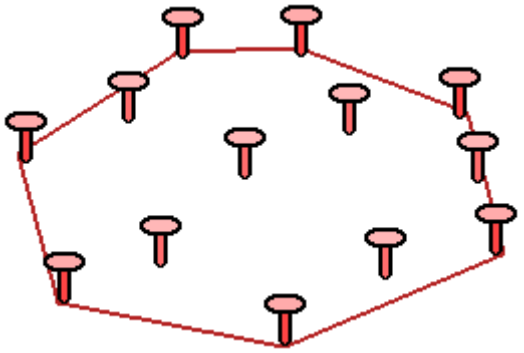
Convex Hull of N points

Convex Hull of a set of N points is the smallest convex polygon that contains all the given points.It has a property that all the internal angles of the polygon should be **convex, i.e < 180°**.



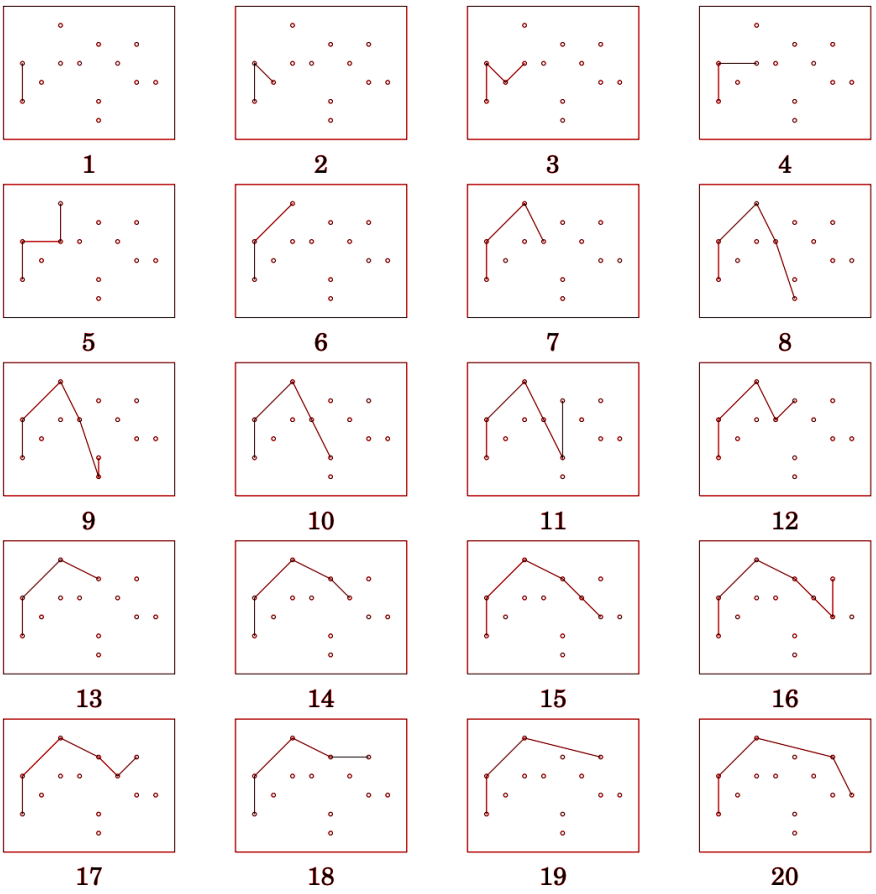
Here is a nice visualization to understand the concept of convex Hull. Consider each of the N points as a nail and the hull as the rubber band that covers all the points. It is formed when we stretch the band over all the nails and allows it to take the smallest possible convex shape.

We will discuss the **Graham's Scan Algorithm** to find the convex Hull of the given set of points.

Graham's Scan Algorithm

The algorithm first locates the leftmost and rightmost points, and then constructs the convex hull in two parts: first the upper hull and then the lower hull. Both parts are similar, so we can focus on constructing the upper hull:

- First, we sort the points primarily according to x coordinates and secondarily according to y coordinates.
- After this, we go through the points and add each point to the hull. Always after adding a point to the hull, we make sure that the orientation of the last two points and the current point is clockwise. As long as it is anticlockwise we pop the last point from the stack.



The above figure shows the construction of the upper hull algorithm using Graham's Scan.

So the Algorithm can be summarized as follows :-
Let points[0..n-1] be the input array.

- First sort all the n points with respect to their x coordinates. If there are two points with the same x value, then the point with a smaller y coordinate value is considered first.
- Let the left-most point be P0. Put P0 at first position in the output hull
- If n is less than 3, return (Convex Hull not possible)
- Create an empty stack 'S' and push points[0], points[1] and points[2] to S.
- Process remaining n-3 points one by one. Do following for every point 'points[i]'
    - Keep removing points from the stack while the orientation of the following 3 points is not counterclockwise (or they don't make a left turn).
        - Point next to top in stack
        - Point at the top of stack
        - points[i]
    - Push points[i] to S
- Print contents of S

Here is the pseudo code for the above idea .

Pseudocode:

```
/*
        function to check if the three points
        a, b and c are in clockwise orientation
        about point b by calculating the cross product
        (AB)x(BC)
*/
function cw(pt a, pt b, pt c)
        return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;


/*
        function to check if the three points
        a, b and c are in counter clockwise orientation
        about point b by calculating the cross product
        (AB)x(BC)
*/
function ccw(pt a, pt b, pt c)
        return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;


/*
        function to return the convex hull of the
        array a
*/

function convex_hull(Points[] a)

        n = a.size

        //  no convex hull is possible
        if (n <= 2)
                return

        //  sort a according to x coordinate
        sort(a)

        //  take the leftmost and rightmost points
        p1 = a[0], p2 = a.back()

        //  arrays to store the upper and the lower hull
```

```
up = []
down = []
//   add the leftmost point to both the hulls
up.append(p1)
down.append(p1)

//   loop from 1 to n  - 1
for i from 1 to n - 1

        if (i == n - 1 || cw(p1, a[i], p2))
                /*
                        pop the points from the upper hull if the
                        last 3 points are not clockwise oriented
                */
                while (up.size >= 2 && !cw(up[up.size-2], up[up.size-1], a[i]))
                        up.pop_back()

                //   add the new point to the hull
                up.append(a[i])

        if (i == n - 1 || ccw(p1, a[i], p2))
                /*
                        pop the points from the lower hull if the
                        last 3 points are not counterclockwise oriented
                */
                while(down.size >= 2 && !ccw(down[down.size-2], down[down.size-1], a[i]))
                        down.pop_back()
                //   add the new point to the hull
                down.append(a[i])



//    merge the upper and the lower hull
a = merge(up, down)

//    return the final convex hull
return a
```

TIme complexity : O(NlogN), where N is the total number of points in the input array. We use an O(NlogN) algorithm to sort the input array and then use the monotonic stack (where each element is pushed and popped at most once) to compute the convex hull in O(N) time. Hence the final time complexity is O(NlogN) + O(N) = O(NlogN).

Space complexity : O(N), where N is the total number of points in the input array. We use two arrays ‘up’ and ‘down’ to store the upper and lower hulls, sizes of both of which are bounded by O(N).