Graham:

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

struct Point {

    int x, y;

};

Point p0;

void swap(Point &p1, Point &p2) {

    Point temp = p1;

    p1 = p2;

    p2 = temp;

}

int khoangcach(Point p1, Point p2) {

    return (p1.x - p2.x) \* (p1.x - p2.x) + (p1.y - p2.y) \* (p1.y - p2.y);

}

int goclonnhat(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 : -1;

}

bool sosanh(Point p1, Point p2) {

    int o = goclonnhat(p0, p1, p2);

    if (o == 0)

        return khoangcach(p0, p2) >= khoangcach(p0, p1);

    return (o == -1);

}

vector<Point> Graham(vector<Point> &points) {

    int n = points.size();

    int ymin = points[0].y, min = 0;

    for (int i = 1; i < n; i++) {

        int y = points[i].y;

        if ((y < ymin) || (y == ymin && points[i].x < points[min].x)) {

            ymin = points[i].y;

            min = i;

        }

    }

    swap(points[0], points[min]);

    p0 = points[0];

    sort(points.begin() + 1, points.end(), sosanh);

    vector<Point> baoloi;

    baoloi.push\_back(points[0]);

    baoloi.push\_back(points[1]);

    baoloi.push\_back(points[2]);

    for (int i = 3; i < n; i++) {

        while (baoloi.size() > 1 && goclonnhat(baoloi[baoloi.size() - 2], baoloi.back(), points[i]) != -1) {

            baoloi.pop\_back();

        }

        baoloi.push\_back(points[i]);

    }

    return baoloi;

}

double dientich(const vector<Point> &baoloi) {

    int n = baoloi.size();

    double dientich = 0;

    for (int i = 0; i < n - 1; i++) {

        dientich += baoloi[i].x \* baoloi[i + 1].y - baoloi[i].y \* baoloi[i + 1].x;

    }

    dientich += baoloi[n - 1].x \* baoloi[0].y - baoloi[n - 1].y \* baoloi[0].x;

    return abs(dientich) / 2.0;

}

int main() {

    vector<Point> points = {{0, 0}, {1, 1}, {2, 2}, {2, 0}, {3, 1}, {0, 3}, {3, 3},{4,3},{6,7},{2,9},{1,10},{1,9},{1,3},{1,5},{4,1}};

    vector<Point> baoloi = Graham(points);

    cout << "Convex baoloi: \n";

    for (auto &p : baoloi) {

        cout << "(" << p.x << ", " << p.y << ")\n";

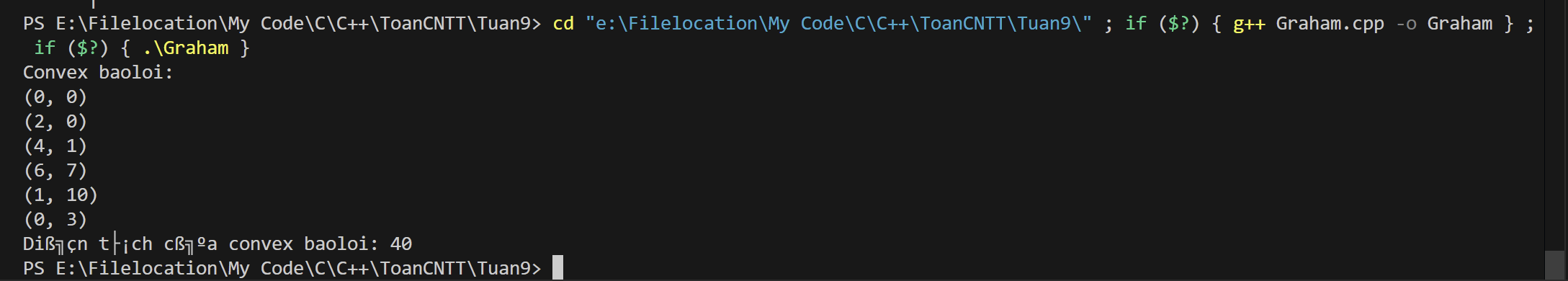
    }

    double area = dientich(baoloi);

    cout << "Diện tích của convex baoloi: " << area << endl;

    return 0;

