"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOG

Lab Report

Submitted By:
Name: Khandoker Sefayet Alam
Roll:2003121
Department: Computer Science & Engineering
Section-C

Course code: CSE 2202
Course name: Algorithm Analysis and Design Sessional

Submitted To:

A. F. M. Minhazur Rahman
Lecturer, Department of CSE,RUET

Problem statement: *Given, a set of points, find the convex hull.*

Algorithm-1*: Quick Hull.*

Algorithm description:

The convex hull of a set of points in a 2-dimensional space can be efficiently computed using the Quickhull algorithm, a computational geometry technique. The smallest convex form that encompasses all of the specified points is represented by the convex hull. The Quickhull algorithm was created as a divide-and-conquer strategy. It recursively divides the point set into subgroups and then connects the extreme points of each subset to create the convex hull. Because of its divide-and-conquer approach, Quickhull is particularly effective at managing enormous datasets since it reduces time complexity.

In order to create an initial convex hull, the procedure starts by locating the points along each axis that have the lowest and maximum coordinates. Then, on both sides of the line that joins these extreme points, the point set is divided into two subsets. After that, until the convex hull is completely generated, the algorithm is applied iteratively to every subset.

Code:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define pb push_back
#define ll long long


#define nn '\n'



#define all(x) x.begin(), x.end()

typedef unsigned long long ull;
typedef pair<ll, ll> pll;
typedef vector<ll> vl;
typedef vector<pll> vpll;
typedef vector<vl> vvl;
```

These are some header files, macros and definitions to make my code shorter and faster.

```
void setIO()
{
#ifndef ONLINE_JUDGE
    freopen("input_05.txt", "r", stdin);

    freopen("output_05.txt", "w", stdout);
#endif // ONLINE_JUDGE
}
```

This function takes input from an input file and writes in the output file.

```
vector<pll> points;
set<pll> convex_hull;

ll n;
pll lowp;
map<pll, double> mpp;
```

These are some data structures to store out points and convex hull point and reference point.

```
bool cmp(pll a, pll b)
{
    return a.second < b.second;
}
ll getDist(pll a, pll b, pll p)
{
    return abs((p.second - a.second) * (b.first - a.first) - (b.second -
a.second) * (p.first - a.first));
}
ll getDist_2(pll a,pll b){
    return (a.first - b.first)*(a.first - b.first) +
          (a.second - b.second)*(a.second - b.second);
}
bool cmp2(pll a, pll b)
{
    if((a.second - lowp.second) * (b.first - lowp.first) == (b.second -
lowp.second) * (a.first - lowp.first)) return
getDist_2(a,lowp)<getDist_2(b,lowp);
    return (a.second - lowp.second) * (b.first - lowp.first) < (b.second -
lowp.second) * (a.first - lowp.first);
}
```

These are custom comparatos that returns Boolean values to sort a set of points. The getDist() and getDist_2 functions return absolute and non-absolute distance between two points accordingly.

```
ll getside(pll a, pll b, pll p)
{
    ll val = (p.second - a.second) * (b.first - a.first) - (b.second - a.second)
* (p.first - a.first);

    if (val > 0)
        return 1;
    if (val < 0)
        return -1;
    return 0;
}
```

This function returns if point p is on the upper side of the line ab or lower or is on the line.

```
void QuickHull(pll a, pll b, ll side)
{
    ll pos = -1;
    ll max_dist = 0;

    for (int i = 0; i < n; i++)
    {
        ll temp = getDist(a, b, points[i]);
        if (getside(a, b, points[i]) == side && temp > max_dist)
        {
            pos = i;
            max_dist = temp;
        }
    }

    if (pos == -1)
    {
        convex_hull.insert(a);
        convex_hull.insert(b);
        return;
    }
    QuickHull(a, points[pos], side);
    QuickHull(points[pos], b, side);
}
```

This is the recursive quickhull function. Each time, the algorithm chooses two points and finds the point on the respected side with maximum distance from the connecting line of the two points.

The recursive call ends when there are no point between the selected two points in the respected side.

