Assignment 07 | Advance Algorithms CE-092

Assignment submission for Advance Algorithms subject week 7.

nevilparmar24@gmail.com

Task 1:

Write a program to find the Convex Hull for a given set of points in 2-D using Graham's Scan Method.

Code:

```
/*
* @Author: nevil
* @Date:   2020-09-04 16:11:58
* @Last Modified by:   nevil
* @Last Modified time: 2020-09-04 16:18:05
*/
#include<bits/stdc++.h>
using namespace std;

struct Point
{
    int x, y;
};

Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
```

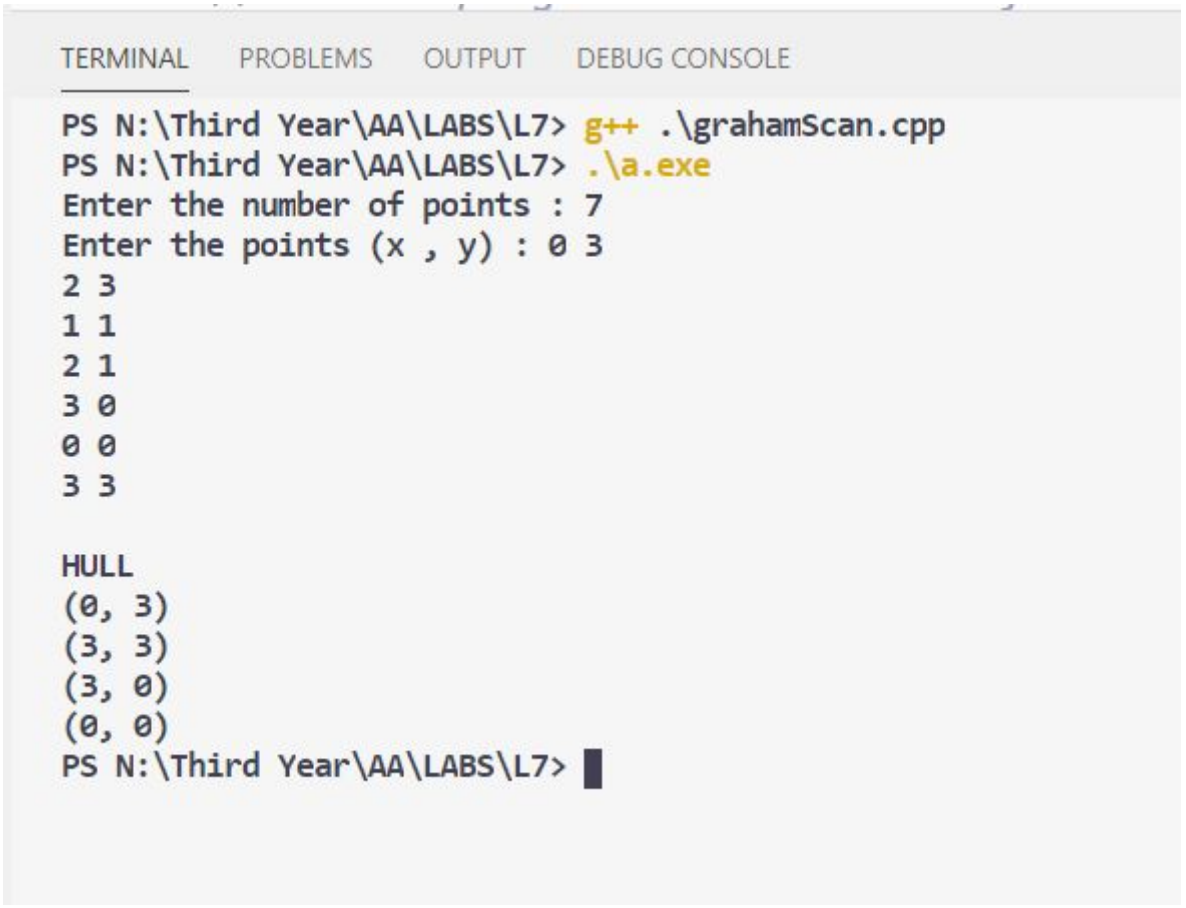
```
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
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; // colinear
    return (val > 0) ? 1 : 2; // clock or counterclock
wise
}
```

```
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; // Initialize size of modified array
```

```
    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++; // Update size of modified array
    }

    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();
    }
}

// Driver program to test above functions
int main()
{
    cout << "Enter the number of points : ";
    int n;
    cin >> n;
    Point points[n];
    cout << "Enter the points (x , y) : " ;
    for(auto i = 0 ; i < n; i++)
        cin >> points[i].x >> points[i].y;

    convexHull(points, n);
    return 0;
}
```

Output:



Complexity :

Let n be the number of input points.
The algorithm takes O(nLogn) time if we use a O(nLogn) sorting algorithm.

The first step (finding the bottom-most point) takes O(n) time. The second step (sorting points) takes O(nLogn) time. The third step takes O(n) time. In the third step, every element is pushed and popped at most one time. So the sixth step to process points one by one takes O(n) time, assuming that the stack operations take O(1) time. Overall complexity is O(n) + O(nLogn) + O(n) + O(n) which is O(nLogn)

Nevil Parmar
CE-092
https://nevilparmar.me