[Convex Hull in 2D]
Computational Geometry (CSc 635)

Jagdish Bhatta

Central Department of Computer Science & Information Technology
Tribhuvan University

Convex hull in 2D:

Definition:

- The convex hull of a finite set of points, S in plane is the smallest convex polygon P, that encloses S. (Smallest area)
- The convex hull of a set of points, S in the plane is the union of all the triangles determined by points in S.
- The convex hull of a finite set of points, S, is the intersection of all the convex polygons (sets) that contain S.

There are wide ranges of application areas where it comes use of convex hulls such as;

- In pattern recognition, an unknown shape may be represented by its convex hull, which is then matched to a database of known shapes.
- In motion planning, if the robot is approximated by its convex hull, then it is easier to plan collision free path on the landscape of obstacles.
- Smallest box, fitting ranges & so on.

What does it mean to compute the Convex Hull?

- As we have seen, the convex hull of point set S is a convex polygon. A natural way to represent a polygon is by listing its vertices in counter clockwise order, starting with an arbitrary one. So the problem we want solve is;

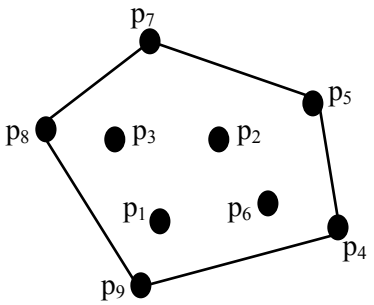
  Given a set S = { p_1, p_2, p_3, --------, p_n} of points in the plane, compute a list that contains those points from S that are vertices of convex hull of P, listed in counter clockwise order.

  Input = set of points:

  { p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9 }

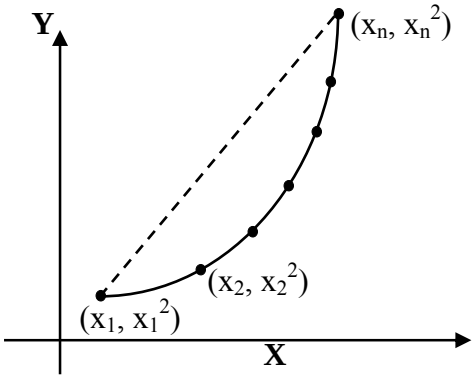
  Output = representation of the convex hull:

  { p_4, p_5, p_7, p_0, p_9 }



Theorem: The convex hull of a set of n-points in the plane be computed in Ω (nlogn) time.

Proof:

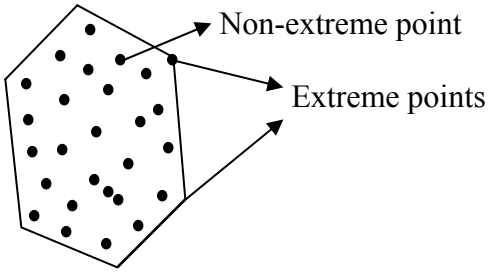


- Suppose we have given n positive numbers to sort. Let x_1, x_2, x_3, --------, x_n are given numbers.
- From a set o f 2D points as; (x_1, x_1^2), (x_2, x_2^2), ---------, (x_n, x_n^2). These pairs when plotted form a parabola arm as in the figure.
- Finding convex hull of these points, we get, x-coordinates sorted. Since every point is on the hull. Identify lowest point, say $a(x_i,y_i)$, on the hull. The order in which points occur on the hull counterclockwise from a, is their sorted order.
- This transformation takes linear time.
- The convex hull of n-points could be computed in less than Ω (nlogn) time. Then, sorting problem could be solved in less than Ω (nlogn).
- But, sorting problem has complexity Ω (nlogn) time, so computation of convex hull must also take Ω (nlogn) time.

  ∴ Lower bound for convex hull is Ω (nlogn)

Extreme point of convex hull:

- The points corresponding to the vertices of the convex hull of a set of points are called extreme points.
- The interior angle at extreme points is less than π.



Algorithms for computing convex hull

1) Naive algorithm for extreme point (Elimination Method)

Extreme Point:

A point x is said to be extreme point of the set of points S if x does not lie inside any triangle formed by any three points p, q, r ∈ S. If x lies inside any triangle formed by p, q, r ∈ S then x is non-extreme point. The naïve algorithm for computing convex hull examines all the points in S for all $\binom{n}{3} = O(n^3)$ triangles. If the point is non-extreme, it is eliminated.