```
void calculate_convex_hull()
{

    cin >> n;
    if (n < 3)
    {
        cout << "No convex hull found" << nn;
        return;
    }
    ll x, y;
    map<pll,ll>vis;
    for (ll i = 0; i < n; i++)
    {
        cin >> x >> y;
        if(!vis[{x,y}]) points.push_back({x, y});
        vis[{x,y}]=1;
    }
    n=points.size();

    sort(all(points));
    QuickHull(points[0], points[n - 1], 1);
    QuickHull(points[0], points[n - 1], -1);

    // printing all points
    vector<ll> xs, ys;

    for (auto it : points)
    {
        xs.push_back(it.first);
        ys.push_back(it.second);
    }
    cout << "x=[";

    for (ll i = 0; i < xs.size(); i++)
    {
        cout << xs[i];
```

```
        if(i!=xs.size()-1) cout<< ",";
    }
    cout <<  "]" << nn;

    cout<<nn<<nn<<nn;
    cout << "y=[";

    for (ll i = 0; i < xs.size(); i++)
    {
        cout << ys[i];
        if(i!=ys.size()-1) cout << ",";
    }
    cout << "]" << nn;
    cout<<nn<<nn<<nn;

    cout<<" plt.scatter(x,y,c='r') "<<nn;


    xs.clear();
    ys.clear();

    // convex hull sorting and line printing
    vector<pll> ans;
    for (auto it : convex_hull)
        ans.push_back(it);

    sort(all(ans), cmp);
    lowp = ans[0];
    sort(all(ans), cmp2);

    for (auto it : ans)
    {
        // cout<<it<<nn;
        xs.push_back(it.first);
        ys.push_back(it.second);
        // vis[{it.first,it.second}]=1;
    }
    cout << "x=[";

    for (ll i = 0; i < xs.size(); i++)
    {
        cout << xs[i] << ",";
    }
    cout << lowp.first << "]" << nn;
     cout<<nn<<nn<<nn;
```

```
    cout << "y=[";

    for (ll i = 0; i < ys.size(); i++)
    {
        cout << ys[i] << ",";
    }
    cout << lowp.second<<"]" << nn;
    cout<<nn<<nn<<nn;
    cout<<"plt.plot(x,y,c='b')"<<nn;

}
```

This function takes input of the points and finds the leftmost and the rightmost point and calls quickhull function for both sides on the two points. The rest of the function outputs a suitable code for python language that plots all the points and the points of the convex hull.

Algorithm-2: Graham Scan

Algorithm description:

One popular method for determining the convex hull of a set of points in a two-dimensional space is the Graham Scan algorithm. Known for its ease of use and effectiveness, Graham Scan uses an angular-sweep method to methodically build the convex hull. The pivot point of the convex hull is formed by the algorithm's initial selection of the point with the lowest y-coordinate, or leftmost in the event of a tie.

After that, the other points are arranged according to their polar angles concerning the pivot point. The technique can effectively track the convex hull's border thanks to this angular ordering. Following sorting, Graham Scan iteratively examines each point, taking into account how it relates to the previously chosen points in the convex hull. By cleverly eliminating points that don't add to the convex hull, the approach lowers the total computing cost.

Throughout its execution, Graham Scan keeps a convex hull stack and dynamically modifies it as it examines each point. The algorithm effectively locates and eliminates concave areas of the hull by repeatedly analyzing the angles created by subsequent points. A sorted series of points representing the convex hull in anticlockwise order is the end result.

Code:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

// VVI
#define fast                        \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);                   \
    cout.tie(0);
#define pb push_back
#define ll long long
#define nn '\n'
#define all(x) x.begin(), x.end()



/// Data structures
typedef unsigned long long ull;
typedef pair<ll, ll> pll;
typedef vector<ll> vl;
typedef vector<pll> vpll;
typedef vector<vl> vvl;
```

