COMPUTATIONAL GEOMETRY

Lec 3

We will discuss several other algorithms related to Convex Hulls in order to get different geometric insights which will be helpful while solving other problems pertaining to the subject.
For this reason, we will stick with the construction of convex hulls in 2D for few more lectures.

GRAHAM'S SCAN

Lets have a look at Graham's scan algorithm.
First paper on convex hull algorithms which has a time complexity of $O(n \lg n)$.

↳ Better than Gift wrapping / Jarvis March

Idea is simple but makes few assumptions initially.
We are given a set of n points in arbitrary order.

$$P = \{p_1, p_2, \cdots, p_n\}$$

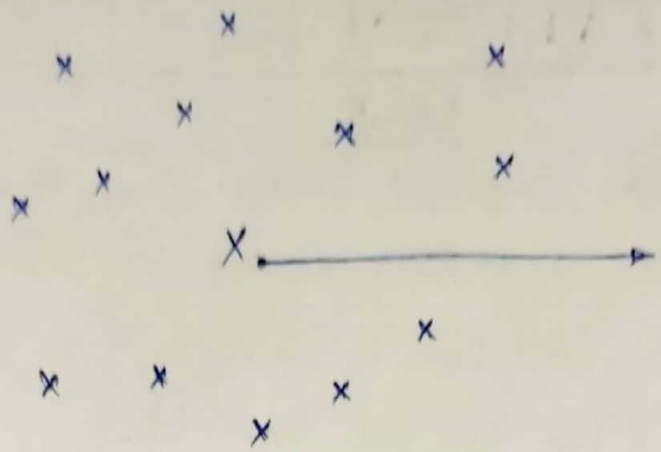
This algo assumes that a point X is given which lies inside the convex hull of these points.

$$X \text{ is inside } CH(P)$$

We will remove these assumptions in the later part of the algorithm.
Another assumption is that points are in general position.

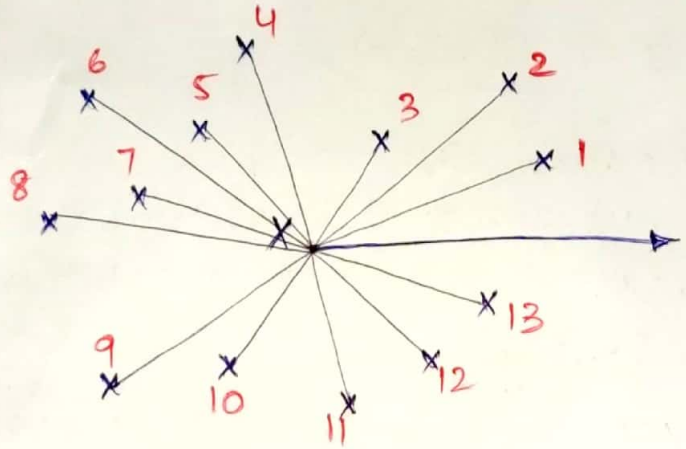
↳ In this case, it means that no 3 points are collinear.

Let us take an example.

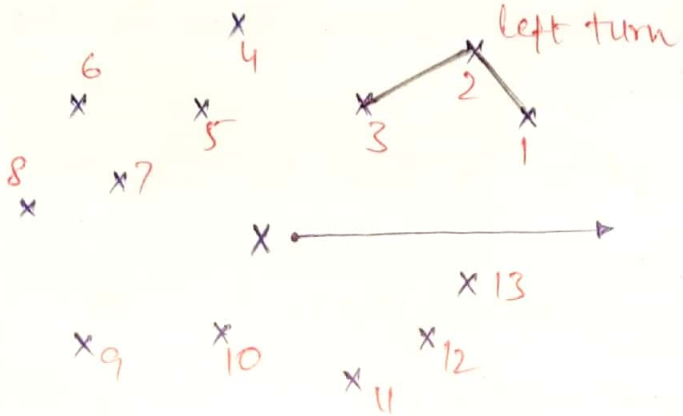


These points are sorted angularly wrt point X.
(All points are sorted angularly wrt the ray starting at X & going in +ve X dir^n).

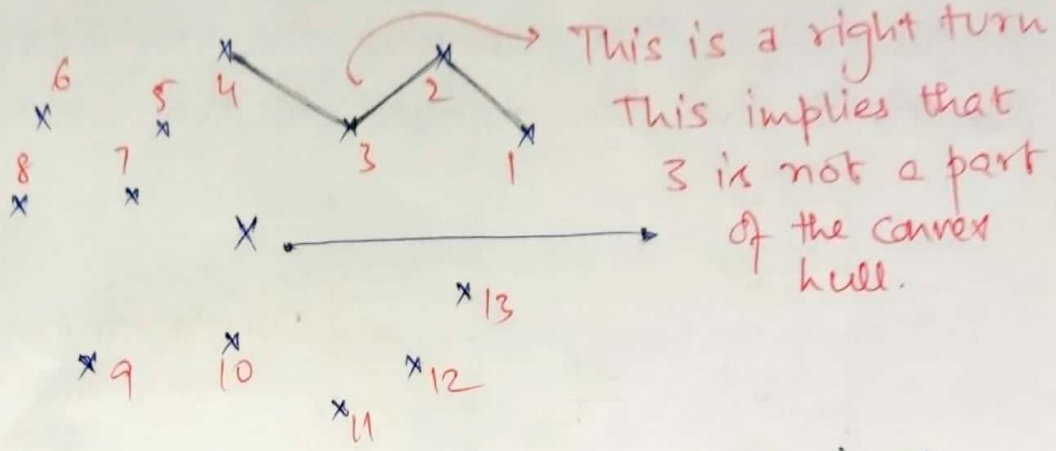
Assuming that angles increase in counter clockwise order, the points will be ordered as:



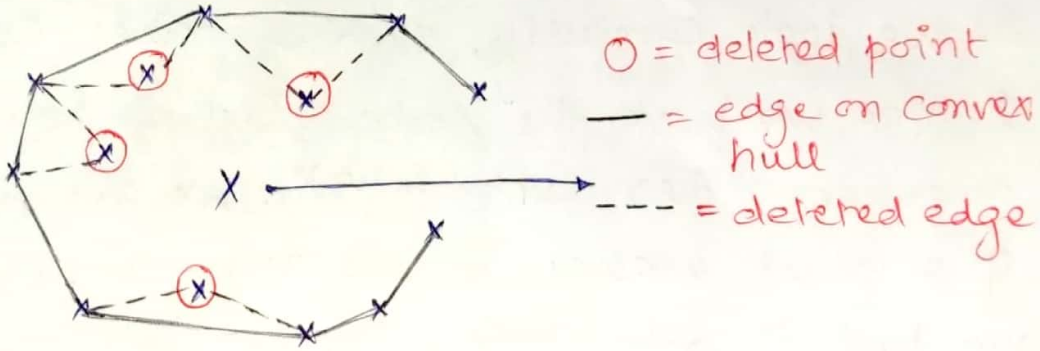
In this algorithm, we will walk through this sorted order. Lets consider the first 3 points.



If we go from 1 to 2, then 2 to 3, it makes a left turn at 2.
In this algorithm, if a left turn is made at some point, we proceed further. If a right turn is made, we remove that point & try ~~for~~ with the previous point which had not been removed earlier.



Since, a right turn is made at 3 which isn't acceptable, 3 is removed & now we will try with 2.

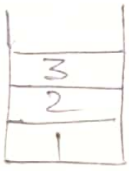


Similarly, on repeating the procedure, we end up with the above figure.

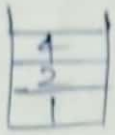
IMPLEMENTATION

We use a stack.
Push the first two points onto the stack.

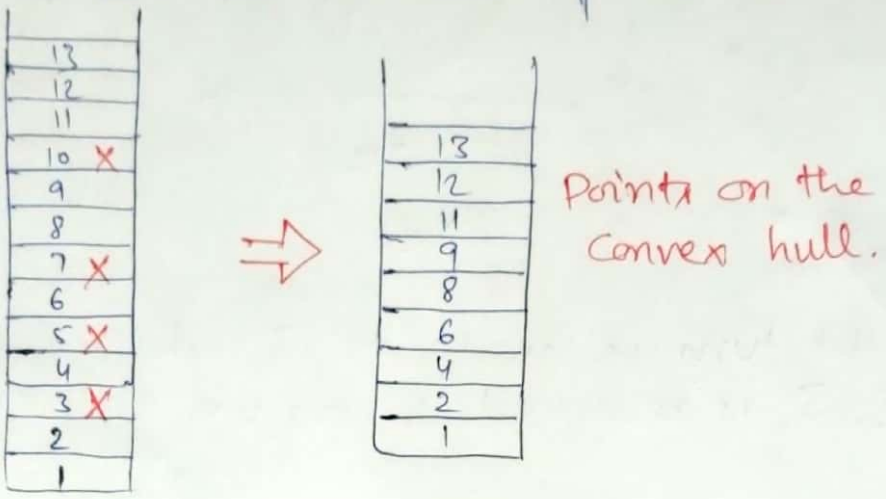
Now, if the 3rd point makes a left turn wrt 2, push it onto the stack.
Else remove 2nd point.



Now we check for 4, since 4 makes a right turn at 3, we pop 3 out of the stack and now check with 2. Since 4 makes a left turn wrt 2, 4 gets pushed.



Similarly, continue for other vertices also.
At the end of this scanning, the vertices left in the stack are the vertices of the convex hull.



