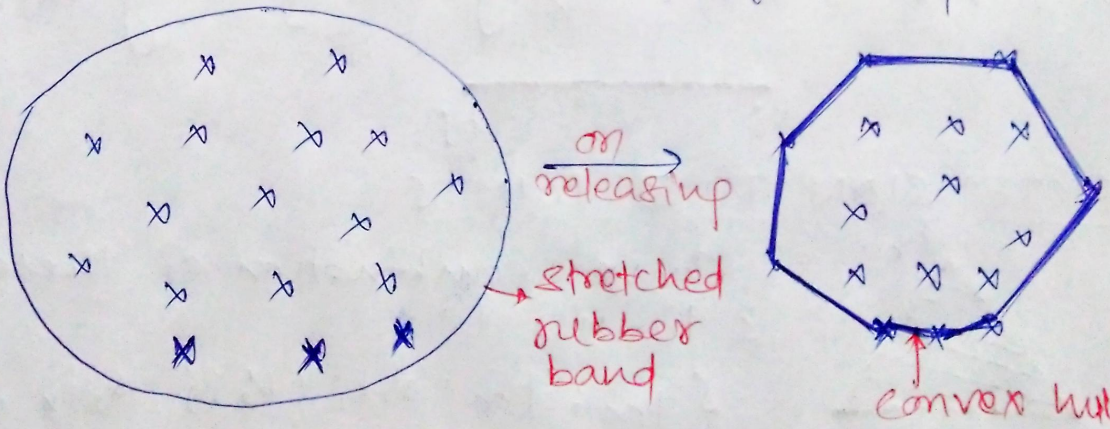
Our first problem is:

Given n arbitrary points on a plane, find their convex hull.



For more understanding of this problem, let us take it in an informal way.

Assume the points to be pins on a whiteboard. If we stretch a rubberband around these points, the shape it will take on releasing will be a convex polygon and is called a convex hull of these points



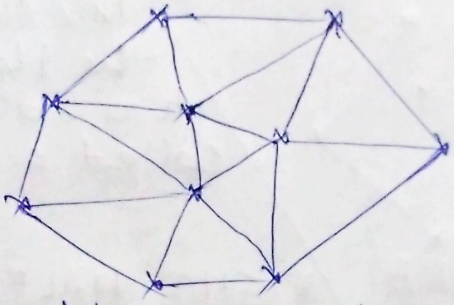
Let us look at another aspect of these convex hulls. The convex hull for d-dimensional points will be the union of convex combinations of all possible (d+1) tuples of points.

Since we are dealing with 2D points, we need 3-tuples.

Take the convex combination of all 3-point tuples. Union of all of these convex combinations will be the convex hull of these n points.

Hence, convex hull is the union of convex combinations of all possible (d+1) for lower points.

Here, 'lower' is redundant, 'd+1' is the sufficient condition.

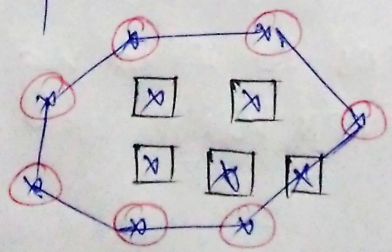


Now, the problem requires us to output the convex hull. It can be described using

(1) listing all of its boundary points (also called extreme points)

(2) listing the edges in some particular order (CLOCKWISE or COUNTER CLOCKWISE)

Note:- A point lying on the edge of the boundary is not considered as an extreme point

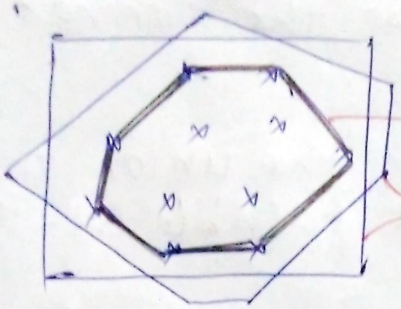


○ EXTREME POINTS

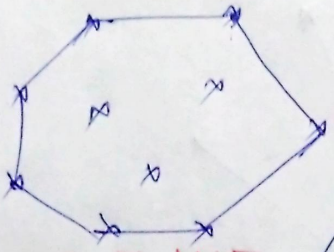
□ NOT EXTREME POINTS

Let us look at some other definitions of convex hull.

(1) convex hull is the intersection of all convex polygons enclosing the given n points.