Algorithm

1) Identify the points that are non-extreme and eliminate.
2) Angularly sort the remaining points about an interior point. The so sorted set of points will form the convex hull.

   To compute Step 1, we can do it as;

   - If a point x lies inside a triangle formed by three points p, q, r ∈ S then x can not be extreme point.
   - Examine x for all possible $\binom{n}{3} = O(n^3)$ triangles for determining as non-extreme point.

   So, the code structure may be as;

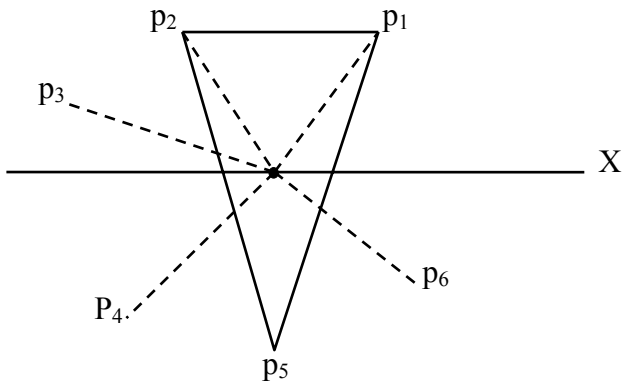
```
For each i
        for each j ≠ i do
                for each k ≠ j ≠ i do
                        for each l ≠ k ≠ j ≠ i do
                                if p_l ∈ Δ (p_i, p_j, p_k)
                                        then p_l is non extreme
                                end if
                        end for
                end for
        end for
```

The inclusion of point inside the triangle can be done by implementing left turn test.

In step 2, we have to sort angularly about an interior point. For this,

- Pick any three points from input set say a, b, c the centroid of three points give interior point.
  i.e. $P_c = \left(\frac{a.x+b.x+c.x}{3}, \frac{a.y+b.y+c.y}{3}\right)$

- Find the angle between the lines, from centroid point p_c to candidate point p_i, with x-axis.
- Use these angles for angular sorting. (i.e. sort about p_c).



Analysis:

This approach for computing convex hull is very slow. Here, the time complexity is;

Step 1: each candidate point should be examined with $\binom{n}{3} = O(n^3)$ triangles. So it takes $O(n^4)$ times.

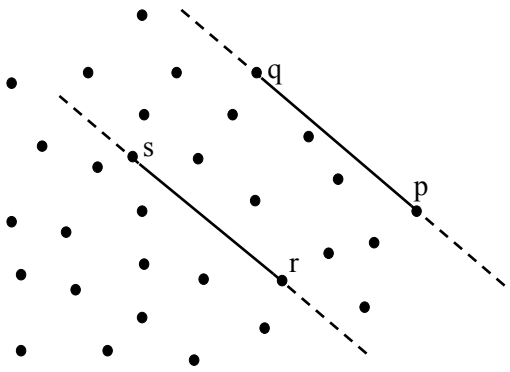
Step 2: sorting can be done in Ω (nlogn)

$\therefore$ Total time complexity = O (n4). [*Not applicable for practical application.*]

2) Algorithm based on extreme edge:

Extreme Edge:

Given a set of points, S, in plane, an edge e formed by joining the two points of S is called extreme edge, if all of the remaining points other than two end points of the edge lie on the same side of the line passing through e.



Here, *pq* is an extreme edge while *rs* is non-extreme edge. To determine the extreme edge, method of left turn/right turn test can be used.

The algorithm for finding convex hull using the idea of extreme edge is as;

Step 1: -Identify all the extreme edges, taking at most $\binom{n}{2} = O(n^2)$ edges, by checking the turn test with candidate point.

Step 2: Rearrange the edges (extreme) to form boundary of convex hull.

Step 1, can be done as;

```
for each i do
    for each j ≠ i do
        for each k ≠ j ≠ i do
            if pk is not left or on (pi, pj)
                then (pi, pj) is not extreme.
            end if
        end for
    end for
end for
```

Step 2 includes just rearranging the remaining extreme edges;

Analysis:

In step 1, each candidate edge examined with O (n) points. There are $\binom{n}{2} = O(n^2)$ candidate edges.

So algorithm step 1, takes $O(n^3)$ time.

Step 2 takes O (n2) for rearrangement.

Hence, the complexity of algorithm is O (n3). This is faster than naïve algorithm of extreme point by a factor of n.