}

Kết quả:

Chia để trị:

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <stack>

#include <math.h>

#include <cmath>

using namespace std;

int x[101] = {-7, -10, -7, -9, 4, 6, -2, -4, 10, 0, 8, 8, -10, 8, 6},

    y[101] = {-1, 7, -9, -4, -6, 1, 6, -5, -1, 10, -6, 10, -8, 1, -5}, n;

int arr[101], c;

void sinhCapSoNguyen(int n) {

std::srand(std::time(0));

    for (int i = 0; i < n; ++i) {

        bool trungLap;

        do {

            trungLap = false;

            x[i] = (std::rand() % 21) - 10;

            y[i] = (std::rand() % 21) - 10;

            for (int j = 0; j < i; ++j) {

                if (x[i] == x[j] && y[i] == y[j]) {

                    trungLap = true;

                    break;

                }

            }

        } while (trungLap);

    }

}

void print(int n)

{

    for(int i = 0; i < n; i++)

        std::cout << "Cap so " << i + 1 << ": (" << x[i] << ", " << y[i] << ")\n";

}

// thực hiện sắp xép theo qsort

bool soSanh(int i, int j)

{

    if(x[i] < x[j])

        return true;

    if(x[i] > x[j])

        return false;

    if(y[i] < y[j])

        return true;

    return false;

}

void SWAP(int a[], int i, int j)

{

    int t = a[i];

    a[i] = a[j];

    a[j] = t;

}

int partition(int p, int r)

{

    int pos = r;

    int i = p - 1;

    for(int j = p; j < r; j++)

    {

        if(soSanh(j, r))

        {

            i++;

            SWAP(x, i, j);

            SWAP(y, i, j);

        }

    }

    SWAP(x, i + 1, r);

    SWAP(y, i + 1, r);

    return i + 1;

}

void quickSort(int p, int r)

{

    if(p < r)

    {

        int q = partition(p, r);

        quickSort(p, q - 1);

        quickSort(q + 1, r);

    }

}

// hàm tìm khoảng cách giữa 2 điểm i, j:

double distance(int i, int j)

{

    return sqrt((x[i] - x[j]) \* (x[i] - x[j]) + (y[i] - y[j]) \* (y[i] - y[j]));

}

int xetViTri(int p1, int p2, int p) // true: P nằm bên phải AB

{

    // Tính tích có hướng

    int D = (x[p2] - x[p1]) \* (y[p] - y[p1]) - (y[p2] - y[p1]) \* (x[p] - x[p1]);

    // Nếu D < 0, P nằm bên phải AB, trả về true

    if(D < 0)

        return 1;

    else if(D == 0)

        return 0;

    return -1;

}

void DivideAndConquer(int arr[101], int s, int f)

{

    // xử lí trường hợp có 2 - 3 điểm

    if(f - s == 3)

    {

        arr[0] = 3;

        arr[1] = s;

        arr[2] = s + 1;

        arr[3] = f - 1;

        arr[4] = 3;

        if(xetViTri(s, s + 1, f - 1) == 1)

        {

            SWAP(arr, 2, 3);

            arr[4] = 2;

        }

        else

        {

            if(xetViTri(s, s + 1, f - 1) == 0)  // 3 điểm thẳng hàng

            {

                arr[0] = 2;

                arr[2] = s + 2;

                arr[3] = 2;

            }

        }

        //cout << 3;

        return;

    }

    else if(f - s == 2)

    {

        arr[0] = 2;

        arr[1] = s;

        arr[2] = f - 1;

        arr[3] = 2;

        //cout << 2;

        return;

    }

    else // trường hợp nhiều hơn 2-3 điểm

    {

        int leftArr[101], rightArr[101];

        int p = (s + f)/2;

        // chia thành 2 bao lồi con

        DivideAndConquer(leftArr, s, p);

        DivideAndConquer(rightArr, p, f);