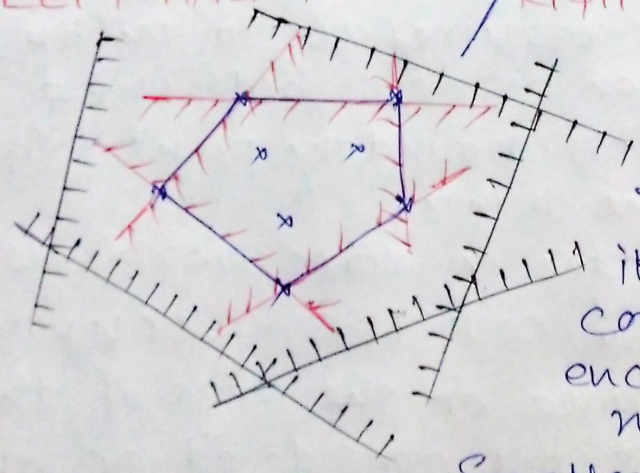
(2) Convex hull is the intersection of all half planes enclosing the n points.



This line divides the plane into two halves, each called a half plane.

LEFT HALF PLANE

RIGHT HALF PLANE



Convex hull has the property that it is the smallest convex polygon enclosing these n points.

Smallest in terms of ① perimeter ② area

Our problem reduces to finding out the vertices of the convex hull or extreme points.

for simplicity, we assume that these n points are in general position.

↳ NO 3 POINTS ARE COLLINEAR

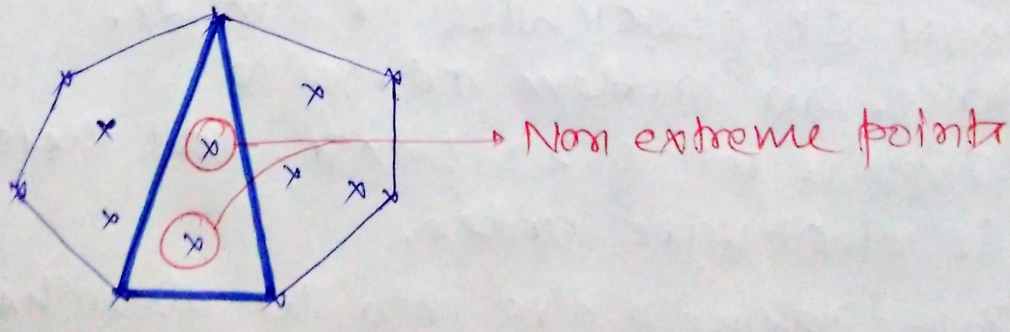
General position varies from situation to situation and problem to problem.

NOTE: This assumption is only to understand the algorithm. Once we understand it, we can make minor modifications to it or in the code to handle these cases as well.

FINDING THE EXTREME POINTS

One approach to find the extreme points is to remove all those points which are non-extreme. The remaining points are the extreme points.

We classify points to be non extreme if it lies inside a triangle with 3 different points as its vertices.



We now look at the pseudo code for the algorithm shown above

PSEUDO CODE:

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            for l = 1 to n do
                if (l ≠ i ≠ j ≠ k) then
                    if pl ∈ Δ pi pj pk then
                        pl is non extreme
```

Since there are 4 for loops in the above code, each ranging from 1 to n.

TOTAL TIME COMPLEXITY = $O(n^4)$

THIS IS VERY HIGH TIME COMPLEXITY.

The output we get from this algo contains the extreme points of the convex hull in arbitrary order. If we want them in a particular order (CLOCKWISE or COUNTER CLOCKWISE) then we need to do some extra work.

Let us take the case of CLOCKWISE ordering. COUNTER CLOCKWISE is similar.

Hence, our problem reduces to

Given a set of 2D points, sort these points in clockwise order.

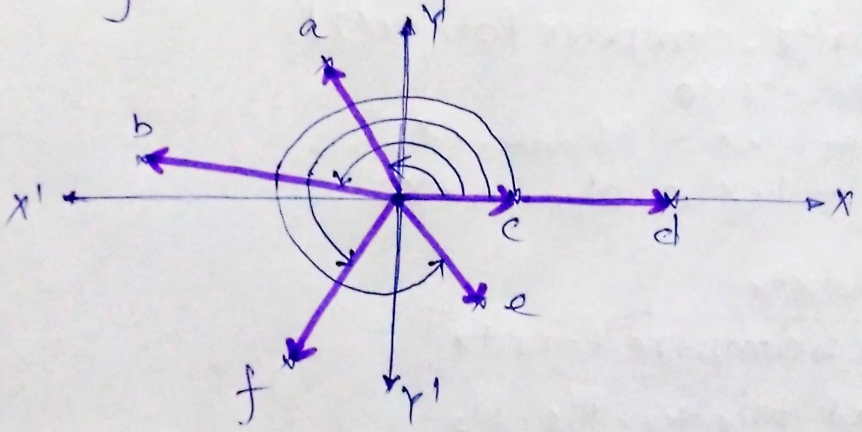
Before taking a dive into the algorithm, let us have a look at a concept named center point. It is an arbitrary point on the 2D plane wrt which we view all of the given points.

ALGO

We need to choose a center point. In our case we choose it to be the mean of the given points.

Then we estimate the angle between the line segment joining center to point and the horizontal line passing through center.

Now, we sort the points based on ~~decreasing~~ angles. In case of any ties, we break the tie by distance from the center



Hence, the order is

e, f, b, a, c, d,

d is farther from the center.

Let us now look at the pseudo code for the algorithm mentioned above.

Algorithm 1 : sortPoint(W)

Data: Read Points
Result: Return the points ordered clockwise
$pt_{center} \leftarrow \{0,0\}$;
for p in Points do
    $pt_{center}.x \leftarrow pt_{center}.x + p.x$;
    $pt_{center}.y \leftarrow pt_{center}.y + p.y$;
end
$pt_{center}.x \leftarrow pt_{center}.x / size(Points)$;
$pt_{center}.y \leftarrow pt_{center}.y / size(Points)$;
for p in Points do
    $p.x \leftarrow p.x - pt_{center}.x$;
    $p.y \leftarrow p.y - pt_{center}.y$;
end
sort (points, comparePoints())
for p in Points do
    $p.x \leftarrow p.x + pt_{center}.x$;
    $p.y \leftarrow p.y + pt_{center}.y$;
end
return Points;

Algorithm 2 : comparePoints

Data: Read pt_{center}, pt_1, pt_2
Result: Return true if first point order is less than second point
$angle_1 \leftarrow getAngle(\{0,0\}, pt_1)$;
$angle_2 \leftarrow getAngle(\{0,0\}, pt_2)$;
if $angle_1 < angle_2$ then
    return true;
end
$d1 \leftarrow getDistance(\{0,0\}, pt_1)$;
$d2 \leftarrow getDistance(\{0,0\}, pt_2)$;
if $(angle_1 == angle_2)$ and $(d1 < d2)$ then
    return true;
end
return false;

Algorithm 3: getAngle

Data: Read ptcenter, pt
Result: return angle from 0 to 2π for pt relative to ptcenter

```
x ← pt.x − ptcenter.x;
y ← pt.y − ptcenter.y;
angle ← arctan2(y, x);
if angle ≤ 0 then
    angle ← 2π + angle;
end
return angle;
```

Algorithm 4: getDistance

Data: Read pt_1, pt_2
Result: return distance between the two points

```
x ← pt1.x − pt2.x;
y ← pt1.y − pt2.y;
return sqrt(x*x + y*y);
```

TIME COMPLEXITY

$O(n)$ + $O(n \log n)$ + $O(n)$

finding point relative to center — sorting — finding point relative to origin

$= O(n \log n)$

TOTAL TIME COMPLEXITY

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

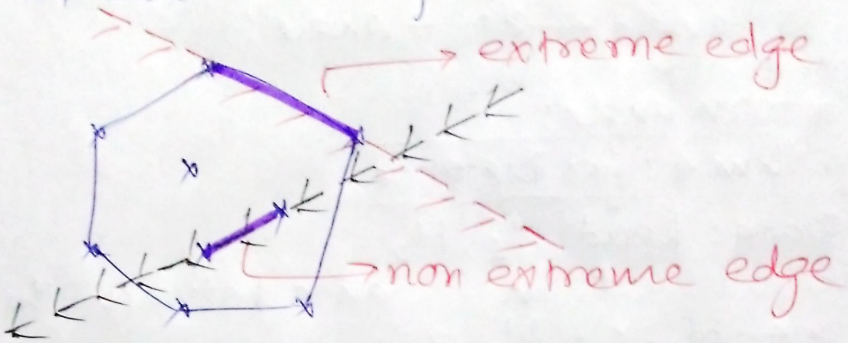
Finding center — sorting points in CW relative to center.

SPACE COMPLEXITY

Since no extra space is required, space complexity is $O(1)$

Coming back to convex hulls, we now look at a different approach to finding convex hulls. We do this by finding the extreme edges (edges whose endpoints lie on convex hull).

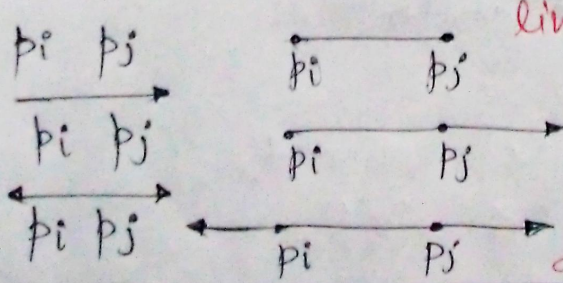
Consider a line passing through an edge. If all of the points are in one half plane of that line, then this edge is an extreme edge.



Let us have a look at the pseudo code for this algorithm.

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            if (i ≠ j ≠ k) then
                if pk is not on the left
                    of pipj or lies on the
                    line segment pipj then
                        pipj is not an extreme edge
```