If we look carefully, sorting takes O(NlgN) time & once we have the sorted list of points i.e., in angular order wrt point X, we are only checking if a point makes a left turn or not. If yes, we push it onto stack, else we pop the topmost element of stack.
Any element once popped from the stack cannot be a part of the convex hull. Hence, each element is pushed & popped at most once from the stack.
Hence, overall algorithm excluding sorting (assuming checking for left turns is O(1)) is O(1) per point.

pushing into stack → O(1)
popping from stack → O(1)
checking for left turn → O(1)

Hence, for all points, time complexity will be O(N), linear

TOTAL TIME COMPLEXITY = SORTING + TIME COMPLEXITY OF SCANNING

$= O(N \lg N) + O(N)$

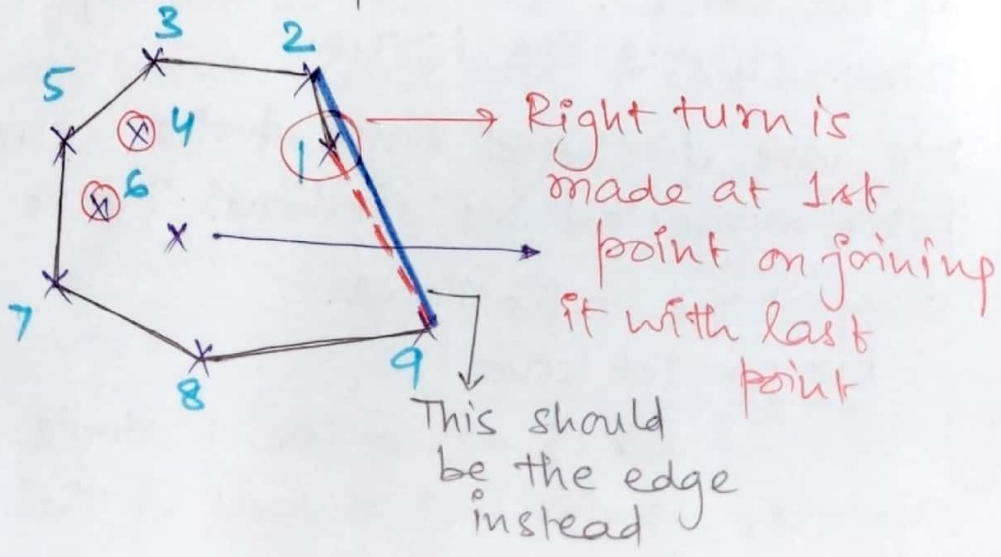
$= O(N \lg N)$

In this, we made a few assumptions such as we are given a point X which lies inside the convex hull.

In the original paper, Graham gave a linear time algorithm to find this point X from the given set of points. That X becomes a floating point number. But if we introduce floating point numbers, an associated precision error will accompany. This is one issue with this algo.

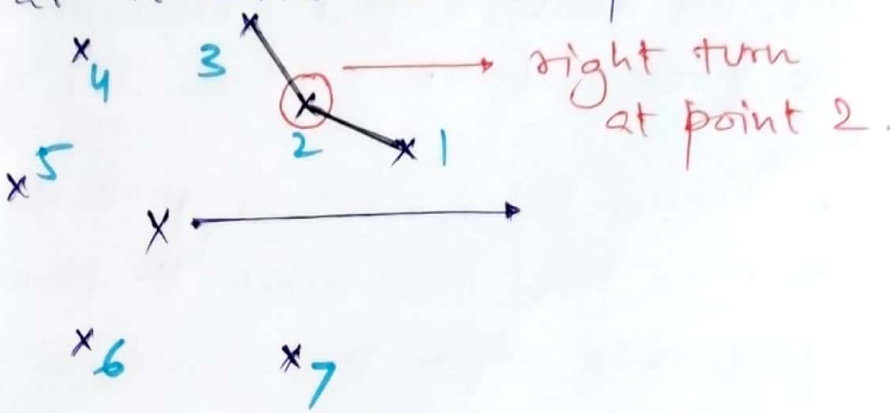
On some inspection, we will notice that the 1st & last point aren't necessarily a part of the hull because we never checked if the hull makes a left turn at these points.

Let us look at an example.



(1, 2, 3) forms a left turn at 2, hence we will proceed. When this algorithm terminates, 1, 2 & 9 are also a part of the hull. But that is incorrect since 1 isn't a part of the convex hull.

The algorithm has 1 more issue.
Let us look at it with an example.



Initially we kept points 1 & 2 in the stack & check whether 1-2-3 forms a left turn at 2 or not. Since it doesn't make a left turn at 2, we cannot push 3 onto the stack. So, we remove 2 from the stack. Now, only 1 element remains in the stack. For checking left turn, we used to look at the top 2 points of the stack. But there is only a single element now. This is the issue.

We have discussed some of the issues in the paper suggested by Graham. There are various suggestions to fix these.

FIX for 1st issue

Instead ~~for~~ of going for a single scan of the points, go for two rounds of the points.

This ensures that the starting points & ending points also get checked for making left turns.

FIX for 2nd issue

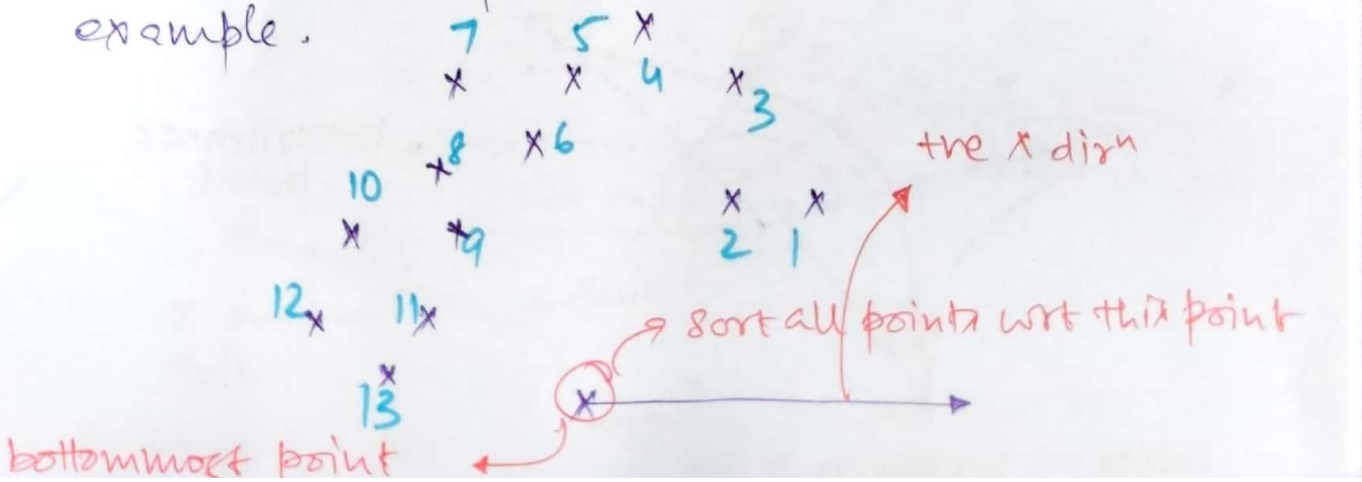
When there is a single point in the stack, push the current point onto the stack without going for left turn check procedure.

NOTE: The issue mentioned here is the case when the starting points of the hull are not a part of the convex hull. The same issue can occur for the ending points, when the ending points aren't a part of the hull.

A different solution to the above problems is also suggested which eliminates the search for a point inside the convex hull (Point X).

⇒ The solution is to sort the points based on one point that lies on the convex hull.

As we know that the bottommost point, leftmost point, rightmost point & topmost point, all are part of the convex hull, we can choose any one of them. In our example, we will take the bottommost point. Let's take a look at an example.

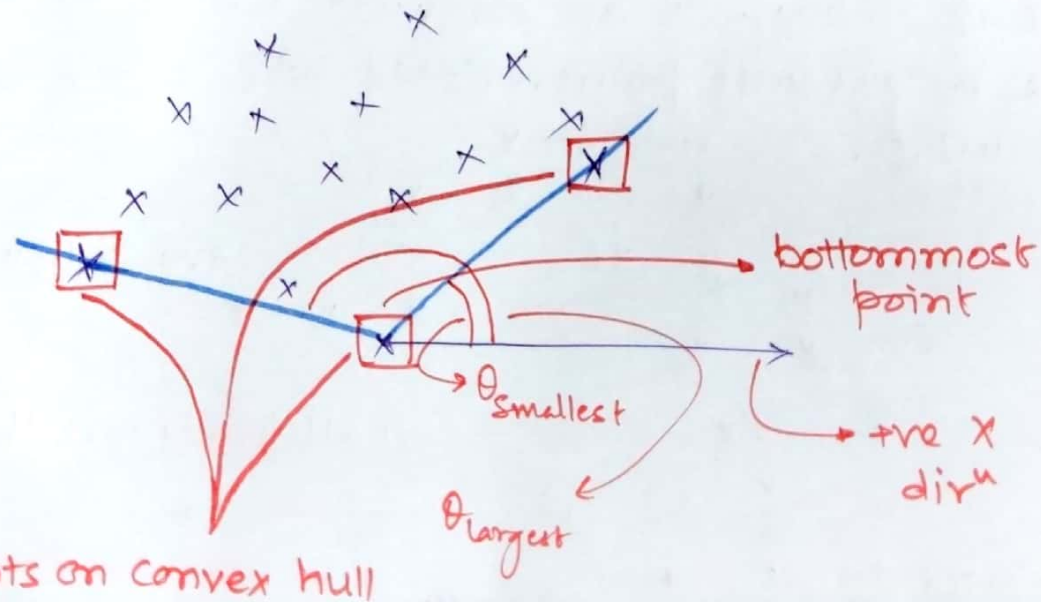


NOTE: If there are multiple points having the minimum y coordinate, take the rightmost point of these points.

What the solution says is that after sorting all the points wrt the bottommost point, the 1st point is also a part of the convex hull.

If we consider Gift wrapping algorithm, the 1st point has the smallest angle which is also being followed here.
Now, we have the first 2 points of the convex hull. If we proceed with Graham's scan, we will get the entire result.
Another advantage with this approach is that the last point is also a part of the convex hull. This is also ensured automatically.

Talking in terms of angle, points making smallest & largest angles with the bottommost point chosen as pivot are also a part of the convex hull.



last point, pivot point & first point is a part of the convex hull. Hence, the issues that we previously faced in Graham's scan will be eliminated by using this approach.

