BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	2B

AIM:	Convex Hull Problem - You are given 100 points in the 2D plane to find a convex hull out of 100 2D points.

Program 1

PROBLEM STATEMENT :	In Computational Geometry, the convex hull/convex envelope/convex closure/convex polygon of n points is the smallest polygon which covers all n points. There are many algorithms to find convex hulls e.g. Brute Force, Graham's Scan and Divide Conquer Convex based solution. Each student has to generate a set of 100 2D points using the rand() function and use this input to three algorithms namely Brute-force, Graham's Scan and Divide and Conquer. The x and y values of these 2D points can have integer values in the range of 1-100.

PROGRAM (hull.cpp):

```
#include <bits/stdc++.h>

using namespace std;
using namespace chrono;

struct pt {
        int x, y;

        pt(int x = 0, int y = 0) : x(x), y(y) {}

        bool operator<(const pt& p) const {
        return x < p.x || (x == p.x && y < p.y);
        }

        bool operator==(const pt& p) const {
        return x == p.x && y == p.y;
        }
};
```

```
// Orientation: 0 (COLL), 1 (CW), -1 (CCW)
int orientation(pt a, pt b, pt c) {
    int val = (b.y - a.y) * (c.x - b.x) - (b.x - a.x) * (c.y - b.y);
    if (val == 0) return 0;
    return (val > 0) ? 1 : -1;
}

int distance(pt a, pt b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}

pt anchor;
bool polarOrderCompare(pt a, pt b) {
    int o = orientation(anchor, a, b);
    if (o == 0) return (distance(anchor, a) < distance(anchor, b));
    return o == -1;
}

// Brute Force Algorithm
vector<pt> bruteForce(vector<pt>& pts) {
    set<pt> hull;
    int n = pts.size();

    for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
    bool valid = true;
    int side = 0;
    for (int k = 0; k < n; k++) {
        if (k == i || k == j) continue;
        int o = orientation(pts[i], pts[j], pts[k]);
        if (o == 0) continue;
        if (side == 0) side = o;
        else if (side != o) {
        valid = false;
        break;
        }
    }
```

```
        if (valid) {
                hull.insert(pts[i]);
                hull.insert(pts[j]);
        }
        }
        }
        return vector<pt>(hull.begin(), hull.end());
}

// Graham's Scan Algorithm
vector<pt> grahamScanConvexHull(vector<pt>& pts) {
        int n = pts.size();
        if (n < 3) return {};

        // Find the anchor point (lowest y-coordinate)
        int minIdx = 0;
        for (int i = 1; i < n; i++) {
        if (pts[i].y < pts[minIdx].y || (pts[i].y == pts[minIdx].y && pts[i].x <
pts[minIdx].x)) {
        minIdx = i;
        }
        }
        swap(pts[0], pts[minIdx]);
        anchor = pts[0];

        // Sort points by polar angle
        sort(pts.begin() + 1, pts.end(), polarOrderCompare);

        // Build the convex hull
        vector<pt> hull;
        for (auto p : pts) {
        while (hull.size() > 1 && orientation(hull[hull.size() - 2],
hull.back(), p) != -1) {
        hull.pop_back();
        }
        hull.push_back(p);
        }
        return hull;
```

```
}

// Divide and Conquer Algorithm
void mergeHulls(vector<pt>& leftHull, vector<pt>& rightHull,
vector<pt>& mergedHull) {
        int n1 = leftHull.size(), n2 = rightHull.size();

        int rightmost_leftHull = 0;
        for (int i = 1; i < n1; i++) {
        if (leftHull[i].x > leftHull[rightmost_leftHull].x)
        rightmost_leftHull = i;
        }

        int leftmost_rightHull = 0;
        for (int i = 1; i < n2; i++) {
        if (rightHull[i].x < rightHull[leftmost_rightHull].x)
        leftmost_rightHull = i;
        }

        // Find the upper tangent
        int upperLeft = rightmost_leftHull, upperRight = leftmost_rightHull;
        bool done = false;
        while (!done) {
        done = true;
        while (orientation(rightHull[upperRight], leftHull[upperLeft],
leftHull[(upperLeft + 1) % n1]) == -1) {
        upperLeft = (upperLeft + 1) % n1;
        }
        while (orientation(leftHull[upperLeft], rightHull[upperRight],
rightHull[(n2 + upperRight - 1) % n2]) == 1) {
        upperRight = (n2 + upperRight - 1) % n2;
        done = false;
        }
        }

        // Find the lower tangent
        int lowerLeft = rightmost_leftHull, lowerRight = leftmost_rightHull;
        done = false;
        while (!done) {
        done = true;
        while (orientation(leftHull[lowerLeft], rightHull[lowerRight],
rightHull[(lowerRight + 1) % n2]) == -1) {
        lowerRight = (lowerRight + 1) % n2;
        }
        while (orientation(rightHull[lowerRight], leftHull[lowerLeft],
leftHull[(n1 + lowerLeft - 1) % n1]) == 1) {
        lowerLeft = (n1 + lowerLeft - 1) % n1;
        done = false;
        }
        }

        // Merge the hulls
        mergedHull.clear();
        for (int i = 0; i <= upperLeft; i++) {
        mergedHull.push_back(leftHull[i]);
        }
        for (int i = upperRight; i != (lowerRight + 1) % n2; i = (i + 1) % n2)
{
        mergedHull.push_back(rightHull[i]);
        }
        for (int i = (lowerLeft + 1) % n1; i != 0; i = (i + 1) % n1) {
        mergedHull.push_back(leftHull[i]);
        }
}

vector<pt> divideAndConquer(vector<pt> pts) {
        int n = pts.size();
        if (n <= 3) return bruteForce(pts);

        // Sort points by x-coordinate
        sort(pts.begin(), pts.end());

        // Divide into two halves
        vector<pt> leftHalf(pts.begin(), pts.begin() + n / 2);
        vector<pt> rightHalf(pts.begin() + n / 2, pts.end());
```

```
    // Recursively find convex hulls
    vector<pt> leftHull = divideAndConquer(leftHalf);
    vector<pt> rightHull = divideAndConquer(rightHalf);

    // Merge the two convex hulls
    vector<pt> mergedHull;
    mergeHulls(leftHull, rightHull, mergedHull);
    return mergedHull;
}

vector<pt> generateRandomPoints(int n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, 100);

    vector<pt> points;
    points.reserve(n);
    for (int i = 0; i < n; ++i) {
    points.emplace_back(dis(gen), dis(gen));
    }
    return points;
}

void writePointsToFile(const string& filename, const vector<pt>& original,
const vector<pt>& hull) {
    ofstream file(filename);
    if (!file) {
    cerr << "Error opening file: " << filename << endl;
    return;
    }

    file << "Original Points:\n";
    for (const auto& p : original) {
    file << p.x << " " << p.y << "\n";
    }

    file << "\nConvex Hull:\n";
    for (const auto& p : hull) {
```