This is the basic header files,namespace and definition part. Here I use some macros to make my code short and effective.

```
void setIO()
{
#ifndef ONLINE_JUDGE
    freopen("input_06.txt", "r", stdin);

    freopen("output_06.txt", "w", stdout);
#endif // ONLINE_JUDGE
}
```

By this function input and output file streams are set.

```
vector<pll> points;
vector<pll>final_points;
ll n, x, y;
pll p0;
```

In this part, some variables and vectors are declared to store the points and the points of the convex hull (named as final_points) and a reference point p0 is declared.

```
pll nextToTop(stack<pll> &S)
{
    pll p = S.top();
    S.pop();
    pll res = S.top();
    S.push(p);
    return res;
}

ll getDist(pll a,pll b){
    return (a.first - b.first)*(a.first - b.first) +
          (a.second - b.second)*(a.second - b.second);
}
int angle(pll p, pll q, pll r)
{

    int val = (q.second - p.second) * (r.first - q.first) - (q.first - p.first) *
(r.second - q.second);

    if (val == 0)
        return 0;             // collinear
    return (val > 0) ? 1 : 2; // clock or counterclock wise
}
```

nextToTop function returns the second top stack element and it doesn't change the stack. getDist function returns the distance between two cartesian points. The angle function takes 3 points as input, p ,q and r. It returns if p,q and r are co linear or not. If not, if returns 1 if pq line's angle with x axis is less than pr line's

angle with x axis. It is basically a derived formula from the cross product: pq X pr = |pq||pr| sin (α). It returns 1 if the angle, alpha is greater than 0 , else retuns 2.

```
bool cmp2(pll a, pll b)
{
    // if(getDist(a,p0)!=getDist(b,p0)) getDist(a,p0)<getDist(b,p0);
    if((a.second - p0.second) * (b.first - p0.first)==(b.second - p0.second) *
(a.first - p0.first)) return getDist(a,p0)<getDist(b,p0);
    return (a.second - p0.second) * (b.first - p0.first) < (b.second - p0.second)
* (a.first - p0.first);
}
```

This custom comparators compares two points a and b with respect to the reference point p0 and returns which should come first in order to form a convex hull order. To do this, it checks the angle created with x axis taking the points a and b. if the angles are equal it returns the point closer to the reference point.

```
void print(vector<pll> &vec,char c){
    vector<ll>xs,ys;
    for(auto it:vec){
        xs.push_back(it.first);
        ys.push_back(it.second);
    }
    cout << "x=[";

    for (ll i = 0; i < xs.size(); i++)
    {
        cout << xs[i];
         if(i!=xs.size()-1) cout<< ",";
    }
    cout <<  "]" << nn;

    cout<<nn<<nn<<nn;
    cout << "y=[";

    for (ll i = 0; i < xs.size(); i++)
    {
        cout << ys[i];
```

```
        if(i!=ys.size()-1) cout << ",";
    }
    cout << "]" << nn;
    cout<<nn<<nn<<nn;

    cout<<"plt.scatter(x,y,c='"<<c<<"') "<<nn;
}
```

This function takes a vector of point as input and a color defining character c. Then it formats the output suitable for a python code to plot the points in cartesian plane.

```
void convex_hull_graham()
{
    cin >> n;
    for (ll i = 0; i < n; i++)
    {
        cin >> x >> y;
        points.push_back({x, y});
    }
    print(points,'b');
    ll ymin = points[0].second, min = 0;
    for (ll i = 1; i < n; i++)
    {
        if (ymin > points[i].second || (ymin == points[i].second &&
(points[min].first < points[i].first)))
        {
            ymin = points[i].second;
            min = i;
        }
    }
    swap(points[0], points[min]);
    p0=points[0];
    sort(points.begin()+1,points.end(),cmp2); // sort by p0 according to angles

    for(ll i=1;i<n;i++){
         while (i < n-1 && angle(p0, points[i], points[i+1]) == 0) i++;
         final_points.push_back(points[i]);
    }
    // final_points.resize(n);
    // final_points=points;
    ll m=final_points.size();
```

```
    if(m<3){
        cout << "No convex hull found" << nn;
        return;
    }
    stack<pll> S;
    S.push(final_points[0]);
    S.push(final_points[1]);
    S.push(final_points[2]);

   for (ll i = 3; i < m; i++)
   {
      while (S.size()>1 && angle(nextToTop(S), S.top(), final_points[i]) != 2)
S.pop();
      S.push(final_points[i]);
   }
   vector<pll>ans;
   while (!S.empty())
   {
       pll p = S.top();
       ans.push_back(p);
       S.pop();
   }
   ans.push_back(points[0]);
   ans.push_back(ans[0]);
   print(ans,'r');
   cout<<"plt.plot(x,y,c='g')"<<nn;

}
```

This is the main graham scan function. It takes n points as input and finds the point with lowest y axis value and swaps it with the first element. So, the reference point is the first element and then it only keeps the points that are not co linear in final_points vector. Now, it decides, if a convex hull is possible by checking if the final_points vector has more than 3 elements. At first graham scan pushes 3 points in the stack and for the rest of the points it checks if we take the point, whether we'll take a clockwise turn or an anticlockwise turn. If we take anticlockwise turn, we push it to the stack or else we keep popping the top element from the stack. Thus, for each point we check only once if it is a part of the convex hull or not. Finally we push the reference point and the first point of the to the answer to make a complete convex hull (we're adding the last point to make it suitable for our python code).

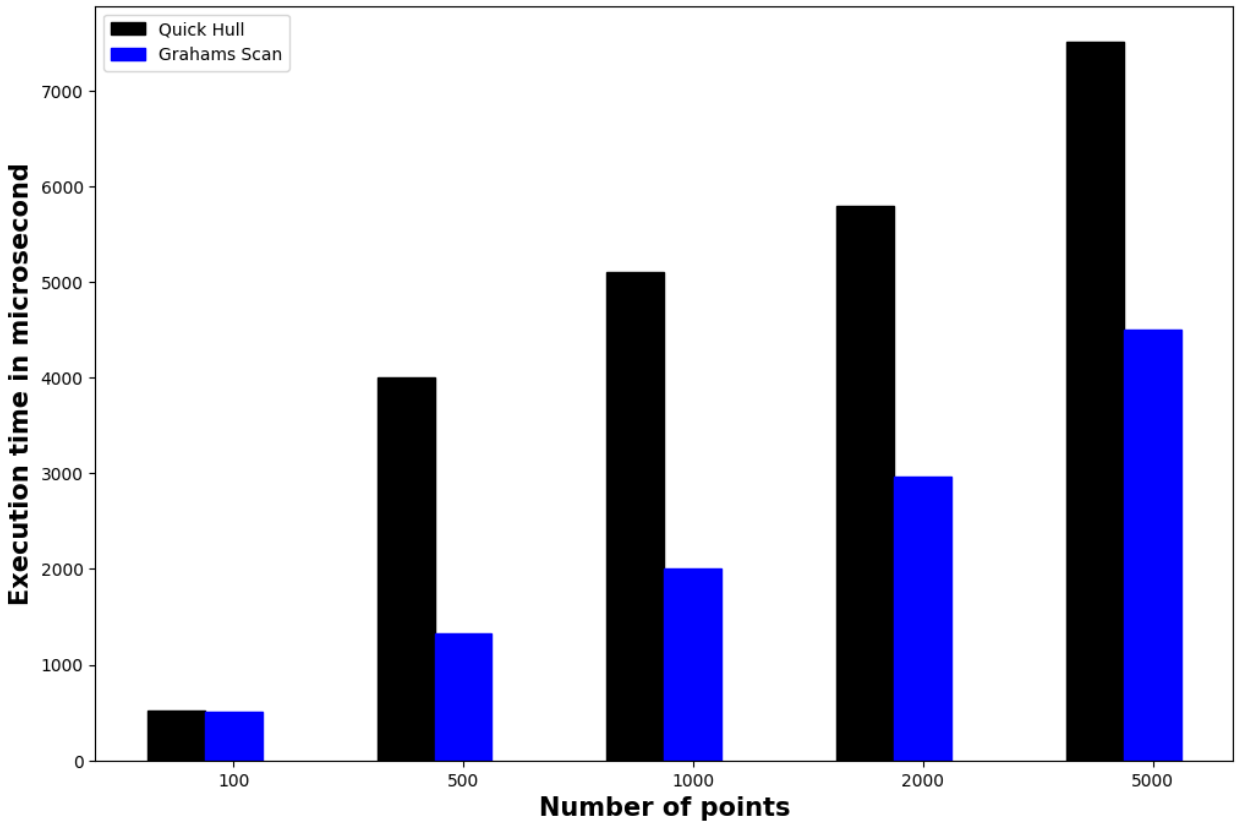
```
using namespace std::chrono;
int main()
{
    fast;
    ll t;
    setIO();
    // ll tno=1;;
    t = 1;
    // cin>>t;

    while (t--)
    {
        auto start = high_resolution_clock::now();
        convex_hull_graham();
        auto end = high_resolution_clock::now();
        auto duration = duration_cast<microseconds>(end - start);
        double dur=(double)duration.count();
        cout<<fixed<<setprecision(10)<<"Duration for graham's algorithm for
"<<n<<"points= "<<dur<<"microseconds"<<nn;
    }

    return 0;
}
```

In the main function we call the graham scan function and find the duration to execute the whole program. We call the set IO function to take input from input file and write in output file.

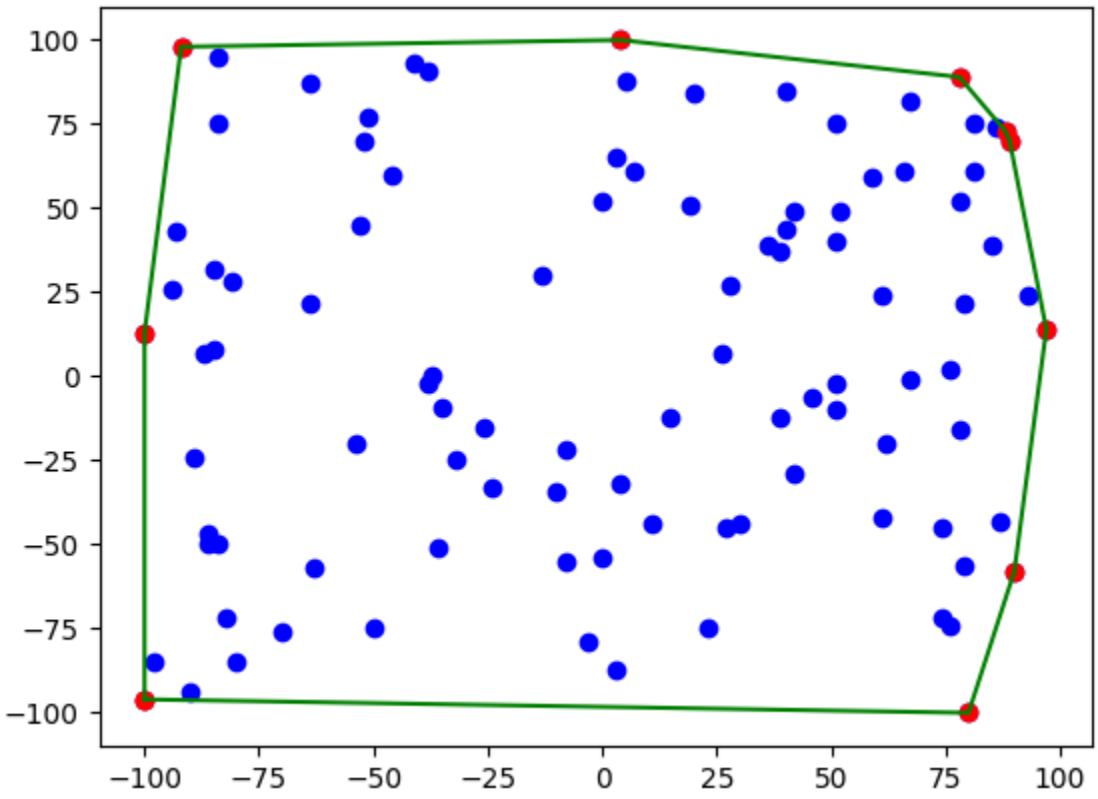
Time comparison :



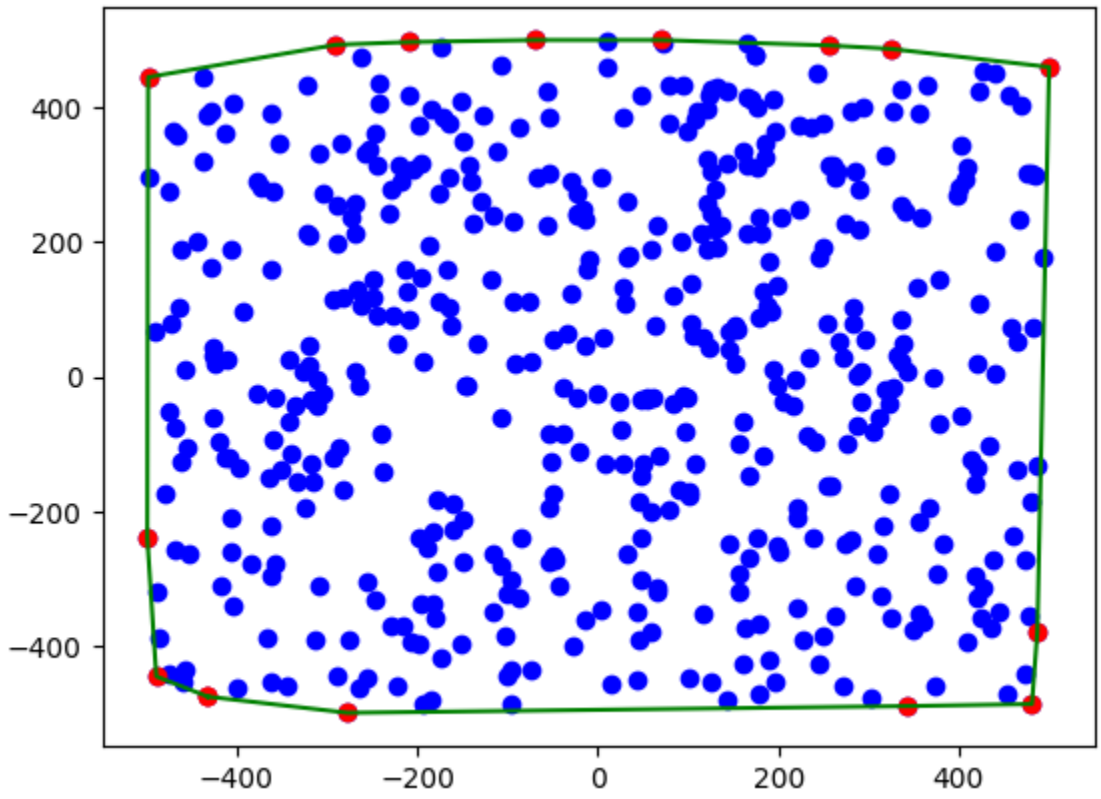
Output with visualization:

The output is same for both algorithms. Here's a few visualization for different number of points.

For 100 points:



For 500 points:



For 1000 points: