Table of Contents

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Convex hull and various algorithms for obtaining the 2D convex hull

Git Repository : https://github.com/bhavinkotak07/convex-hull

Abstract

Convex hull is popular concept from geometry. The objective of this project is to study at least three algorithms for obtaining a convex hull in the 2-d setting and compare the practical efficiency of the algorithms.

The main aim of the project is to optimize standard algorithms so that we can reduce the time complexity of the those algorithms by a constant factor.

Definitions

We are given a set P of n points in the plane. We want to compute something called the convex hull of P. Intuitively, the convex hull is what you get by driving a nail into the plane at each point and then wrapping a piece of string around the nails. More formally, the convex hull is the smallest convex polygon containing the points:

• Polygon

A region of the plane bounded by a cycle of line segments, called edges, joined end-to-end in a cycle. Points where two successive edges meet are called vertices.

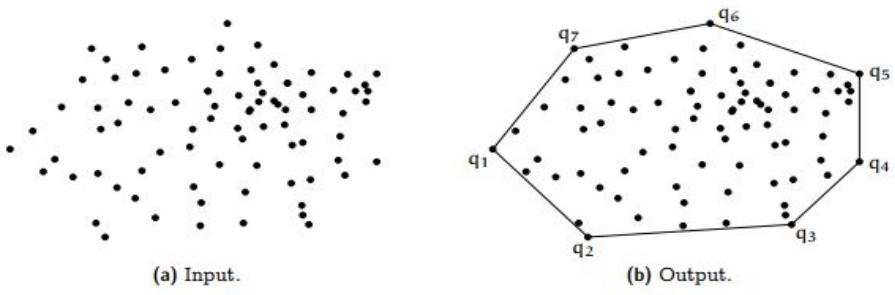
• Convex

For any two points p, q inside the polygon, the line segment pq is completely inside the polygon.

• Smallest

The minimal set of points that forms a polygon that covers all other points in the plane.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071



(a) Input. (b) Output.

We Studied three algorithms for finding the convex hull and analysed their performance in terms of time complexity.

1. Jarvis March (Gift wrapping) algorithm
2. Graham's Algorithm
3. QuickHull Algorithm

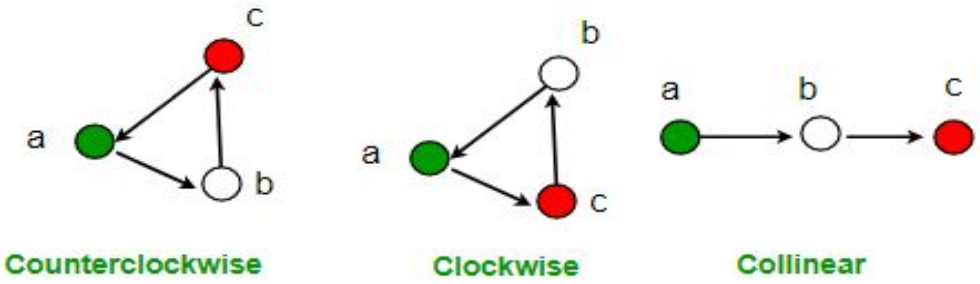
Before diving deep into any of the algorithm, here is the basic explanation of what orientation of points means in the algorithm and how to find orientation of three points.
(Finding orientation of third point with respect to a given two points i.e. given line segment is very much important in all the three algorithms)

Orientation

Orientation of an ordered triplet of points in the plane can be

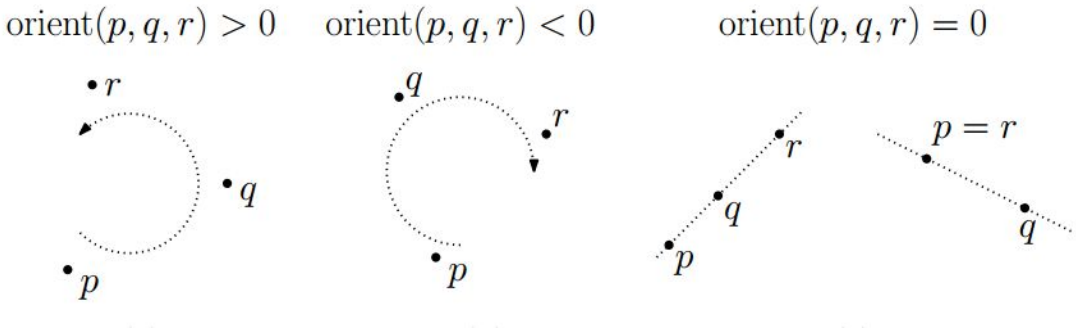
- counterclockwise
- clockwise
- collinear

The following diagram shows different possible orientations of (a,b,c)



We can find the orientation of three points by cross product of vectors formed by the points. The result will be either negative, positive or zero based on the angle formed between the vectors and hence will denote the orientation.

$$\text{Orient}(p,q,r) = \det \begin{pmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{pmatrix}.$$



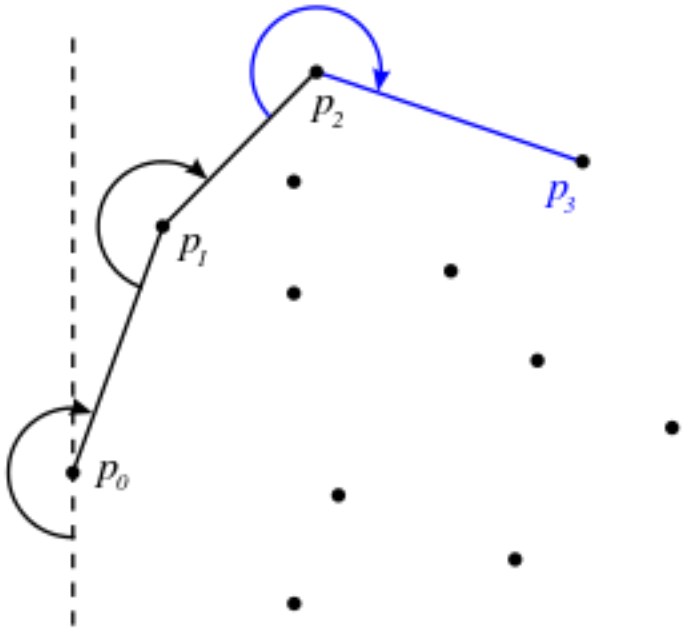
Thus, finding out whether the points p,q,r are making a left turn or a right turn is a simple calculation of a determinant.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Jarvis's March Algorithm

The Jarvis March algorithm builds the convex hull in O(nh) where h is the number of vertices on the convex hull of the point-set. Note that if h≤O(nlogn) then it runs asymptotically faster than Graham's algorithm.

The gift wrapping algorithm begins with i=0 and a point p_0 known to be on the convex hull, e.g., the leftmost point, and selects the point p_{i+1} such that all points are to the right of the line p_i p_{i+1}. This point may be found in O(n) time by comparing polar angles of all points with respect to point p_i taken for the center of polar coordinates. Letting i=i+1, and repeating with until one reaches p_h=p_0 again yields the convex hull in h steps. In two dimensions, the gift wrapping algorithm is similar to the process of winding a string (or wrapping paper) around the set of points.



The point is on the left of present line segment or on right of it, can be decided from orientation discussed above.

Pseudo Code

```
jarvis(S)
  // S is the set of points
  pointOnHull = leftmost point in S // which is guaranteed to be part
of the CH(S)
  i = 0
  repeat
    P[i] = pointOnHull
    endpoint = S[0]      // initial endpoint for a candidate edge on the
hull
    for j from 1 to |S|
      if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to
endpoint)
        endpoint = S[j]   // found greater left turn, update endpoint
    i = i+1
    pointOnHull = endpoint
  until endpoint == P[0]      // wrapped around to first hull point
```

Time Complexity

The inner loop checks every point in the set *S*, and the outer loop repeats for each point on the hull. Hence the total run time is O(nh). The run time depends on the size of the output, so Jarvis's march is an output-sensitive algorithm.

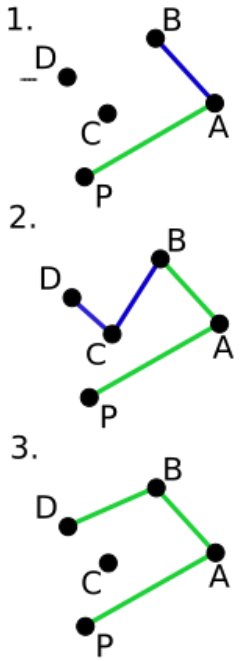
However, because the running time depends linearly on the number of hull vertices, it is only faster than O(n*logn) algorithms such as Graham scan when the number *h* of hull vertices is smaller than log *n*. Chan's algorithm, another convex hull algorithm, combines the logarithmic dependence of Graham scan

with the output sensitivity of the gift wrapping algorithm, achieving an asymptotic running time O(n*logh) that improves on both Graham scan and gift wrapping.

Worst Case : When all the given vertices are part of the convex hull $O(n^2)$

Graham's Algorithm

- Graham scan is a method of computing the convex hull of a finite set of points in the plane with time complexity O(n logn).
- The algorithm finds all vertices of the convex hull ordered along its boundary.
- Graham's scan solves the convex-hull problem by maintaining a stack S of candidate points. Each point of the input set Q is pushed once onto the stack.
- And the points that are not vertices of CH(Q) are eventually popped from the stack.
- When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary.
- When we traverse the convex hull counter clockwise, we should make a left turn at each vertex. Each time the while loop finds a vertex at which we make a non-left turn ,the vertex is popped from the vertex.



As one can see, PAB and ABC are counterclockwise, but BCD isn't. The algorithm detects this situation and discards previously chosen segments until the turn taken is counterclockwise (ABD in this case.)

Pseudo Code:

```
GRAHAM-SCAN(Q)
1. Let p0 be the point in Q with the minimum y-coordinate,
   or the leftmost such point in case of a tie.
2. Let (p1, p2, . . . , pm) be the remaining points in Q,
   sorted by polar angle in counterclockwise order around p0(if
more than point has the same angle, remove all but the one that is farthest
from p0)
3. top[S]¬ 0
4. PUSH(p0,S)
5. PUSH(p1,S)
6. PUSH(p2,S)
7. for i ¬ 3 to m
8. do while the angle formed by points PREV-TO-TOP(S),
   TOP(S), and pi makes a nonleft turn
9.  do POP(S)
10. PUSH(S,pi)
11. return S
```

TIme Complexity

Sorting the points has time complexity O(n log n). While it may seem that the time complexity of the loop is O(n^2), because for each point it goes back to check if any of the previous points make a "right turn", it is actually O(n), because each point is considered at most twice in some sense. Each point can appear only once as a point (x2,y2) in a "left turn" (because the algorithm advances to the next point (x3,y3) after that), and as a point (x2,y2) in a "right turn" (because the point (x2,y2) is removed). The overall time complexity is therefore O(n log n), since the time to sort dominates the time to actually compute the convex hull.

Worst Case Complexity will also be O(n*logn) as it is not dependent on number of output vertices like Jarvis’s march algorithm

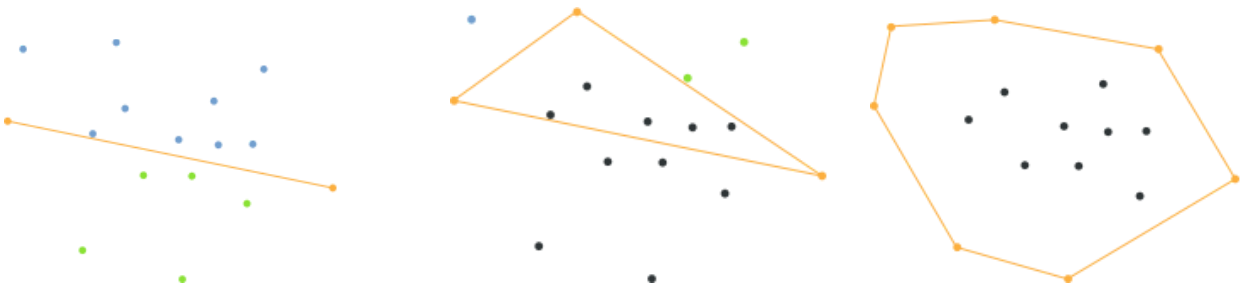
Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Quickhull Algorithm

Quickhull is a method of computing the convex hull of a finite set of points in the plane. It uses a divide and conquer approach similar to that of quicksort, from which its name derives. Its best case complexity is considered to be O(n * log(n)), whereas in the worst case it takes $O(n^2)$ (quadratic). However, unlike quicksort, there is no obvious way to convert quickhull into a randomized algorithm. Thus, its average time complexity cannot be easily calculated.

Under average circumstances the algorithm works quite well, but processing usually becomes slow in cases of high symmetry or points lying on the circumference of a circle. The algorithm can be broken down to the following steps:

1. Find the points with minimum and maximum x coordinates, as these will always be part of the convex hull.
2. Use the line formed by the two points to divide the set in two subsets of points, which will be processed recursively.
3. Determine the point, on one side of the line, with the maximum distance from the line. The two points found before along with this one form a triangle.
4. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.
5. Repeat the previous two steps on the two lines formed by the triangle (not the initial line).
6. Keep on doing so on until no more points are left, the recursion has come to an end and the points selected constitute the convex hull.



Steps 1-2: Divide points in two subsets

Steps 3-5: Find maximal distance point, ignore points inside triangle and repeat it

Step 6: Recurse until no more points are left

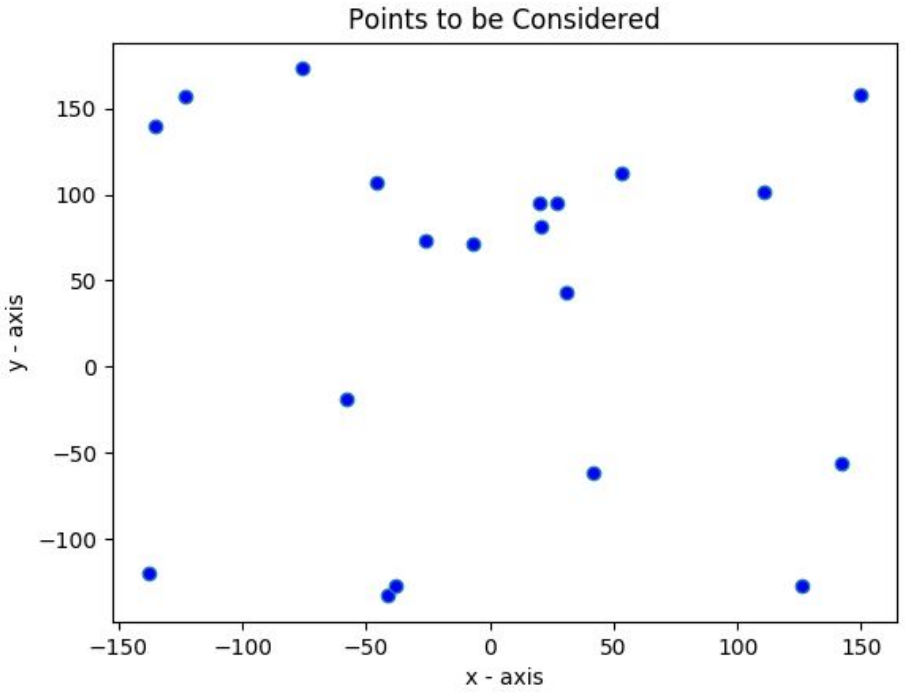
Time Complexity:

As stated above time complexity of quickhull, similar to quicksort in worst case is O(n2) also there is no method of randomizing the algorithm which is the case in quick sort.

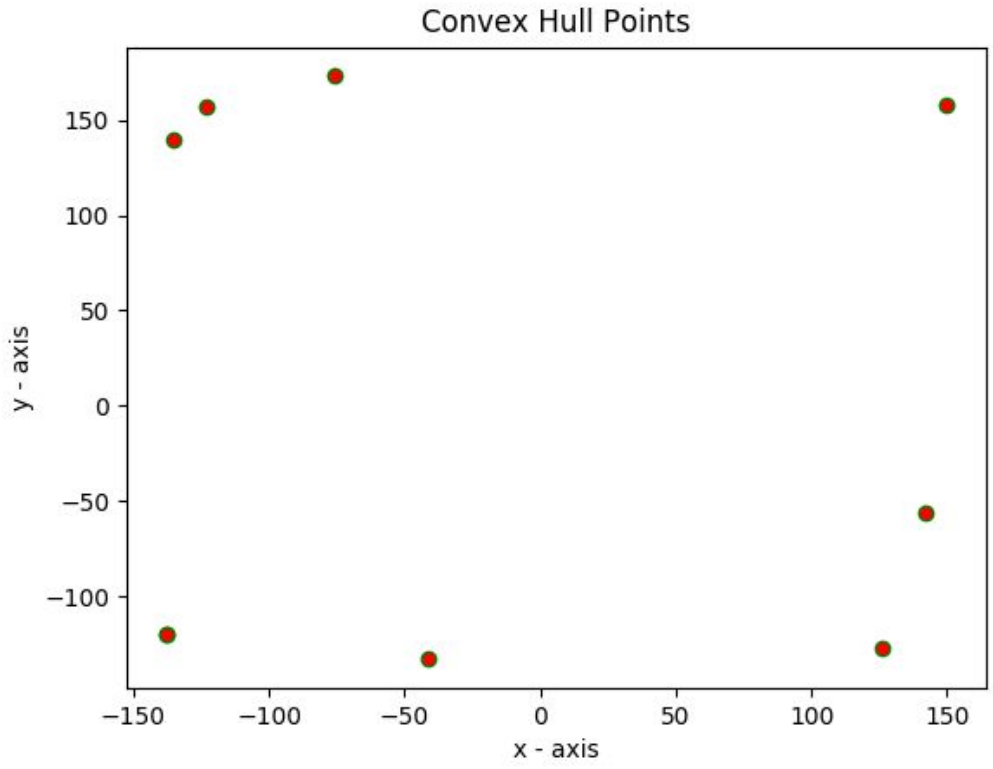
Comparison Between all the three algorithms :

We implemented all the three algorithms and also we analysed the results and the time taken by each of them for different inputs. To verify the results we build a vision module in python which gives the exact idea by plotting points and displaying the resultant convex hull.

Images from our vision module : Input:



Output:

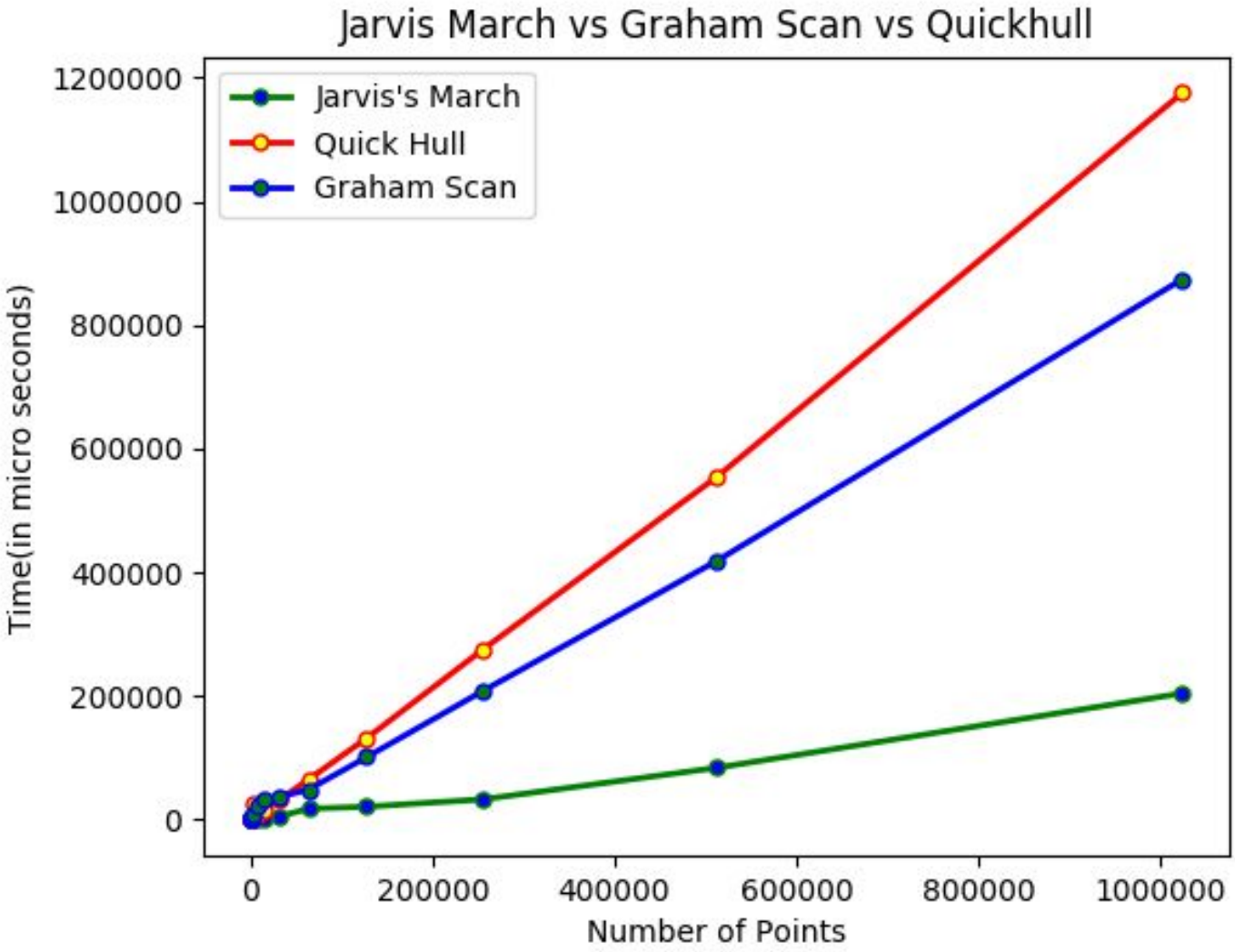


Comparison between three algorithms:

Worst case complexity of quick hull is O(n^2).
Worst case complexity of Graham Scan is O(nlogn).
Worst case complexity of Jarvis March is O(n^2).

Important thing to note here is the complexity of jarvis's march is O(nh), where h is the number of vertices that are the part of the output i.e. are part of convex hull. Hence jarvis march will always perform better in the case where h is smaller than logn. To generate worst case of jarvis march, all the points of input set should be part of convex hull. That will lead it to O(n2) solution.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071



The above graph was generated by us with number of points in the range [10 - 1024000] on X-axis and their respective time in terms of microseconds. As discussed above, it’s not possible to manually generate large worst case for jarvis's march as a result it is showing better results.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Optimization

There are many points in the input set which can be easily pruned on the basis of some calculations. So how to prune them?

We decided to find four points :

A = (Ax, Ay) which maximizes x-y

B = (Bx, Xy) which maximizes x+y

C = (Cx, Cy) which minimizes x-y

D = (Dx, Dy) which minimizes x+y

These four points are the extreme points which are definitely part of the convex hull. And any point inside the quadrilateral formed by these four points can be discarded as they are already covered by the region boundary formed by these four points. We identified the non-hull points and discarded them before applying any convex hull algorithm.

Algorithm:

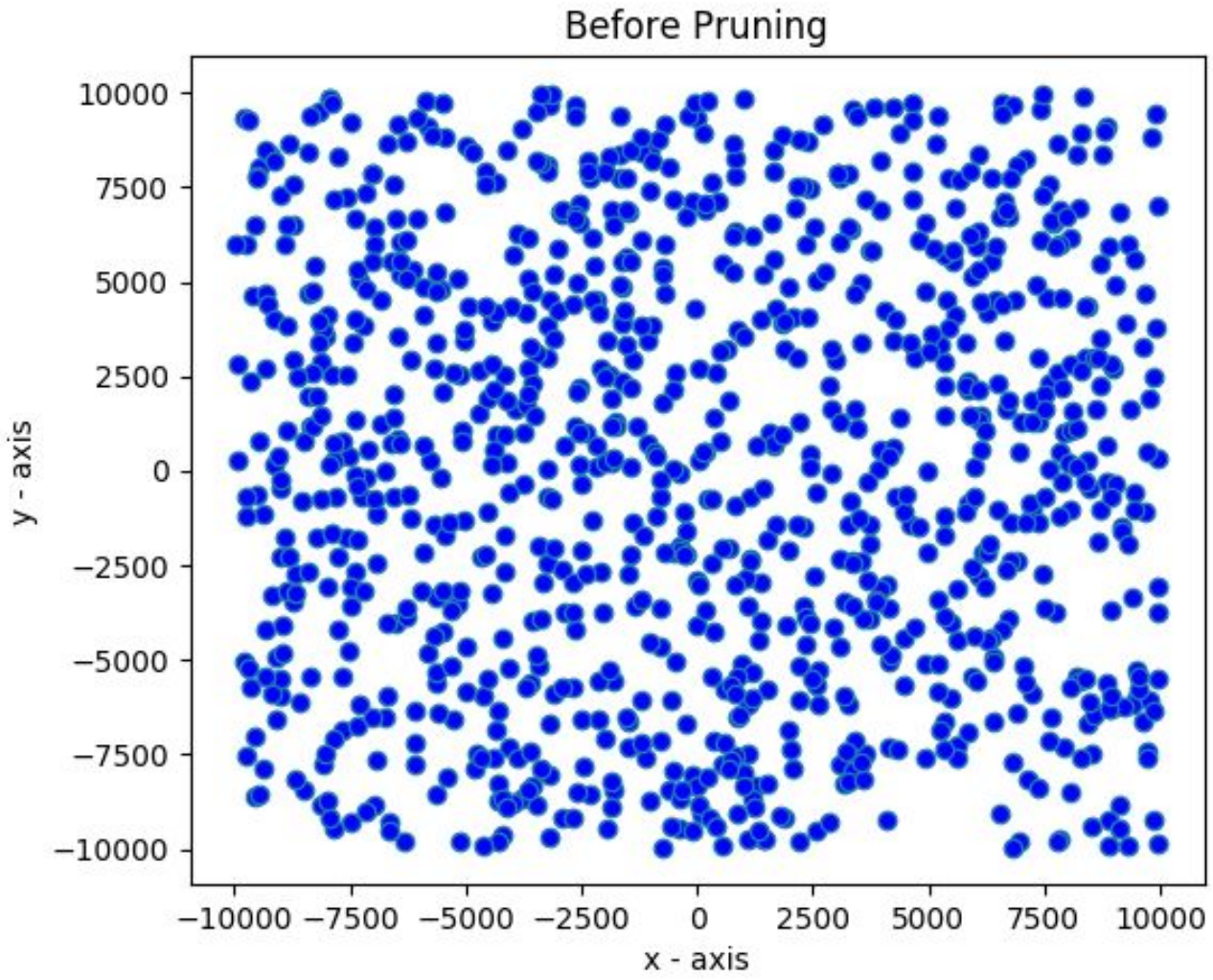
1. Traverse the input set and find A,B,C,D
2. Traverse the input set again and check if a point is inside the quadrilateral A,B,C,D prune them i.e. remove them.
3. The remaining points are the candidate points for convex hull and we can pass them to any of the algorithm discussed above.

Below is the results obtained from our vision module, which will make the approach more clear to understand.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

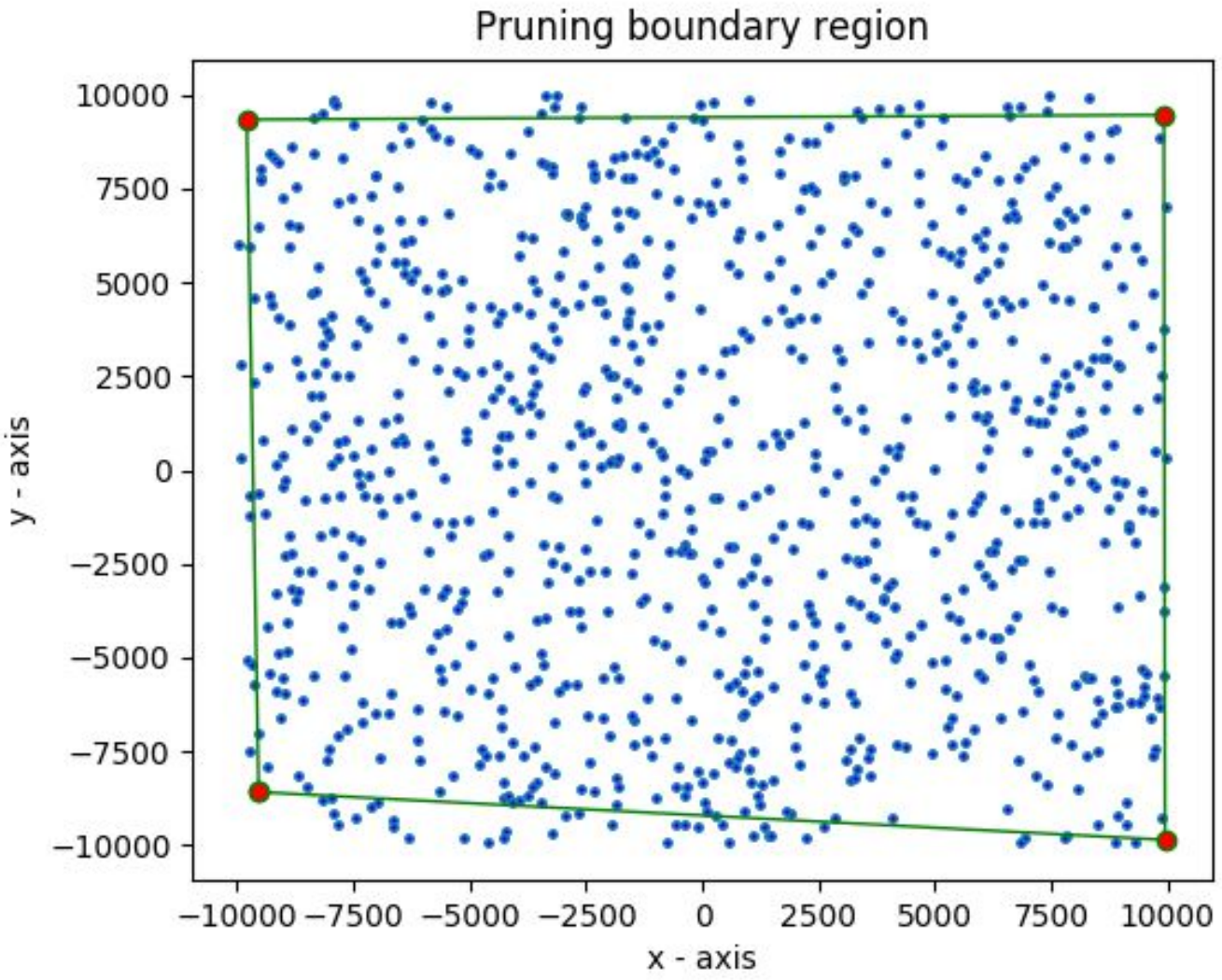
Test 1 :

1. Input set



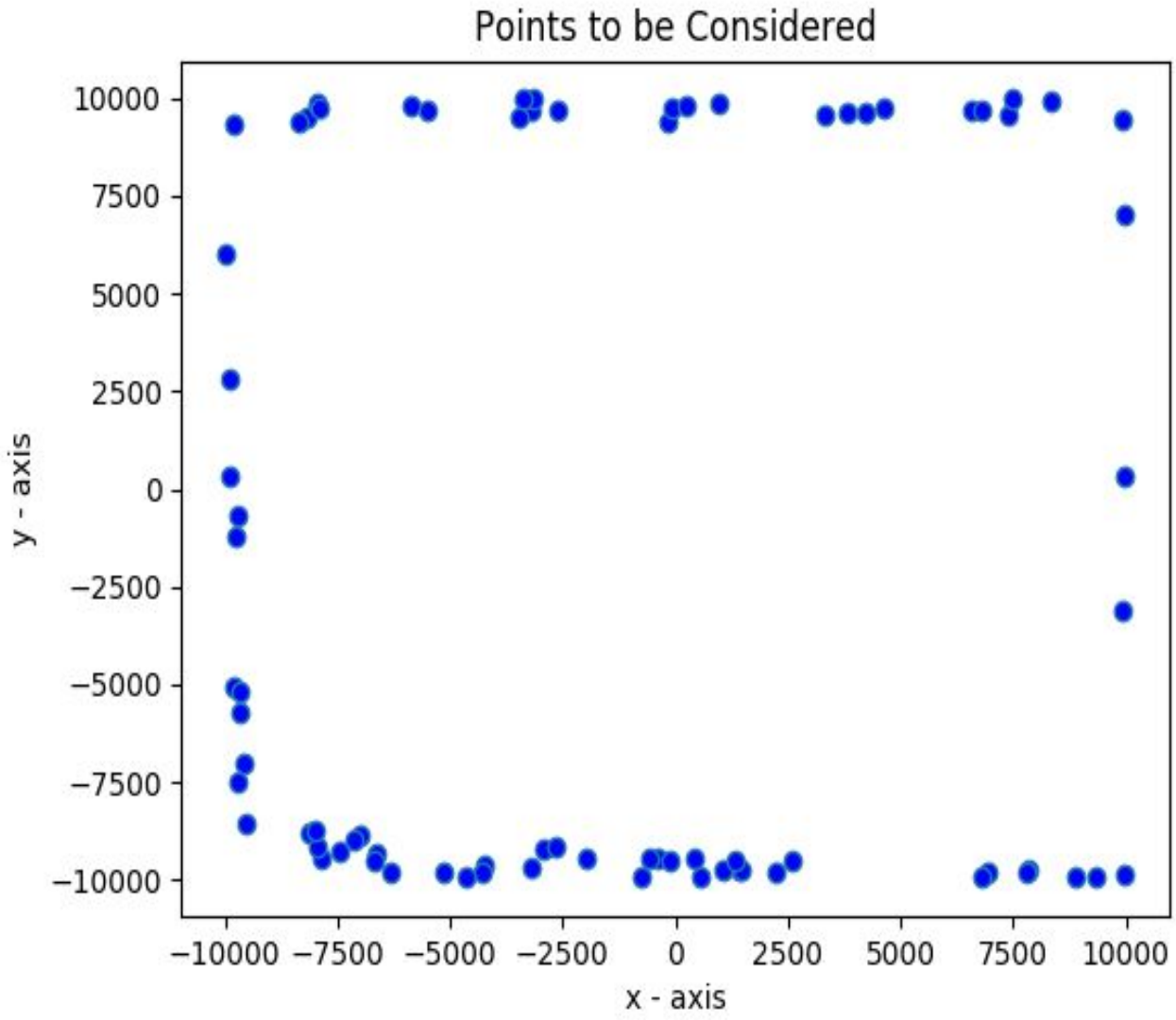
Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

2. Pruning boundary region



Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

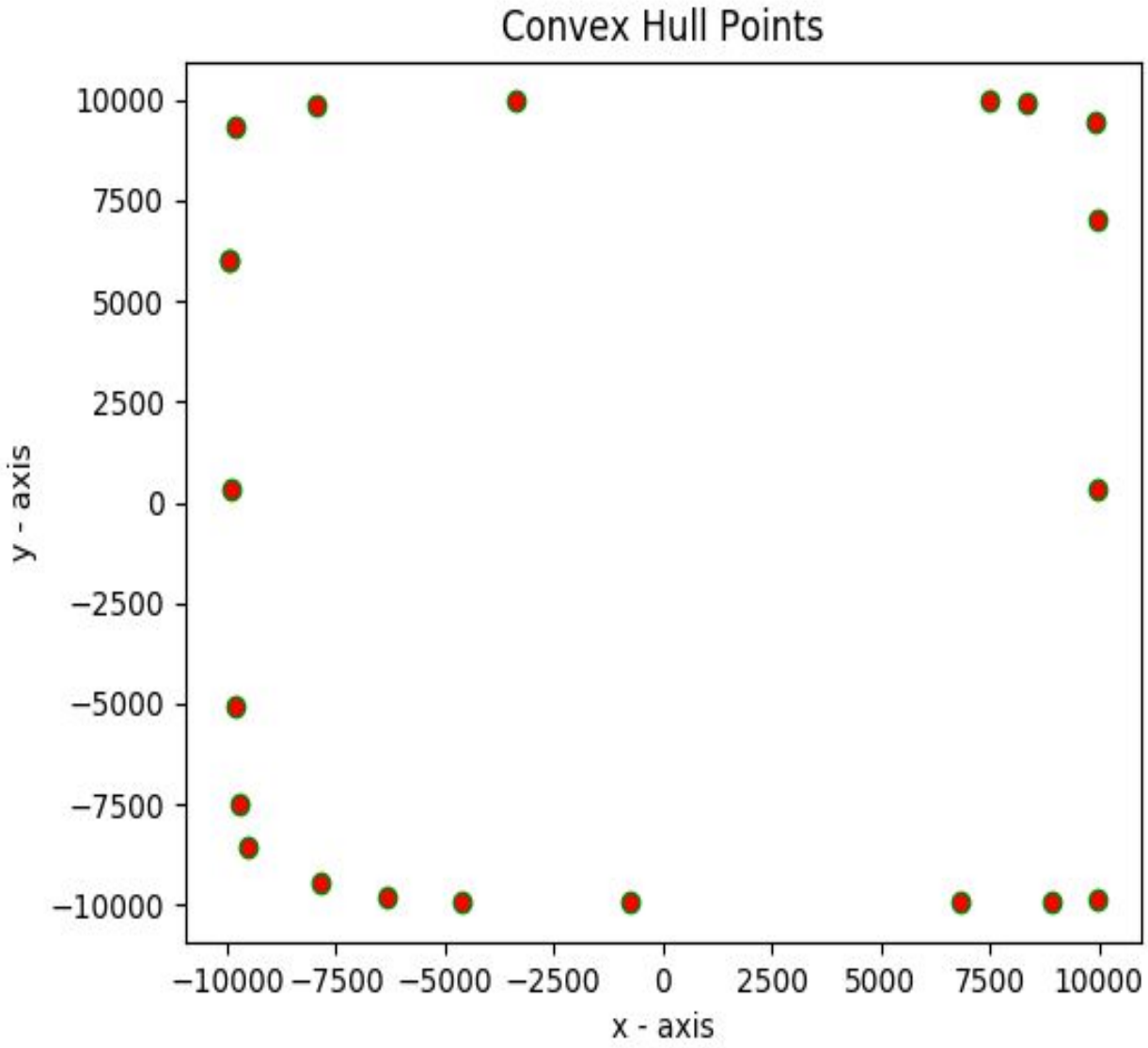
3. After pruning



This reduced 91% of points in this test after pruning.

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

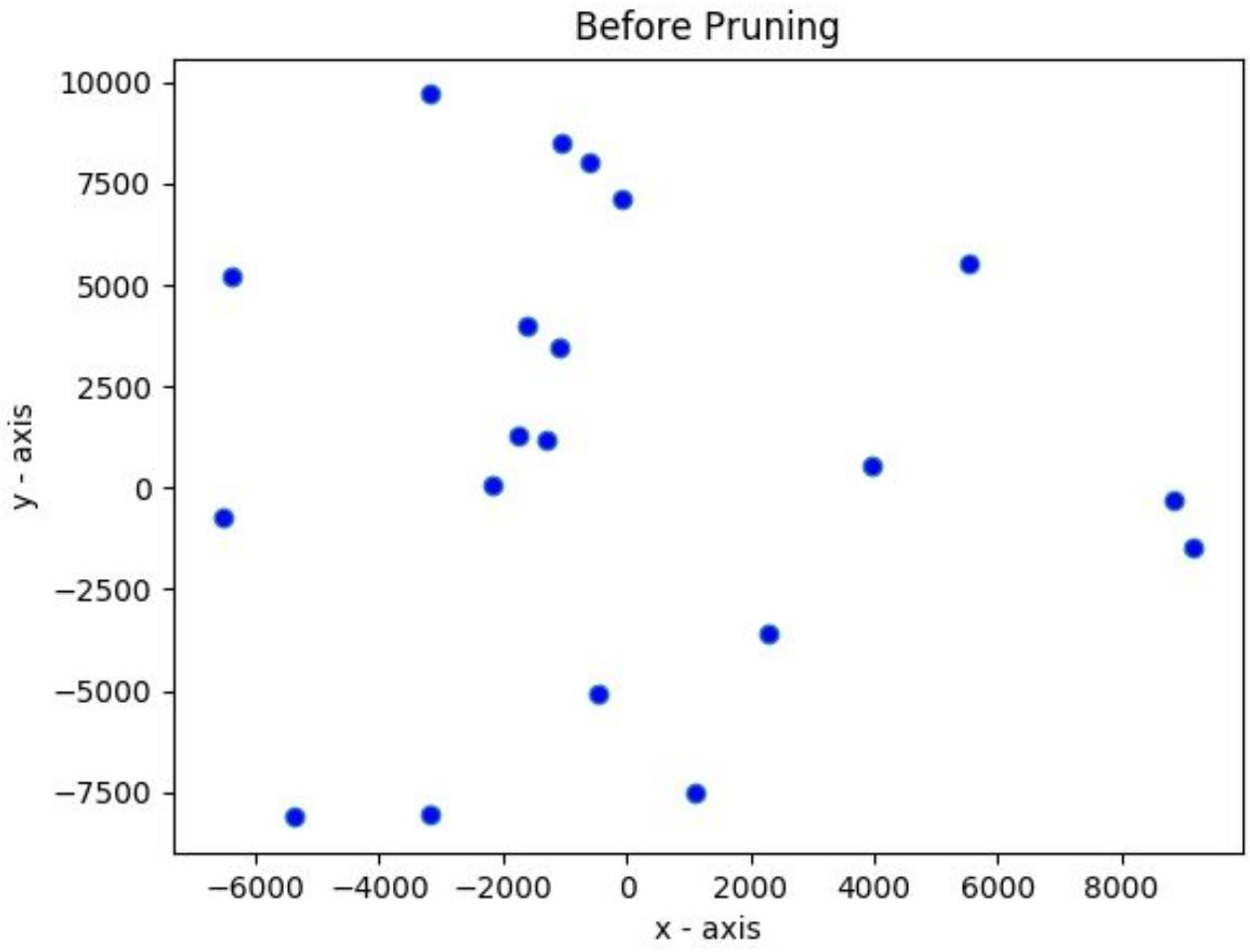
4. Final result



Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

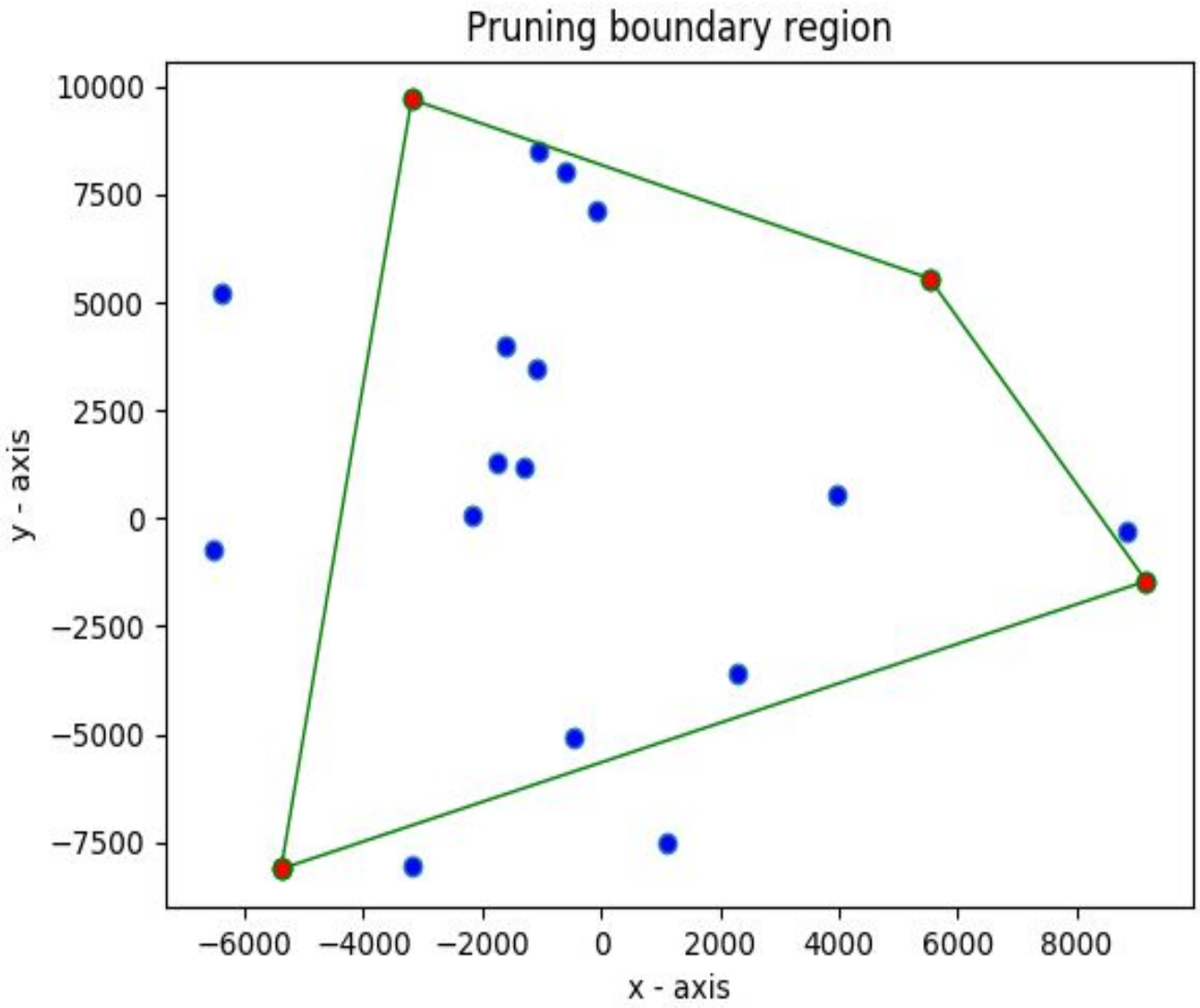
Test 2 :

1. Input set



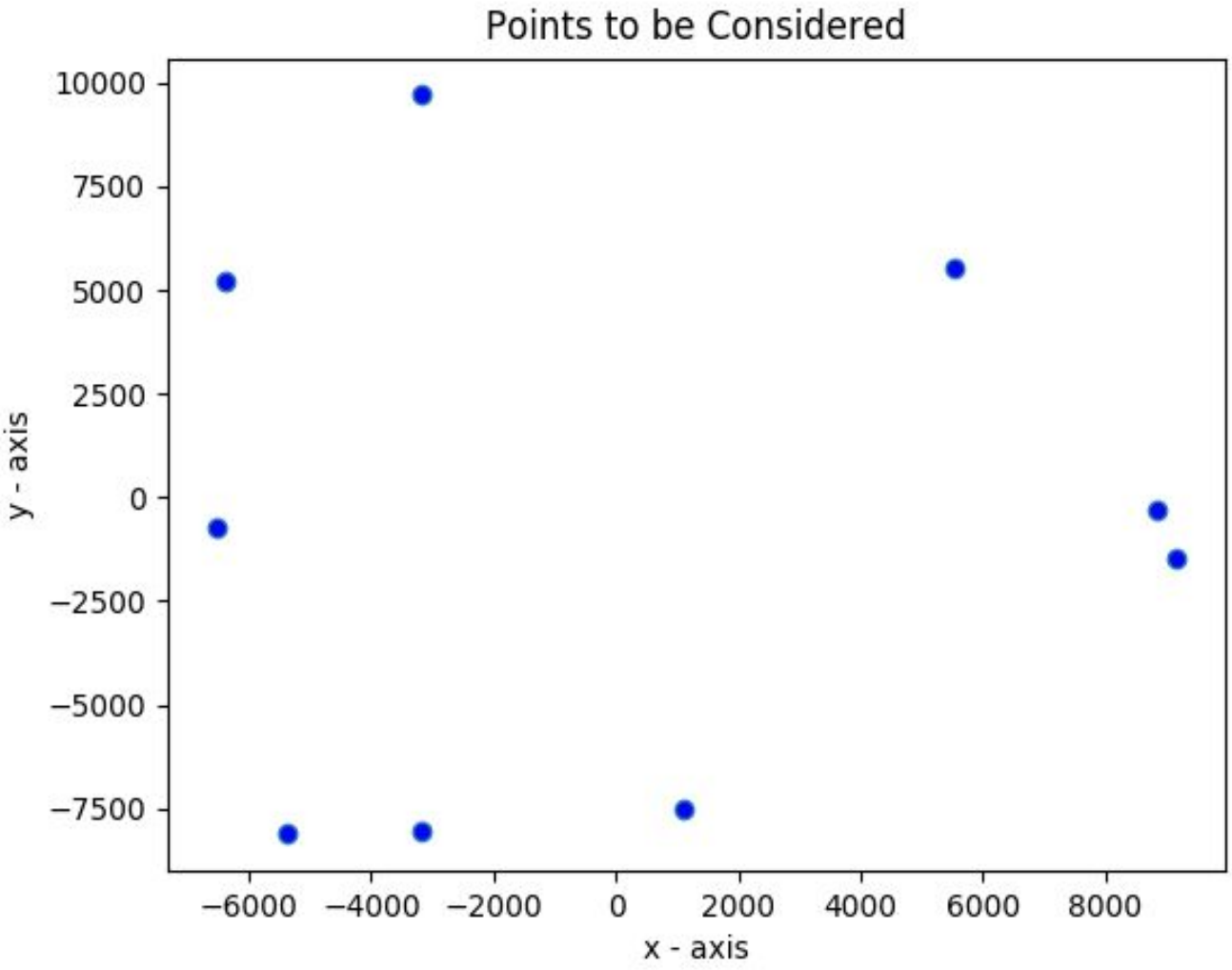
Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

2. Pruning Boundary region

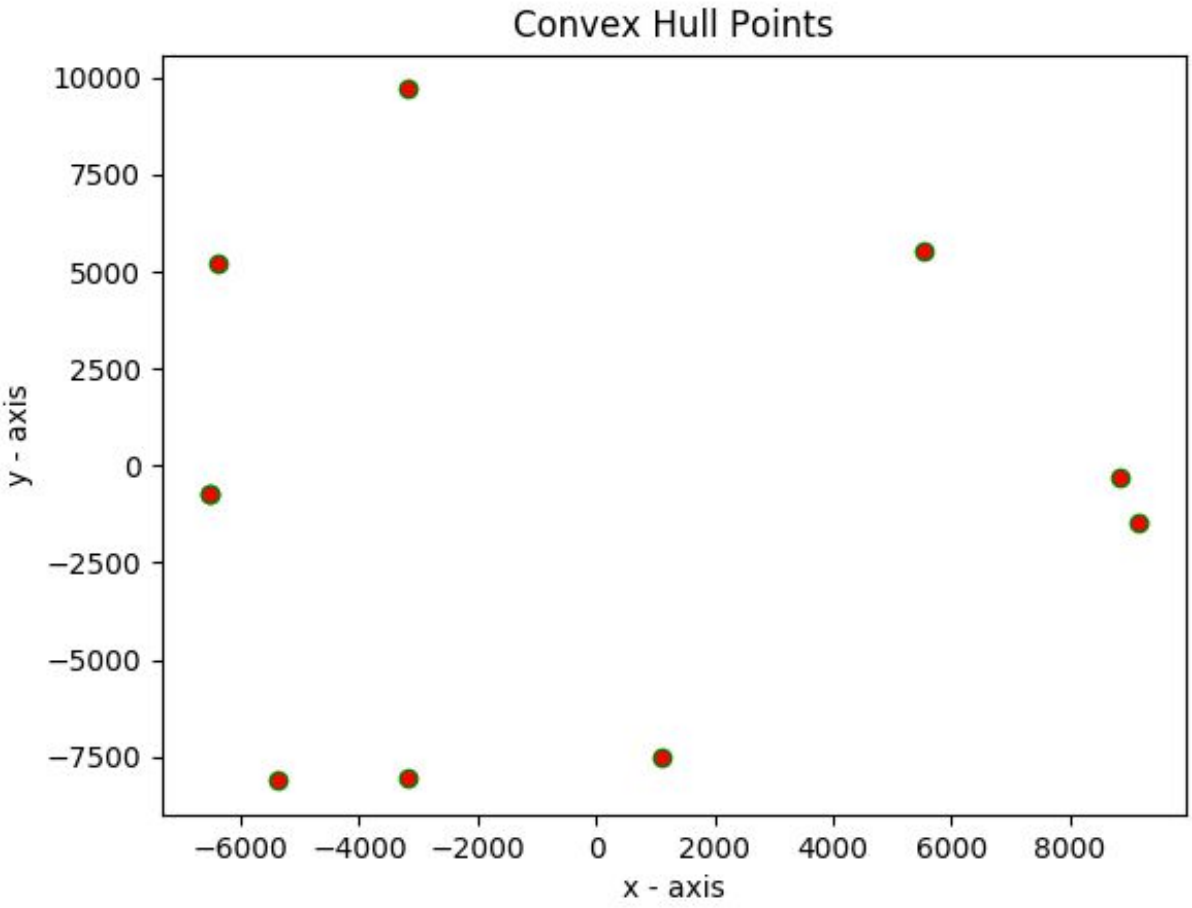


Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

3. After Pruning



4. Final Result



Even on the sparse set of points, it removes considerable amount of points. As seen above, the points left after pruning were part of the final result itself. Hence it is effective on scattered points as well.

Time Complexity of Pruning

The first step traverses the input set once - O(n)

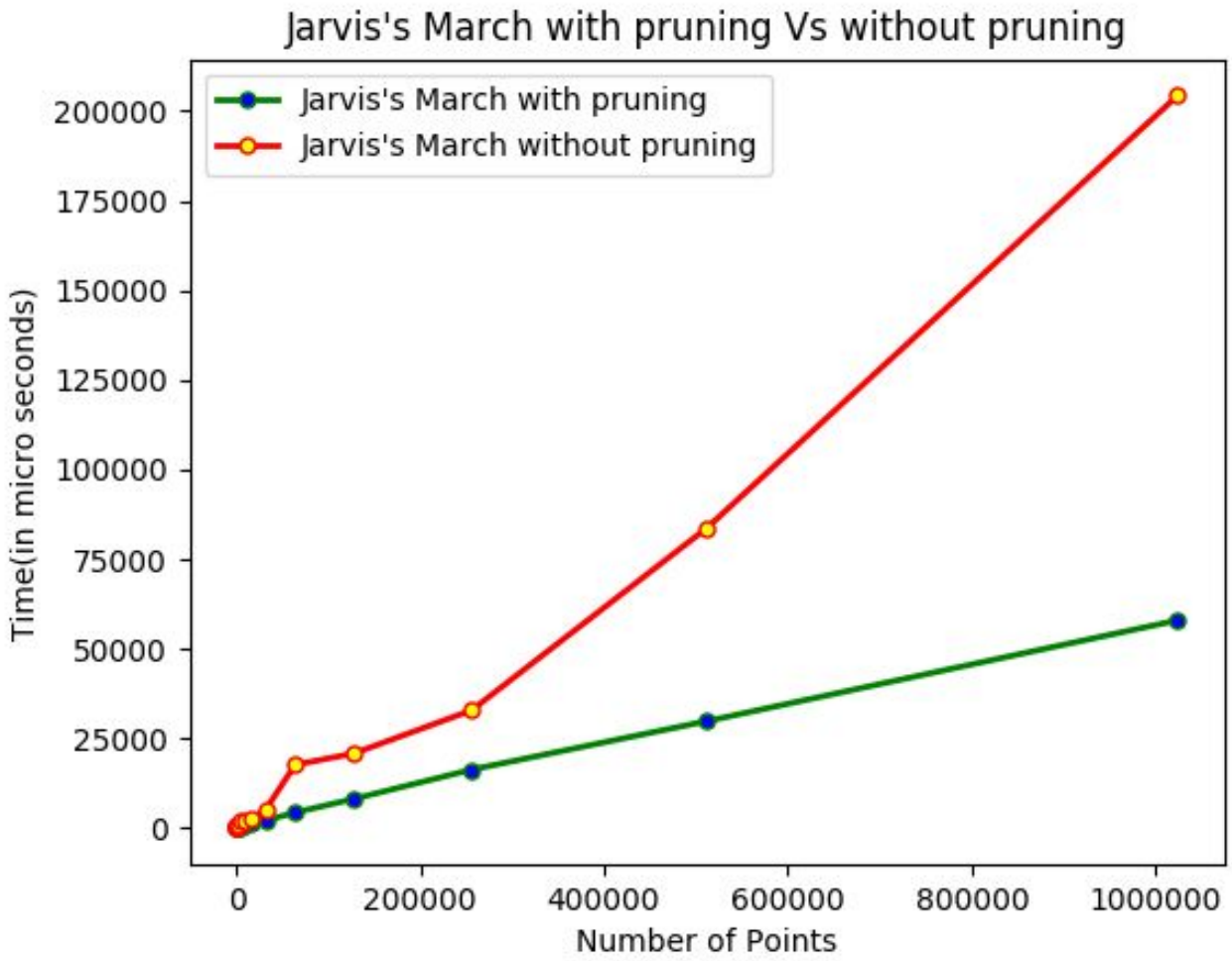
The second step traverses the input set once - O(n)

Total time complexity = O(n)

We compared the time taken with and without pruning step for all the three algorithms, there was drastic difference in the results obtained by both.

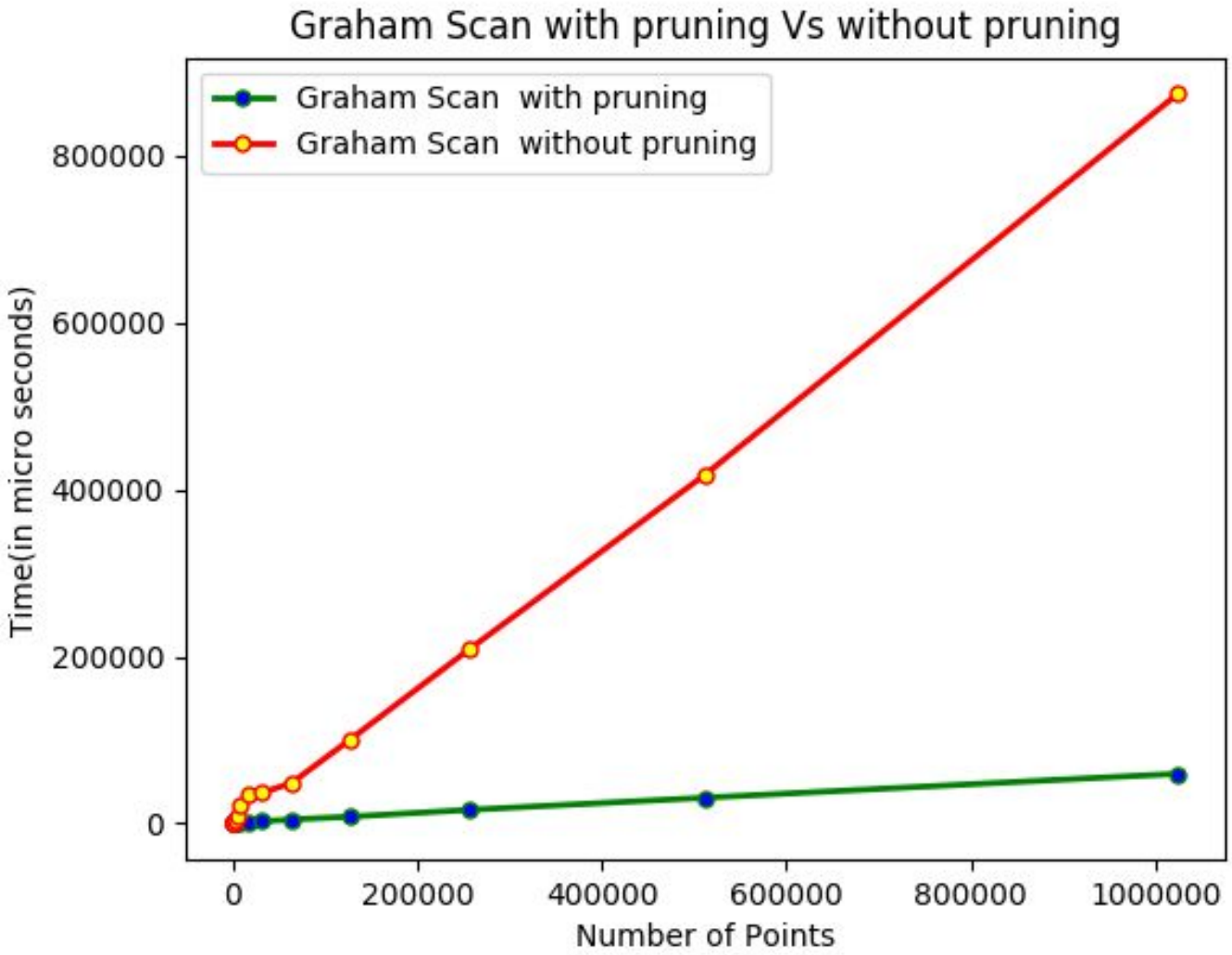
Performance of convex of algorithm with and without pruning

1. Jarvis's March



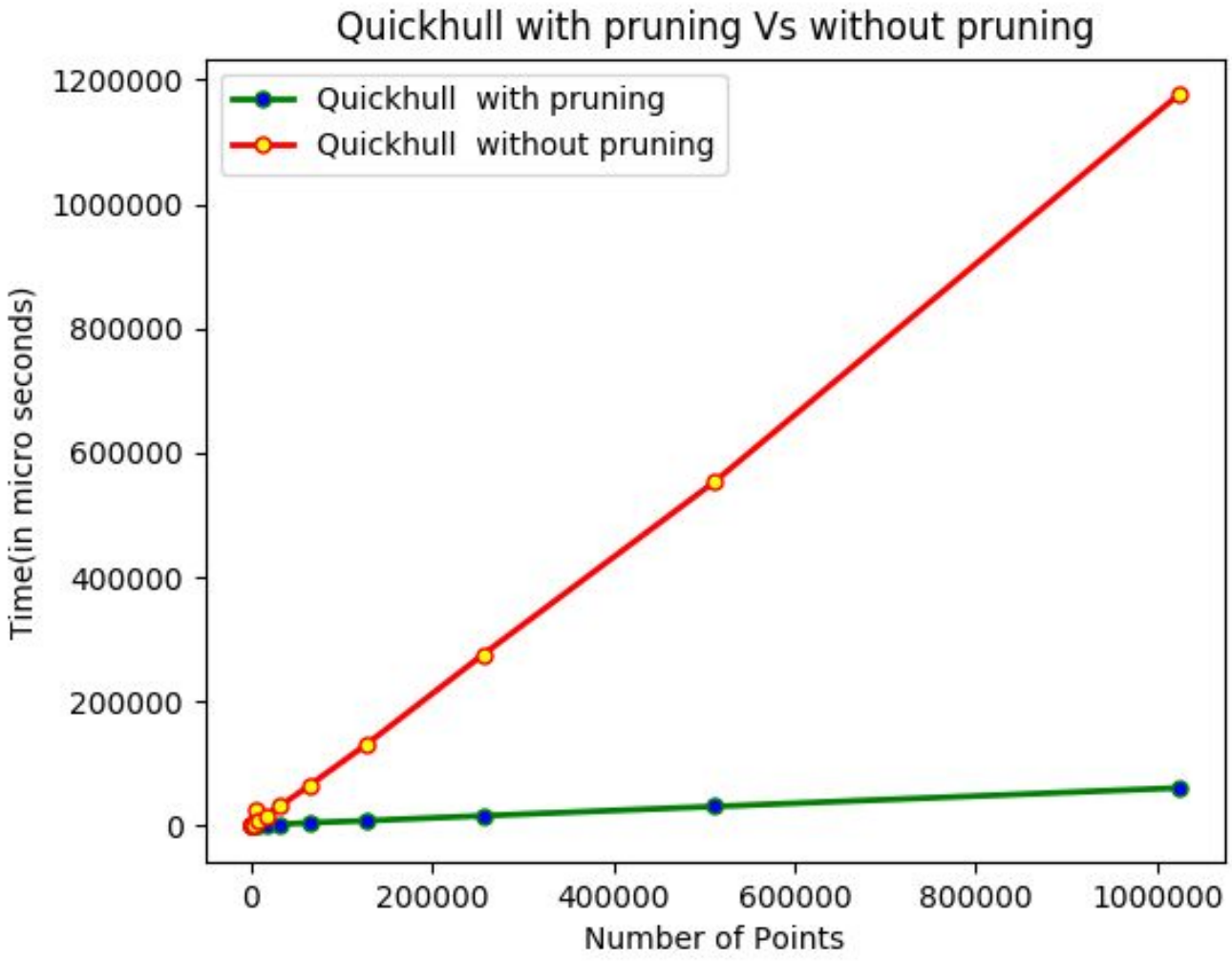
Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

2. Graham Scan



Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

3. Quickhull



Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Results

Number of Points	Graham Scan without Pruning (Time in microseconds)	Graham Scan with Pruning (Time in microseconds)	Performance Gain(%)
10	38	33	13.01
100	158	89	43.67
500	840	246	70.71
1000	1733	466	73.11
2000	3759	265	92.95
4000	8032	570	92.90
8000	20758	731	96.47
16000	34536	1412	95.91
32000	36690	2491	93.21
64000	48603	4698	90.33
128000	101186	8072	92.02
256000	208423	16207	92.22
512000	418022	30776	92.63
1024000	874097	59454	93.19

Vishal Bidawatka - 2018201004
Bhavin Kotak - 2018201071

Conclusion

We studied different algorithms namely Jarvis March, Graham Scan and Quick Hull for finding convex hull from a given set of points.We added an optimization in terms of pruning the input points. The most important metric is how many points are pruned, or what's the survival rate at the end of the second pass. This depends on the distribution of the points. For the input data the we generated, only 10% (approx) of the points were left after running the pruning. Any convex hull algorithm can be applied to the remaining points.

The effectiveness of pruning highly depends on the distribution of the input. It works well for a random distribution, but it might not for others. Eventually, if no points inside quadrilateral (ABCD) are found, the algorithm cannot prune anything. In this unfortunate case, not only does the algorithm not introduce any advantage in terms of size, but it even worsen the performance because the time to run the pruning has to be added to the total time. The good thing is that even in the worst case, it cannot increase the computational complexity, since O(n) + O(n log n) is still O(n log n). And anyway the constant factor is so low that can be almost neglected.

The pruning algorithm can be used in conjunction with *any* convex hull algorithm to dramatically reduce the size of the input data, and consequently the running time.

References

1. https://en.wikipedia.org/wiki/Convex_hull
2. https://www.youtube.com/watch?v=VP9ylElm1yY
3. http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf
4. http://cs.indstate.edu/~jtalasila/convexhull.pdf