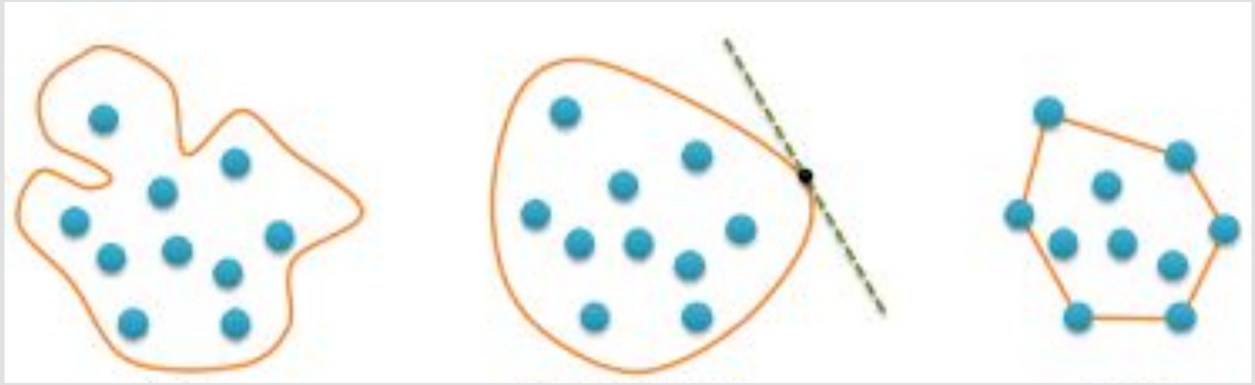
IT300: Design and Analysis of Algorithms

Convex Hull and its Applications

Ajay Bharadwaj (7619359184) - 181IT103
Udbhav Bisarya (9743032120) - 181IT150
Deep Dhanuka (9967939403) - 181IT214
Vishwas Parekh (+971503672455) - 181IT252

What is a Convex Hull:

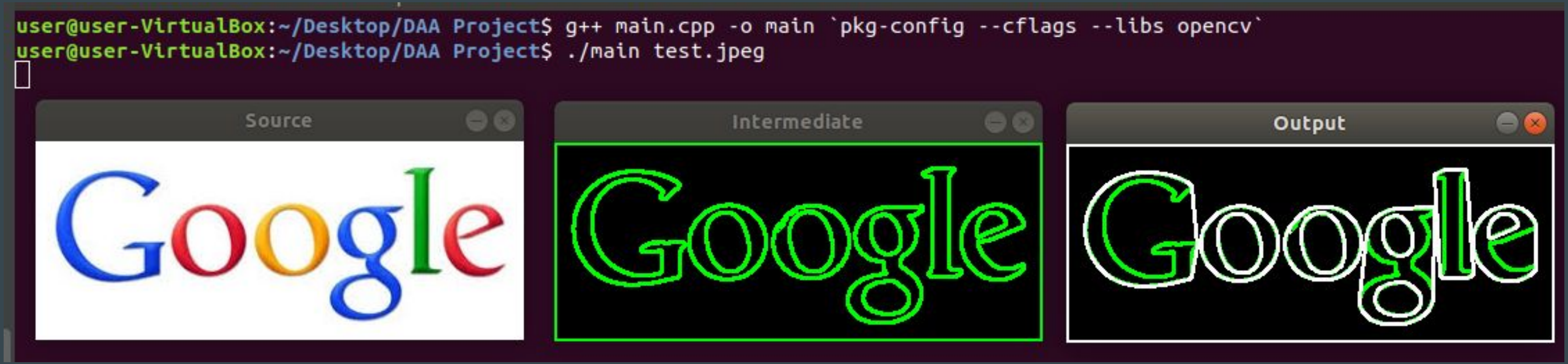
The convex hull of a set of points (In a 2D plane) is the subset of points that forms the smallest convex polygon that contains the rest of the points in it.



Convex Hull Application (Outlining Program)

1. Find the points on the edges of "objects" (Based on colour) in the input image
   a. More optimal than running the convex hull algorithm on all the points of an "object"
2. For each "object":
   a. Run the convex hull algorithm on these "edge points"
   b. Draw the convex hull over the image

The convex hull algorithm used here is Sklansky's algorithm.



```
user@user-VirtualBox:~/Desktop/DAA Project$ g++ main.cpp -o main `pkg-config --cflags --libs opencv`
user@user-VirtualBox:~/Desktop/DAA Project$ ./main test.jpeg
```

How to Find the Convex Hull?

1. Divide and Conquer method
2. Sklansky's algorithm
3. Graham Scan algorithm
4. Quickhull algorithm
5. Jarvis' algorithm

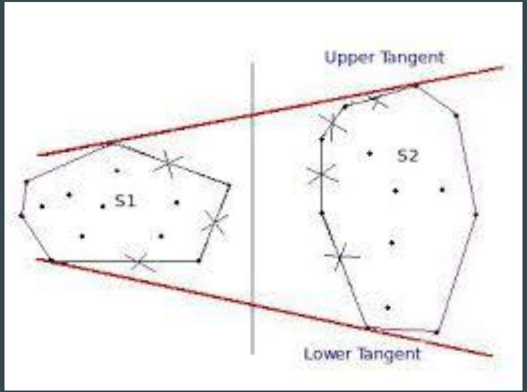
Divide and Conquer Method

Base case: When there are only 3 points (It is easier to call the base case when there are 5 or fewer points and naively finding the convex hull).

Divide stage: Split the points into 2 halves - Left half and Right half [Based on the X coordinates].

Conquer/Merge stage: Find the endpoints of the upper and lower tangents of the Left and Right convex hulls. Use these endpoints to discard points on the “interior” (Refer to the image on the next slide). This happens in O(n) where n is the number of points.

Divide and Conquer Method



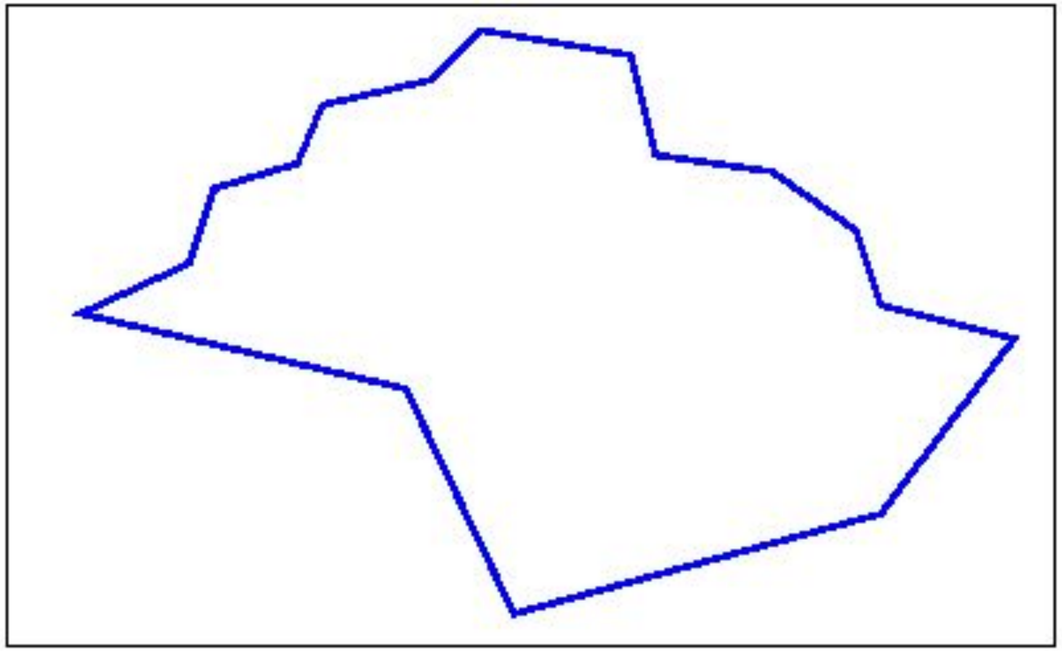
Time Complexity:

T(n) = 2*T(n/2) + O(n)

O(n*log(n))

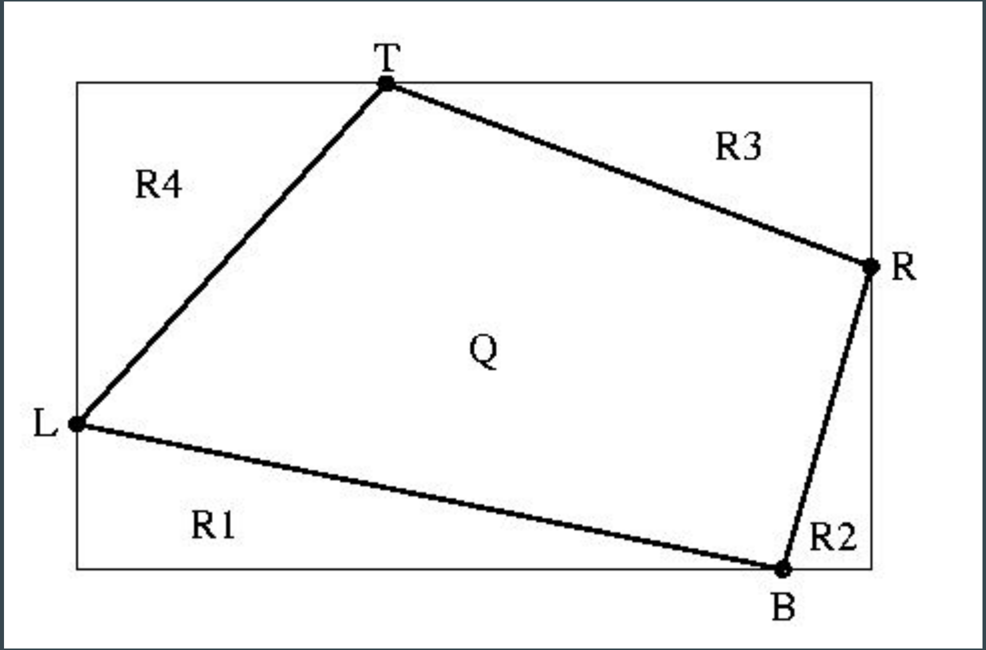
[Where n is the number of points]

Sklansky Algorithm


Maximal Polygon

Sklansky Algorithm - Step 1 and Step 2

- Step 1:- Find the vertices B,R,T and L
- Step 2:- Find the square circumscribing these points. The square will have 4 triangular regions

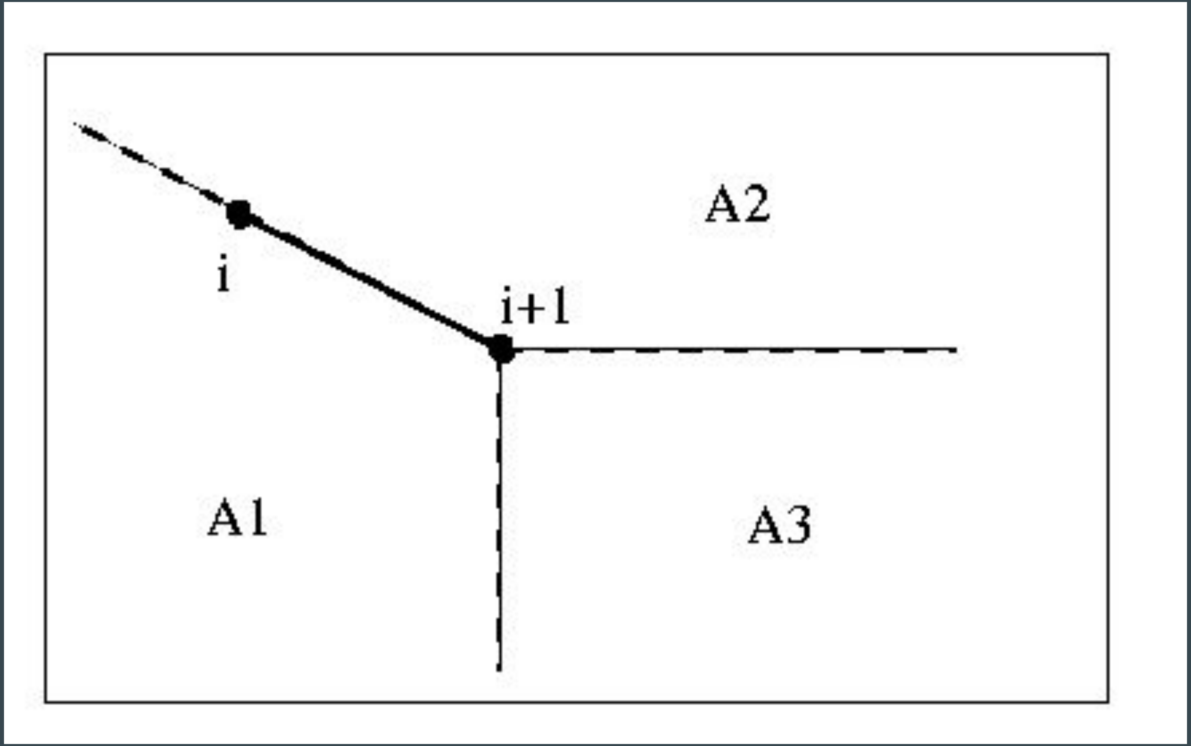


Sklansky Algorithm - Steps 3,4 and 5

- Considering R_1 ,
- $V_1 = V_L$ is included in convex hull
- V_2 is included in convex hull if it belongs to R_1
- For remaining vertices,
  - If I+2 is is not in R_1, discard I+2.
  - 2) Else if I+2 is in A3, retain I+2 and re-iterate.
  - 3) Else if (I+2 is in A2) OR (I+2 is in A1 AND I+2 is above I+1 AND a vertex was discarded due to line 3 on the immediately preceding iteration), discard I+2.
  - 4) Else if I+2 is in A1, discard I+1.

