**Convex Hull**

Convex Hull of a set of points, in 2D plane, is a convex polygon with minimum area such that each point lies either on the boundary of polygon or inside it. Now given a set of points the task is to find the convex hull of points.

**Input:**  
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains an integer N denoting the no of points. Then in the next line are N\*2 space separated values denoting the points ie x and y.

**Output:**  
For each test case in a new line print the points x and y of the convex hull separated by a space in sorted order where every pair is separated from the other by a ','. If no convex hull is possible print -1.

**Constraints:**  
1 <= T <= 100  
1 <= N <= 100  
1 <= x,y <= 1000

**Example:  
Input:**  
2  
3  
1 2 3 1 5 6  
3  
1 2 4 4 5 1  
**Output:**  
1 2, 3 1, 5 6  
1 2, 4 4, 5 1

* Solution

#include <bits/stdc++.h>

using namespace std;

struct Point

{

    int x, y;

};

// To find orientation of ordered triplet (p, q, r).

// The function returns following values

// 0 --> p, q and r are colinear

// 1 --> Clockwise

// 2 --> Counterclockwise

int orientation(Point p, Point q, Point r)

{

    int val = (q.y - p.y) \* (r.x - q.x) -

              (q.x - p.x) \* (r.y - q.y);

    if (val == 0) return 0;  // colinear

    return (val > 0)? 1: 2; // clock or counterclock wise

}

// Prints convex hull of a set of n points.

void convexHull(Point points[], int n)

{

    // There must be at least 3 points

    if (n < 3) return;

    // Initialize Result

    vector<Point> hull;

    // Find the leftmost point

    int l = 0;

    for (int i = 1; i < n; i++)

        if (points[i].x < points[l].x)

            l = i;

    // Start from leftmost point, keep moving counterclockwise

    // until reach the start point again.  This loop runs O(h)

    // times where h is number of points in result or output.

    int p = l, q;

    do

    {

        // Add current point to result

        hull.push\_back(points[p]);

        // Search for a point 'q' such that orientation(p, x,

        // q) is counterclockwise for all points 'x'. The idea

        // is to keep track of last visited most counterclock-

        // wise point in q. If any point 'i' is more counterclock-

        // wise than q, then update q.

        q = (p+1)%n;

        for (int i = 0; i < n; i++)

        {

           // If i is more counterclockwise than current q, then

           // update q

           if (orientation(points[p], points[i], points[q]) == 2)

               q = i;

        }

        // Now q is the most counterclockwise with respect to p

        // Set p as q for next iteration, so that q is added to

        // result 'hull'

        p = q;

    } while (p != l);  // While we don't come to first point

    // Print Result

    for (int i = 0; i < hull.size(); i++)

        cout << "(" << hull[i].x << ", "

              << hull[i].y << ")\n";

}

// Driver program to test above functions

int main()

{

    Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1},

                      {3, 0}, {0, 0}, {3, 3}};

    int n = sizeof(points)/sizeof(points[0]);

    convexHull(points, n);

    return 0;

}

Convex Hull | Set 1 (Jarvis’s Algorithm or Wrapping)

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

We strongly recommend to see the following post first.  
[How to check if two given line segments intersect?](https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/)

The idea of Jarvis’s Algorithm is simple, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. The big question is, given a point p as current point, how to find the next point in output? The idea is to use [orientation()](https://www.geeksforgeeks.org/orientation-3-ordered-points/) here. Next point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have “orientation(p, q, r) = counterclockwise”. Following is the detailed algorithm.

<https://media.geeksforgeeks.org/wp-content/uploads/ConvexHull.png>

We strongly recommend to see the following post first.  
[How to check if two given line segments intersect?](https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/)

The idea of Jarvis’s Algorithm is simple, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. The big question is, given a point p as current point, how to find the next point in output? The idea is to use [orientation()](https://www.geeksforgeeks.org/orientation-3-ordered-points/) here. Next point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have “orientation(p, q, r) = counterclockwise”. Following is the detailed algorithm.

1. Initialize p as leftmost point.  
   **2)** Do following while we don’t come back to the first (or leftmost) point.  
   …..**a)** The next point q is the point such that the triplet (p, q, r) is counterclockwise for any other point r.  
   …..**b)** next[p] = q (Store q as next of p in the output convex hull).  
   …..**c)** p = q (Set p as q for next iteration).

<https://media.geeksforgeeks.org/wp-content/uploads/JarvisAlgorithm.png>

**Time Complexity:** For every point on the hull we examine all the other points to determine the next point. Time complexity is ?(m \* n) where n is number of input points and m is number of output or hull points (m <= n). In worst case, time complexity is O(n 2). The worst case occurs when all the points are on the hull (m = n)