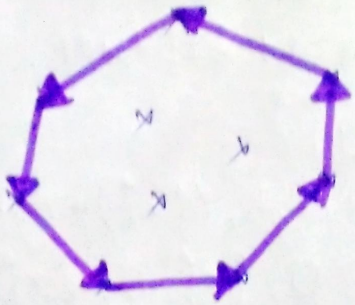
NOTATION



line segment b/w pi & pj

ray emanating from pi passing through pj

line extending in both dirns & passing through pi & pj



EXTREME EDGES
(dirⁿ is COUNTER CLOCKWISE)

For every extreme edge p_ip_j, we will consider both $\overrightarrow{p_ip_j}$ & $\overrightarrow{p_jp_i}$. One of them will definitely be in our result. For one of the two, all points will be towards left & for the other, all points will be towards right.

TIME COMPLEXITY

Since there are 3 for loops, each ranging from 1 to n, time complexity is $O(n^3)$.

Now, the only point left to discuss is how to determine the location of a point wrt a point, i.e. whether a point lies to the LEFT, RIGHT or ON THE LINE.

To do this, we take the help of CROSS PRODUCT.

Definition

Cross product or vector product is a binary operation on two vectors & is denoted by the symbol '×'. Given two linearly independent vectors a and b, the cross product, a×b is a vector that is perpendicular to both a & b & thus normal to the plane containing them. The magnitude of cross product is equal to the area of the parallelogram with a & b as its sides.

Let a & b be two 2D vectors. We derive an expression for their cross product.

$$\vec{a} \times \vec{b} = (a_x \hat{i} + a_y \hat{j}) \times (b_x \hat{i} + b_y \hat{j})$$
$$= a_x b_x (\hat{i} \times \hat{i}) + a_x b_y (\hat{i} \times \hat{j}) + a_y b_x (\hat{j} \times \hat{i}) + a_y b_y (\hat{j} \times \hat{j})$$
$$= (a_x b_y - a_y b_x) \hat{k} \quad \{\hat{i} \times \hat{i} = 0, \hat{j} \times \hat{j} = 0, \hat{i} \times \hat{j} = \hat{k} = -\hat{j} \times \hat{i}\}$$

Another formula for $\vec{a} \times \vec{b}$ is

$$\vec{a} \times \vec{b} = \|\vec{a}\| \|\vec{b}\| \sin\theta \, \hat{n}$$

For 2D vectors, $\hat{n}$ is $\hat{k}$ (+ve z axis).
Hence, sign of $\vec{a} \times \vec{b}$ is determined by $\sin\theta$ where θ is angle from a to b in COUNTER CLOCKWISE direction.

$$\sin\theta = \begin{cases} +ve, & 0 < \theta < \pi \\ -ve, & \pi < \theta < 2\pi \\ 0, & \theta = 0 \text{ or } \theta = \pi \end{cases}$$

$0 < \theta < \pi$ ⟶ b lies to the left of a
$\pi < \theta < 2\pi$ ⟶ b lies to the right of a
$\theta = 0$ or $\theta = \pi$ ⟶ b lies on a

Let us see how we apply this to our problem.
Let the edge be between p_i & p_j ⟶
we have to determine location of p_k wrt $p_i p_j$.

Let $\vec{a} = \overrightarrow{p_i p_k} = \overrightarrow{p_k} - \overrightarrow{p_i} = (x_k - x_i)\hat{i} + (y_k - y_i)\hat{j}$
Let $\vec{b} = \overrightarrow{p_i p_j} = \overrightarrow{p_j} - \overrightarrow{p_i} = (x_j - x_i)\hat{i} + (y_j - y_i)\hat{j}$

if sign of $\vec{a} \times \vec{b}$ is
- +ve: p_k lies to the left of the line
- -ve: p_k lies to the right of the line
- 0: p_k lies on the line.