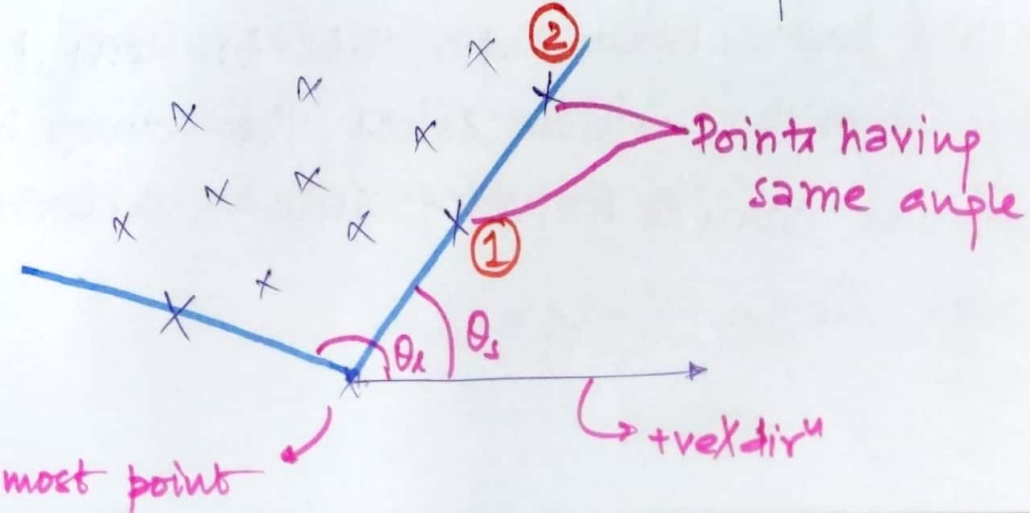
Of course, the pivot point can be any point that lies on the convex hull. So we can also sort the points wrt the topmost point but the sorting might be slightly different.

Instead of taking the +ve X dir^n, we take the -ve X dir^n & sort all the points wrt this dir^n.

Similarly, the same procedure can also be done using leftmost & rightmost point as pivots.

Hence, we have an $O(N \lg N)$ convex hull algorithm. But we assumed the points to be in general position (no 3 points are collinear). What if this condition isn't followed?

Let us take a look at an example.



Solution

In sorted order, for points having the same angle, the points lying closer comes first in order as compared to the points at a greater distance.

In this case, when we are checking for left turns, we have to check for strict left turns (angle made should be strictly less than 180°). In the figure, since an angle of 180° is made at 1, it does not make a strict left turn & hence is removed from the stack.

Now, the assumption of general position is also taken care of. Hence, we get a complete algorithm.

THE TIME COMPLEXITY OF THIS ALGORITHM IS $O(N \lg N)$

CAN WE DO BETTER?

If we can do better, we have to come up with an algorithm. If not, we have to come up with a lower bound for this problem, i.e., any algorithm which solves the convex hull problem cannot have a time complexity lower than $O(n \lg n)$.

The fact is we cannot do better than this. We can prove that the lower bound for this problem exists and is $\Omega(n \lg n)$.

Proving the lower bound for a problem can be done in two ways:-

① Proof by Reduction

Use the existing lower bound for another problem to prove the lower bound for this problem.

② Proof by Adversarial arguments

For every algorithm A & input of size n, there exists an input for which A takes at least L(n) steps.

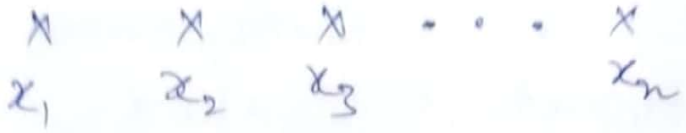
L(n) ⇕ lower bound for problem size n

For this problem, we take the first approach & reduce the sorting problem to this problem so that if any algorithm runs in complexity better than $\Omega(n \lg n)$, then Sorting can also be done in time complexity better than $\Omega(n \lg n)$ which violates the lower bound for Sorting problem.

→ lower bound for comparison sort.

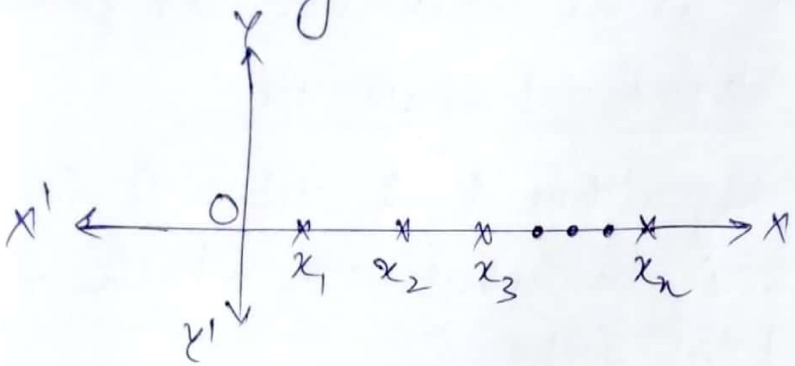
PROOF ON NEXT PAGE

Proof:-



Assume that we have been given n elements to sort. The figure above shows the points to be in sorted order, but the order can be arbitrary.

We assume they all are +ve X coordinates



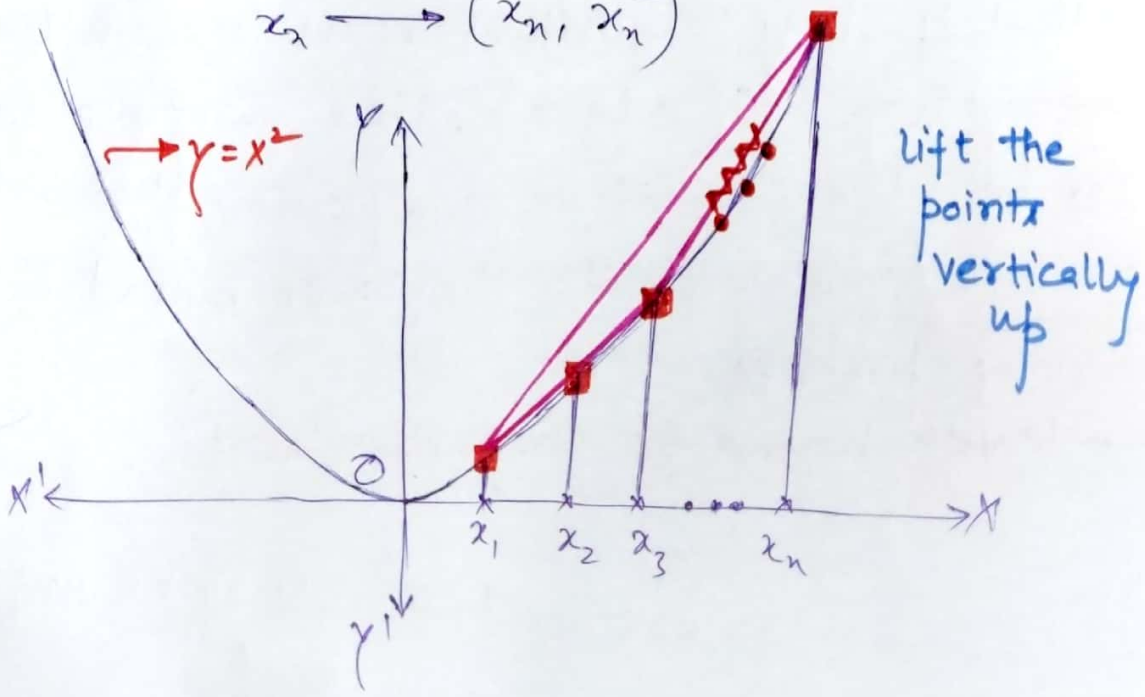
Take the parabola $y = x^2$ & map each point on to this parabola.

$x_1 \longrightarrow (x_1, x_1^2)$

$x_2 \longrightarrow (x_2, x_2^2)$

$\vdots$

$x_n \longrightarrow (x_n, x_n^2)$



The transformation $x_i \longrightarrow (x_i, x_i^2)$ $\forall$ $1 \le i \le n$ can be done in linear time.

Now we compute the convex hull of the points. Let us assume that the complexity is lower than $n \lg n$. In asymptotic notation, complexity is $o(n \lg n)$

Once our convex hull is constructed, the output is the points on the boundary of the convex hull in clockwise or counter clockwise direction.

Now, do the following steps:

$O(n)$ → Find the leftmost point on the convex hull.

$O(n)$ → Output the x coordinate of this point followed by all the points in convex hull order.

THE ABOVE STEPS CAN BE DONE IN $O(n)$ TIME

This output is nothing but the sorted sequence of initial input.

The time complexity is bounded by minimum complexity between convex hull computation ($o(n \lg n)$) & $O(n)$. Hence, overall complexity remains $o(n \lg n)$.

This shows that given an input size n, it can be sorted in time complexity lower than $O(n \lg n)$.

This violates the lower bound for comparison sort.

Hence, our assumption that there exists an algorithm having time complexity lower than $O(n \lg n)$ & solves the convex hull problem is not correct.

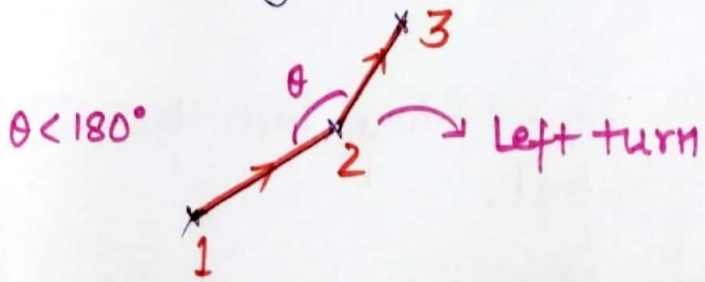
Hence, lower bound of convex hull problem is $\Omega(n \lg n)$

Proved

$\Omega(n \lg n)$ → Lower bound for convex hull

A very crucial part of Graham's scan algorithm is

Given 3 points, determine whether there is a left turn or right turn at the middle point.



This can be solved using simple geometry

$$M_{3\times3} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

The determinant of this matrix M denotes 2× area vector of triangle formed by 3 points.

The determinant can be positive or negative.

RESULT:

If $|M| > 0$ ⟶ left turn at 2

If $|M| < 0$ ⟶ right turn at 2

If $|M| = 0$ ⟶ collinear points (180° at 2)

This result plays a very crucial role in many other geometric calculations.

ex: used in determining if 2 line segments intersect each other or not.