        // ghép lại

        int p1 = leftArr[leftArr[0] + 1], p2 = 1;

        // ghép trên

        int trai = xetViTri(leftArr[p1], rightArr[p2], leftArr[p1 % leftArr[0] + 1]);

        int phai = xetViTri(leftArr[p1], rightArr[p2], rightArr[(p2 + rightArr[0] - 2) % rightArr[0] + 1]);

        //cout << p1 << ", " << p2 << ", " << p1 % leftArr[0] + 1 << ", " << (p2 + rightArr[0] - 2) % rightArr[0] + 1  << ", " << trai << ", " << phai << endl;

        while((trai <= 0 || phai <= 0) )

        //while((trai <= 0 || phai <= 0) && ((leftArr[leftArr[0] + 1] != p1 % leftArr[0] + 1) || (1 != (p2 + rightArr[0] - 2) % rightArr[0] + 1)))

        {

            if(trai <= 0 && (leftArr[leftArr[0] + 1] != p1 % leftArr[0] + 1))

            {

                p1 = p1 % leftArr[0] + 1;

                trai = xetViTri(leftArr[p1], rightArr[p2], leftArr[p1 % leftArr[0] + 1]);

                phai = xetViTri(leftArr[p1], rightArr[p2], rightArr[(p2 + rightArr[0] - 2) % rightArr[0] + 1]);

            }

            else if(phai <= 0 && (1 != (p2 + rightArr[0] - 2) % rightArr[0] + 1))

            {

                p2 = (p2 + rightArr[0] - 2) % rightArr[0] + 1;

                trai = xetViTri(leftArr[p1], rightArr[p2], leftArr[p1 % leftArr[0] + 1]);

                phai = xetViTri(leftArr[p1], rightArr[p2], rightArr[(p2 + rightArr[0] - 2) % rightArr[0] + 1]);

            }

            else

                break;

                   // cout << p1 << ", " << p2 << ", " << p1 % leftArr[0] + 1 << ", " << (p2 + rightArr[0] - 2) % rightArr[0] + 1  << ", " << trai << ", " << phai << endl;

            //cout << p1 << ", " << p2 << endl;

        }

        //cout << "Oke" << endl;

        //cout << p1 << ", " << p2 << endl;

        //cout << "-----------------" << endl;

        // ghép dưới

        int p3 = leftArr[leftArr[0] + 1], p4 = 1;

        trai = xetViTri(leftArr[p3], rightArr[p4], leftArr[(p3 + leftArr[0] - 2) % leftArr[0] + 1]);

        phai = xetViTri(leftArr[p3], rightArr[p4], rightArr[p4 % rightArr[0] + 1]);

        //cout << p3 << ", " << p4 << ", " << trai << ", " << phai << endl;

        while((trai >= 0 || phai >= 0))

        //while((trai >= 0 || phai >= 0) && (leftArr[leftArr[0] + 1] != (p3 + leftArr[0] - 2) % leftArr[0] + 1) && (1 != p4 % rightArr[0] + 1))

        {

            if(trai >= 0 && (leftArr[leftArr[0] + 1] != (p3 + leftArr[0] - 2) % leftArr[0] + 1))

            {

                p3 = (p3 + leftArr[0] - 2) % leftArr[0] + 1;

                trai = xetViTri(leftArr[p3], rightArr[p4], leftArr[(p3 + leftArr[0] - 2) % leftArr[0] + 1]);

                phai = xetViTri(leftArr[p3], rightArr[p4], rightArr[p4 % rightArr[0] + 1]);

            }

            else if(phai >= 0 && (1 != p4 % rightArr[0] + 1))

            {

                p4 = p4 % rightArr[0] + 1;

                trai = xetViTri(leftArr[p3], rightArr[p4], leftArr[(p3 + leftArr[0] - 2) % leftArr[0] + 1]);

                phai = xetViTri(leftArr[p3], rightArr[p4], rightArr[p4 % rightArr[0] + 1]);

            }

            else

                break;

            //cout << p3 << ", " << p4 << endl;

