Heavens' Light is Our Guide


# Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering


Algorithms Analysis & Design Sessional

# Convex Hull


Submitted by

Kaif Ahmed Khan

Roll: 2103163

Submitted to

A. F. M. Minhazur Rahman
Assistant Professor


November 22, 2024

# Contents

# List of Listings



# List of Figures

# 1 Convex Hull

## 1.1 Problem Statement

- Implement Quickhull and Graham Scan algorithm to find the convex hull of 2D points.
- Visualize the input 2D points and the convex hull boundary.
- Compare Quickhull and Graham Scan based on various input size on randomly generated points. The comparison metric should be the execution time of each sorting algorithm.

## 1.2 Code

Listing 1: *generate_points.cpp*

```cpp
#include <bits/stdc++.h>
#include <cstdlib>
#include <ctime>
#include <fstream>
using namespace std;

int main() {
  ofstream myfile;
  long long n;
  cout << "How many points? ";
  cin >> n;
  myfile.open("./points.txt");
  if (myfile.is_open()) {
    srand(time(0));
    for (int k = 0; k < n; k++) {
      long long x = rand() % n + 1;
      long long y = rand() % n + 1;
      myfile << x << " " << y << "\n";
    }
    myfile.close();
  } else {
    cout << "Error opening file." << endl;
  }
  return 0;
}
```

Listing 2: *quick_hull.cpp*

```cpp
#include <algorithm>
#include <bits/stdc++.h>
#include <chrono>
#include <utility>
#include <vector>
using namespace std;
using namespace std::chrono;

typedef long long ll;
```

```cpp
typedef pair<ll, ll> Point;
typedef vector<Point> vpii;

vpii hull_points;

bool order(Point p1, Point p2) {
  Point ref = hull_points[0];
  ll dx1 = p1.first - ref.first, dy1 = p1.second - ref.second;
  ll dx2 = p2.first - ref.first, dy2 = p2.second - ref.second;

  ll cross = dx1 * dy2 - dy1 * dx2;

  if (cross == 0)
    return (dx1 * dx1 + dy1 * dy1) < (dx2 * dx2 + dy2 * dy2);

  return cross > 0;
}

ll get_side(Point p1, Point p2, Point p) {
  ll cross_prod = (p2.first - p1.first) * (p.second - p1.second) -
                  (p2.second - p1.second) * (p.first - p1.first);
  if (cross_prod > 0)
    return 1;
  else if (cross_prod < 0)
    return -1;
  return 0;
}

Point get_min(const vpii pts) { return *min_element(pts.begin(), pts.end()); }
Point get_max(const vpii pts) { return *max_element(pts.begin(), pts.end()); }

ll dist(Point p1, Point p2, Point p) {
  return abs(((p2.first - p1.first) * (p1.second - p.second)) -
             ((p1.first - p.first) * (p2.second - p1.second)));
}

void quick_hull(const vpii pts, ll n, Point p1, Point p2, ll side) {
  // find point with max dist
  ll max_dist = 0;
  ll max_index = -1;
  for (ll i = 0; i < n; i++) {
    ll temp_dist = dist(p1, p2, pts[i]);
    if (get_side(p1, p2, pts[i]) == side && temp_dist > max_dist) {
      max_index = i;
      max_dist = temp_dist;
    }
  }
  // push 2 extreme points
  if (max_index == -1) {
```

```
    hull_points.push_back(p1);
    hull_points.push_back(p2);
    return;
  }

  quick_hull(pts, n, pts[max_index], p1, -get_side(pts[max_index], p1, p2));
  quick_hull(pts, n, pts[max_index], p2, -get_side(pts[max_index], p2, p1));
}

void read_points_from_file(const string file_name, vpii &points);
void print_points(vpii points, ll n);

int main() {
  vpii P;
  read_points_from_file("points.txt", P);
  ll n = P.size();
  // print_points(P, n);

  auto st = high_resolution_clock::now(); // clock start

  Point min = get_min(P);
  Point max = get_max(P);

  quick_hull(P, n, min, max, 1);
  quick_hull(P, n, min, max, -1);

  auto et = high_resolution_clock::now(); // clock end
  double time_taken =
      chrono::duration_cast<chrono::nanoseconds>(et - st).count();
  time_taken *= 1e-6;

  sort(hull_points.begin(), hull_points.end(), order);
  hull_points.erase(unique(hull_points.begin(), hull_points.end()),
                    hull_points.end());

  // cout << "Convex hull points using Quick Hull:" << endl;
  // print_points(hull_points, hull_points.size());
  cout << "Time taken for Quick hull: " << time_taken << " ms" << endl;
  cout << "No. of Points: " << n << endl;
}

void read_points_from_file(const string file_name, vpii &points) {
  ifstream inputFile(file_name);
  if (!inputFile.is_open()) {
    cerr << "Error opening the file!" << endl;
    exit(1);
  }
  string line;
  while (getline(inputFile, line)) {
    stringstream ss(line);
    ll x, y;
```

```cpp
    ss >> x >> y;
    points.push_back(make_pair(x, y));
  }
  inputFile.close();
}

void print_points(const vpii points, ll n) {
  cout << "X = [ ";
  for (ll i = 0; i < n; i++) {
    cout << points[i].first << ", ";
  }
  cout << points[0].first << " ]" << endl;
  cout << "Y = [ ";
  for (ll i = 0; i < n; i++) {
    cout << points[i].second << ", ";
  }
  cout << points[0].second << " ]" << endl;
}
```

Listing 3: *graham_scan.cpp*

```cpp
#include <algorithm>
#include <bits/stdc++.h>
#include <chrono>
#include <utility>
#include <vector>
using namespace std;
using namespace std::chrono;

typedef long long ll;
typedef pair<ll, ll> Point;
typedef vector<Point> vpii;

vpii points;  // Points are stored as (y, x)
vpii hull_pts;
Point p0;     // Reference point

ll get_y(const Point &p) { return p.first; }
ll get_x(const Point &p) { return p.second; }

ll cross_product(const Point &p1, const Point &p2, const Point &p3) {
  return (get_x(p2) - get_x(p1)) * (get_y(p3) - get_y(p1)) -
         (get_y(p2) - get_y(p1)) * (get_x(p3) - get_x(p1));
}

bool polar_order(const Point &p1, const Point &p2) {
  ll cross = cross_product(p0, p1, p2);
  if (cross == 0) {
    return (get_x(p1) - get_x(p0)) * (get_x(p1) - get_x(p0)) +
           (get_y(p1) - get_y(p0)) * (get_y(p1) - get_y(p0)) <
           (get_x(p2) - get_x(p0)) * (get_x(p2) - get_x(p0)) +
           (get_y(p2) - get_y(p0)) * (get_y(p2) - get_y(p0));
  }
```

```
  return cross > 0;
}

void read_points_from_file(const string &file_name, vpii &points);
void print_points(const vpii &points);

int main() {
  read_points_from_file("points.txt", points);

  p0 = *min_element(points.begin(), points.end(),
                     [](const Point &p1, const Point &p2) {
                       return p1 < p2;
                     });

  auto start_time = high_resolution_clock::now(); // clock start
  sort(points.begin(), points.end(), polar_order);

  vpii filtered_points = {p0};
  for (size_t i = 1; i < points.size(); i++) {
    while (i < points.size() - 1 &&
           cross_product(p0, points[i], points[i + 1]) == 0) {
      i++;
    }
    filtered_points.push_back(points[i]);
  }

  vector<Point> hull;
  hull.push_back(filtered_points[0]);
  hull.push_back(filtered_points[1]);
  hull.push_back(filtered_points[2]);

  for (ll i = 3; i < filtered_points.size(); i++) {
    while (hull.size() > 1 &&
           cross_product(hull[hull.size() - 2], hull.back(),
           ↪  filtered_points[i]) <= 0) {
      hull.pop_back();
    }
    hull.push_back(filtered_points[i]);
  }

  auto end_time = high_resolution_clock::now();

  hull_pts = hull;

  double time_taken = duration_cast<nanoseconds>(end_time -
  ↪  start_time).count() * 1e-6;
  cout << "Time taken for Graham Scan: " << time_taken << " ms" << endl;

  // cout << "Convex hull points:" << endl;
  // print_points(hull_pts);
  return 0;
```

```
}

void read_points_from_file(const string &file_name, vpii &points) {
  ifstream inputFile(file_name);
  if (!inputFile.is_open()) {
    cerr << "Error opening the file!" << endl;
    exit(1);
  }
  ll x, y;
  while (inputFile >> x >> y) {
    points.emplace_back(y, x);
  }
  inputFile.close();
}

void print_points(const vpii &points, ll n) {
  cout << "X = [ ";
  for (ll i = 0; i < n; i++) {
    cout << get_x(points[i]) << ", ";
  }
    cout << get_x(points[0]) << ", ";
  cout << "Y = [ ";
  for (ll i = 0; i < n; i++) {
    cout << get_y(points[i]) << ", ";
  }
    cout << get_y(points[0]) << ", ";
}
```

Listing 4: *Makefile*

```
CC=g++

convex: points graham quick

points: generate_points.cpp
    $(CC) $^ -o points.out
    ./points.out
graham:    graham_scan.cpp
    $(CC) $^ -o graham_scan.out
    ./graham_scan.out

quick: quickhull.cpp
    $(CC) $^ -o quickhull.out
    ./quickhull.out

clean:
    rm *.out
```

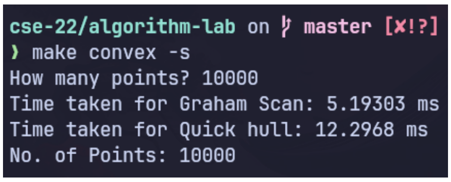
Listing 5: *visualization.py*

```
import matplotlib.pyplot as plt
%matplotlib inline
```

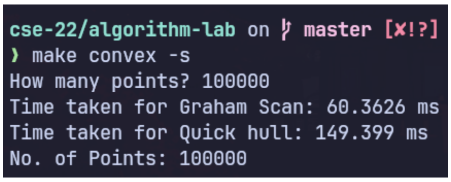
```python
X = [ 44, 63, 90, 43, 54, 26, 42, 47, 57, 2, 61, 72, 24, 88, 82, 78, 33, 74, 55, 19, 99, 24, 42, 73, 18, 32, 41, 43, 64, 49, 8, 73, 66, 13, 66, 32, 27, 8, 82, 69, 5, 80, 59, 12, 56, 70, 86, 7, 40, 74, 54, 20, 65, 51, 59, 96, 76, 60, 100, 60, 83, 75, 23, 22, 4, 18, 57, 89, 16, 18, 11, 90, 43, 71, 24, 1, 11, 78, 60, 46, 51, 72, 51, 79, 100, 93, 12, 99, 82, 47, 51, 64, 26, 97, 92, 100, 56, 100, 31, 24 ]
Y = [ 55, 98, 64, 46, 32, 64, 98, 29, 44, 83, 16, 14, 57, 82, 26, 77, 40, 22, 68, 61, 44, 93, 23, 50, 74, 30, 55, 16, 83, 97, 26, 92, 46, 72, 31, 64, 8, 20, 80, 99, 53, 97, 74, 74, 60, 16, 42, 3, 72, 5, 58, 80, 28, 46, 72, 64, 27, 34, 24, 8, 29, 20, 33, 62, 48, 58, 37, 21, 40, 75, 65, 86, 49, 94, 30, 7, 27, 33, 52, 63, 63, 7, 61, 67, 96, 62, 82, 54, 69, 6, 100, 13, 41, 85, 42, 42, 71, 6, 78, 82 ]
x = [ 100, 69, 51, 42, 24, 2, 1, 7, 74, 100, 100 ]
y = [ 96, 99, 100, 98, 93, 83, 7, 3, 5, 6, 96]

plt.scatter(X,Y, label="n=100")
plt.plot(x,y,color="red")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.savefig("convex_100pt.png", dpi=300, bbox_inches="tight")
```

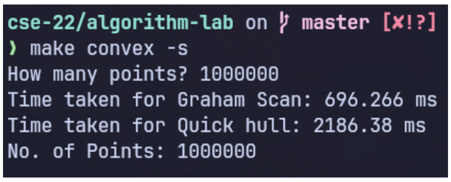
## 1.3 Output



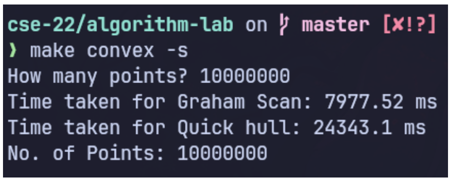

(a) *Execution time for* $n$=$10^4$




(b) *Execution time for* $n$=$10^5$




(c) *Execution time for* $n$=$10^6$




(d) *Execution time for* $n$=$10^7$

Figure 1: *Convex hull algorithm execution time*

## Summary of Execution time

Table 1: *Comparison table for Graham Scan & Quick Hull algorithm execution time*

| Input Size | Graham Scan (ms) | Quick Hull (ms) |
|---|---|---|
| $10^3$ | 0.440 | 1.000 |
| $5 \times 10^3$ | 2.385 | 5.553 |
| $10^4$ | 5.193 | 12.297 |
| $5 \times 10^4$ | 30.383 | 81.619 |
| $10^5$ | 60.363 | 149.399 |
| $10^6$ | 696.266 | 2186.380 |
| $10^7$ | 7977.520 | 24 343.100 |

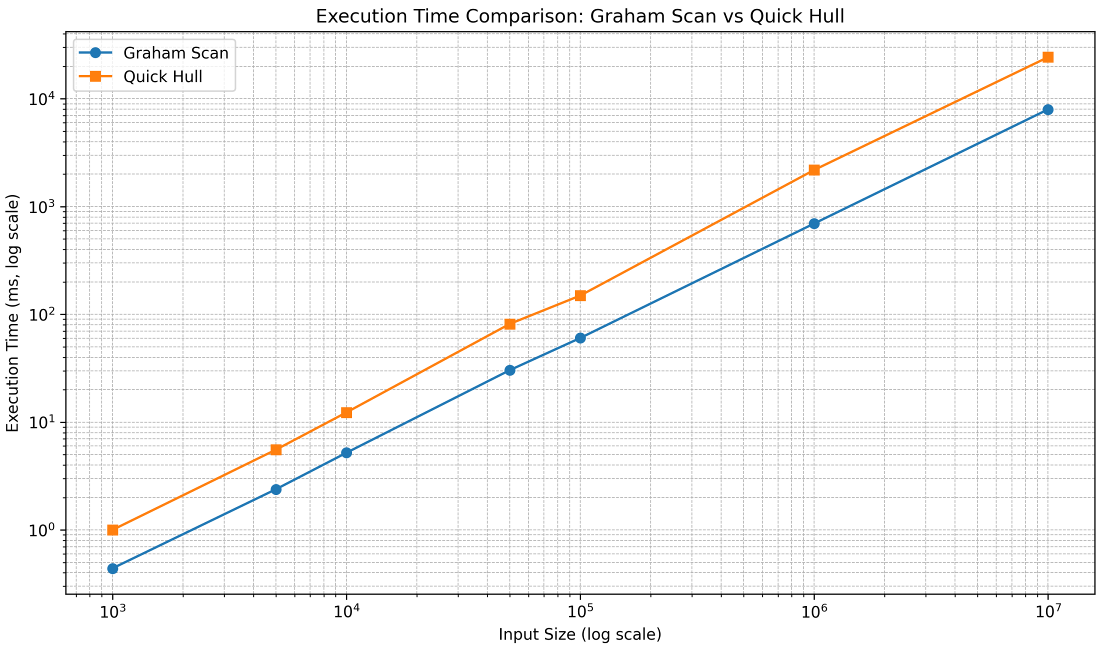
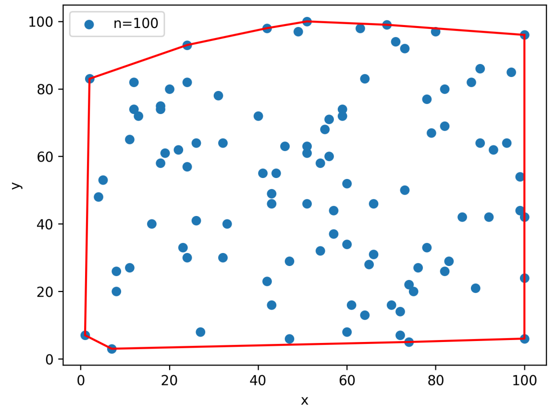
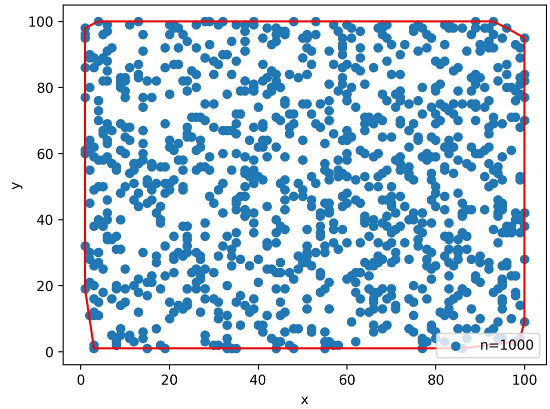



Figure 2: *Execution time comparison*

### Visualization




(a) *Convex hull visualization using graham scan for 100 points*




(b) *Convex hull visualization using quick hull for 1000 points*

Figure 3: *Convex hull Visualization*

## 1.4 Analysis & Discussion

The implemented code sucessfully finds the convex hull points using Graham scan and Quick hull algorithms (see fig. 3). In case of execution time, from the comparison table 1 and fig. 2 it is clear that Graham scan outperforms quick hull for any input size. With increasing number of input sizes Graham scan is almost 3 times faster than quick hull.

This is because Graham scan has $O(n \log n)$ complexity for any input size. Whereas Quick Hull algorithm uses a divide and conquer approach having the the complexity of $O(n \log n)$ for avarage case and $O(n^2)$ in worst case.

Graham scan always sorts the points in polar angle order. This step takes the largest cost. More efficient algorithm for sorting can also imporve this step's complexity.

In conclusion, Graham scan is more efficent in comparison to quick hull. Also the simplicity in implementation for Graham scan is much more preferred over quick hull algorithm.