Of course, this idea can be extended to 3 dimensions also.

COMPUTATIONAL GEOMETRY

Lec 4

Let us look at a Divide & Conquer approach to convex hull problem.

DIVIDE & CONQUER

Input:- n points are given in arbitrary order.

Algorithm:-

(1) Sort the points based on x-coordinate.

(2) Divide point set into 2 halves.

DIVIDE STEP (first $\frac{n}{2}$ in left half, remaining in right half)

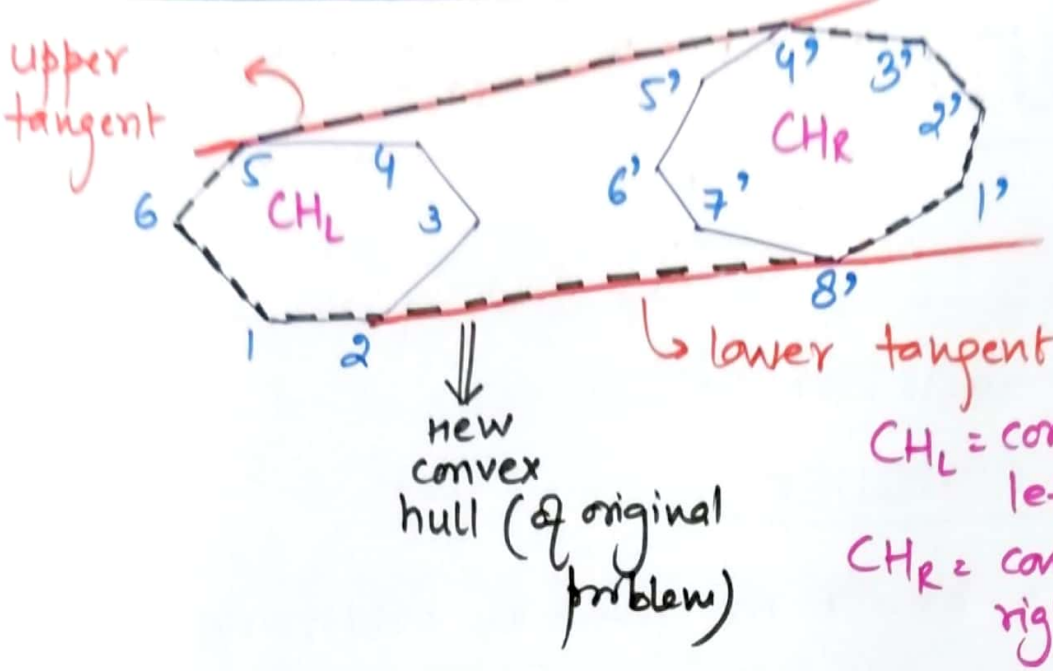
Assumption: No two points lie on same vertical line. No three points are collinear. This is known as general position for this algorithm.

(3) Recursively find convex hull for left and right half.

(4) Once both subproblems are solved, combine the solutions of these two problems to obtain solution of original problem.

MERGE STEP

Let us take an example to see how to combine the two convex hulls to get convex hull of original problem.



new convex hull (of original problem)

CH_L = convex hull of left part

CH_R = convex hull of right part

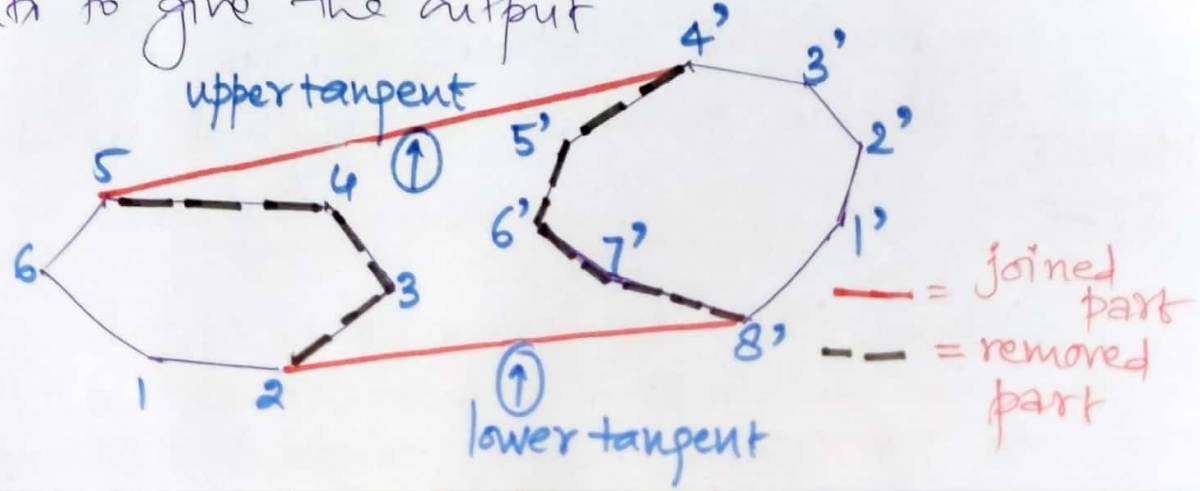
So basically we need to find the upper tangent & lower tangent.

Here, we assume that the points on the convex hull of left & right halves are given in counter clockwise (CCW) order.

So, the output that we need to give is

1	2	8'	1'	2'	3'	4'	5	6

Basically, what it does is that it removes points 3, 4 from left half & 5', 6', 7' from right half & combines the remaining points to give the output



Now the question is how to find these lower & upper tangents.

Here, we will only discuss how to find the upper tangents & the same procedure can be done with some minor modifications to find the lower tangent.

NOTE: The convex hulls of the left & right halves would never intersect.

(1) The points are sorted according to x-coordinate

(2) We assumed no two points lies on the same vertical line.

These 2 arguments lead to a result that there will always be a vertical line b/w the convex hulls of left & right halves.

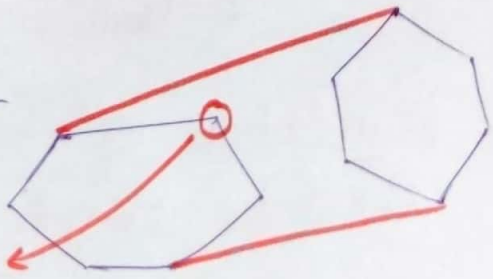
Now, we rephrase the problem,

Given two convex polygons separated by a vertical line, find the upper & lower tangents.

Claim 1: The lines joining the topmost points of the two hulls will be upper tangent.

DISCARDED

Counter example:



Approach :-

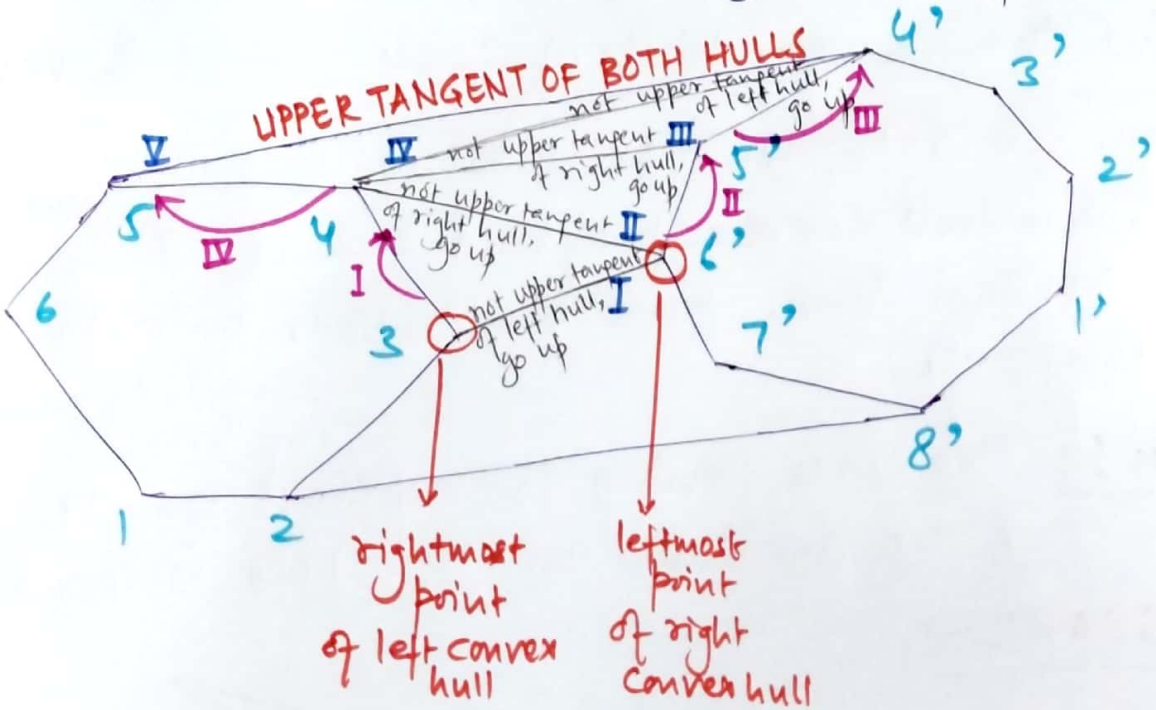
① Take the rightmost point on left convex hull & leftmost point on right convex hull.

② Now check if the line joining these 2 points becomes upper tangent or not. If yes, terminate.

③ Keep moving on the left convex hull on points which were above the previous line & check everytime if the current line is an upper tangent for the left convex hull. If it isn't repeat this step.

④ Do step 3 for right hull.

⑤ If current line is upper tangent for both halves, terminate. Else go to step 3.