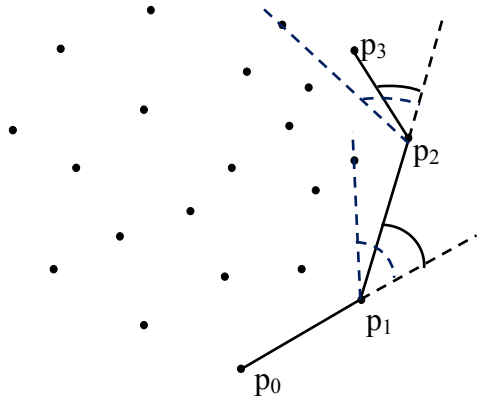
[*Selecting p_i, p_{i+1} with smallest slope means that p_{i+1} will have smallest polar angle with respect to the positive horizontal ray from p_i, in counterclockwise direction. If there will be tie, the farthest point will be chosen*]

3) Gift-wrapping Algorithm (Jarvi's March)

This algorithm is a variation of extreme edge approach with enhancement in computation by a factor of n. it builds the hull by a process called "gift-wrapping". This algorithm

operates by considering any one point on the hull, say, the lowest point and then finds next edge on the hull in counterclockwise order.

The idea is; if a line segment pq is an extreme edge i.e. edge of convex hull, then there must be another boundary edge with q as one of the end point.



Algorithm:

Step 1: find a base point p_0, with smallest y-coordinate. This point p_0 is vertex of convex hull of given points.

Step 2: find next point p_1 such that p_0p_1 has smallest slope among all possible p_i's then p_1 will be another vertex of the hull.

In general, find p_{i+1} from pi such that p_ip_{i+1} has smallest slope. Continue the process until $p_{i+1} = p_0$.

Selecting p_ip_{i+1} with smallest slope means that p_{i+1} will have smallest polar angle with respect to the positive horizontal ray from p_i , in counter clockwise direction. If there will be tie, the farthest point will be chosen.

Complexity Analysis:

- Step 1 takes O (n) time since there are at most n-comparisons.
- Step 2 computes for n- times and there are at most n-comparisons at each pass. So it will take O (n^2).

Hence, time complexity for Gift-wrap algorithm is O (n^2).

Output Sensitivity:

We can actually perform a slightly more nuanced analysis of the Jarvis March algorithm, if we introduce the parameter h, the actual size of convex hull (since it's a polygon, this can be either the number of vertices or the number of edges). In each linear time step, one new

convex hull vertex is added. Therefore, the running time can be rewritten as O (nh). This running time expressed both n terms of input size (n) and output size (n) is called output-sensitive; the algorithm is said to be output sensitive.

So, if h is known in advance then it becomes constant, so complexity will be O(n).
If n = o(log n) [little oh], then this algorithm is asymptotically taster than Graham's Scan.
In worst case, h = n so bound reduces to $O(n^2)$.
Thus, gift-wrapping is efficient when output is known in advance or the convex hull is small enough.

4.) Quick Hull Approach:

This quick hull approach for convex hull generation is generalization of the quick sort sorting procedure. The idea behind Quick Hull is to discard points that are not on the hull as quickly as possible. Here, we have following steps:

Step 1: Find two distinct extreme points that may be rightmost lowest and leftmost highest point say x & y respectively. (As there is at least one convex vertex; lemma)
Step 2: Recursively find the upper hull above xy and lower hull below xy.
Step 3: combine upper hull & lower hull to obtain complete hull of S.

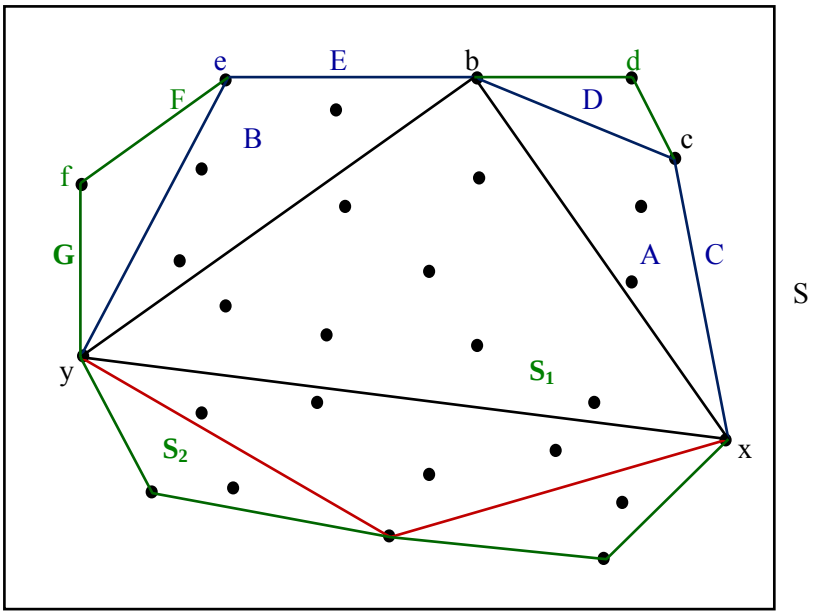
To find the hull in upper half above xy we have;

