**Title of Project**

Convex Hull

**Team Members**

- Ritik Agarwal - 2018202017
- Suchismith Roy - 2018201101

**Deliverables**

- Graham Scan algorithm
- Jarvis March algorithm
- Quickhull algorithm
- Comparison of time and space complexity.
- Improvements over existing algorithms.
- Pruning of unnecessary points.

**Project Delivery Plan**

- Convex hull is popular concept from geometry. The objective is to implement the various algorithms for convex hull. Also after this step , work on improvement over standard algorithms.
- The algorithms being studied inlcude Graham Scan , Jarvis March and Quickhull.
- The idea is to understand and make an abstract design of the algorithm and a comparison between them.
- It uses different methods for solving the problem which includes incremental method , divide and conquer method and prune and search method.
- Applications of it includes finding of farthest pair of points in any set of n points.
- Understand correctness of algorithm using standard proofs.

**Technologies to be used**

C++

**Online Resources**

- https://ieeexplore.ieee.org/document/1555560
- tmc.web.engr.illinois.edu/pub_ch.html
- cs.indstate.edu/ jtalasila/convexhull.pdf

**Repository where work is being committed**

https://github.com/ritik001/Convex-Hull/

**Jarvis March Algorithm**

Jarvis march algorithm starts by computing the leftmost point i.e the point whose x-coordinate is smallest ,say l.This point must be a convex hull vertex. We can find this point in O(n) time. The algorithm then performs a series of pivoting steps to find each successive convex hull vertex, starting with l and continuing until it reaches l again. We can find each successive vertex in linear time by performing a series of O(n) counterclockwise tests. In each pivoting step, we randomly choose the next point,say p among all the available point.Now considering the line segment lp, we choose the point the is leftmost to the the current line segment.

Since the algorithm spends O(n) time for each convex hull vertex, the worst-case running time is O($n^2$). However,If the convex hull has very few vertices, Jarviss march is extremely fast. A better way to write the running time is O(nh), where h is the number of convex hull vertices. In the worst case, h = n. This an output-sensitive algorithm; the smaller the output, the faster the algorithm.

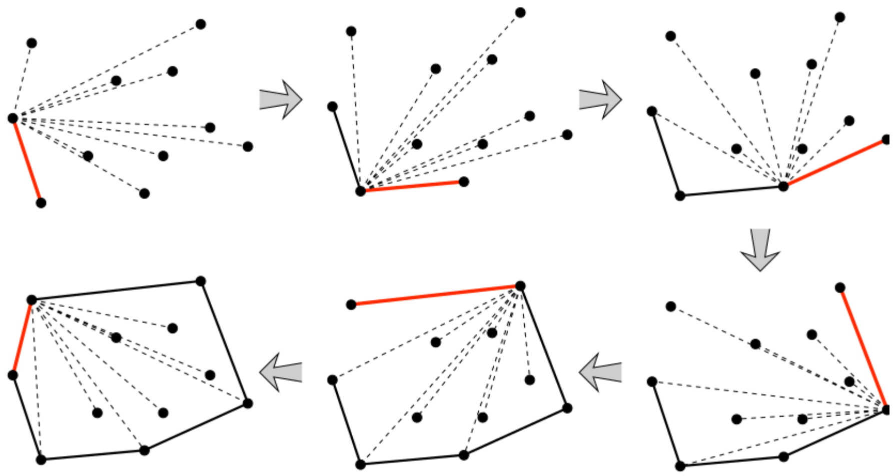


Figure 1: The execution of Jarvis March.

**Quick Hull Algorithm**

The QuickHull algorithm is a Divide and Conquer algorithm similar to QuickSort. The algorithm finds the point with minimum x-coordinate lets say, min_x and similarly the point with maximum x-coordinate, max_x. Make a line joining these two points, say L. This line will divide the the whole set into two parts. Take both the parts one by one and proceed further.

The algorithm finds the point P with with maximum distance from the line L. P forms a triangle with the points min_x, max_x. The points residing inside this triangle can never be the part of convex hull. The above step divides the problem into two subproblems (solved recursively). Now the line joining the points P and min_x and the line joining the points P and max_x are new lines and the points residing outside the triangle is the set of points. The algorithm has a worst case run-time of O($n^2$) and an average case run-time of O(n log n). However, it is possible to design data for which this algorithm will take O($n^2$) time. For example, the algorithm has a bad run time when the points are located on the circumference of a circle.

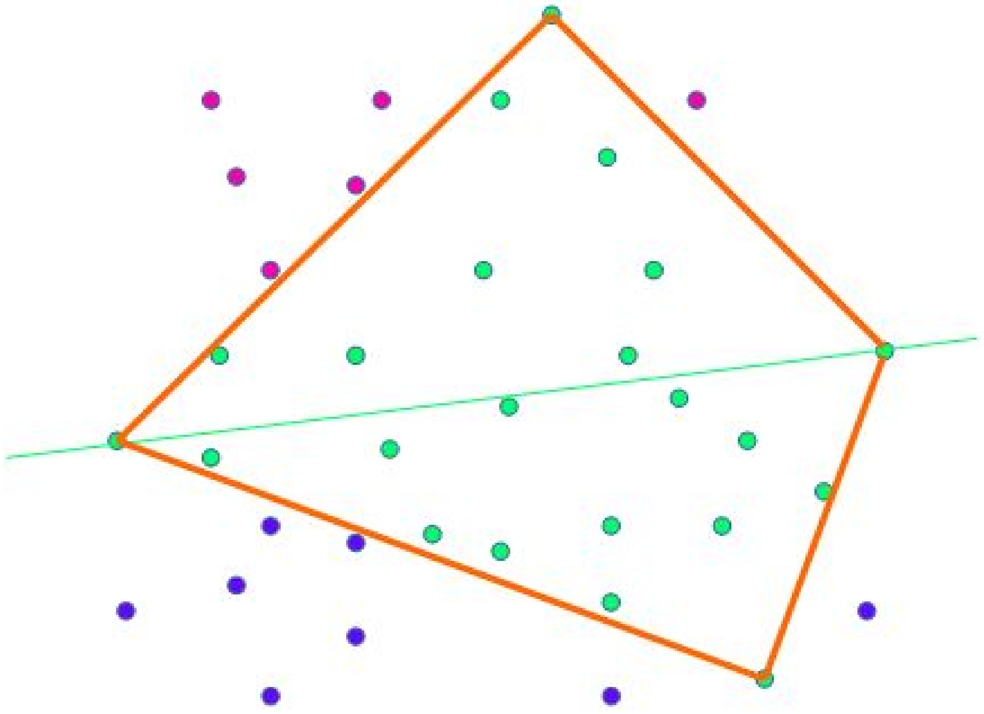


Figure 2: The execution of QuickHull Algorithm.

## Graham Scan Algorithm

Grahams scan algorithm first explicitly sorts the points in O(n log n) and then applies a linear-time scanning algorithm to finish building the hull. We start Grahams scan by finding the leftmost point l, just as in Jarviss march. Then we sort the points in counterclockwise order around l. We can do this in O(n log n) time with any comparison-based sorting algorithm (quicksort, mergesort, heapsort, etc.). To compare two points p and q, we check whether the triple l, p, q is oriented clockwise or counterclockwise. Once the points are sorted, we connect them in counterclockwise order, starting and ending at l. The result is a simple polygon with n vertices. While trajecting the convex hull, the algorithm maintains a stack, to compare the orientation of the chosen point with the top of the stack and the element to next of top of the stack. If the orientation is a non-left turn, we pop the top element from the stack and push the currently chosen element. Hence in this way, we ensure we are eliminating the points which cannot be a part of the final convex hull.

Checking orientation for each point requires O(1) time. For adding each new point we have to make the orientation test and once a point is removed from the stack it is never considered again so the total time complexity after the sorting step is O(n).

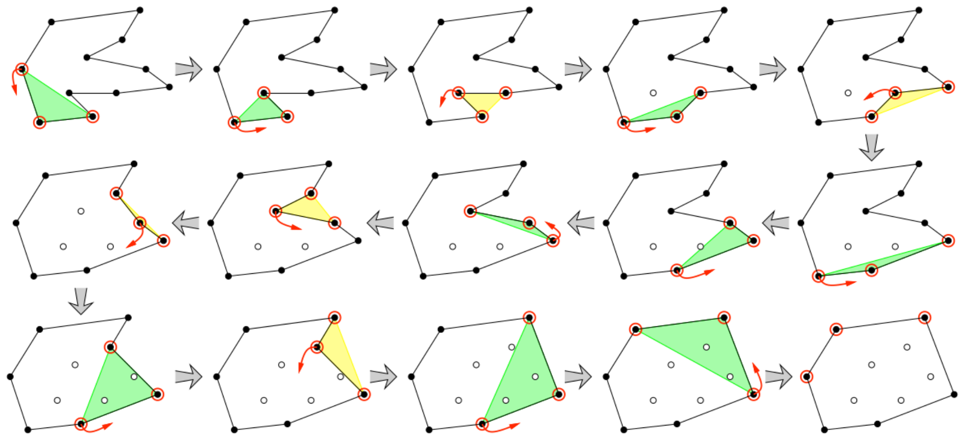


Figure 3: The scanning phase of Grahams scan.

## Optimized Graham Scan Algorithm

We can improve the running time of the Graham Scan Algorithm , by eliminating points that will never be a part of the convex hull. We find out the farthest points in North, South ,East and West directions. We eliminate the points which lies interior to the quadrilateral formed by these 4 points. This reduces the number of points by a good margin for large inputs. Generally $\sqrt[2]{n}$ points are left on average. We can perform Graham Scan on these points. However, the algorithm might fail to improve the running time, when no points lie inside the quadrilateral,i.e when all the points are spread across the boundary of the quadrialteral.

## Plan for testing of Input

- Input will be a set of points and we need to verify output is smallest polygon which contains a set of points.

- For this constraints will be on number of points for negative testing.
- Verify using breakpoints if orientation is correct on every step i.e. left or right move and push or pop depending on requirement of algorithm.
- Input will consist of T test cases. Each test case will contains a integer N denoting the no of points. Then in the next line are N*2 values denoting the points ie x and y.
- If no convex hull is possible it will return -1 else the set of points in the hull.

**Basic Test Cases.**

- **Testing Variation 1**
  Input N points - 8
  {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1, 2}, {3, 1}, {3, 3}}
  Points after xmin & xmax and ymin & ymax condition {{0,0},{4,4}}
  Output - 0,3,4,4,3,1,0,0
  Explanation - This results in a straight line case.We have two points (0,0) and (4,4).So we find out farthest points from it on both sides. This addes (0,3) and (4,4) to our stack and returns the convex hull.

- **Testing Variation 2**
  Input N points - 4
  {{0, 2}, {2,4}, {4,2}, {2,0}}
  Points after xmin & xmax and ymin & ymax condition {{0,2},{2,0},{2,4},{4,2}}
  Output - {{0,2},{2,0},{2,4},{4,2}}
  Explanation - This results in a four side polygon case.As all point are extremes and no other point is left to check , it directly returns the convex hull.

- **Testing Variation 3**
  Input N points - 8
  {{0, 2}, {2,4}, {4,2}, {2,0},{1,2},{2,2},{2,1},{2,3}}
  Points after xmin & xmax and ymin & ymax condition {{0,2},{2,0},{2,4},{4,2}}
  Output - {{0,2},{2,0},{2,4},{4,2}}
  Explanation - This results in a four side polygon case.After this it checks remaining points if they lie inside this polygon.If they lie inside it , eliminate it.
  {1,2},{2,2},{2,1},{2,3}
  All the above points lie inside and hence are eliminated.

- **Testing Variation 4**
  Input N points - 6

{{0, 2}, {2,4}, {4,2}, {2,0},{3,5},{2,3}}
Points after xmin & xmax and ymin & ymax condition {{0,2},{2,0},{3,5},{4,2}}
Output - {{0,2},{2,0},{3,5},{4,2}}
Explanation - This results in a four side polygon case.After this it checks remaining points if they lie inside this polygon.If they lie inside it , eliminate it.
{2,4},{2,3}
All the above points lie inside and hence are eliminated.

**Initial Optimizations**

- When we consider a set of points and a polygon (in our case three or four sided) formed using extreme points. There are two possibility.

  **1. Rectangle** - In best case we can get a rectangle where points are distributed uniformly and we are able to prune a major part of points.
  **2. Circle** - It would result in worst case. If points are in a densed way , outside polygon formed from extremes ( in a circular manner) , it would result in worst case as pruning won't be able to remove any such points.
- Initial plan was to use each line segment formed using extreme points( maximum four) , and apply Jarvis March on it. Jarvis March takes O(n) in each pass. So for 4 passes it would take O(n) time , but this approach was not effective because of random distribution of points.
- If we calculate slope of line with a point to be checked and an extreme point , it will help us to compare it with slope of other boundary in the polygon.This can help us to find if point is in polygon in O(1) time.

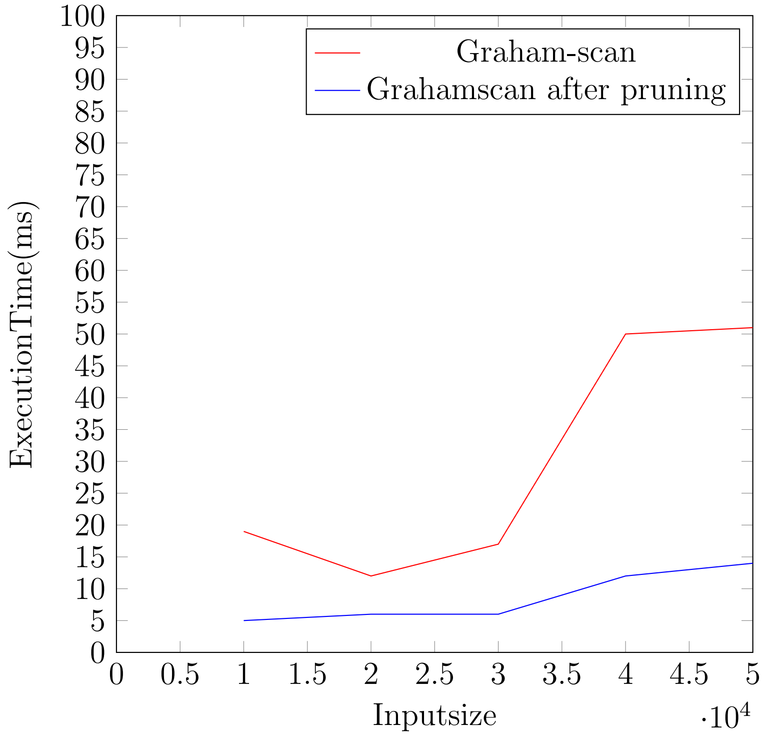
**Final Optimizations**

- The approach was to find Point with minimum and maximum points with x and y co-ordinates.
  **1.** For finding ymax point if more than one point has similar ymax point , choose one with max x co-ordinate.
  **2.** For finding ymin point if more than one point has similar ymin point , choose one with min x co-ordinate.
  **3.** For finding xmax point if more than one point has similar xmax point , choose one with max y co-ordinate.
  **4.** For finding xmin point if more than one point has similar xmin

point , choose one with min y co-ordinate.

- **Cases Possible**
  **1. Straight Line** - If Points at xmin & ymin and xmax & ymax are same. We will get two distinct ordered pairs. In this case we need to search for farthest point on upper(positive) and lower(negative) side.
  **i)** If no such point exists , we return -1 as we can't make a polygon with less than three points.
  **ii)** If both points exists , our polygon will contain 4 points and it would provide us the answer.
  **iii)** If one of the point exists , the output of convex hull would be a triangle.
  **2. Triangle** - If using Points at xmin & ymin and xmax & ymax are we get three distinct ordered pairs. In this case our output of convex hull would be a triangle.
  **3. Quadilateral** - After performing check based on xmin , xmax , ymin and ymax Points , if we get four set of points( maximum number of points we can get). Our next step is to prune points which are inside this Quadilateral.
  **i)** One best case would be these extreme points forming a rectangle and all points inside it are pruned.
  **ii)** Once all points inside this polygon are eliminated , we can apply convex hull on remaining points which runs in O(nlogn).In this case complexity would be same , but are n( set of points) would be less as we had already pruned some of the points.

Comparision of running time of Graham scan and optimised graham scan after pruning points

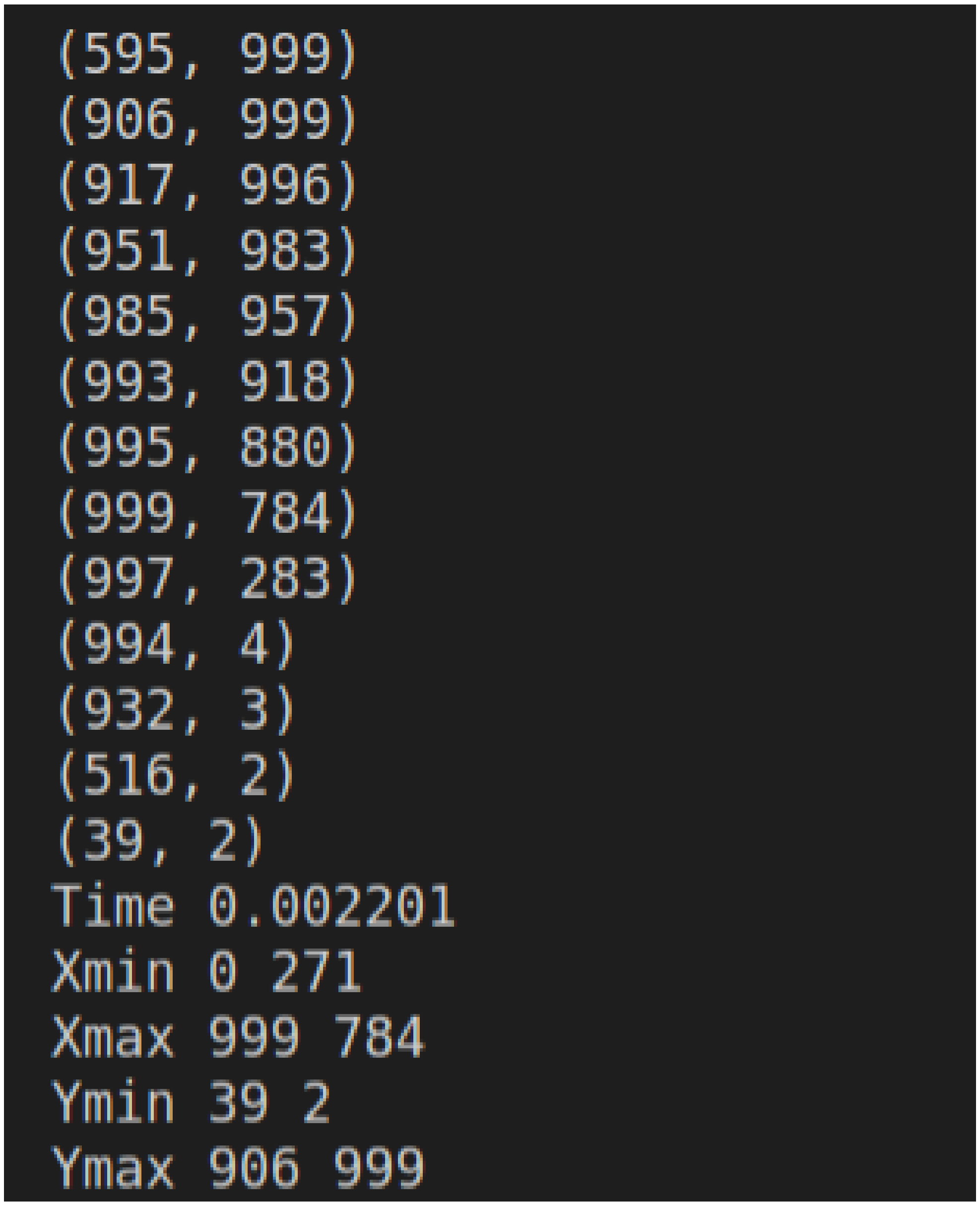


**Analysis of pruning optimization using time functions.**

- **Variation 1**
  Input N points - 10000
  Points were generated using rand() function in range of 0-999
  Output - (5,14),(0,271),(0,942),(43,986),(152,996),(595,999),(906,999),(917,999),(951,983),(98
  **Time Analysis -**
  Time was calculated using clock_t start, end and time_taken as **((double)(end - start)) / CLOCKS_PER_SEC**
  Execution using Graham Scan took 0.002201 seconds and execution using pruning and Graham Scan after pruning took 0.000948 seconds. Also time taken for Graham Scan alone after pruning was around 0.00002 seconds.

**End user documentation. How can your system be used to test that it works.**

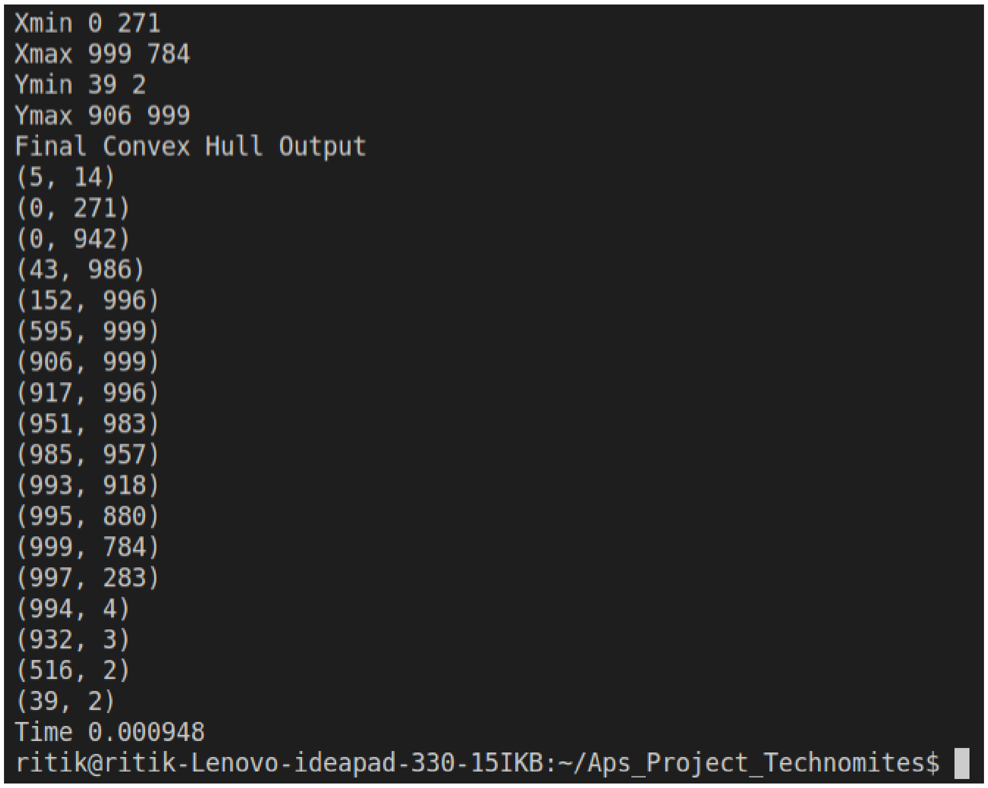
- **Small Inputs** 1. For small inputs , enter value of N (which is the number of points).
  2. Now enter x and y co-ordinate of N points.
  3. In Pruning_optimization_convexhull.cpp , time_taken with and without optimization , both are displayed.
- **Large Inputs** 1. For testing against large inputs , uncomment



```
(595, 999)
(906, 999)
(917, 996)
(951, 983)
(985, 957)
(993, 918)
(995, 880)
(999, 784)
(997, 283)
(994, 4)
(932, 3)
(516, 2)
(39, 2)
Time 0.002201
Xmin 0 271
Xmax 999 784
Ymin 39 2
Ymax 906 999
```

Figure 4: Graham Scan Execution over 1000 random points in range(0,999).

the rand() code in main() which will randomly generate points in given range and test both the algorithm.



```
Xmin 0 271
Xmax 999 784
Ymin 39 2
Ymax 906 999
Final Convex Hull Output
(5, 14)
(0, 271)
(0, 942)
(43, 986)
(152, 996)
(595, 999)
(906, 999)
(917, 996)
(951, 983)
(985, 957)
(993, 918)
(995, 880)
(999, 784)
(997, 283)
(994, 4)
(932, 3)
(516, 2)
(39, 2)
Time 0.000948
ritik@ritik-Lenovo-ideapad-330-15IKB:~/Aps_Project_Technomites$
```

Figure 5: Pruning and Graham Scan complete time.