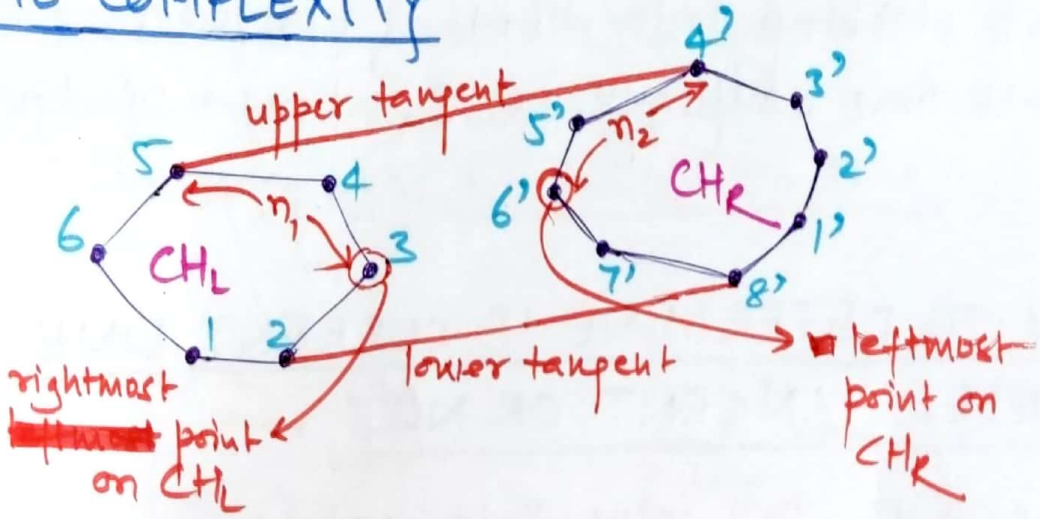
The simulation of above algorithm is shown in above figure.

Let us write a formal procedure for this algorithm.

Algo upper Tangent

a ← rightmost point on CH_L
b ← leftmost point on CH_R
while T = $\overline{ab}$ not upper tangent to CH_L, CH_R
  while T not upper tangent to CH_L
    a ← a+1
  while T not upper tangent to CH_R
    b ← b-1

TIME COMPLEXITY



n_1 = number of points b/w first point & final point on CH_L

n_2 = number of points b/w first point & final point on CH_R

TIME COMPLEXITY $\propto O(n_1)$
$\propto O(n_2)$
$\propto O(n_1+n_2)$

↳ This can be at most n

Hence, the time complexity is $O(n)$ ← LINEAR

But this time complexity is for finding the upper & lower tangents.

Let us look at the time complexity for the complete divide & conquer algorithm.

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

$$= O(n \lg n)$$

Since we did a sorting of all the points before stepping into divide & conquer, overall time complexity is $O(n \lg n) + O(n \lg n)$

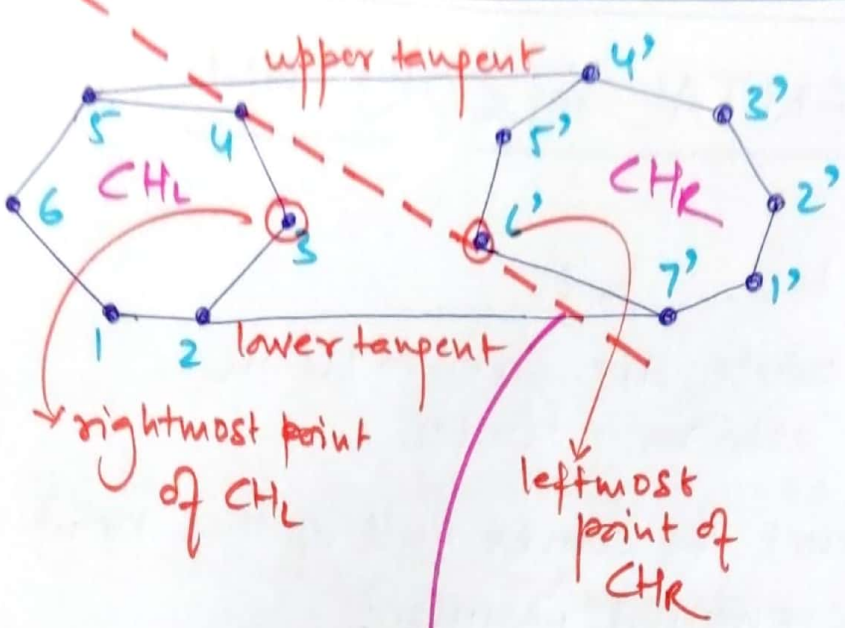
$$= O(n \lg n)$$

HOW TO DETERMINE IF CURRENT LINE IS UPPER TANGENT OR NOT

Check if two immediate neighbours of a are on the same side of the line. Similarly, check the immediate neighbours of b.

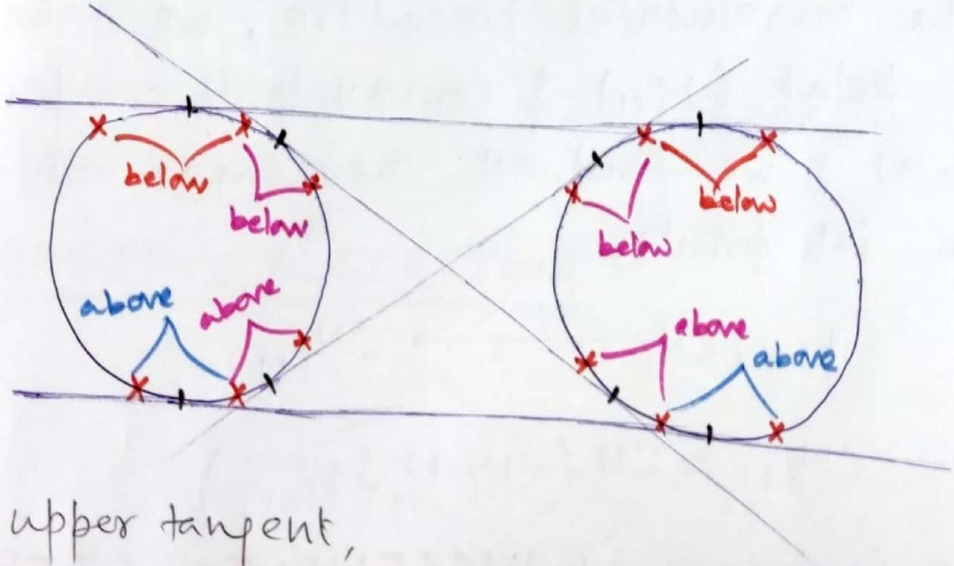
This can be done in $O(1)$ time.

But there is a flaw in this approach. The figure drawn on the next page explains this flaw.



This algorithm will also classify this line as upper tangent which is wrong. This line is a cross tangent.

We handle this by checking if all the 4 points (2 immediate neighbours of a & two immediate neighbours of b) lie "below" the line



For upper tangent,
all 4 points lie below the line
For lower tangent,
all 4 points lie above the line
For cross tangent,
1 pair of points lies above the line
& other pair lies below the line

INCREMENTAL ALGORITHM

Input:

$\{p_1, p_2, \ldots, p_n\}$

set of points are given to us in arbitrary order

We construct the convex hull of this point set in an incremental manner.

Let us assume that we have constructed the convex hull for points $\{p_1, p_2, \ldots, p_i\}$ & we denote the convex hull of these i points by CH_i, i.e.,

$$CH(\{p_1, p_2, \ldots, p_i\}) = CH_i$$

In the incremental procedure, we take the next point (p_{i+1}) & convex hull computed till now & we find out the convex hull of these $i+1$ points.

$$p_{i+1}, CH_i \longrightarrow CH_{i+1}$$

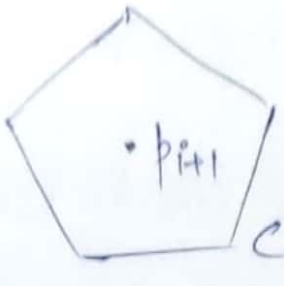
$$CH_{i+1} = CH(CH_i \cup \{p_{i+1}\})$$

This is known as **INCREMENTAL STEP.**

Procedure:

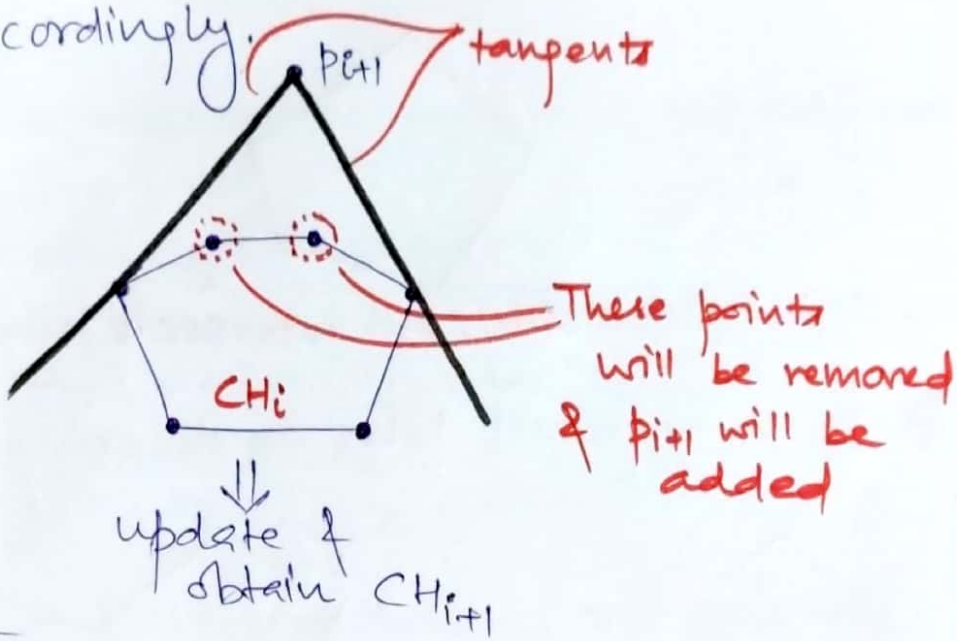
First we check if p_{i+1} lies inside the convex hull CH_i

If yes, $CH_{i+1} = CH_i$ & hence we don't need to do anything



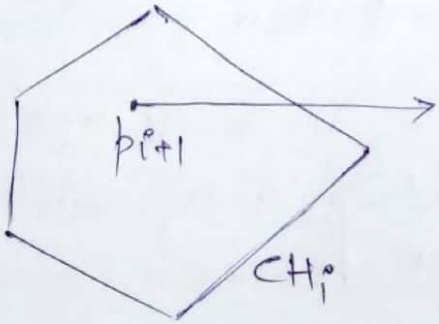
$CH_{i+1} = CH_i$

If no, then we find two tangents from p_{i+1} to CH_i & update the convex hull accordingly.



