```
/*
Convex Hull (Graham Scan)

-> Let points[0.....n-1] be the input array.

1) Find the bottom-most point by comparing y coordinates of all points. If there
are two points with same y value, then the point with smaller x-coordinate value
is considered. Let the bottommost point be P0. Put P0 in first position in
output hull.

2) Considering the remaining n-1 points and sort them by poor angle in counter-
clockwise order around points[0]. If poor angle of two points is same, then put
the nearest point first.

3) After sorting, check if two or more points have same angle. If so, then
remove all same angle points except the farthest from P0. Let the size of new
array be 'm'.

4) If m is less than 3, return (Convex Hull not possible).

5) Create an empty stack 'S' and push points[0], points[1] and points[2] to S.

6) Process remaining m-3 points one by one. Do following for every point
'points[i]' :
      6.1) Keep removing points from stack while orientation of following 3
points is not counter- clockwise (or they don't make a left
      turn).
            a) Point next to top in stack
            b) Point at the top of stack
            c) points[i]
      6.2) Push points[i] to S.

7) Print contents of S.
*/

#include<bits/stdc++.h>

using namespace std;

struct Point
{
      int x,y;
};

struct Point p0;

Point nextToTop(stack<Point> &S)
{
      Point p=S.top();
      S.pop();
      Point res=S.top();
      S.push(p);
      return res;
}

int swap(Point &p1, Point &p2)
{
      Point temp=p1;
      p1=p2;
      p2=temp;
}

int distSq(Point p1, Point p2)
{
      return (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
}

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;
      return (val>0)?1:2;
}

int compare(const void *vp1, const void *vp2)
{
      Point *p1=(Point *)vp1;
      Point *p2=(Point *)vp2;

      int o=orientation(p0,*p1,*p2);
      if(o==0)
            return (distSq(p0,*p2)>=distSq(p0,*p1))?-1:1;
      return (o==2)?-1:1;
}

void convexHull(Point points[], int n)
{
      int ymin=points[0].y,min=0;

      for(int i=1;i<n;i++)
      {
            int y=points[i].y;
            if((y<ymin) || (ymin==y && points[i].x<points[min].x))
                  ymin=points[i].y,min=i;
      }

      swap(points[0],points[min]);
      p0=points[0];
      qsort(&points[1],n-1,sizeof(Point),compare);

      int m=1;
      for(int i=1;i<n;i++)
      {
            while(i<n-1 && orientation(p0,points[i],points[i+1])==0)
                  i++;
            points[m]=points[i];
            m++;
      }

      if(m<3)
            return;

      stack<Point> S;
      S.push(points[0]);
      S.push(points[1]);
      S.push(points[2]);

      for(int i=3;i<m;i++)
      {
            while(orientation(nextToTop(S),S.top(),points[i])!=2)
                  S.pop();
            S.push(points[i]);
      }

      while(!S.empty())
      {
            Point p=S.top();
            cout<<"("<<p.x<<", "<<p.y<<")"<<endl;
            S.pop();
      }
}

int main()
{
      Point points[]={{0,3},{1,1},{2,2},{4,4},{0,0},{1,2},{3,1},{3,3}};
      int n=sizeof(points)/sizeof(points[0]);
      convexHull(points,n);
      return 0;
}

/*

Output :
(0, 3)
(4, 4)
(3, 1)
(0, 0)

*/
```