This should be done till V_B is reached

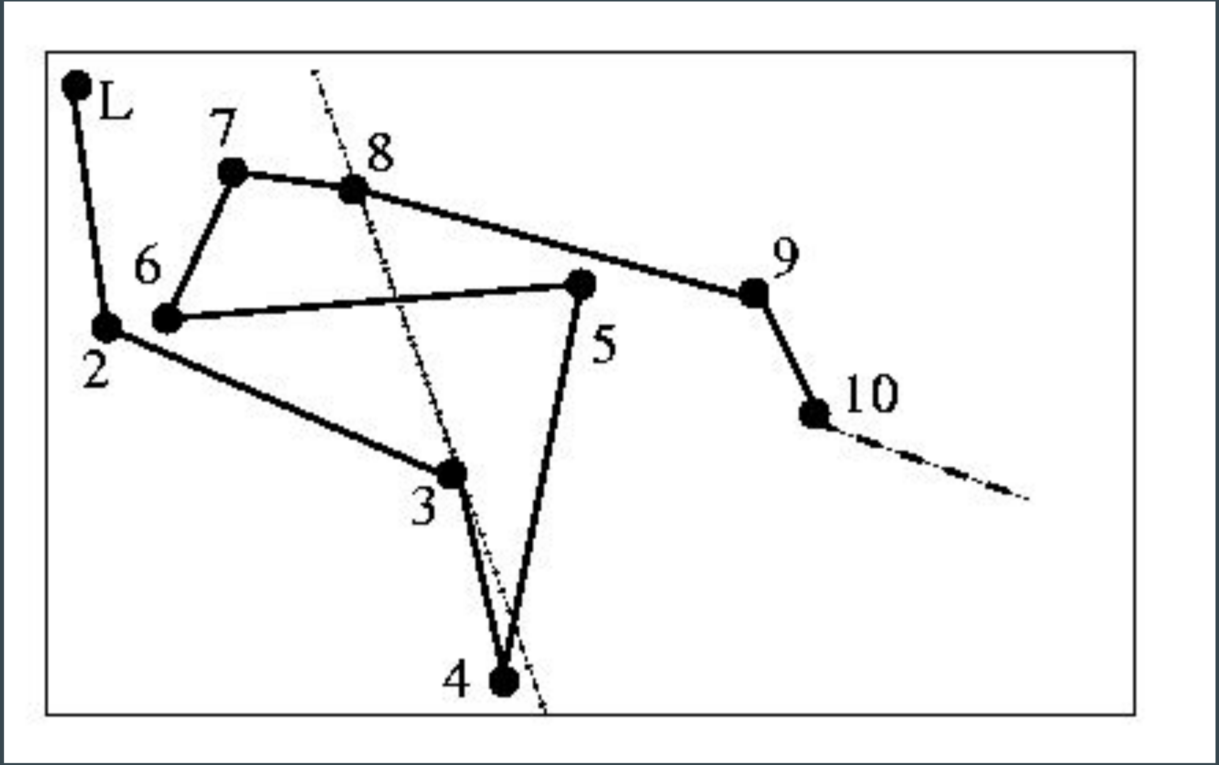
Sklansky Algorithm - Step 3,4 and 5



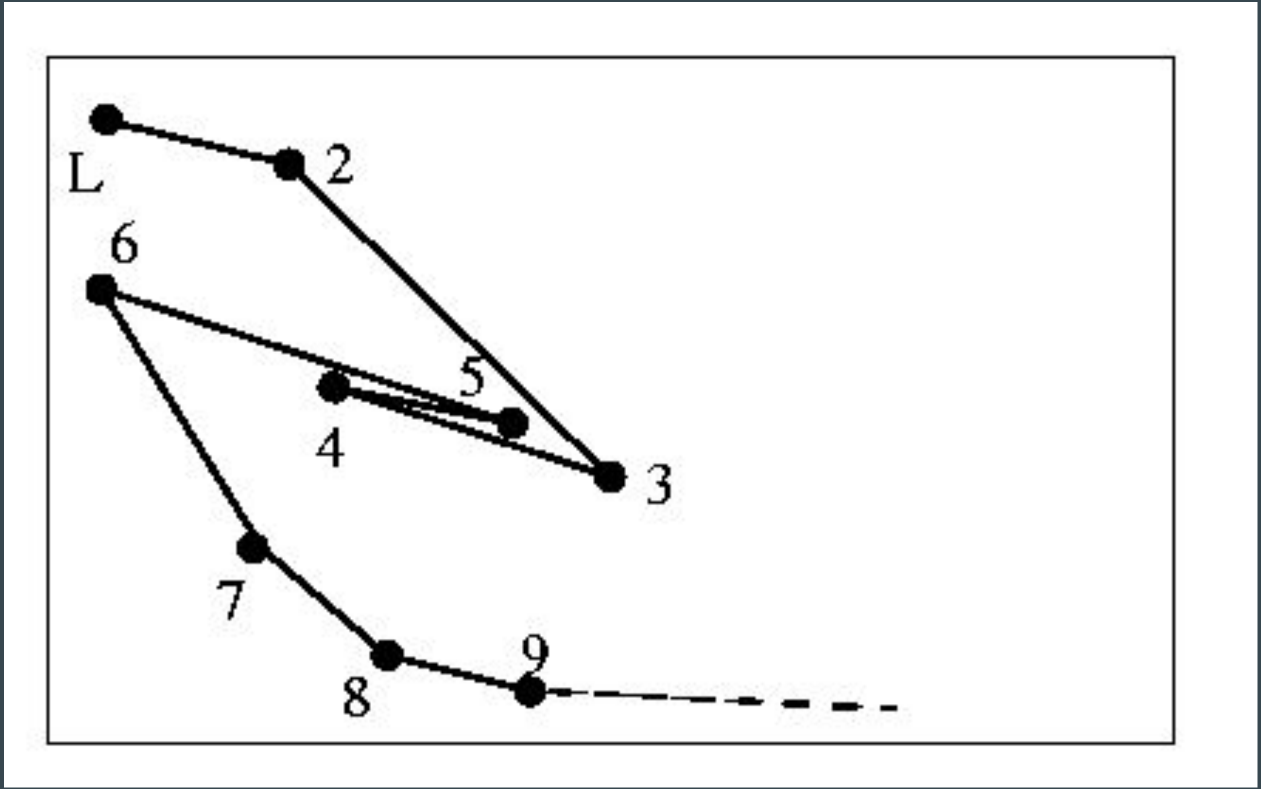
Sklansky Algorithm - Step 6

- After getting some set of vertices from the above steps, we apply the initial algorithm proposed by sklansky(1972)
- For a vertex V_i ,
  - $S_i = (x_{i+1} - x_{i-1})*(y_{i-1} - y_i) + (y_{i+1} - u_{i-1})*(x_i - x_{i-1})$
  - if($S_i < 0$) - V_i is a convex vertex

Counter-Examples to Sklansky Algorithm



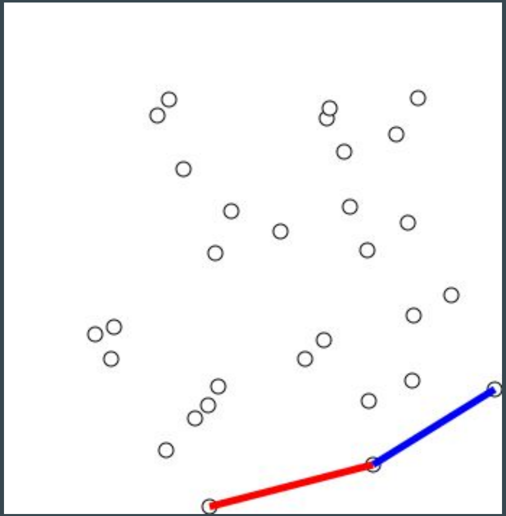
Counter-Examples to Sklansky Algorithm



Graham Scan Algorithm

Graham Scan Algorithm can be divided in two phases:

1. Sort Points
2. Accept or Reject Points

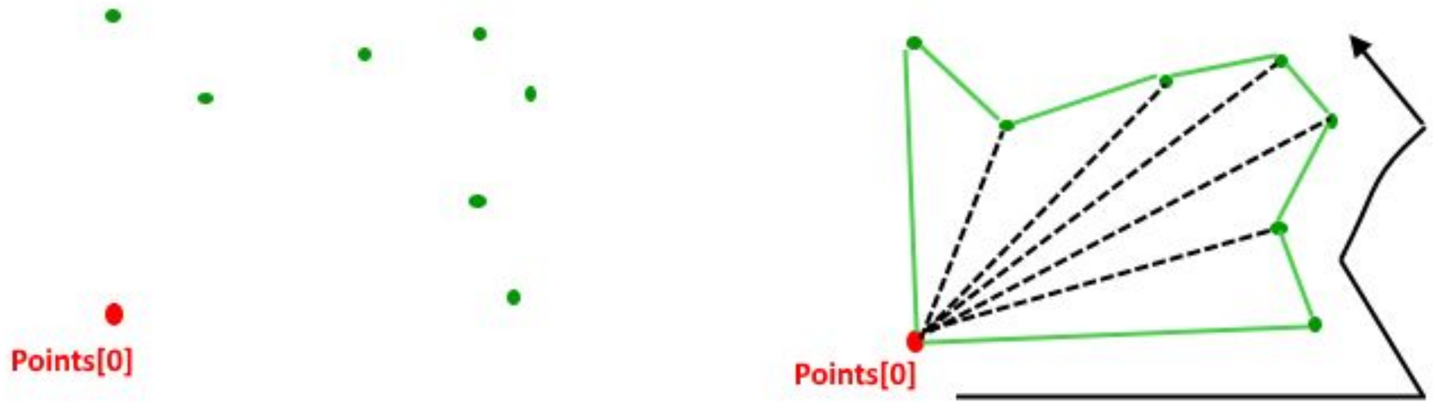


Graham Scan Algorithm

Phase 1:

- Find the bottommost and leftmost point amongst the given points by performing a linear search on the list of n points.
- Sort the rest of n-1 points based on their angles w.r.t pt0 in counter clockwise direction and their distance from pt0.
- Delete elements from this sorted list having same angle with pt0 except the farthest one.
    - Let m be the new size of the array after phase 1.
    - If m<3, return no convex hull present.

Graham Scan Algorithm



Given points

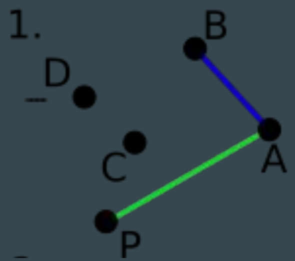
Points consider according to increasing angle with respect to points[0] from a simple closed path

Graham Scan Algorithm

Phase 2:

- Create an empty stack and push first three elements in it.
- Iterate over rest of the points from i=3 to m:
    - While second to the top element, the top element of the stack and point i :
        - Make right turn: pop the top element of the stack and continue
        - Make left turn: break the loop
    - Push the point i in the stack.
- Print the points remaining in the stack on terminal which represent the convex hull.

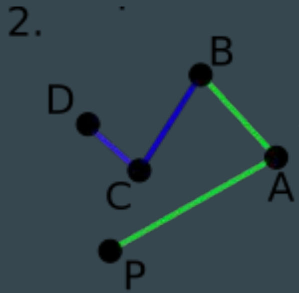
Graham Scan Algorithm



Stack: top => B A P]

Point[i] = C

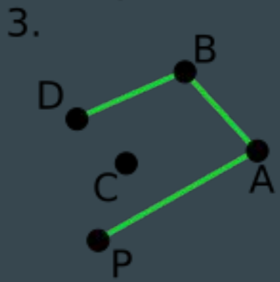
Left Turn(counter clockwise)



Stack: top => C B A P]

Point [i] = D

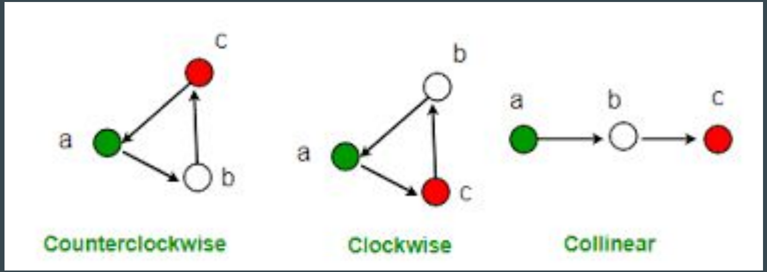
Right turn(clockwise)



Stack: top=> D B A P]

Pop C from the stack and push D

Graham Scan Algorithm



Using the slopes of the line segments formed by a-b and b-c the formula to determine the orientation for a triplet of points are:

0 : collinear

(y2 - y1)*(x3 - x2) - (y3 - y2)*(x2 - x1) = >0 : clockwise

<0 : counterclockwise

Where, x(i) and y(i) are coordinates for each point.

Graham Scan Algorithm

Time Complexity

- Step 1 : O(n)
- Step 2: O(n*log(n))
- Step 3: O(n)
- Step 5: O(n)
- Step 7: O(n)

The overall time complexity of Graham Scan algorithm is:

O(n) + O(n*log(n)) + O(n) + O(n) + O(n) ~ **O(n*log(n))**

where, n is the number of points

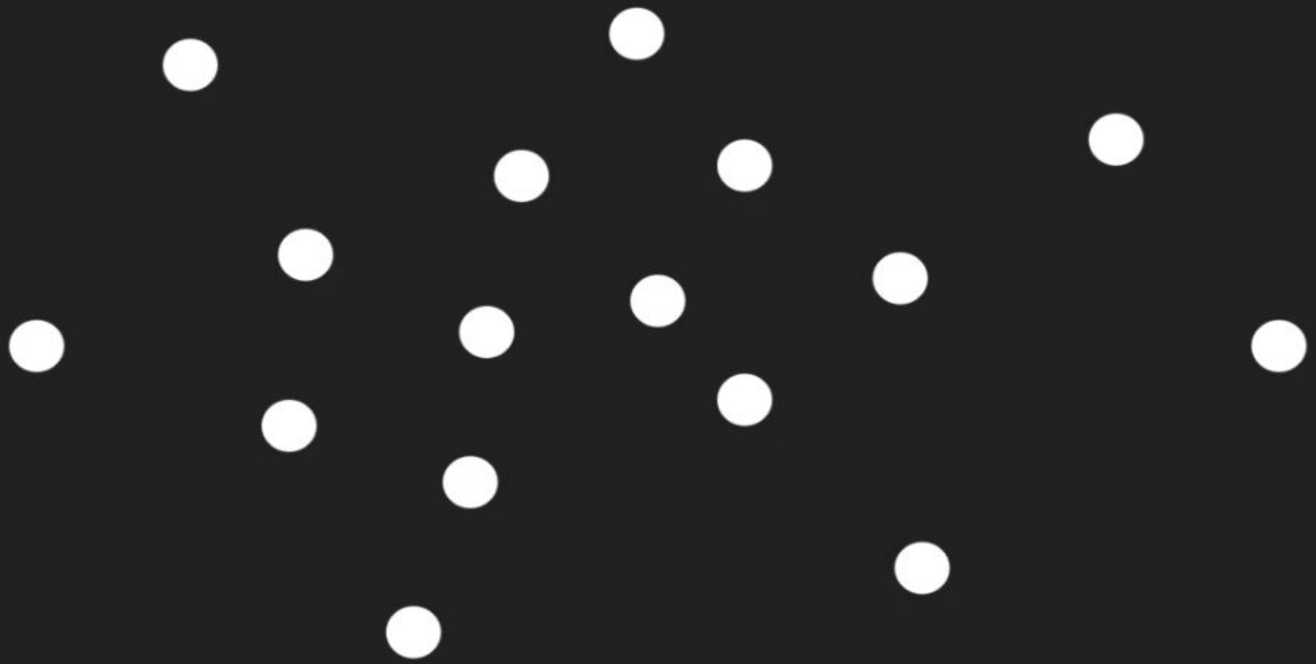
Quickhull Algorithm

This algorithm is similar in sense to the quicksort. On average, we get time complexity as O(n Log n), but in worst case, it can become O(n^2).

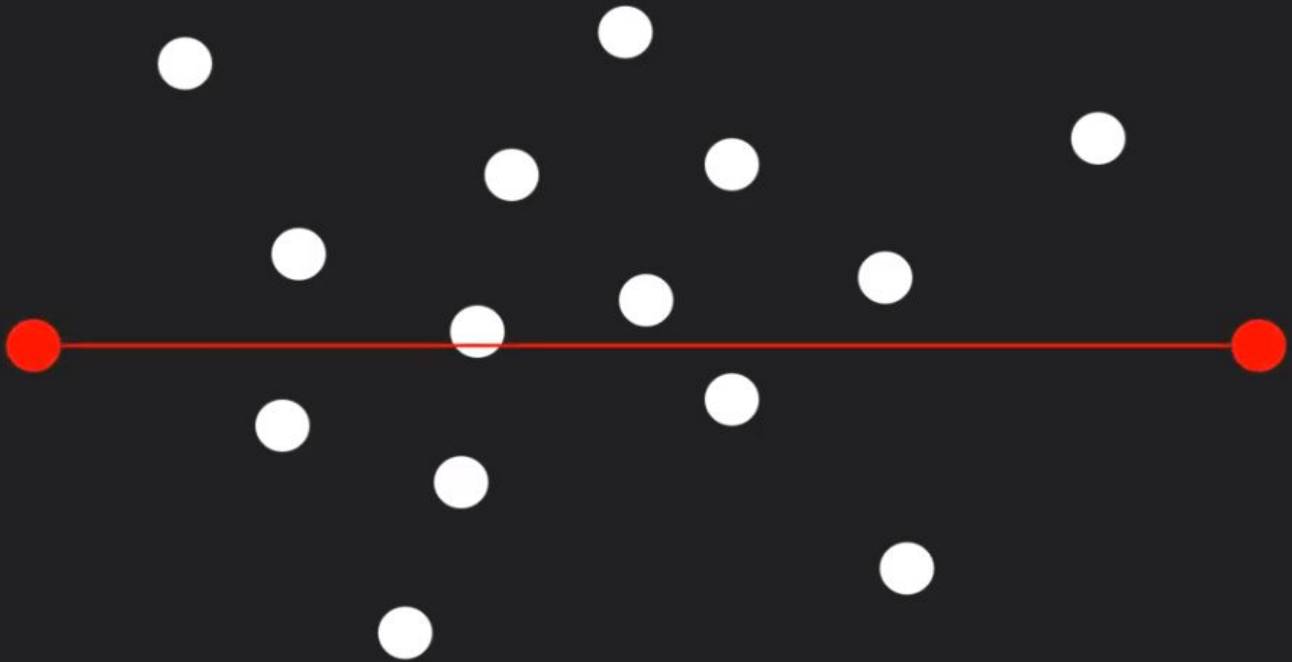
1. Find the point with minimum x-coordinate min_x and similarly the point with maximum x-coordinate max_x.
2. Make a line joining these two points, say L. This line will divide the whole set into two parts. Take both the parts one by one and proceed further.
3. For a part, find the point P with maximum distance from the line L. P forms a triangle with the points min_x, max_x. It is clear that the points residing inside this triangle can never be the part of convex hull.
4. The above step divides the problem into two sub-problems (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. Repeat point no. 3 till there no point left with the line. Add the end points of this point to the convex hull.

The worst case is when all the points are a part of the convex hull.

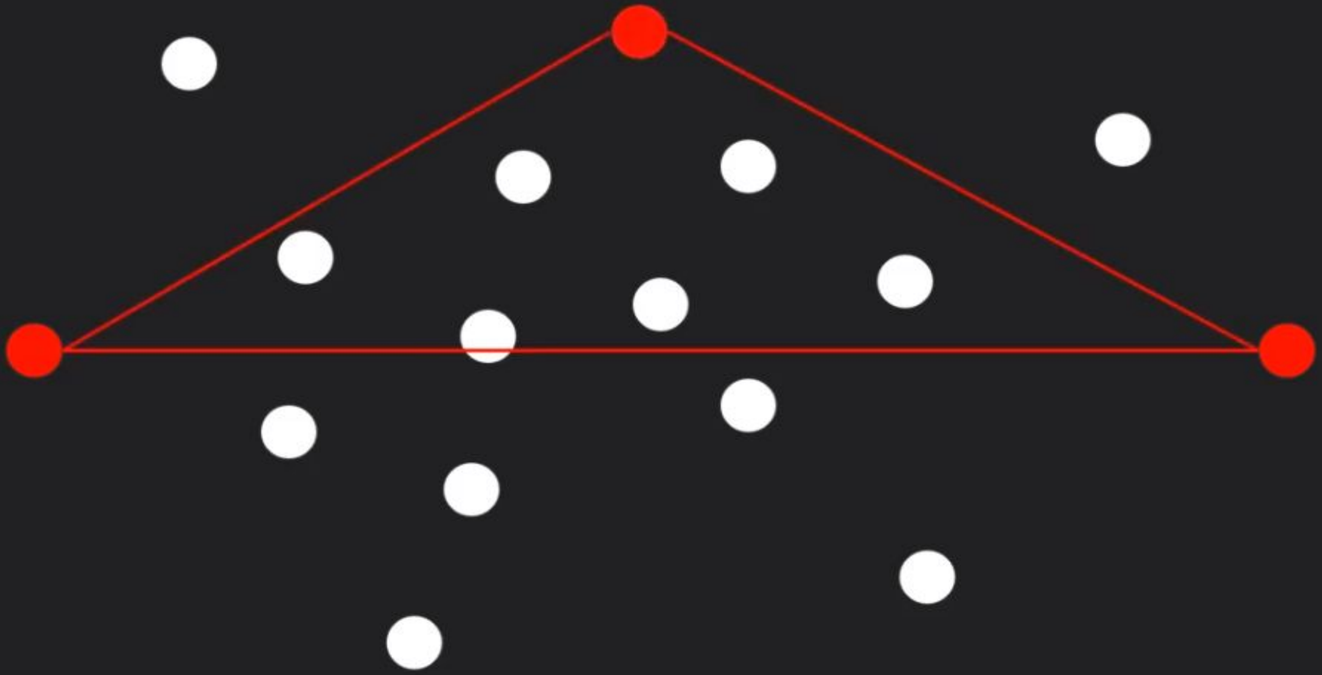
Quickhull: Divide and Conquer



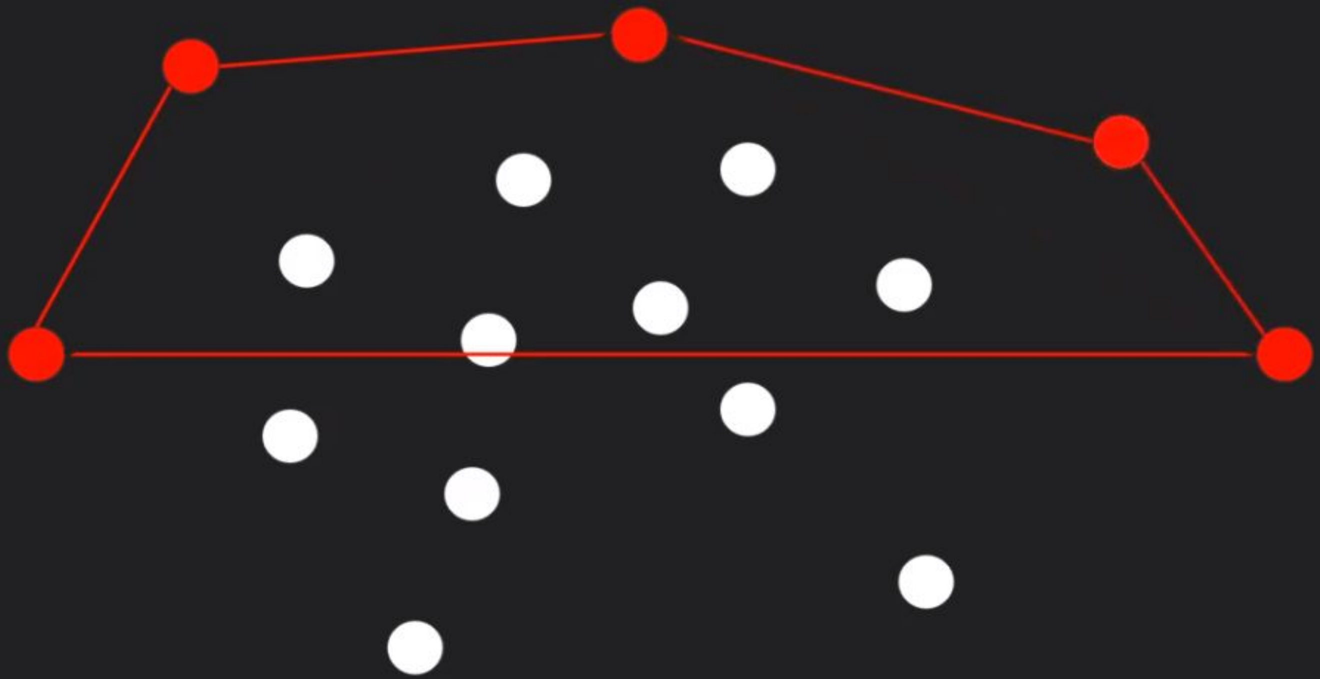
Quickhull: Divide and Conquer



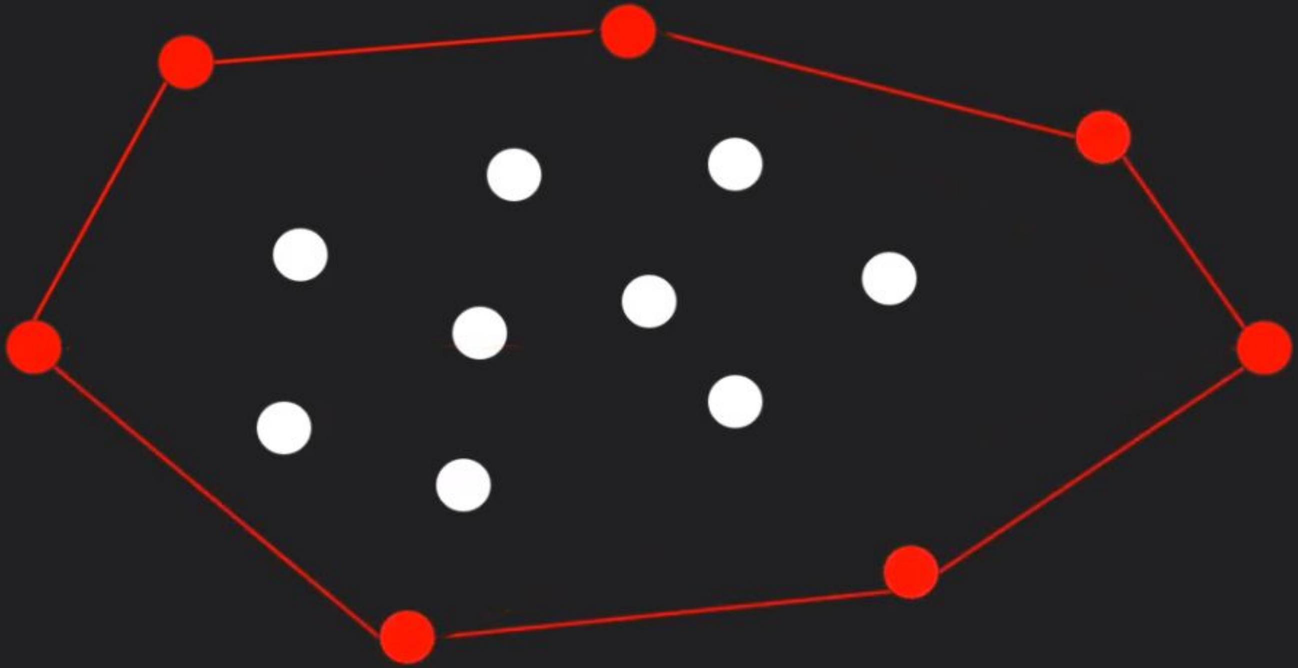
Quickhull: Divide and Conquer

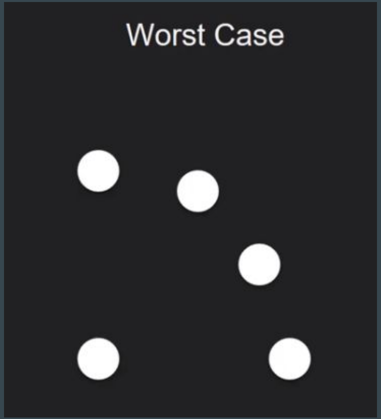
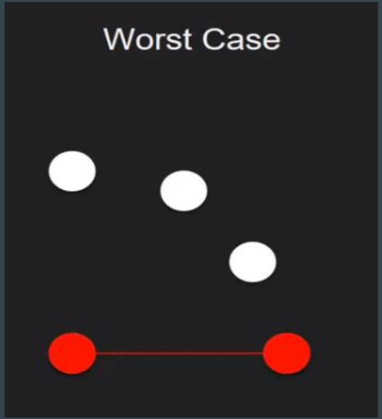
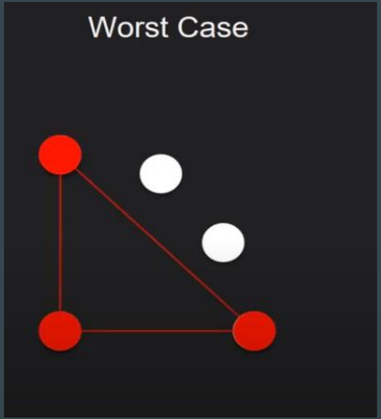
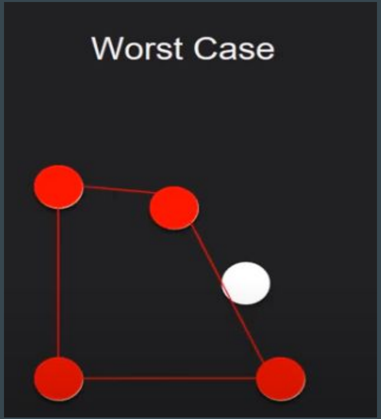
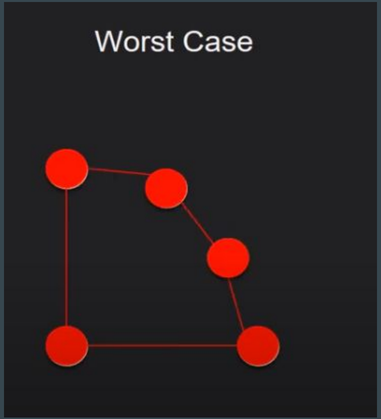


Quickhull: Divide and Conquer



Quickhull: Divide and Conquer




Worst Case

Worst Case

Worst Case

Worst Case

Worst Case

Time complexity of the quickhull algorithm

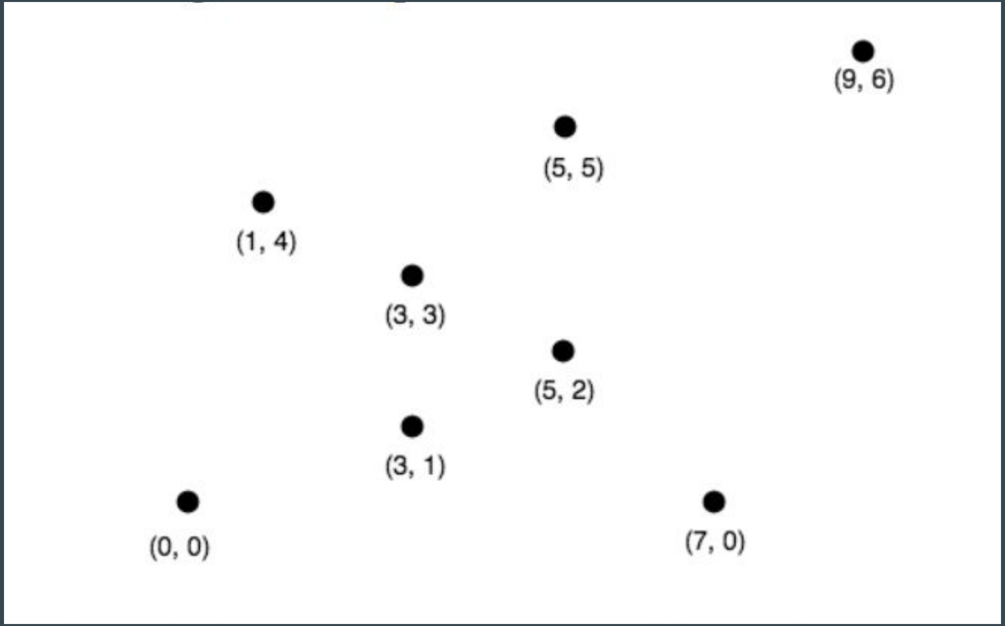
The average case gives the recurrence relation of T(n)=2*T(n/2)+O(n),this it the famous recurrence relation which is the same as O(n log n).

The worst case gives us the relation T(n)=T(n-1)+O(n), which gives us the complexity of O(n^2)

The average case occurs when we are able to split the subproblem in the cases where the triangle contains several inner points that can be neglected in calculations for the next point in the convex hull.

Jarvis algorithm (a.k.a Gift Wrapping Algorithm)

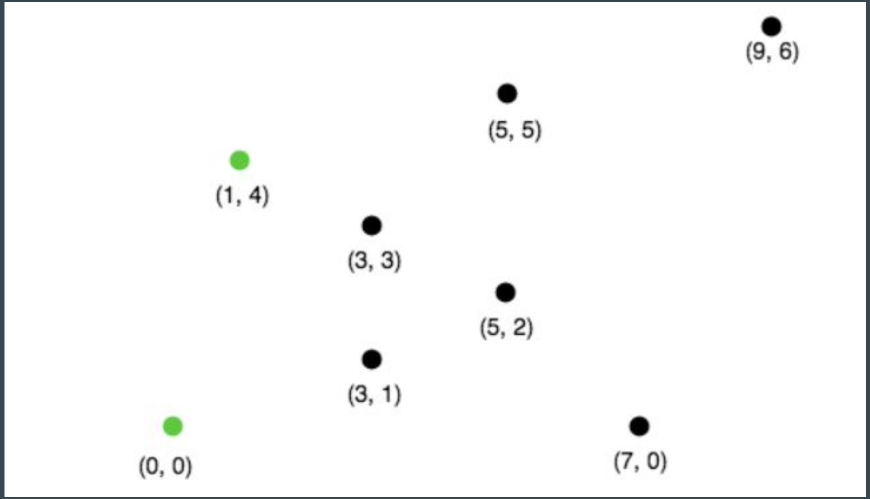
Jarvis’s march algorithm uses a process called gift wrapping to find the convex hull. It is one of the simplest algorithms for computing convex hull. The working of Jarvis’s march resembles the working of selection sort. In selection sort, in each pass, we find the smallest number and add it to the sorted list. Similarly, in Jarvis’s march, we find the leftmost point(This point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have “orientation(p, q, r) = counterclockwise”, as we are wrapping it in that direction, it could have been clockwise also but then we would have to consistently use clockwise direction to search for each point, we cannot use then alternatively or randomly) and add it to the convex hull vertices in each pass. The algorithm ends when we reach back to the start point.

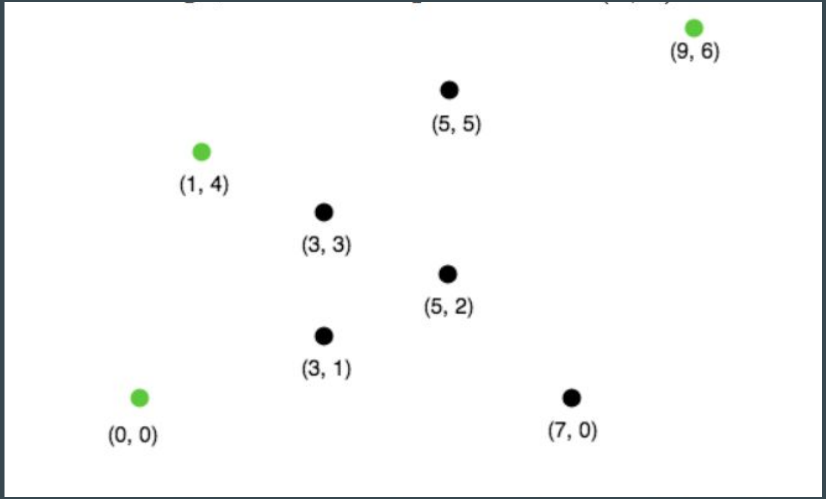


Step 1: Select the leftmost point (In this case the point (0,0) is selected as the leftmost point)

Step 2: If (P,I,Q) turns left, continue with same Q value, if (P,I,Q) turns right change Q value to I value, If (P,I,Q) are collinear then change Q to I if distance between I and P is greater than distance between Q and P

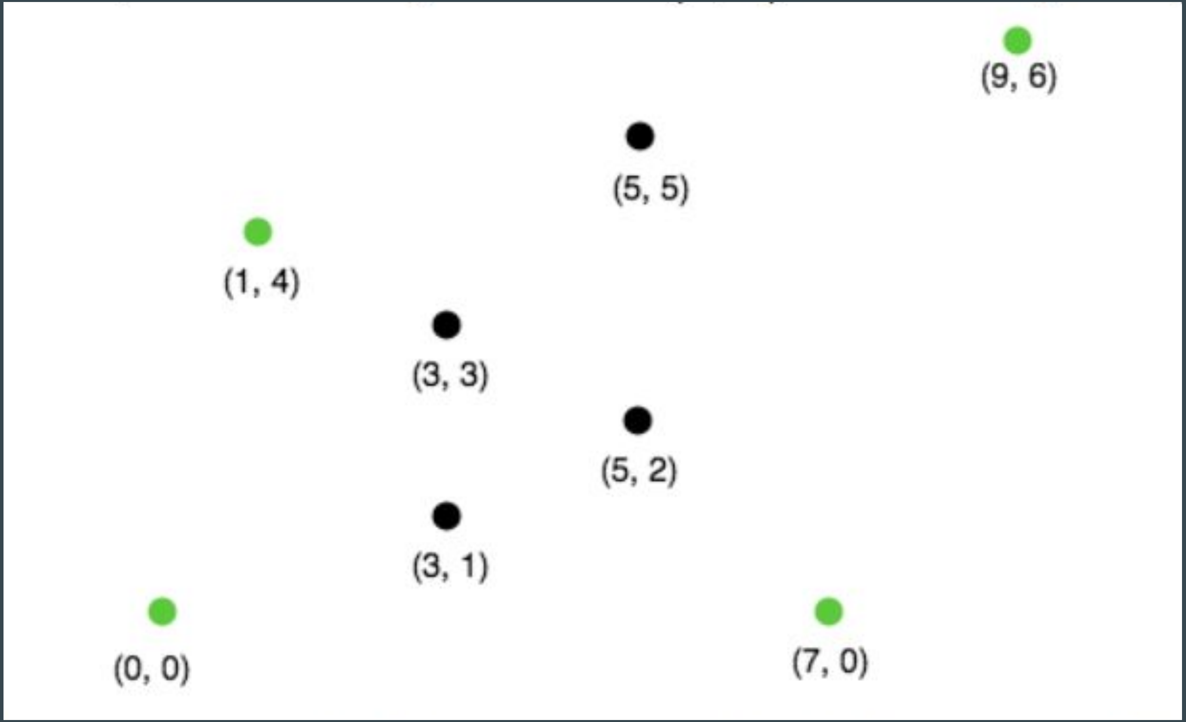
1. Next we find the leftmost point from point l=(0,0) The step by step process of finding the leftmost point from P=(0,0) is as follows.
2. We pick a point following P and call it Q. Let Q be the point (3,3)(We can pick any point, but in the code we generally pick the next point in the array)
3. Let all other points except P and Q be I. Now we check whether the sequence of points (P,I,Q) turns right. If it turns right, we replace Q by I and repeat the same process for remaining points.
4. Let I=(7,0). The sequence ((0, 0), (7, 0), (3, 3)) turns left. Since we only care about right turn, we don't do anything in this case and simply move on.
5. Next I=(5,5). The sequence ((0, 0), (5, 2), (3, 3)) is collinear. In the case of collinear, we replace Q with I only if distance between P and I is greater than distance between Q and L. In this case the distance between (0,0) and (5,5) is greater than the distance between (0,0) and (3,3) we replace Q with point (5,5).
6. Let next I=(1,4). The sequence ((0, 0), (1, 4), (5, 5)) turns right. We replace Q by point (1,4).
7. Finally Q=(1,4) is the leftmost point.
8. We add point (1,4) to the convex hull.





After the first 2 steps, we have point (0,0) and (1,4) as part of the convex hull. We then repeat step 2 with P=(1,4) as it was the latest point to be added to the hull

The next point added is (9,6), we repeat all steps in step 2 with P=(9,6)



The next point added is (7,0) and on repeating the process we get point(0,0) as the next value, and once we reach there we have successfully completed the algorithm.

Complexity of Jarvis March Algorithm

Finding the complexity here is pretty straight forward, as in each iteration of the algorithm, we go through all ‘n’ points, and the number of iterations is the number of points in the convex hull say ‘h’ points. Thus the complexity can be given as O(n*h).

In the worst case , the ‘n’ points each belong in the convex hull as shown in the prior example, thus ‘h’=‘n’ and the time complexity is thus in worst case O(n^2)

Thank You