        }

        //cout << p1 << ", " << p2 << ", " << p3 << ", " << p4 << endl;

        // ghép toàn bộ

        int c = 0;

        int cmax = 1;

        for(int i = 1; i <= p3; i++)

        {

            c++;

            arr[c] = leftArr[i];

        }

        if(p4 <= p2)

        {

            for(int i = p4; i <= p2; i++)

            {

                c++;

                arr[c] = rightArr[i];

                if(rightArr[rightArr[0] + 1] == i)

                    cmax = c;

            }

        }

        else

        {

            for(int i = p4; i <= rightArr[0]; i++)

            {

                c++;

                arr[c] = rightArr[i];

                if(rightArr[rightArr[0] + 1] == i)

                    cmax = c;

            }

            for(int i = 1; i <= p2; i++)

            {

                c++;

                arr[c] = rightArr[i];

                if(rightArr[rightArr[0] + 1] == i)

                    cmax = c;

            }

        }

        for(int i = p1;p1 != 1 && i <= leftArr[0]; i++)

        {

            c++;

            arr[c] = leftArr[i];

        }

        arr[0] = c;

        arr[c + 1] = cmax;

        //cout << f << " " << c << endl;

    }

}

void xuLiSau()

{

    c = arr[0];

    for(int i = 1; i <= c; i++)

    {

        arr[i - 1] = arr[i];

    }

    arr[c] = arr[0];

}

int dienTich()

{

    int res = 0;

    for(int i = 0; i < c; i++)

    {

        res += x[arr[i]]\*y[arr[i + 1]] - x[arr[i + 1]]\*y[arr[i]];

    }

    return res;

}

// in ra các đỉnh của bao lồi

void inKetQua()

{

    cout << "Bao loi " << c << " dinh:" << endl;

    for(int i = 0; i < c; i++)

        cout << x[arr[i]] << "  " << y[arr[i]] << endl;

}

// hàm thực hiện tìm 2 điểm ngắn nhất trong C đỉnh của BAO LỒI

// do something -  thực hiện thuật toán chia để trị

// tìm điểm ngắn nhất trong đoạn [s, f)

double closetPoint(int s, int f)

{

    double minDistance = 1000000000;

    // trong trường hợp có tối đa 3 phần tử --> bruteForce

    if(f - s <= 3)

    {

        for(int i = s; i < f; i++)

        {

            for(int j = i + 1; j < f; j++)

                minDistance = min(distance(arr[i], arr[j]), minDistance);

        }

        return minDistance;

    }

    int p = (s + f)/2;

    minDistance = min(closetPoint(s, p), closetPoint(p, f)); // khoảng cách ngắn nhất giữa 2 điểm trong 2 bên

    // tìm khoảng cách ngắn nhất giữa 2 điểm với mỗi điểm 1 bên:

    int l = p - 1;

    while (l > s && x[arr[p]] - x[arr[l - 1]] < minDistance)

    {

        l--;

    }

    for(int i = l; i < p; i++)

        for(int j = p; j < f && x[arr[j]] - x[arr[i]] <= minDistance; j++)

            minDistance = min(minDistance, distance(arr[i], arr[j]));

    return minDistance;

}

int compare(const void \*a, const void \*b) {

    return (\*(int\*)a - \*(int\*)b);

}

int main() {

    n = 15;

    print(n);

    quickSort(0, n - 1);

    DivideAndConquer(arr, 0, n);

    xuLiSau();

    inKetQua();

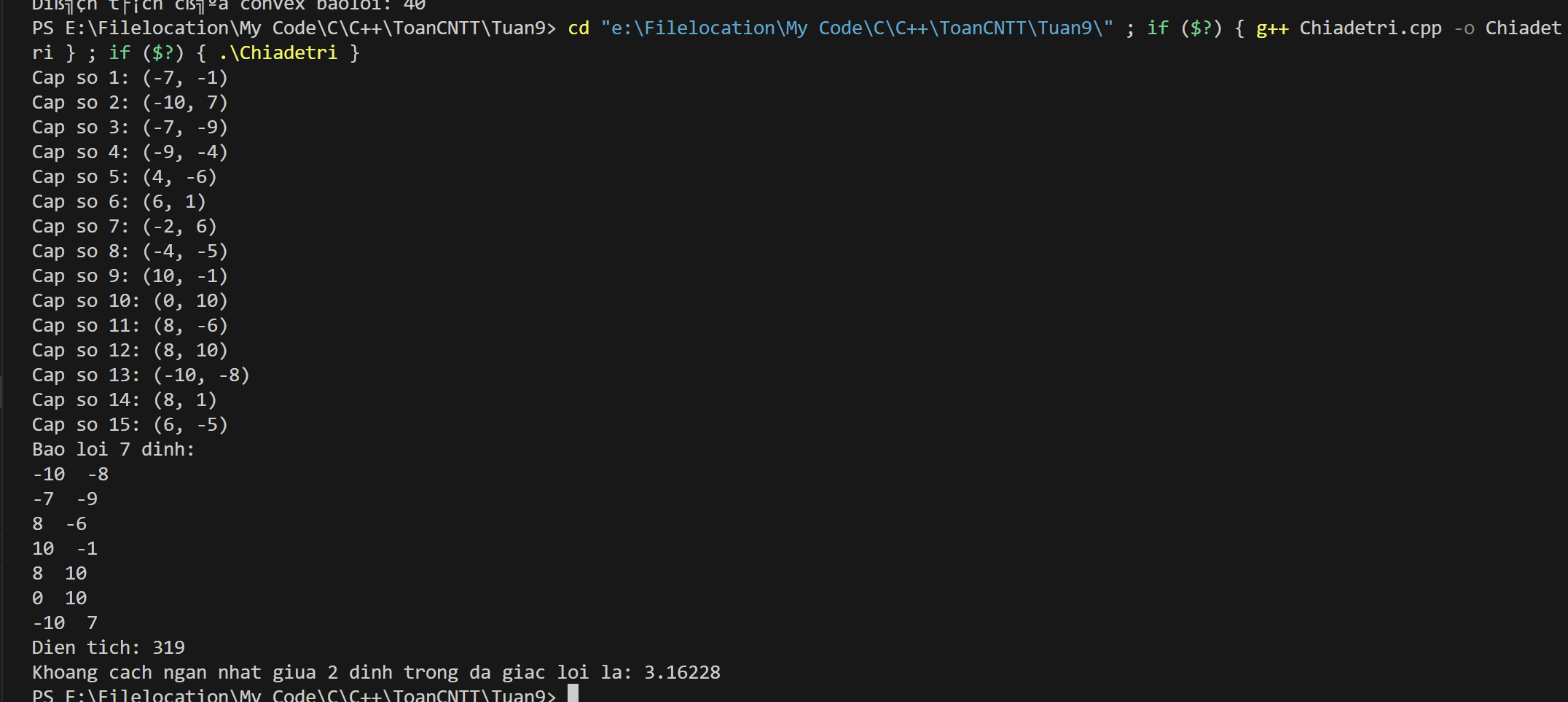
    cout << "Dien tich: " << fabs(((float)dienTich())/2) << endl;

    qsort(arr, c, sizeof(int), compare);

    cout << "Khoang cach ngan nhat giua 2 dinh trong da giac loi la: " << closetPoint(0, c) << endl;

    return 0;

}

Kết quả: 

Javi:

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <algorithm>    // For qsort() algorithm

#include <utility>      // For pair() STL

#define RIGHT\_TURN -1  // CW

#define LEFT\_TURN 1  // CCW

#define COLLINEAR 0  // Collinear

using namespace std;

/\*

Class to handle the 2D points!