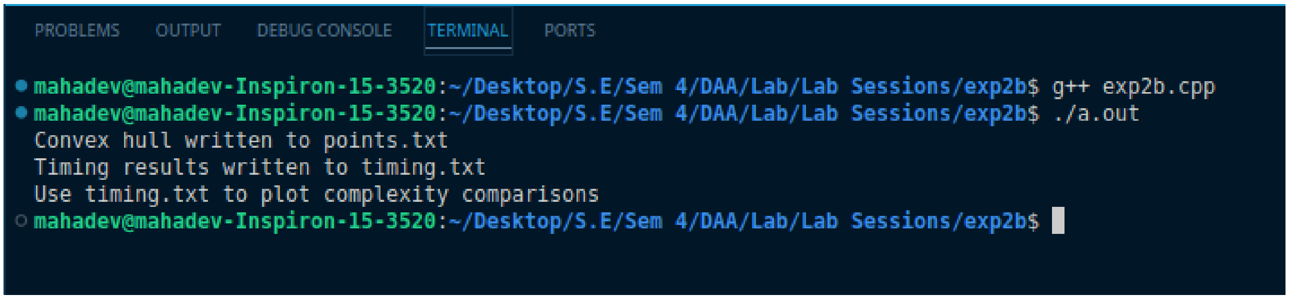
```
	file << p.x << " " << p.y << "\n";
	}
}

template<typename Func>
double measureTime(Func f, vector<pt>& points, int iterations = 100) {
	double totalTime = 0;
	for (int i = 0; i < iterations; i++) {
	vector<pt> points_copy = points;
	auto start = high_resolution_clock::now();
	f(points_copy);
	auto end = high_resolution_clock::now();
	totalTime += duration_cast<microseconds>(end - start).count();
	}
	return totalTime / iterations;
}

void performTimingAnalysis(const string& filename, const vector<pt>&
points) {
	ofstream file(filename);
	if (!file) {
	cerr << "Error opening file: " << filename << endl;
	return;
	}

	file << fixed << setprecision(2);
	file << "Points BruteForce(us) DivideConquer(us)
GrahamScan(us)\n";

	for (int n = 4; n <= 100; n++) {
	vector<pt> samplePoints(points.begin(), points.begin() + n);

	double timeBF = measureTime(bruteForce, samplePoints);
	double timeDC = measureTime(divideAndConquer, samplePoints);
	double timeGS = measureTime(grahamScanConvexHull,
samplePoints);

	file << n << " " << timeBF << " " << timeDC << " " << timeGS <<
```

```
"\n";
        }
}

int main() {
        int NUM_POINTS = 100;

        vector<pt> points = generateRandomPoints(NUM_POINTS);
        vector<pt> hull = grahamScanConvexHull(points);

        writePointsToFile("points.txt", points, hull);
        performTimingAnalysis("timing.txt", points);

        cout << "Convex hull written to points.txt\n"
        << "Timing results written to timing.txt\n"
        << "Use timing.txt to plot complexity comparisons\n";


        return 0;
}
```

plot.ipynb:

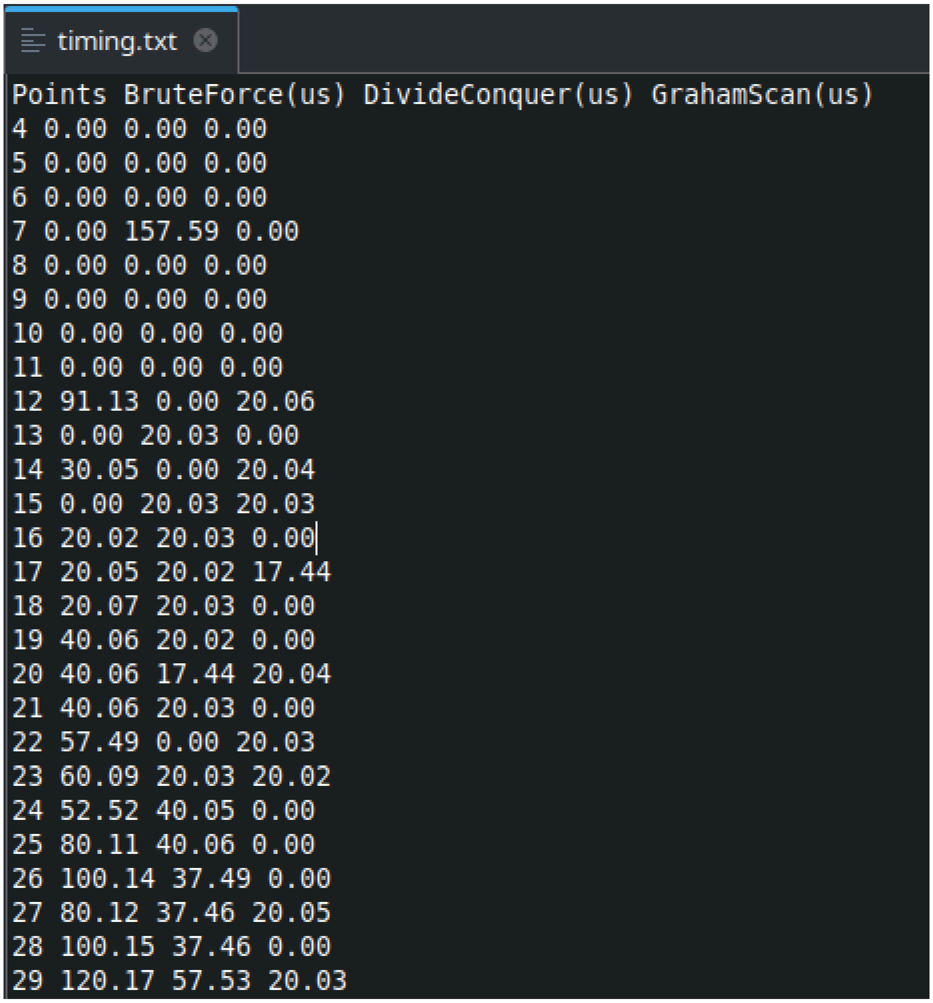
```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('timing.txt', delim_whitespace=True, header=0)

n_points = data['Points']
bf_time = data['BruteForce(us)']
dc_time = data['DivideConquer(us)']
gs_time = data['GrahamScan(us)']

plt.plot(n_points, bf_time, label='Brute Force', color='blue')
plt.title('Brute Force Runtime')
plt.xlabel('Number of Points')
plt.ylabel('Time (µs)')
plt.grid(True)
plt.legend()
plt.plot(n_points, dc_time, label='Divide and Conquer', color='green')
plt.title('Divide and Conquer Runtime')
plt.xlabel('Number of Points')
plt.ylabel('Time (µs)')
plt.grid(True)
plt.legend()


plt.plot(n_points, gs_time, label='Graham Scan', color='red')
plt.title('Graham Scan Runtime')
plt.xlabel('Number of Points')
plt.ylabel('Time (µs)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 6))
plt.plot(n_points, bf_time, label='Brute Force', color='blue')
plt.plot(n_points, dc_time, label='Divide and Conquer', color='green')
plt.plot(n_points, gs_time, label='Graham Scan', color='red')
plt.title('Comparison of Convex Hull Algorithms')
plt.xlabel('Number of Points')
plt.ylabel('Time (µs)')
plt.grid(True)
plt.legend()
plt.show()
```

RESULT:

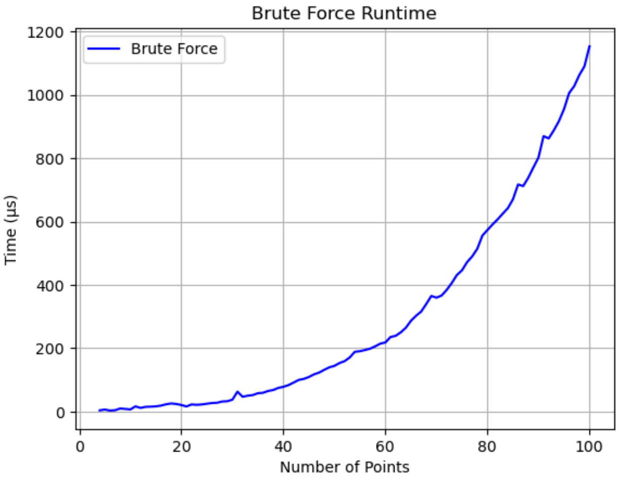


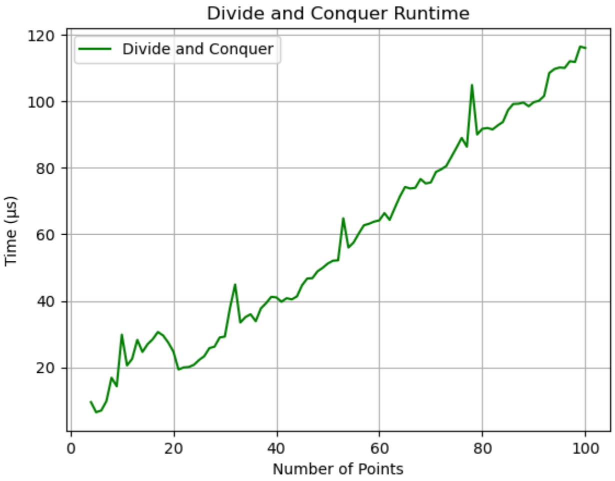
```
● mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2b$ g++ exp2b.cpp
● mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2b$ ./a.out
  Convex hull written to points.txt
  Timing results written to timing.txt
  Use timing.txt to plot complexity comparisons
○ mahadev@mahadev-Inspiron-15-3520:~/Desktop/S.E/Sem 4/DAA/Lab/Lab Sessions/exp2b$ 
```

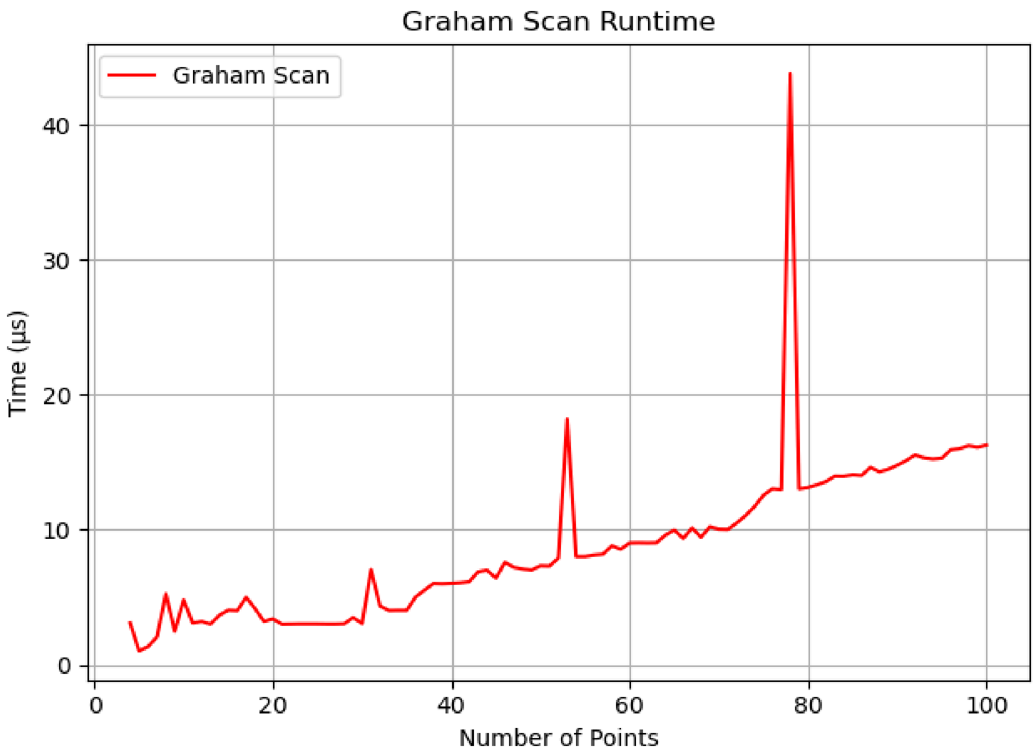
OUTPUT:

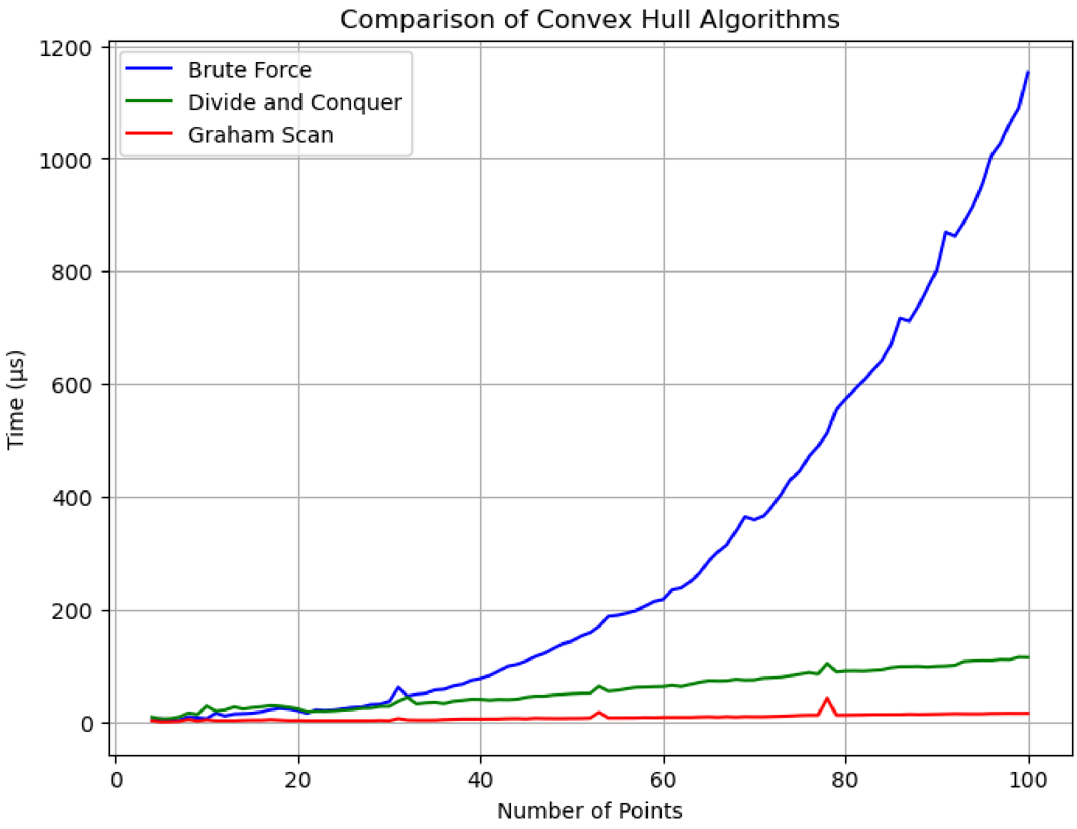


```
Points BruteForce(us) DivideConquer(us) GrahamScan(us)
4 0.00 0.00 0.00
5 0.00 0.00 0.00
6 0.00 0.00 0.00
7 0.00 157.59 0.00
8 0.00 0.00 0.00
9 0.00 0.00 0.00
10 0.00 0.00 0.00
11 0.00 0.00 0.00
12 91.13 0.00 20.06
13 0.00 20.03 0.00
14 30.05 0.00 20.04
15 0.00 20.03 20.03
16 20.02 20.03 0.00
17 20.05 20.02 17.44
18 20.07 20.03 0.00
19 40.06 20.02 0.00
20 40.06 17.44 20.04
21 40.06 20.03 0.00
22 57.49 0.00 20.03
23 60.09 20.03 20.02
24 52.52 40.05 0.00
25 80.11 40.06 0.00
26 100.14 37.49 0.00
27 80.12 37.46 20.05
28 100.15 37.46 0.00
29 120.17 57.53 20.03
```

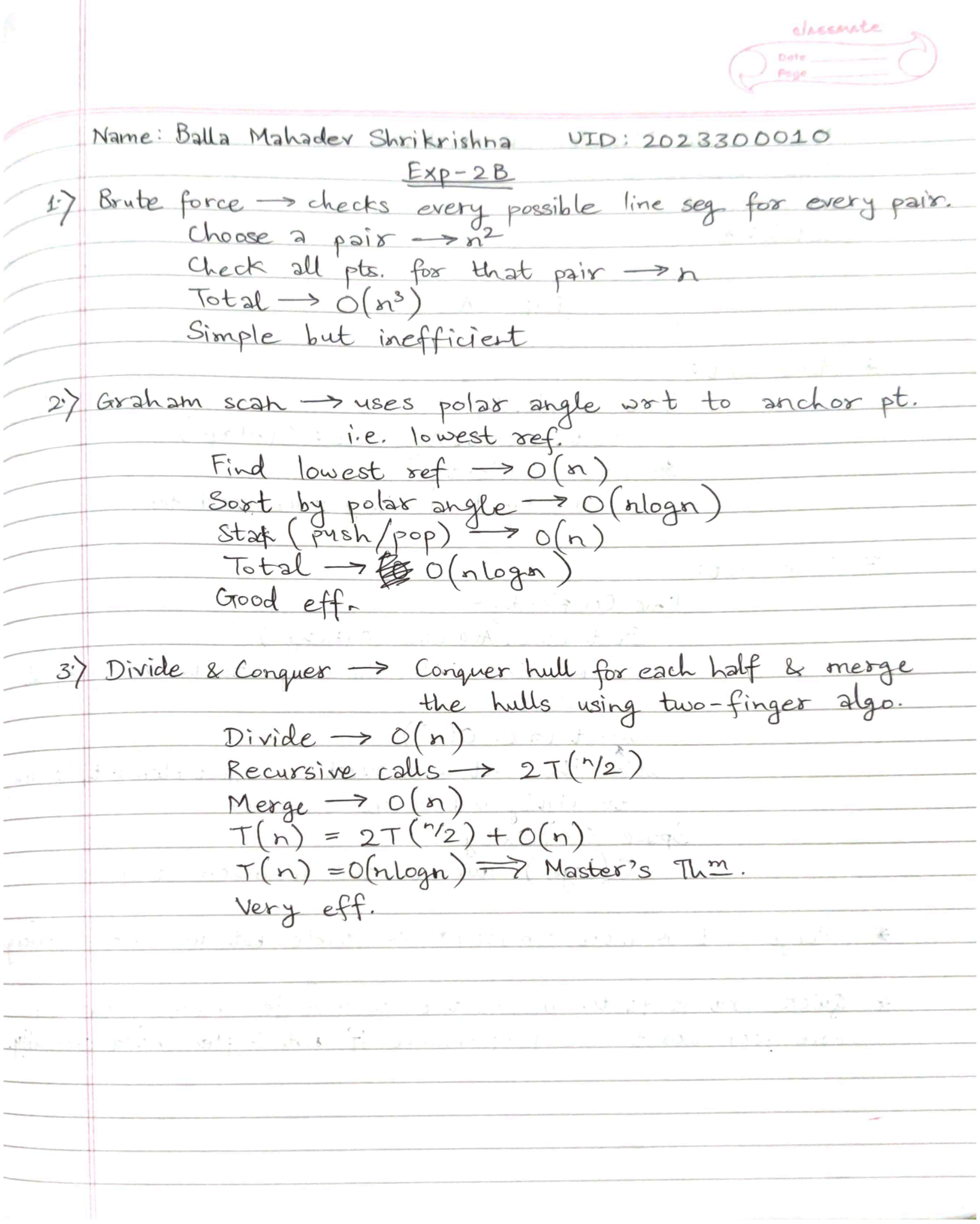





Graham Scan Runtime
Graham Scan
Time (µs)
0
10
20
30
40
0
20
40
60
80
100
Number of Points


Comparison of Convex Hull Algorithms
Brute Force
Divide and Conquer
Graham Scan
Time (µs)
0
200
400
600
800
1000
1200
0
20
40
60
80
100
Number of Points

CONCLUSION:



Name: Balla Mahadev Shrikrishna   UID: 2023300010

Exp-2B

1.) Brute force → checks every possible line seg. for every pair.
Choose a pair $\rightarrow n^2$
Check all pts. for that pair $\rightarrow n$
Total $\rightarrow O(n^3)$
Simple but inefficient

2.) Graham scan → uses polar angle wrt to anchor pt.
i.e. lowest ref.
Find lowest ref $\rightarrow O(n)$
Sort by polar angle $\rightarrow O(n\log n)$
Stack (push/pop) $\rightarrow O(n)$
Total $\rightarrow O(n\log n)$
Good eff.

3.) Divide & Conquer → Conquer hull for each half & merge the hulls using two-finger algo.
Divide $\rightarrow O(n)$
Recursive calls $\rightarrow 2T(n/2)$
Merge $\rightarrow O(n)$
$T(n) = 2T(n/2) + O(n)$
$T(n) = O(n\log n) \Longrightarrow$ Master's Th$^{\underline{m}}$.
Very eff.