CROSS PRODUCT can be easily calculated using $a_x b_y - a_y b_x$. Hence, TIME & SPACE COMPLEXITY = O(1)

COMPUTATIONAL GEOMETRY

Lec 2

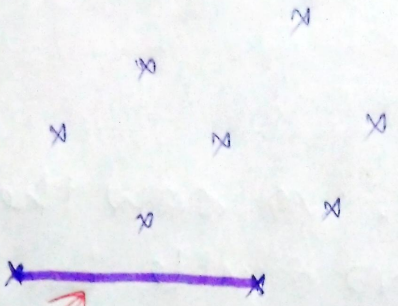
In the last lecture, we saw the definition of convex hull & two simple brute force algorithms that compute the extreme points & extreme edges respectively. But their complexities were $O(n^4)$ & $O(n^3)$ respectively which are quite high. In this lecture, we discuss other algorithms which give a better time complexity than the ones offered by the previous algorithms.

Today, we will look at GIFT WRAPPING or JARVIS MARCH algorithm.

It is named so because wrapping a gift in a gift wrapping paper follows a similar procedure.

This isn't entirely a new algorithm but a slight improvement over the previous ones.

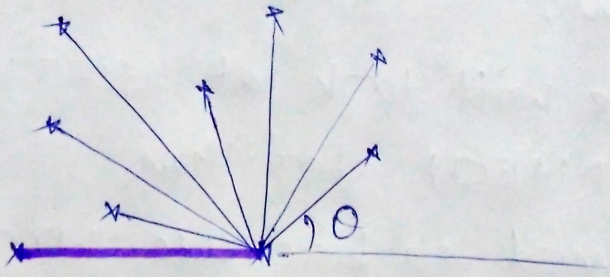
In this algorithm, we assume that we know one edge on the boundary of the convex hull.



Let's assume that this edge was given to us.

Then, we anchor our search for the next vertex on the boundary of the convex hull on this edge.

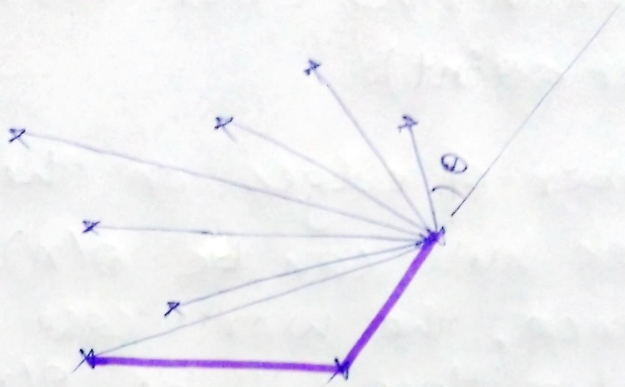
Then, we measure the angles of each point wrt the line passing through that edge.



The point making the smallest angle is the next vertex on the boundary of the convex hull.

Add the edge b/w this newly found point and the pivot point to the convex hull.

Continue the same procedure taking the newly formed edge as the anchor this time



Keep repeating the procedure until you reach the starting vertex again.

Taking a close look at this, it is quite similar to the previous algorithm. In the previous algorithm, we used to check if every point is to the left of this edge or not. Here, we choose the vertex with the smallest angle and this is sufficient to claim that all other points are to the left of this edge.

This is an easy observation & we exploit this observation to lower the time complexity by a factor of n.

TIME COMPLEXITY ANALYSIS

Finding the angle of $(n-2)$ points relative to the anchoring line takes $\Theta(n)$ time. ($\Theta(1)$ per point)

The number of edges on the boundary of the convex hull can be $\Theta(n)$ in the worst case. Hence, the worst case time complexity of the algorithm will be $\Theta(n^2)$.

Let us consider a general case scenario. If there are h edges on the boundary of the convex hull, the time complexity comes out to be $\Theta(nh)$.

In the best case, h could be a constant, i.e., $h = \Theta(1)$.

In the worst case, h could be very close to n, i.e., $h = \Theta(n)$.

GENERAL CASE TIME COMPLEXITY $\boxed{\Theta(nh)}$

Generally, we express the time complexity of any algorithm in terms of number of inputs, but in this particular case, we are also expressing the time complexity in terms of size of output.

BEST CASE : $\Theta(n)$

WORST CASE : $\Theta(n^2)$

The time complexity of the algorithm varies with the size of the output. Hence, this algorithm belongs to the class of **OUTPUT SENSITIVE** algorithms.

Let us assume an output sensitive algorithm has computational complexity $O(\log n)$. Total time complexity will therefore be $O(\log n + k)$

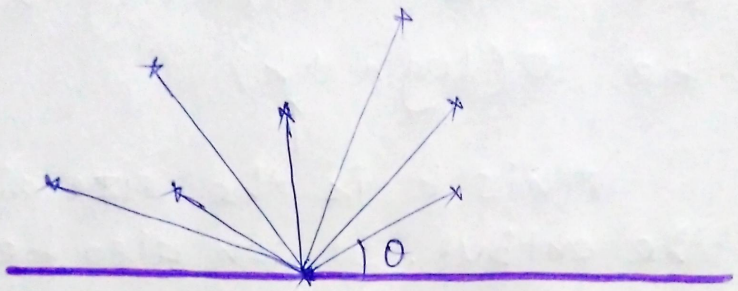
↑

This k is the size of the output. It can also be $O(n^2)$. We do not have any control over this. We only have control over the complexity of computations done. Hence, for output sensitive algorithms, we express the complexity as sum of computational complexity & output complexities. This gives a much better picture than only writing overall time complexity.

