



Ahsania Mission University of Science & Technology

Department of Computer Science and Engineering

1st Batch, 2nd Year 2nd Semester, Spring 2025

Lab Report

Course Code: CSE 2202

Course Title: Computer Algorithms Sessional

AMUST

Experiment No. : 7
Experiment Date : 07-05-2025
Submission Date : 28-05-2025

Submitted To:

Md. Fahim Faisal

Lecturer,

Department of Computer Science and Engineering (CSE)

Faculty of Engineering, Ahsania Mission University of Science & Technology

Submitted By-

Name : -- Md. Aktaruzzaman Aktar --

ID No. : -- 1012320005101015 --

1st Batch, 2nd Year 2nd Semester, Spring 2025

Department of Computer Science and Engineering, AMUST.

Task Number-01: Convex Hull using Brute Force Algorithm

Problem Statement:

Write a program in C++ to implement Convex Hull using the Brute Force method.

Theory:

The convex hull of a set of points is the smallest convex polygon that contains all the points. In the brute force approach, we check every pair of points and determine whether all other points lie on one side of the line formed by the pair. If they do, that edge is part of the convex hull.

Source Code:

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

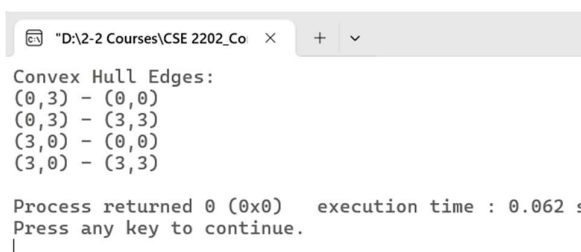
struct Point {
    int x, y;
};

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;
}
```

Output:



```
"D:\2-2 Courses\CSE 2202_Co" x + v
Convex Hull Edges:
(0,3) - (0,0)
(0,3) - (3,3)
(3,0) - (0,0)
(3,0) - (3,3)

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Conclusion:

The program successfully finds all edges of the convex hull by checking every possible pair and validating if all other points lie on one side of the line. This demonstrates the brute force approach.

Task No.-02: Convex Hull using Graham's Scan Algorithm

Problem Statement:

Write a program in C++ to implement Convex Hull using Graham's Scan algorithm.

Theory:

Graham's Scan is an efficient algorithm to compute the convex hull of a set of 2D points. It works in $O(n \log n)$ time and is based on sorting the points by polar angle and then processing them using a stack to ensure convexity.

Source 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]);
}

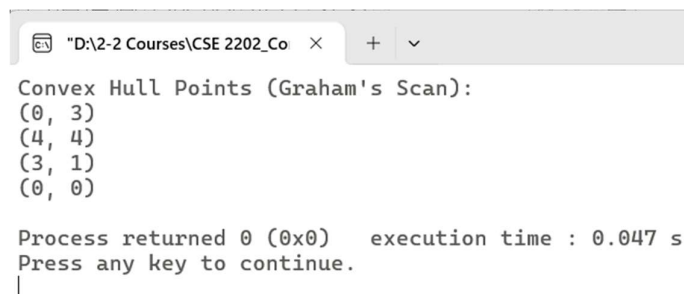
cout << "Convex Hull Points (Graham's Scan):\n";
while (!hull.empty()) {
    Point p = hull.top();
    cout << "(" << p.x << ", " << p.y << ")\n";
    hull.pop();
}

}

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;
}

```

Output:



```

D:\2-2 Courses\CSE 2202_Co  X  +  v
Convex Hull Points (Graham's Scan):
(0, 3)
(4, 4)
(3, 1)
(0, 0)

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
|

```

Conclusion:

This program efficiently constructs the convex hull by first sorting based on polar angles and then processing the sorted list using a stack to maintain convexity.

Task Number-03: Convex Hull using QuickHull Algorithm

Problem Statement:

Write a program in C++ to implement Convex Hull using the QuickHull algorithm.

Theory:

QuickHull is a divide-and-conquer algorithm that finds the convex hull similarly to QuickSort. It recursively identifies the farthest point from a line and partitions the remaining set, efficiently constructing the convex polygon.

Source 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, 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";
}

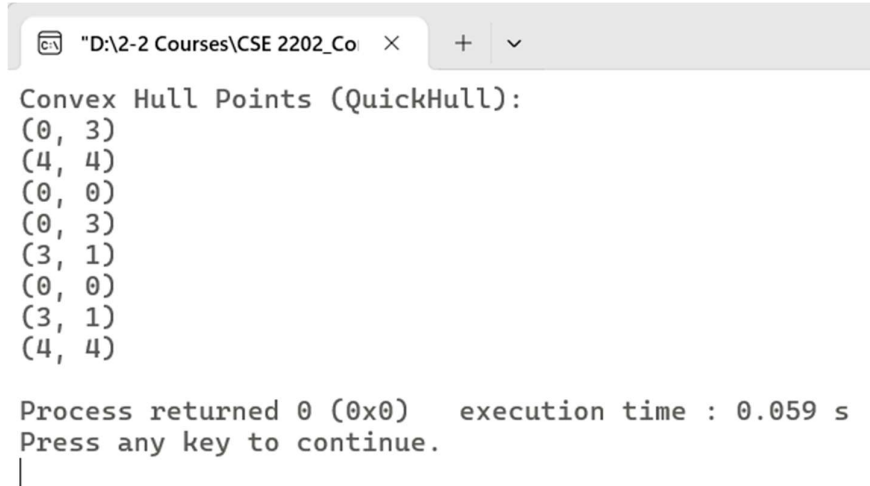
int main() {
```

```

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

```

Output:



```

D:\2-2 Courses\CSE 2202_Co
Convex Hull Points (QuickHull):
(0, 3)
(4, 4)
(0, 0)
(0, 3)
(3, 1)
(0, 0)
(3, 1)
(4, 4)

Process returned 0 (0x0)   execution time : 0.059 s
Press any key to continue.
|

```

Conclusion:

QuickHull efficiently computes the convex hull using a divide-and-conquer approach. It is suitable for most practical use cases and performs well in average cases.