Hence our first task is to check if p_{i+1} lies inside CH_i

RAY CASTING ALGORITHM

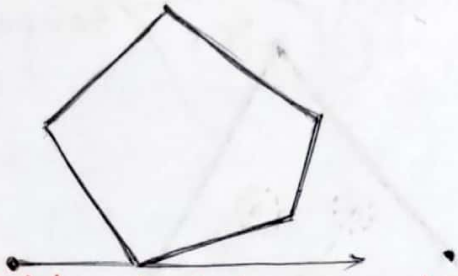


Take a ray from point p_{i+1} in +ve X dir^n (other directions can also be taken but this is standard). If it intersects the convex hull at exactly one point, the point is inside the hull.

If it intersects twice or zero times, the point lies outside

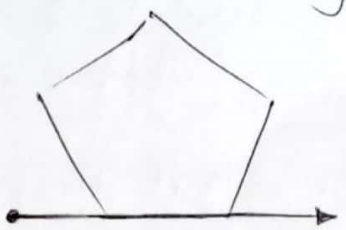
This algorithm has some issues.

① If the ray passes through exactly one vertex of the polygon from an outside point



outside, but intersects only once

② If a segment falls on the ray



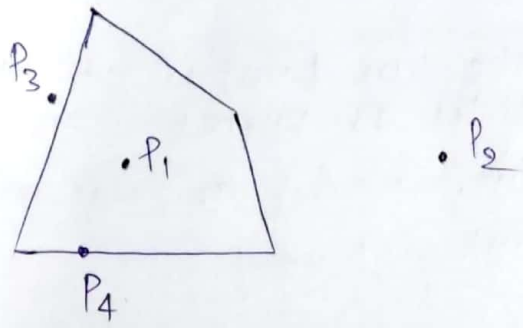
We avoid these cases by incrementing the y coordinate of our point by an infinitesimally small value ϵ & then checking for intersections.

Python code is given on next page for understanding along with some test cases

```
def contains(self, point):
    import sys
    # _huge is used to act as infinity if we divide by 0
    _huge = sys.float_info.max
    # _eps is used to make sure points are not on
    # same line as vertexes
    _eps = 0.00001

    # We start on the outside of the polygon
    inside = False
    for edge in self.edges:
        #Make sure A is the lower point of the edge
        A, B = edge[0], edge[1]
        if (A.y > B.y):
            A, B = B, A

        # Make sure point is not at same
        # height as vertex
        if point.y == A.y or point.y == B.y:
            point.y += _eps

        if (point.y > B.y or point.y < A.y or
                point.x > max(A.x, B.x)):
            # the horizontal ray does not intersect
            # with the edge
            continue
        if (point.x < min(A.x, B.x)):
            #the ray intersects with the edge
            inside = not inside
            continue

        try:
            m_edge = (B.y - A.y)/(B.x - A.x)
```

```
except ZeroDivisionError:
    m_edge = -huge

try:
    m_point = (point.y - A.y)/(point.x - A.x)
except ZeroDivisionError:
    m_point = -huge
if m_point >= m_edge:
    # the ray intersects with the edge
    inside = not inside
    continue

return inside
```

TESTS



Output

```
P1 inside polygon: True
P2 inside polygon : false
P3 inside polygon : false
P4 inside polygon : True
```

NOTE

~~An~~ ~~the~~ Another approach for this problem is by using WINDING NUMBER ALGORITHM. for further reading, refer to the Wikipedia article on "Point in Polygon".

NOTE: In the incremental algorithm discussed above, we did not sort the input points. We carried forward the algorithm with the same order of points as was given in the output. Let us first analyze the time complexity of the above discussed algorithm & then look at another variant of INCREMENTAL ALGORITHM.

TIME COMPLEXITY

Time taken to determine if p_{i+1} lies inside CH_i is $O(i)$ in the worst case as we need to check intersections with every edge. If the point lies outside the convex hull, finding two tangents from this point to the hull can be done similar to what we did in the previous algo. Hence, this can also be done in $O(i)$.

$\therefore T(i+1) = T(i) + O(i)$

By this recurrent relation,

$$\boxed{T(n) = O(n^2)}$$

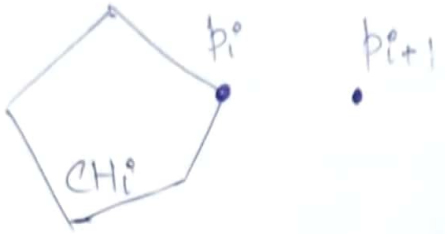
ANOTHER VARIANT OF INCREMENTAL ALGORITHM

In this variant, the input points are sorted wrt x coordinate.

Some important observations:

(1) The next point will never be inside the convex hull. So we won't have to check if point is inside polygon.

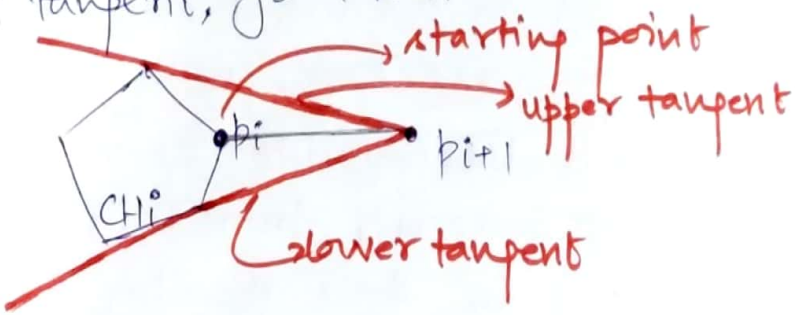
(2) The rightmost point of CH_i will be p_i.



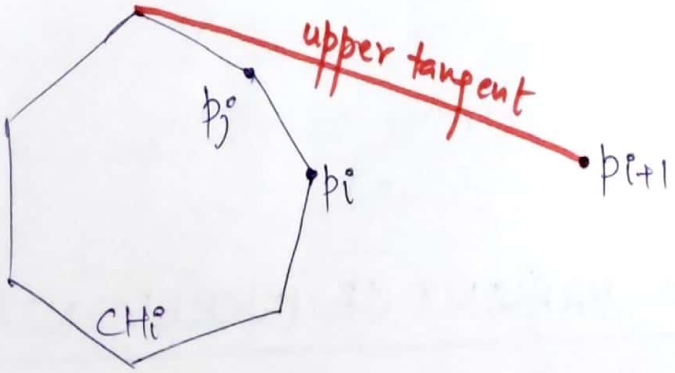
Hence, we only need to find upper & lower tangents.

→ Choose p_i as starting point.
→ Go upwards until current edge is the upper tangent

★ For lower tangent, go down.



TIME COMPLEXITY



$p_{i+1}p_j$ will be checked if it is the upper tangent or not. If it isn't an upper tangent, this point will become an interior point of CH_{i+1} & hence will never be checked again.

Since a point is never checked after it moves to the interior of the convex hull & every point moves inside the convex hull after failing the test for the first time (i.e. neither upper tangent, nor lower tangent), no point fails more than once & at every step, at max 2 points pass the test (1 upper tangent, 1 lower tangent). Hence, the number of times the check for upper or lower tangents is done is bounded by

2 successes per iteration
$2 \times$ no. of iterations $+$ each point fails atmost once
total points

$= 2 \times O(n) + O(n)$

$= O(n)$

Since upper tangent check & lower tangent check take $O(1)$ time, total time complexity is $O(n)$

LINEAR

So, to summarize convex hulls,

① we saw best time complexity $O(n \lg n)$

② we gave a lower bound to the time complexity of convex hull problem as $\Omega(n \lg n)$

③ But can we do better?

There exists an algorithm whose time complexity is $O(n \lg h)$

where h is the size of the CH

If h is some constant,

algorithm becomes linear.

If $h = O(n)$,

complexity becomes $O(n \lg n)$

This is an output sensitive algorithm for convex hull.

In the literature, there are 2 algorithms that give this time complexity.

We will look at them in the next lecture.

COMPUTATIONAL GEOMETRY

Lec 5

CHAN'S ALGORITHM

An output sensitive convex hull algorithm with time complexity $O(n \lg h)$.

Sorting is ruled out (since that will make time complexity as $O(n \lg n)$).

HULL2D (P, m, H)

Input: n points are given in a plane.

Algorithm:

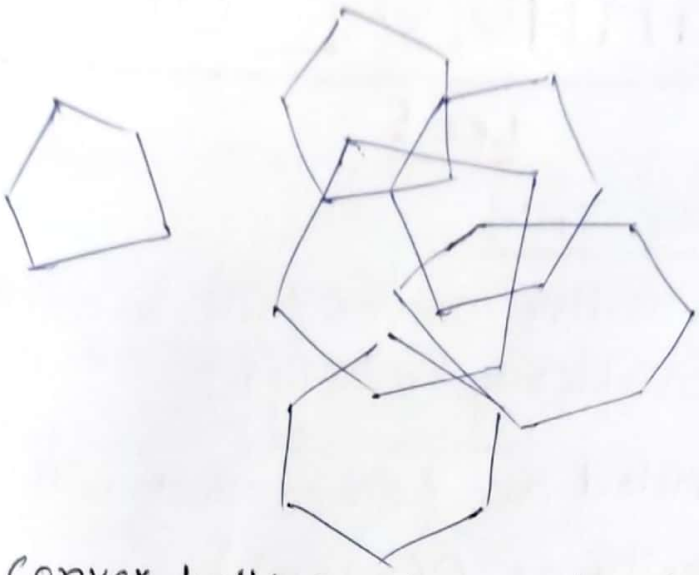
→ Partition the points into subsets of size m each in arbitrary order (since no sorting is allowed)

$P_1 \quad P_2 \quad P_3 \ldots\ldots P_{\lceil \frac{n}{m} \rceil}$

Note: m is a parameter.

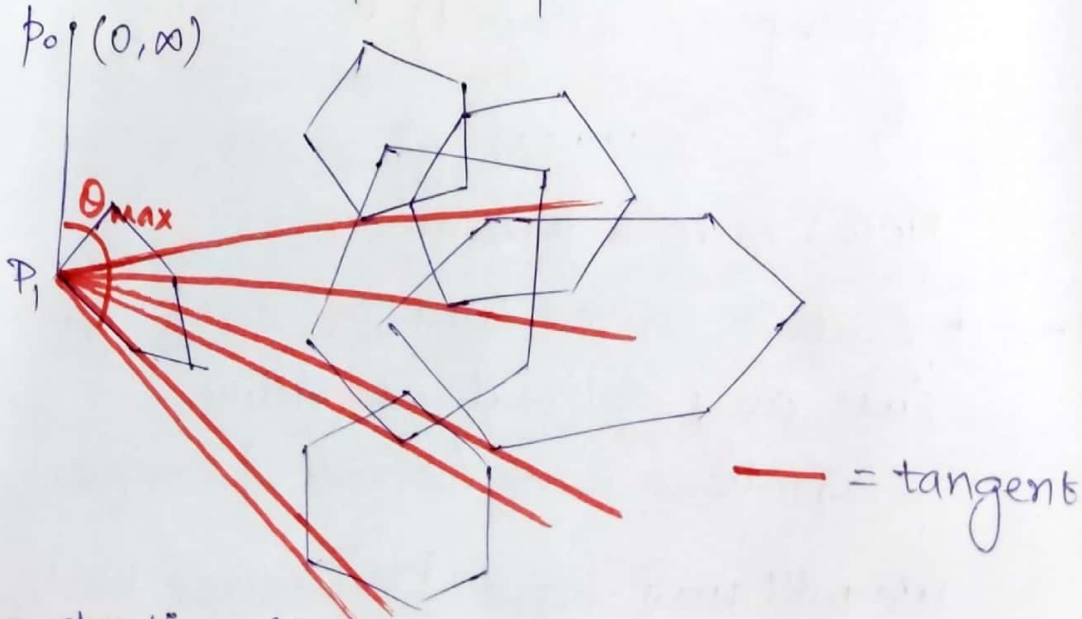
→ Compute convex hull for each group (use any optimal algorithm Graham Scan / Divide & Conquer)

We will now have $\lceil \frac{n}{m} \rceil$ convex hulls. These convex hulls will be overlapping, intersecting or disjoint since the points are assigned to the groups in arbitrary order.



Convex hulls can be overlapping, intersecting or disjoint

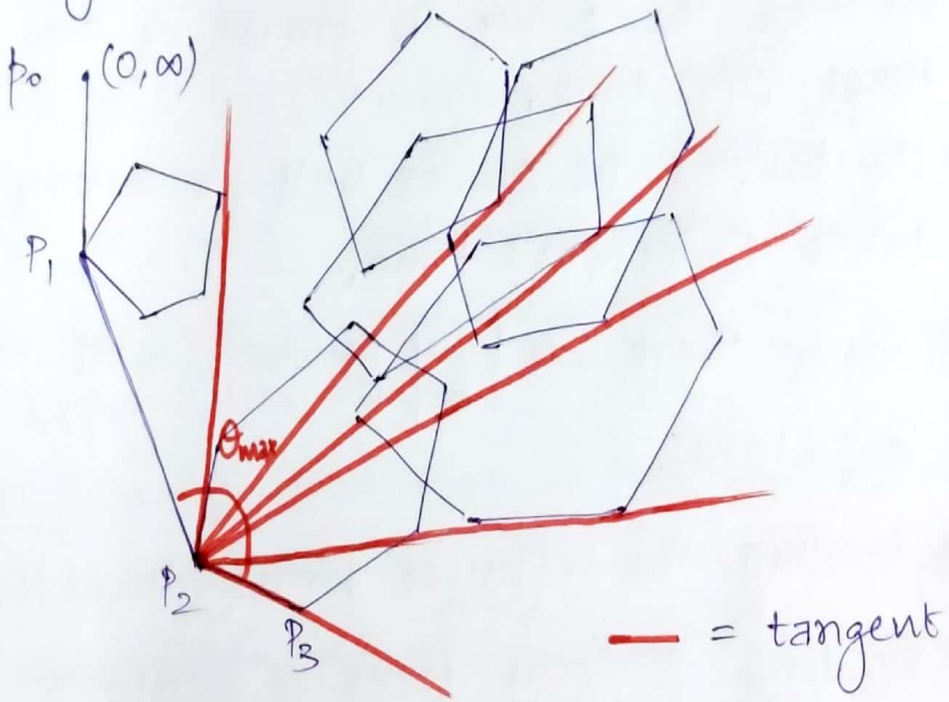
Now, we apply Gift wrapping algorithm. A point has to be chosen which is necessarily on the hull. It can be the topmost / bottommost / leftmost / rightmost point. In the paper, the rightmost point was taken as pivot. Here, we will take the leftmost point.



Here, the starting line is the line p_0p_1 where p_0 is the point $(0, \infty)$ & p_1 is the leftmost point.

To do the gift wrapping among these convex polygons, we draw tangents to every convex polygon from the previous line (P_0P_1)

Out of all the tangents, the one which gives the largest angle is taken as the next edge. In this case, P_2.



Now repeat the same procedure for P_1P_2.
Now, P_2P_3 is the new edge.

In gift wrapping, we keep repeating the procedure until first point is reached.
But here, we continue the procedure till H edges.

H is the number of edges on the convex hull.

Of course, we don't know how many edges will be on the convex hull ∴ assume that this parameter is given to us, somehow we know. We will see how to find this one later.

TIME COMPLEXITY

① Dividing the points in groups of size m takes $O(1)$ time.

② Computing the convex hull for every group takes $O(m \lg m)$ time.

③ Since there are $\lceil \frac{n}{m} \rceil$ groups, total time is $O(n \lg m)$.

④ Finding the leftmost point takes $O(n)$ time.

⑤ Finding tangent from a point to all convex hulls.

Finding tangent per convex hull = $O(\lg m)$

FOUND BY USING BINARY SEARCH

Finding all tangents = $O(\frac{n}{m} \lg m)$

⑥ Finding tangent making largest angle = $O(\frac{n}{m})$

⑦ Hence, finding one edge = $O(\frac{n}{m} \lg m) + O(\frac{n}{m})$

$= O(\frac{n}{m} \lg m)$

⑧ Finding H edges = $O(\frac{Hn}{m} \lg m)$

TOTAL TIME COMPLEXITY (over all STEPS) $= O(n \lg m) + O\left(\frac{Hn}{m} \lg m\right)$

If $m = H = h$, where h is the no. of edges on the convex hull, then time complexity becomes

$$O(n \lg h) + O(n \lg h)$$
$$= O(n \lg h)$$

Algorithm is quite simple, the only issue being that we do not know the value of h beforehand.

Another point of discussion is the algorithm to find a tangent in $O(\lg m)$ time complexity. This will be covered at the end of the lecture.

Now we need to find out the value of h in order to attain the time complexity of $O(n \lg h)$. But we do not know it beforehand. But the solution is simple. Simply enumerate.

CHULL(P)

for $t = 1, 2, \ldots$ do

$L \leftarrow$ HULL2D(P, m, H) where $m = H = \min(2^{2^t}, n)$

if $L \neq$ incomplete, then return L

The value of t will be incremented until we get a complete convex hull, after which we simply return the computed hull.

Let us have a look at the i^{th} iteration of the loop. It takes time $O(n \lg H)$. For the i^{th} iteration, $H = 2^{2^i}$, hence, time complexity is $O(n \cdot 2^i)$

Max. no. of iterations (β)

$$2^{2^\beta} \leq h$$

$$\beta \leq \log\log h$$

Hence, $\beta = \log\log h$.

TOTAL TIME IN COMPUTING CONVEX HULL

$$= \sum_{i=1}^{\lceil \log\log h \rceil} n \cdot 2^i = O(n \log h)$$

Beauty of this algorithm is that it can be extended to 3D with the exact same time complexity of $O(n \log h)$. In 3D, instead of finding tangents, we'll have to find tangent planes.

An improvement can be done in this algorithm to achieve performance speedup. In the current iteration (lets assume i^{th}) of the algorithm, the value of $m = 2^{2^i}$. We will get $\lceil \frac{n}{m} \rceil$ different convex hulls. The points ~~which~~ which lie inside

these $\lceil \frac{n}{m} \rceil$ convex hulls will also not be a part of the final convex hull. Hence, we remove the interior points in all $\lceil \frac{n}{m} \rceil$ convex hulls at the end of current iteration. This will make the next iteration faster.

3D CONVEX HULL ALGORITHMS

Divide & conquer	$O(n \lg n)$
Incremental algo	$O(n^2)$
Gift wrapping algo	$O(n \lg n)$
Chan's algorithm	$O(n \lg h)$

FINDING TANGENT FROM EXTERNAL POINT TO CONVEX HULL IN O(log n) TIME

Definition: A tangent $\overrightarrow{PV_i}$ to convex hull is an upper tangent if all points of hull are to the left of, or on PV_i. Likewise for lower tangent (points to the right).

Assumption: Given points of convex hull sorted in counter clockwise order (CCW)

$$\{V_0, V_1, \ldots, V_{n-1}\}$$

Algo for finding upper tangent

(similar for lower tangent with different directions)

1) Initialise $l \leftarrow 0$ and $r \leftarrow n-1$

Loop → 2) $m = \left(\frac{l+r}{2}\right) \% n$

3) consider $\ell = \overrightarrow{PV_m}$

4) If both $V_{(m+1)\%n}$ and $V_{(m-1)\%n}$ are to the left of $\overrightarrow{PV_m}$, the upper tangent is $\overrightarrow{PV_m}$

5) Otherwise if $V_{(m+1)\%n}$ is to the right of $\overrightarrow{PV_m}$, $l = (m+1)\%n$. Go back to step 2.

6) Otherwise, $r = (m-1)\%n$. Go back to step 2.

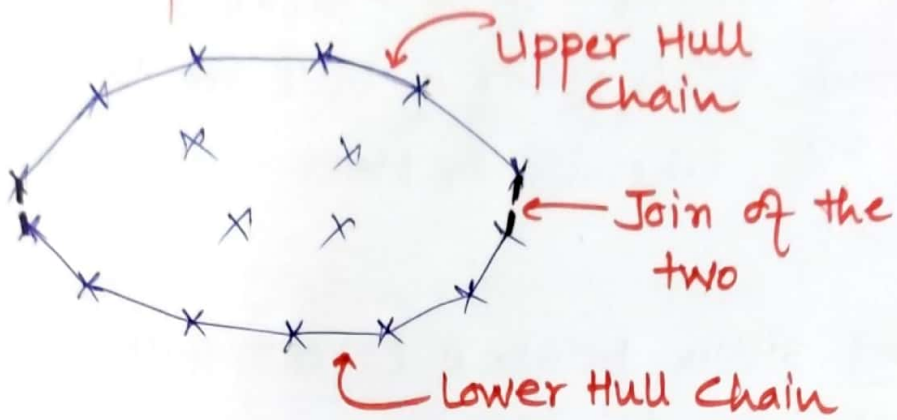
The same algo can be used for finding lower tangent. Just swap 'right' and 'left'.

Interesting fact

If P is to the left of convex hull, the upper tangent is the one on the bottom. (According to the definition of the top).

THE ULTIMATE PLANAR CONVEX HULL ALGORITHM

by Kirkpatrick and Seidal



This algorithm finds the convex hull of a set of 2D points in $O(n \log h)$ time, where h is the number of edges of the hull.

This algorithm works by finding a chain of the upper hull points, and a chain of lower hull points and (for endpoints they don't share) connecting them.

Connecting the two chains isn't difficult. Just join the rightmost points of both (if not the same) & the leftmost points of both (if not the same).

Connection happens in $O(1)$ time. The challenge is finding the two chains. The algorithm for the upper chain can be extended to the lower chain.

Upper chain calculation

Prerequisite: Blum's selection algorithm

Given: n points in arbitrary order. Points in set S.

What the algorithm does is, it finds a vertical line $(x=a)$, such that $\lceil \frac{n}{2} \rceil$ points have $x \leq a$ and $\lfloor \frac{n}{2} \rfloor$ points have $x \geq a$. These points are partitioned

accordingly and a bridge edge (connecting a point on the left and a point on the right such that no point is above the line) is found. This bridge edge is an extreme edge of the convex hull. The other edges are found by recursively solving for the left set and the right set.

Pseudocode (taken directly from the paper)

UPPER_HULL(S)

1) Find indices 'min' and 'max' such that
$x(p_{min}) \leq x(p_i) \leq x(p_{max})$,
$y(p_{min}) \geq y(p_i)$ if $x(p_{min}) = x(p_i)$
$y(p_{max}) \geq y(p_i)$ if $x(p_{max}) = x(p_i)$
Let $T = \{p_{min}, p_{max}\} \cup \{p \in S \mid x(p) \neq x(p_{min}) \& x(p) \neq x(p_{max})\}$

2) CONNECT (min, max, T)
Pseudocode for CONNECT (k, m, S)
2.1) Find a such that $x(p_i) \leq a$ for $\lceil \frac{|S|}{2} \rceil$ points and $x(p_i) \geq a$ for $\lfloor \frac{|S|}{2} \rfloor$ points in S.
2.2) Find bridge over line (x=a) [bridge defined earlier]
(i, j) = BRIDGE (S, a)
↳ Described later. Runs in O(n) time.
2.3) Let $S_{left} = \{p_i\} \cup \{p \in S \mid x(p) < x(p_i)\}$ and
$S_{right} = \{p_j\} \cup \{p \in S \mid x(p) > x(p_j)\}$
2.4) If i=k, add i to upper_hull_array
else CONNECT (k, i, S_{left})
If j=m, add j to upper_hull_array
else CONNECT (j, m, S_{right})

Time Complexity: $O(n \lg h_0)$ where
h_0 is no. of edges in upper hull chain

Proof:

Blum's selection (step 2.1) and Bridge finding (step 2.2) happen in $O(n)$ time.

Recursive relation :-

$O(n)$ twice

$$T(n,h) \leq \begin{cases} cn & \text{if } h = 2 \\ cn + \max\limits_{h_l + h_r = h-1} \{T(\frac{n}{2}, h_l) + T(\frac{n}{2}, h_r)\} \end{cases}$$

$$cn + \max_{h_l+h_r=h-1} \{T(\tfrac{n}{2}, h_l) + T(\tfrac{n}{2}, h_r)\} \leq$$

$$ch + \max_{h_l+h_r=h} \{T(\tfrac{n}{2}, h_l) + T(\tfrac{n}{2}, h_r)\}$$

Guess & Substitute

Claim $T(n,h) = O(n \log h)$

$$\Rightarrow T(n,h) \leq cn + \frac{1}{2} cn \max_{h_l+h_r=h} \{\log h_l + \log h_r\}$$

$$= cn + \frac{1}{2} cn \max_{h_l+h_r=h} \{\log h_l h_r\}$$

A.M-G.M inequality

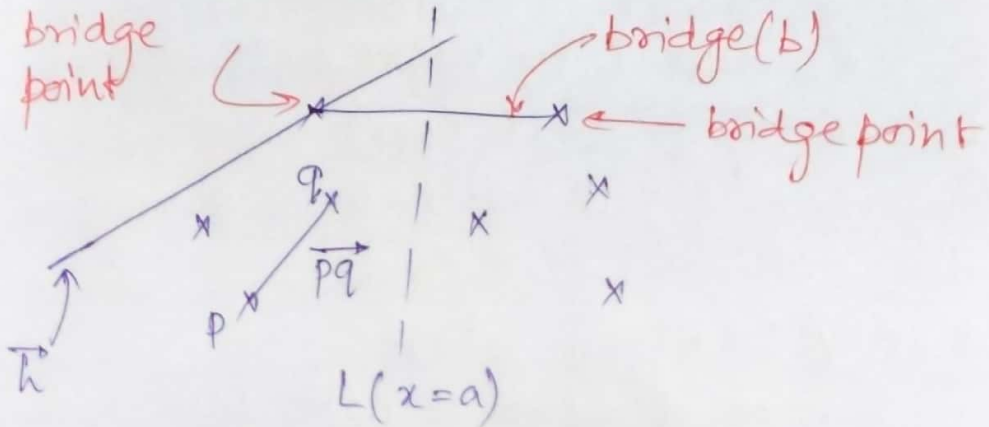
$$\leq cn + \frac{1}{2} cn \log\left(\frac{h}{2}\right)^2$$

$$= cn + cn \log h - cn$$

$$= cn \log h$$

$$\Rightarrow \boxed{T(n,h) = O(n \log h)}$$

FINDING BRIDGE



A supporting line is a line that contains at least one point of S and no point is above this line.

Naturally, a bridge is a supporting line.

Let $\overrightarrow{pq}$ be a line for $p, q \in S$. Let $\overrightarrow{b}$ be the bridge line. Let s_e be slope of line L. Let L be the separating line. Let $\overrightarrow{h}$ be a supporting line.

Properties: (can be proved easily)
(demonstrated in above diagram)

1) If $s_{pq} > s_b$, p cannot be a bridge point
2) If $s_{pq} < s_b$, q cannot be a bridge point
3) If $s_h > s_b$, points on h, only lie on the left of (or on) L. (AND VICE VERSA)
4) If $s_h < s_b$, points on h only lie on the right of L. (AND VICE VERSA)
5) If $s_h = s_b$, h contains a point on the right, and a point on the left (or on) L.

Using these properties (Taken from paper)

BRIDGE (S, a):

1) CANDIDATES = ϕ
2) If $|S| = 2$, return (i, j) where $S = \{p_i, p_j\}$ & $x(p_i) \leq x(p_j)$.
3) Make $\lfloor \frac{|S|}{2} \rfloor$ disjoint pairs of points in S. If a point remains, add it to CANDIDATES. Order points in pairs by x coordinate. Add pairs to PAIRS.
4) Determine slopes of lines through these pairs.

If slope doesn't exist, add the point with larger y-coordinate to PAIRS.

5) Determine median slope of slopes. Call it K.

6) Make three sets, SMALL (contains pairs with slope < k), EQUAL (contains pairs with slope = k) and LARGE (contains pairs with slope > k)

7) Let MAX be the set of points such that $p_i \in S$, such that $y(p_i) - k \times x(p_i)$ is maximum. (HIGHEST LINE)

Let p_k be the minimum point in MAX (by x coordinate), and p_m be the maximum point in MAX (by x coordinate)

8) If $x(p_k) \leq a$ and $x(p_m) > a$ return (k, m)

$p_k p_m$ is bridge

9) If $x(p_m) \leq a$ then:- (BY LEMMA 3, $s_b < k$)
for all $(p_i, p_j) \in$ LARGE $\cup$ EQUAL, insert p_j into ~~candidates~~ CANDIDATES.
for all $(p_i, p_j) \in$ SMALL, insert p_i & p_j into CANDIDATES

LEMMA I

10) If $x(p_m) > a$ then:- (BY LEMMA 4, $s_b > k$)
for all $(p_i, p_j) \in$ SMALL $\cup$ EQUAL, insert p_i into CANDIDATES
for all $(p_i, p_j) \in$ LARGE, insert p_i & p_j into CANDIDATES

LEMMA II

TIME COMPLEXITY ANALYSIS ($O(n)$)

For every recursive call, at least $\frac{n}{4}$ points are removed where n is the input to the current call.

$$\Rightarrow T(n) = \begin{cases} T\left(\frac{3n}{4}\right) + O(n) & n > 2 \\ O(1) & n = 2 \end{cases}$$

Pairing, finding slope, median (using Blum's) partitioning etc.

$$\Rightarrow T(n) \leq cn + c\frac{3n}{4} + c\left(\frac{3}{4}\right)^2 n + c\left(\frac{3}{4}\right)^3 n + \ldots + c$$

where c is some constant

$$= cn\left(1 + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \ldots + \left(\frac{3}{4}\right)^k\right) \text{ for some } k$$

$$< cn\left(1 + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \ldots\right)$$

$$= cn(4) = O(n)$$

$$\Rightarrow \boxed{T(n) = O(n)}$$