- Find the next extreme point, say b, which is farthest from xy. Discard all points that lies inside Δbxy. Split remaining points into two subsets, those that lie outside xb and those that lie outside xb and those that lie outside of by.
- Recursively find extreme points farthest from xb & by each and perform similar operation.

Thus, finding the quick hull can be performed by following recursive procedure;

```
QuickHull (S)              //S is the set of points
{
        Find extreme points x & y and partition S into S1 & S2.
        return Recursive_hull(x, y, S1) ∪ Recursive_hull (y, x, S2)
}
```

```
Recursive_hull(x, y, S)
{
        if S is empty
            return;
        else
        {
            b = index of point with maximum distance from xy.
            A = points strictly to right of (x, b)
            B = points strictly to right of (b, y)
            return Recursive_hull (x, b, A) ∪ (b) ∪ Recursive_ hull (b, y, B)
        }
}
```



Time Complexity:

- Here, finding the extreme points x, y and partitioning can be done in O(n).
- Let S_1 = n, for recursive function it takes n-time to find another extreme point b & then partition into A & B.
- The cost of recursive call depends upon the size of A & B.

Let |A| = α and |B| = β such that α + β = n-1. Here, α + β is at most n-1 because the point b is not within region of A & B.

So, time complexity with quick-hull |S| = n can be obtained by the recursive relation;

$T(n) = T(\alpha) + T(\beta) + O(n)$

For best case;

$\alpha = \beta = \frac{n}{2}$ (equal partition Approx.)

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

Solving this relation we can have, $T(n) = O(n\log n)$.

For worst case;

$\alpha = n-1, \beta = 0$

So, the relation is, $T(n) = T(n-1) + O(n)$

Solving this we have, $T(n) = O(n^2)$.

5.) Graham's Scan Algorithm:

This algorithm computes convex hull of points in 2D by maintaining the feasible candidate points on the stack. If the candidate point is not extreme, then it is removed from the stack. When all points are examined, only the extreme points remain on the stack & which will result the final hull.

Input $\Rightarrow P = \{p_0, p_1, \text{-------}, p_{n-1}\}$ of n-points.

Output $\Rightarrow$ Convex hull of P

Algorithm:

- Find a point pi with lowest y-coordinate, let it be q_0.
- Sort the input points angularly about q_0 let the sorted list is now $\{q_0, q_1, \text{-------}, q_{n-1}\}$
- Push q_0 into stack S and push q_1 into the stack S.
- Initialize i = 2

```
while (i < n)
        if LeftTurn (next top (S), top (S), qi) is true
                push qi into stack S.
                i++
        else
                pop the stack
        end if
end while
```

- Each points popped from stack are not vertex of convex hull.
- Finally, at last when all elements are processed, the points that remain on stack are the vertices of the convex hull.

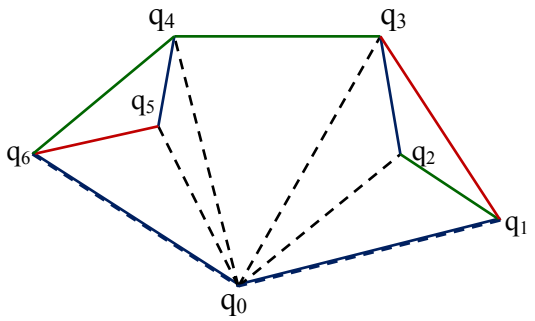
Complexity Analysis:

- Finding minimum y-coordinate point it takes O (n) time.
- Sorting angularly about the point takes O (nlogn) time.
- Pushing & popping takes constant time.
- The while loop runs for O (n) times

Hence, the complexity = O (n) + O (nlogn) + O (1) + O (n)

= O (nlogn).

Consider an example:



(1) At First, Push q_0 & q_1 into stack $\Rightarrow$ | | | q_1 | q_0 |

(2) **Scan q_2**:

$q_0q_1q_2$ is left turn so push q_2 into stack. $\Rightarrow$ | | | q_2 | q_1 | q_0 |

(3) **Scan q_3:**

$q_1q_2q_3$ is not left turn so pop (stack) i.e. pop q_2 $\Rightarrow$ | | | | q_1 | q_0 |

$q_0q_1q_3$ is left turn so push q_3 into stack. $\Rightarrow$ | | q_3 | q_1 | q_0 |

& so on……

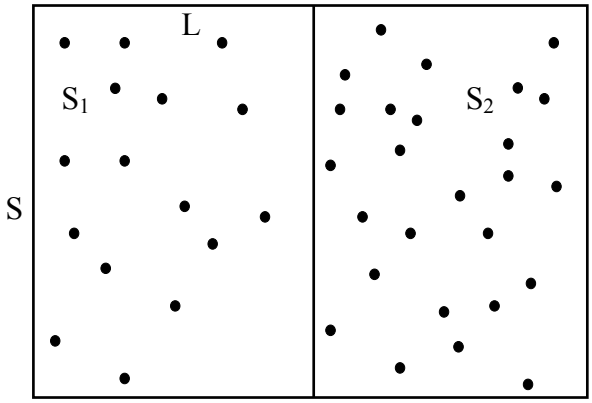
At last, final hull will be $\Rightarrow \{q_0, q_1, q_3, q_4, q_6\}$

6.) Divide & Conquer Algorithm for Convex Hull:

Here, the idea is to partition the given point set into two (nearly) equal halves, solve each half recursively & create a complete solution by merging two half solutions.

Divide Step:

- Divide the input point set S into two parts S_1 & S_2 of approximately equal size.
- It would be convenient to partition S in such a way that all points in $S_1$are to the left of all point in S_2 i.e. partition S by an imaginary vertical live into two equal size sets S_1 & S_2 such that all points in S_1 are left & all points in S_2 are right of the line say L.



Conquer Step:

- Recursively find the convex hull of S_1 & S_2. If S_1 and /or S_2 have only 2 or 3 points, then the convex hull is a line or triangle.

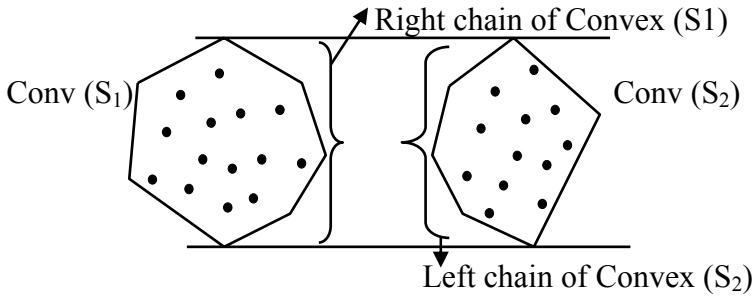
Combine Step:

- Combine the convex hulls of S_1 & S_2 efficiently to obtain the hull of S.

To implement divide step:

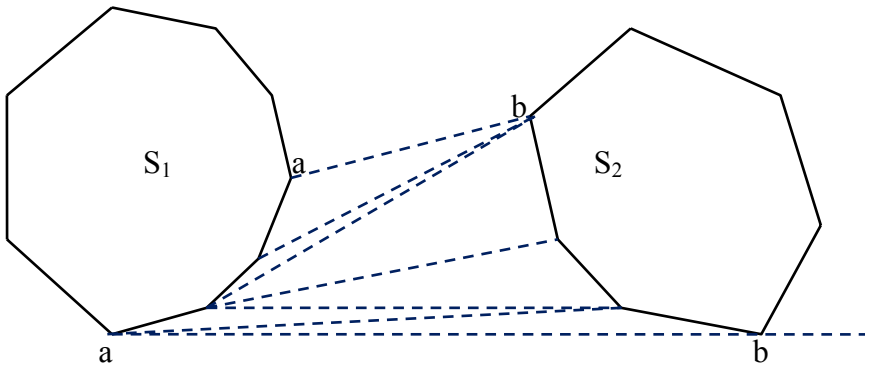
- Sort the p input point set S in non-decreasing order of x-coordinates. The first half of the sorted list is in S_1 & next half is in S_2.

To combine the hulls of S_1 & S_2:



In the resulting convex hulls of S_1 & S_2 are as in the figure above, how to combine these hulls to form the resulting hull of $S = S_1 \cup S_2$, we can do it by obtaining lower tangent and upper tangent and remove right chain of convex hull of $S_1 \cup S_2 = S$.

Computing lower & upper tangents of convex (S_1) & Convex (S_2):



Lower Tangent:

- Choose rightmost point of S_1, Say a
- Chose leftmost point of S_2, say b
- Move b downwards until ab is lower tangent of S_2
- Move a downwards until ab is lower tangent of S_1.
- Continue this process until ab is lower tangent to both S_1 & S_2.

  Note: *T = ab is said to be lower tangent at point a, if both a-1 & a+ 1 lie above T. This is equivalent to say that both of these points are left or on ab.*

So, the algorithm structure to find tangent may be;

a = rightmost point of S_1

b = leftmost point of S_2.

while T = ab not lower tangent to both S_1 & S_2 do

  while T is not lower tangent to S_1 do

    a← a–1 //a is index of rightmost point of S_1 & we are in clockwise so a–1

  while T is not lower tangent to S_2 do

    b← b+1 //b is index of leftmost point in S_2 & we are in counterclockwise so b+1

Similar approach can be used to find the ***upper tangent***.

Complexity analysis:

- Divide step is done in O(nlogn) as it requires sorting.
- Lower & upper tangents can be found in O(n) time.
- Now time for recursive computation can be expressed as;

  $T(n) = 2\,T(n/2) + O(n)$

  Clearly, its solution is O(nlogn).

  Hence, altogether the complexity is O(nlogn).

7.) Incremental Algorithm for computing convex hull:

Here, the general idea is to add the points one at a time and update the hull constructed so far at each step.

Let the given point set be $P = \{p_0, p_1, p_2, p_3, \text{--------}, p_{n-1}\}$

Now, the computation of convex hull comprises:

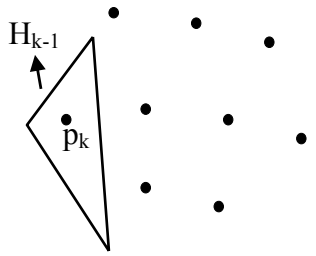
- $H_2 \leftarrow$ convex hull (p_0, p_1, p_2)
- For k=3 to n-1

  $Hk = \text{ConvexHull}\,(H_{k-1} \cup p_k)$

Now computing $H_{k-1} \cup p_k$, we can have two cases;

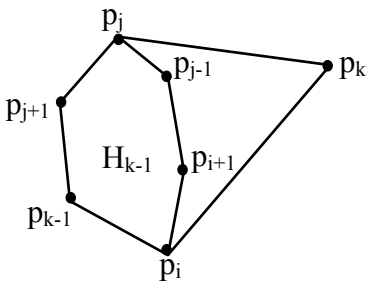
Case I $p_k \in H_{k-1}$

- If $p_k \in H_{k-1}$ then simply discard p_k such that, $H_k = H_{k-1}$
- For this test using left or on test for H_{k-1}.
  If left turn test is true for every edge of H_{k-1}, then
  $p_k \in H_{k-1}$ so discard it.



Case II:

- If $p_k \notin H_{k-1}$, then $H_k = H_{k-1} \cup p_k$.
  For this the left turn test will be false for any edge of H_{k-1}, so, $p_k \notin H_{k-1}$.
- Now, to modify the previous hull i.e. H_{k-1}, find the upper & lower tangent from p_k to H_{k-1}.
  Remove all points between upper & lower tangent-point.
- To find tangent from p_k, check for each point p_i of H_{k-1}



- If lefton (p_{i-1}, p_i, p_k) is true & lefton(p_i, p_{i+1}, p_k) is false then there is a lower tangent at pi from pk.
- If lefton (p_{j-1}, p_j, p_k) is false and lefton (p_j, p_{j+1}, p_k) is true then there is an upper tangent at p_j form p_k.
- Removing points between p_i & p_j & adding p_k at that position, we can get $H_{k-1} \cup p_k$.

Complexity Analysis:

- Here each step takes O(n) time.
- For worst case each $p_k \notin H_{k-1}$ so there will be at most n such steps.

  Hence, complexity turns out be $O(n^2)$.