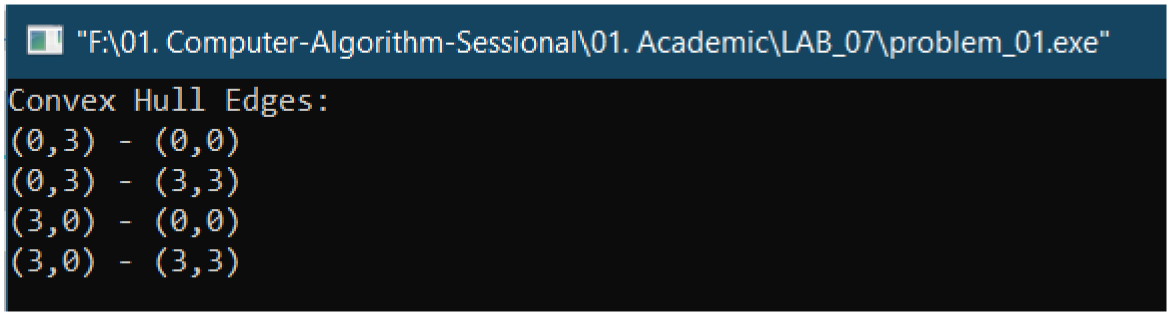
Program No: 01

Program statement: Convex Hull using Brute Force Algorithm

Program Code:

```
#include <iostream>
#include <vector>
using namespace std;
struct Point
{
    int x, y;
};

int direction(Point a, Point b, Point c)
{
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x);
}
void convexHull(vector<Point>& points)
{
    int n = points.size();
    cout << "Convex Hull Edges:\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            int pos = 0, neg = 0;
            for (int k = 0; k < n; k++)
            {
                if (k == i || k == j) continue;
                int d = direction(points[i], points[j], points[k]);
                if (d > 0) pos++;
                else if (d < 0) neg++;
            }
            if (pos == 0 || neg == 0)
                cout << "(" << points[i].x << "," << points[i].y << ") - ("
                     << points[j].x << "," << points[j].y << ")\n";
        }
    }
}
int main()
{
    vector<Point> points = {{0, 3}, {2, 2}, {1, 1}, {2, 1}, {3, 0}, {0, 0}, {3,
3}};
    convexHull(points);
    return 0;
}
```

Program Output:



"F:\01. Computer-Algorithm-Sessional\01. Academic\LAB_07\problem_01.exe"

```
Convex Hull Edges:
(0,3) - (0,0)
(0,3) - (3,3)
(3,0) - (0,0)
(3,0) - (3,3)
```

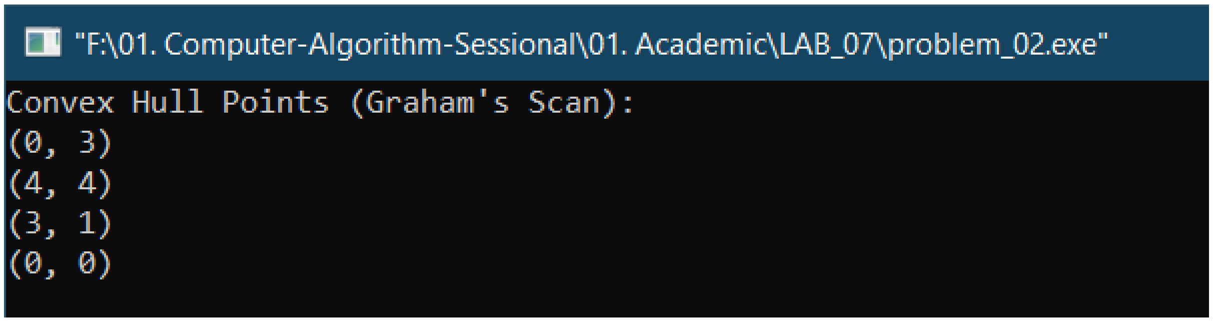
Program No: 02

Program statement: Convex Hull using Graham's Scan Algorithm

Program Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stack>
using namespace std;
struct Point
{
    int x, y;
};
Point p0;
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 distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);
}

bool compare(Point p1, Point p2)
{
    int o = orientation(p0, p1, p2);
    if (o == 0)
        return distSq(p0, p1) < distSq(p0, p2);
    return (o == 2);
}
void grahamScan(vector<Point>& points)
{
    int n = points.size();

    int ymin = points[0].y, minIndex = 0;
    for (int i = 1; i < n; i++)
    {
        if ((points[i].y < ymin) || (points[i].y == ymin && points[i].x <
points[minIndex].x))
        {
            ymin = points[i].y;

            minIndex = i;
        }
    }
    swap(points[0], points[minIndex]);
    p0 = points[0];
    sort(points.begin() + 1, points.end(), compare);
    stack<Point> hull;
    hull.push(points[0]);
    hull.push(points[1]);
    hull.push(points[2]);
    for (int i = 3; i < n; i++)
    {
        while (hull.size() > 1)
        {
            Point top = hull.top();
            hull.pop();
            Point nextToTop = hull.top();
            if (orientation(nextToTop, top, points[i]) != 2)
                continue;
            else
            {
                hull.push(top);
                break;
            }
        }
        hull.push(points[i]);
    }
    cout << "Convex Hull Points (Graham's Scan):\n";
    while (!hull.empty())
    {
        Point p = hull.top();
        cout << "(" << p.x << ", " << p.y << ")\n";
        hull.pop();
    }
}
int main()
{
    vector<Point> points = {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3,
1}, {3, 3}};
    grahamScan(points);
    return 0;
}
```

Program Output:



```
"F:\01. Computer-Algorithm-Sessional\01. Academic\LAB_07\problem_02.exe"
Convex Hull Points (Graham's Scan):
(0, 3)
(4, 4)
(3, 1)
(0, 0)
```

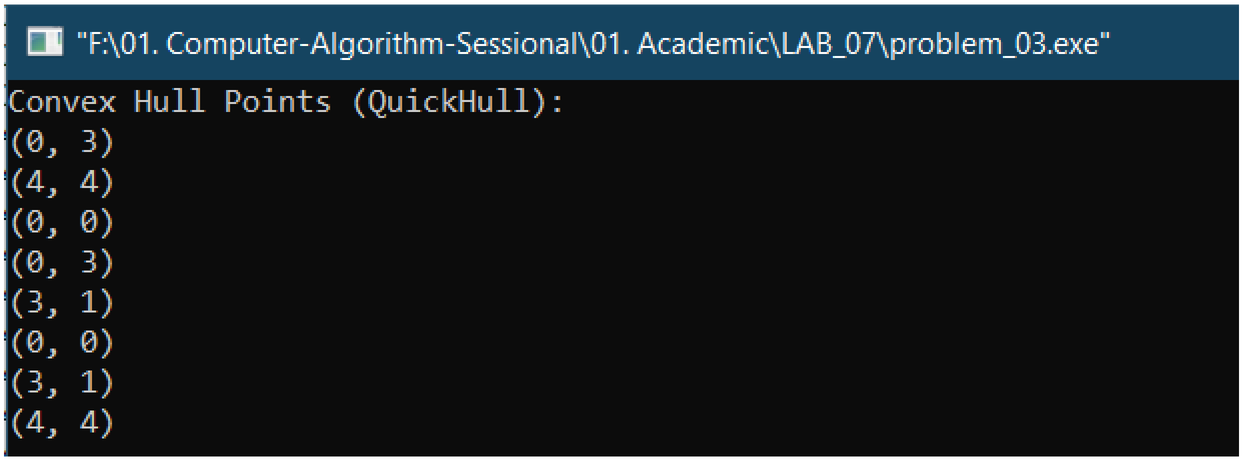
Program No: 03

Program statement: Convex Hull using QuickHull Algorithm

Program Code:

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

struct Point {
    int x, y;
};

int distance(Point A, Point B, Point C) {
    return abs((C.y - A.y) * B.x - (C.x - A.x) * B.y + C.x * A.y - C.y * A.x);
}

int findSide(Point A, Point B, Point P) {
    int val = (P.y - A.y) * (B.x - A.x) - (B.y - A.y) * (P.x - A.x);
    if (val > 0) return 1;
    if (val < 0) return -1;
    return 0;
}

void quickHull(vector<Point>& points, Point A, Point B, int side, vector<Point>&
hull) {
    int index = -1;
    int max_dist = 0;
    for (int i = 0; i < points.size(); i++) {
        int temp = distance(A, B, points[i]);
        if (findSide(A, B, points[i]) == side && temp > max_dist) {
            index = i;
            max_dist = temp;
        }
    }
    if (index == -1) {
        hull.push_back(A);
        hull.push_back(B);
        return;
    }
    quickHull(points, points[index], A, -findSide(points[index], A, B), hull);
    quickHull(points, points[index], B, -findSide(points[index], B, A), hull);
}

void findConvexHull(vector<Point>& points) {
    if (points.size() < 3) {
        cout << "Convex hull not possible\n";
        return;
    }
    int min_x = 0, max_x = 0;
    for (int i = 1; i < points.size(); i++) {
        if (points[i].x < points[min_x].x) min_x = i;
        if (points[i].x > points[max_x].x) max_x = i;
    }
    vector<Point> hull;
    quickHull(points, points[min_x], points[max_x], 1, hull);
    quickHull(points, points[min_x], points[max_x], -1, hull);

    cout << "Convex Hull Points (QuickHull):\n";
    for (auto& p : hull) {
        cout << "(" << p.x << ", " << p.y << ")\n";
    }
}

int main() {
    vector<Point> points = {
        {0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3, 1}, {3, 3}
    };
    findConvexHull(points);
    return 0;
}
```

Program Output:


"F:\01. Computer-Algorithm-Sessional\01. Academic\LAB_07\problem_03.exe"

```
Convex Hull Points (QuickHull):
(0, 3)
(4, 4)
(0, 0)
(0, 3)
(3, 1)
(0, 0)
(3, 1)
(4, 4)
```