\*/

class point

{

public:

    int x;

    int y;

    point (int newx=0,int newy=0){

        x=newx;

        y=newy;

    }

    /\*

        Overloaded == operator to check for equality between 2 objects of class point

    \*/

    friend bool operator== (const point& p1,const point& p2){

        return (p1.x==p2.x && p1.y==p2.y);

    }

    /\*

        Overloaded != operator to check for non-equality between 2 objects of class point

    \*/

    friend bool operator!= (const point& p1,const point& p2){

        return (!(p1.x==p2.x && p1.y==p2.y));

    }

    /\*

        Overloaded ostream << operator to check for print object of class point to STDOUT

    \*/

    friend ostream& operator<<(ostream& output,const point& p){

        output<<"("<<p.x<<","<<p.y<<")";

        return output;

    }

}p0; // Global point class object

/\*

    Returns square of the distance between the two point class objects

    @param p1: Object of class point aka first point

    @param p2: Object of class point aka second point

\*/

int dist(point p1, point p2)

{

    return (p1.x - p2.x)\*(p1.x - p2.x) + (p1.y - p2.y)\*(p1.y - p2.y);

}

/\*

    Returns orientation of the line joining points p and q and line joining points q and r

    Returns -1 : CW orientation

            +1 : CCW orientation

            0 : Collinear

    @param p: Object of class point aka first point

    @param q: Object of class point aka second point

    @param r: Object of class point aka third point

\*/

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;  // Collinear

    return (val > 0)? -1: 1; // CW: -1 or CCW: 1

}

/\*

    Predicate function used while sorting the points using qsort() inbuilt function in C++

    @param p: Object of class point aka first point

    @param p: Object of class point aka second point

\*/

int compare(const void \*vp1, const void \*vp2)

{

    point \*p1 = (point \*)vp1;

    point \*p2 = (point \*)vp2;

    int orient = orientation(p0, \*p1, \*p2);

    if (orient == 0)

        return (dist(p0, \*p2) >= dist(p0, \*p1))? -1 : 1;

    return (orient == 1)? -1: 1;

}

/\*

    Returns the index of the point to which the tangent is drawn from point p.

    Uses a modified Binary Search Algorithm to yield tangent in O(log n) complexity

    @param v: vector of objects of class points representing the hull aka the vector of hull points

    @param p: Object of class point from where tangent needs to be drawn

\*/

int tangent(vector<point> v,point p){

    int l=0;

    int r= v.size();

    int l\_before = orientation(p, v[0], v[v.size()-1]);

    int l\_after = orientation(p, v[0], v[(l + 1) % v.size()]);

    while (l < r){

        int c = ((l + r)>>1);

        int c\_before = orientation(p, v[c], v[(c - 1) % v.size()]);

        int c\_after = orientation(p, v[c], v[(c + 1) % v.size()]);

        int c\_side = orientation(p, v[l], v[c]);

        if (c\_before != RIGHT\_TURN and c\_after != RIGHT\_TURN)

            return c;

        else if ((c\_side == LEFT\_TURN) and (l\_after == RIGHT\_TURN or l\_before == l\_after) or (c\_side == RIGHT\_TURN and c\_before == RIGHT\_TURN))

            r = c;

        else

            l = c + 1 ;

        l\_before = -c\_after;

        l\_after = orientation(p, v[l], v[(l + 1) % v.size()]);

    }

    return l;

}

/\*

    Returns the pair of integers representing the Hull # and the point in that Hull which is the extreme amongst all given Hull Points

    @param hulls: Vector containing the hull points for various hulls stored as individual vectors.

\*/

pair<int,int> extreme\_hullpt\_pair(vector<vector<point> >& hulls){

    int h= 0,p= 0;

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

        int min\_index=0, min\_y = hulls[i][0].y;

        for(int j=1; j< hulls[i].size(); ++j){

            if(hulls[i][j].y < min\_y){

                min\_y=hulls[i][j].y;

                min\_index=j;

            }

        }

        if(hulls[i][min\_index].y < hulls[h][p].y){

            h=i;

            p=min\_index;

        }

    }

    return make\_pair(h,p);

}

/\*

    Returns the pair of integers representing the Hull # and the point in that Hull to which the point lpoint will be joined

    @param hulls: Vector containing the hull points for various hulls stored as individual vectors.