NOTE: This algorithm outputs the vertices of the convex hull in counter clockwise order.

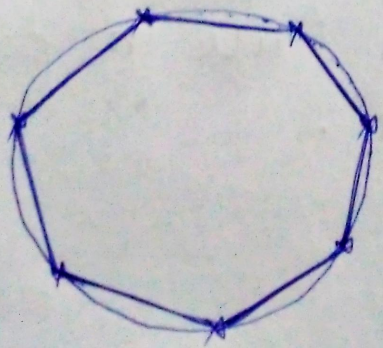
Now, the question boils down to finding the first edge. We assumed that this edge was given to us but in reality, we need to find this edge.

The solution is to find the point with minimum y-coordinate.



Anchor the search for the next point on the horizontal line passing through this point. Choose the point that makes minimum angle with this line. The resulting edge b/w this newly found point and our pivot will be a part of the convex hull.

Worst case time complexity $O(n^2)$ is achieved when all points lie on the convex hull. Let us look at such a case.



All points are extreme points

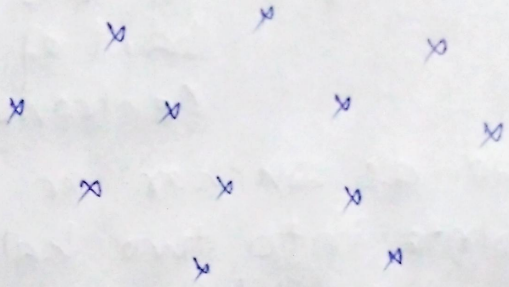
Time complexity is $O(n^2)$.

Let us look at another algorithm for computing the convex hull of a set of n points. It is called **QUICK HULL**

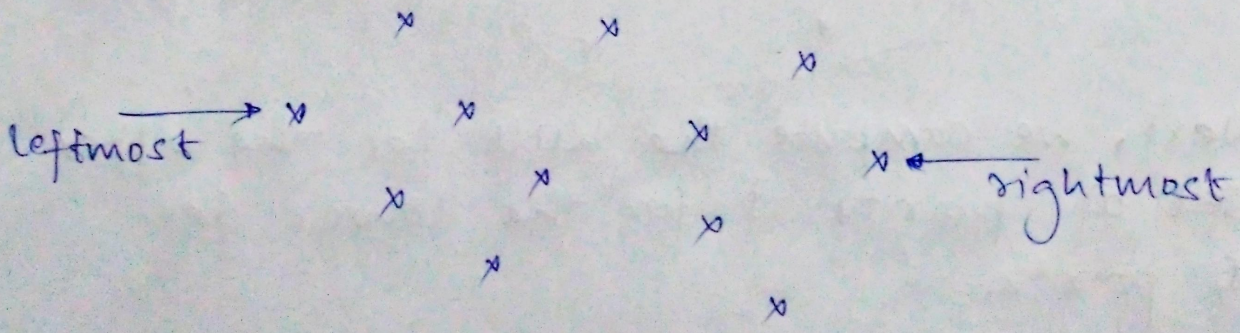
This algorithm is quite similar to QuickSort algorithm, hence the name Quick Hull.

In this algorithm too, we have a pivot & we partition the points into two sets with respect to the pivot.

Let us look at an example

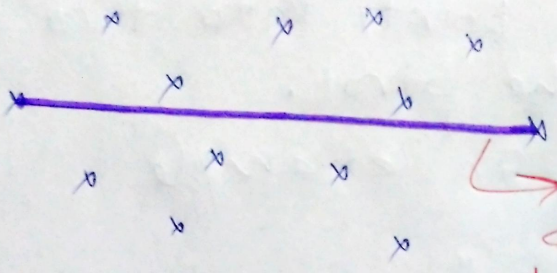


First, we have to identify two points that lie on the convex hull. We choose the point having the minimum x-coordinate & the one having the maximum x-coordinate as the two points



If there are multiple points having minimum x-coordinate, choose the topmost point among them.

If there are multiple points having maximum x-coordinate, choose the bottommost point among them.

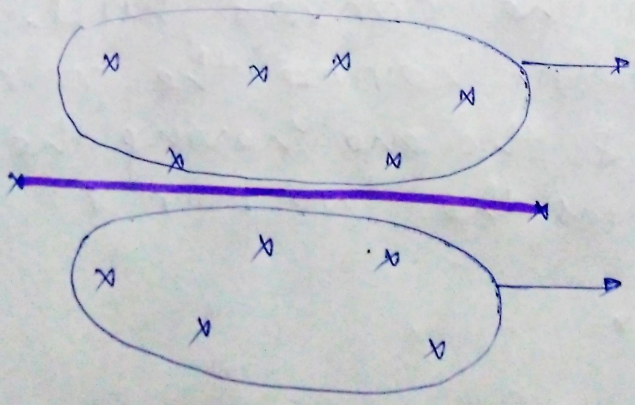


→ both endpoints of this edge are part of convex hull (extreme points)

Take this edge as anchor or pivot & partition the points into two halves:-

(1) all points above this line

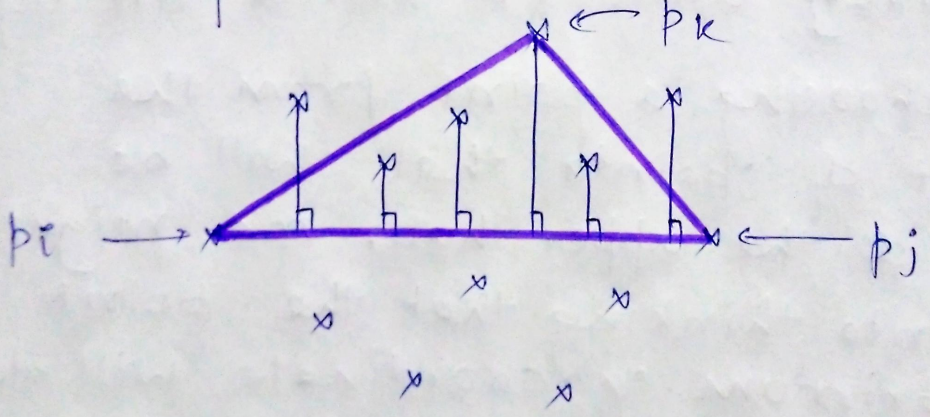
(2) all points below this line



Next, we compute the hull for the upper set of points & for the lower set of points.

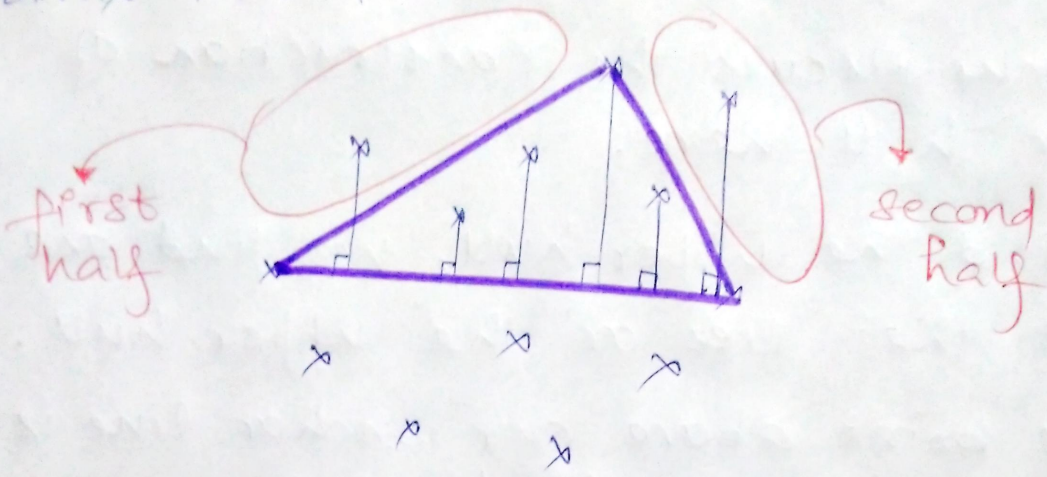
Since the procedure for construction of upper hull & lower hull is similar, we only discuss the construction of upper hull here.

To find the upper hull, we find one point that lies on the upper hull.
Every point above our anchor line is a candidate for the point.
Out of these candidate points, we choose the point having maximum perpendicular distance from the anchor line.



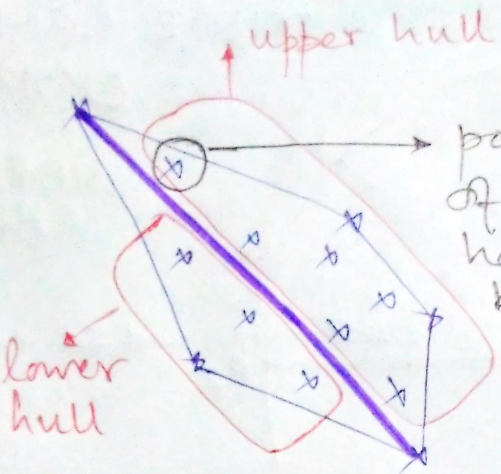
Now consider $\Delta p_i p_k p_j$. All the points that are inside the triangle are not a part of the convex hull. So we can remove them from our search space.

Now this further partitions the points into two halves.



So, again we have two subproblems. We need to find out the convex hull for the points in these two halves recursively until no points are left.

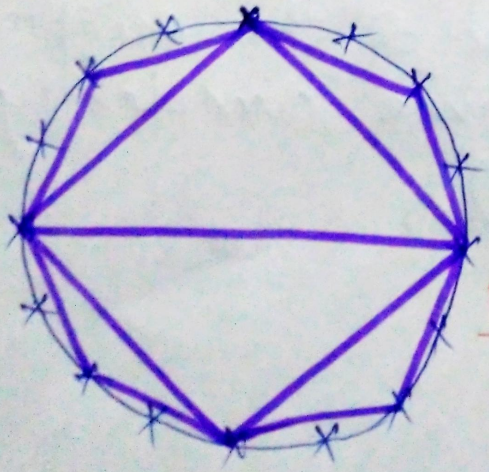
While choosing a point from the upper set of points that will be a part of the upper hull, one might tempt into thinking that the point with maximum y coordinate will always be a part of the upper hull. But that would be wrong. The point having the maximum y-coordinate is not necessarily a part of the upper hull. The diagram on the next page shows a counter example.



point which is a part of the upper hull & has max y coordinate but not a part of the convex hull.

TIME COMPLEXITY ANALYSIS

If the partitions are of almost equal size at every level, we get a time complexity of $O(n \log n)$. But if the partitions are skewed (one side has a very large number of points & the other side has a very small no. of points) at every level, then the time complexity is $O(n^2)$. Let us take a look at the best & worst cases of this algorithm.



Time complexity = $O(n \log n)$

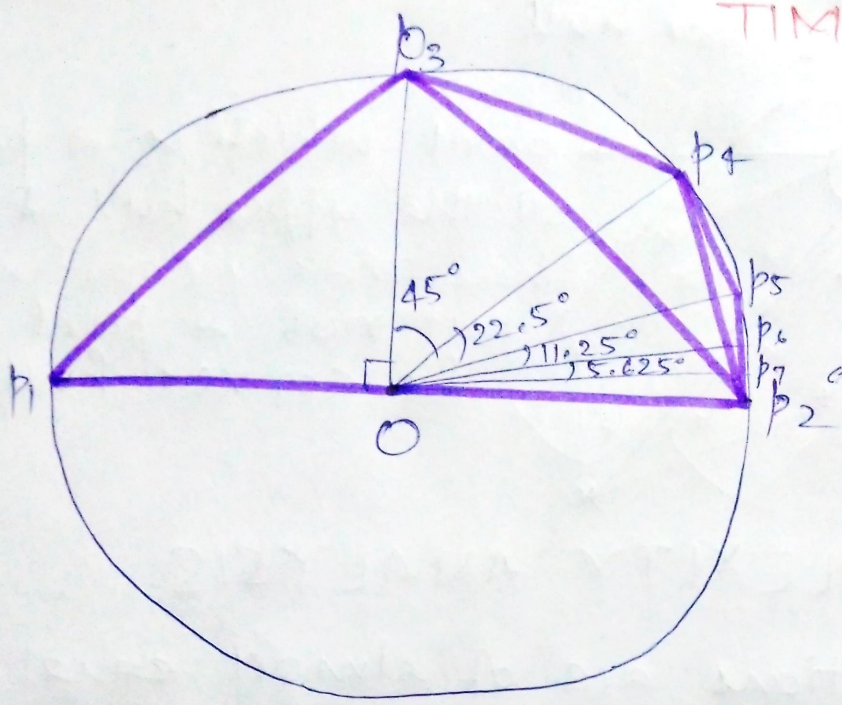
EQUALLY SPACED POINTS ON A CIRCLE

↑

Best Case

TIME COMPLEXITY
$= O(n^2)$

WORST CASE



and so on.

p_i for $i > 3$ lies on the angle bisector of p_{i-1} & p_2

⇧

WORST CASE

The partitions are highly skewed.
The left partition is always empty.
All the remaining points belong to the right partition. Due to this high imbalance in the sizes of the subproblems at every level, the time complexity comes out to be $O(n^2)$.