    @param lpoint: Pair of the Hull # and the leftmost extreme point contained in that hull, amongst all the obtained hulls

\*/

pair<int,int> next\_hullpt\_pair(vector<vector<point> >& hulls, pair<int,int> lpoint){

    point p = hulls[lpoint.first][lpoint.second];

    pair<int,int> next = make\_pair(lpoint.first, (lpoint.second + 1) % hulls[lpoint.first].size());

    for (int h=0; h< hulls.size(); h++){

        if(h != lpoint.first){

            int s= tangent(hulls[h],p);

            point q= hulls[next.first][next.second];

            point r= hulls[h][s];

            int t= orientation(p,q,r);

            if( t== RIGHT\_TURN || (t==COLLINEAR) && dist(p,r)>dist(p,q))

                next = make\_pair(h,s);

        }

    }

    return next;

}

/\*

    Constraint to find the outermost boundary of the points by checking if the points lie to the left otherwise adding the given point p

    Returns the Hull Points

    @param v: Vector of all the points

    @param p: New point p which will be checked to be in the Hull Points or not

\*/

vector<point> keep\_left (vector<point>& v,point p){

    while(v.size()>1 && orientation(v[v.size()-2],v[v.size()-1],p) != LEFT\_TURN)

        v.pop\_back();

    if(!v.size() || v[v.size()-1] != p)

        v.push\_back(p);

    return v;

}

/\*

    Graham Scan algorithm to find convex hull from the given set of points

    @param points: List of the given points in the cluster (as obtained by Chan's Algorithm grouping)

    Returns the Hull Points in a vector

\*/

vector<point> GrahamScan(vector<point>& points){

    if(points.size()<=1)

        return points;

    qsort(&points[0], points.size(), sizeof(point), compare);

    vector<point> lower\_hull;

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

        lower\_hull = keep\_left(lower\_hull,points[i]);

    reverse(points.begin(),points.end());

    vector<point> upper\_hull;

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

        upper\_hull = keep\_left(upper\_hull,points[i]);

    for(int i=1;i<upper\_hull.size();++i)

        lower\_hull.push\_back(upper\_hull[i]);

    return lower\_hull;

}

/\*

    Implementation of Chan's Algorithm to compute Convex Hull in O(nlogh) complexity

\*/

vector<point> chansalgorithm(vector<point> v){

    for(int t=0; t< v.size(); ++t){

        for(int m=1; m< (1<<(1<<t)); ++m){

            vector<vector<point> > hulls;

            for(int i=0;i<v.size();i=i+m){

                vector<point> chunk;

                if(v.begin()+i+m <= v.end())

                    chunk.assign(v.begin()+i,v.begin()+i+m);

                else

                    chunk.assign(v.begin()+i,v.end());

                hulls.push\_back(GrahamScan(chunk));

            }

            cout<<"\nM (Chunk Size): "<<m<<"\n";

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

                cout<<"Convex Hull for Hull #"<<i<<" (Obtained using Graham Scan!!)\n";

                for(int j=0; j<hulls[i].size();++j)

                    cout<<hulls[i][j]<<" ";

                cout<<"\n";

            }

            vector<pair<int,int> > hull;

            hull.push\_back(extreme\_hullpt\_pair(hulls));

            for(int i=0; i<m; ++i){

                pair<int,int> p= next\_hullpt\_pair(hulls,hull[hull.size()-1]);

                vector<point> output;

                if(p==hull[0]){

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

                        output.push\_back(hulls[hull[j].first][hull[j].second]);

                    }

                    return output;

                }

                hull.push\_back(p);

            }

        }

    }

}

int main()

{

    int T=0,x=0,y=0;

    cin>>T;

    if(T<=0)

        return -1;

    point points[T];

    for(int i=0;i<T;++i){

        cin>>x>>y;

        points[i].x=x;

        points[i].y=y;

    }

    vector<point> v(points,points+T);

    vector<point> output = chansalgorithm(v);

    cout<<"\n---------After Using Chan's Algorithm---------------\n";

    cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONVEX HULL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

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

        cout<<output[i]<<" ";

    cout<<"\n";

    return 0;

}

Kết quả:

