Convex Hull

Convex Hull

- Convex Hull of a set Q of points is the smallest convex polygon P for which each point of Q is either on the boundary of P or inside it
- Smallest set with minimum perimeter

Convex Hull – Introduction

- Intuitively, we can think of each point in Q as being a nail sticking out from a board.
- Convex hull is then the shape formed by a tight rubber band that surrounds all the nails.
- Figure 33.6 shows a set of points and its convex hull

Convex Hull – Introduction

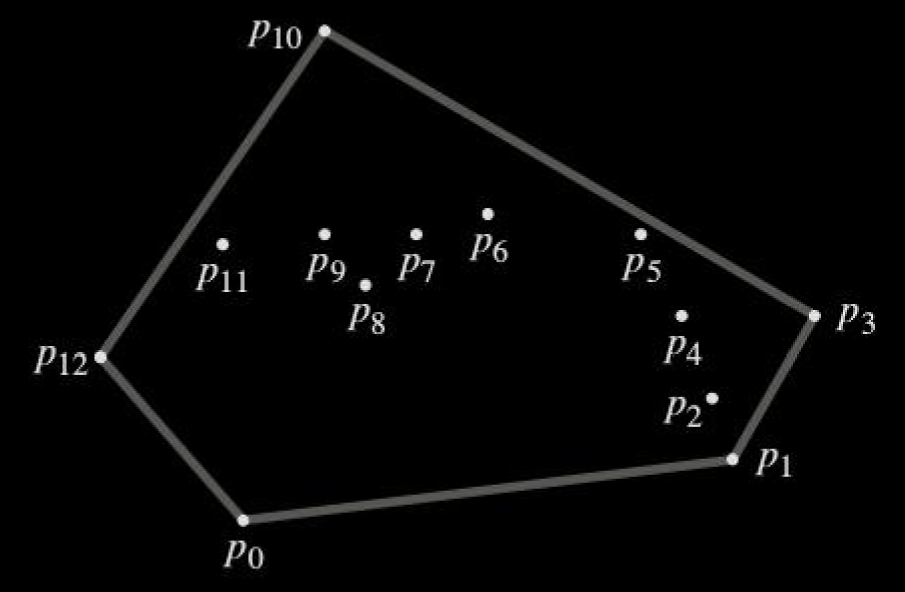
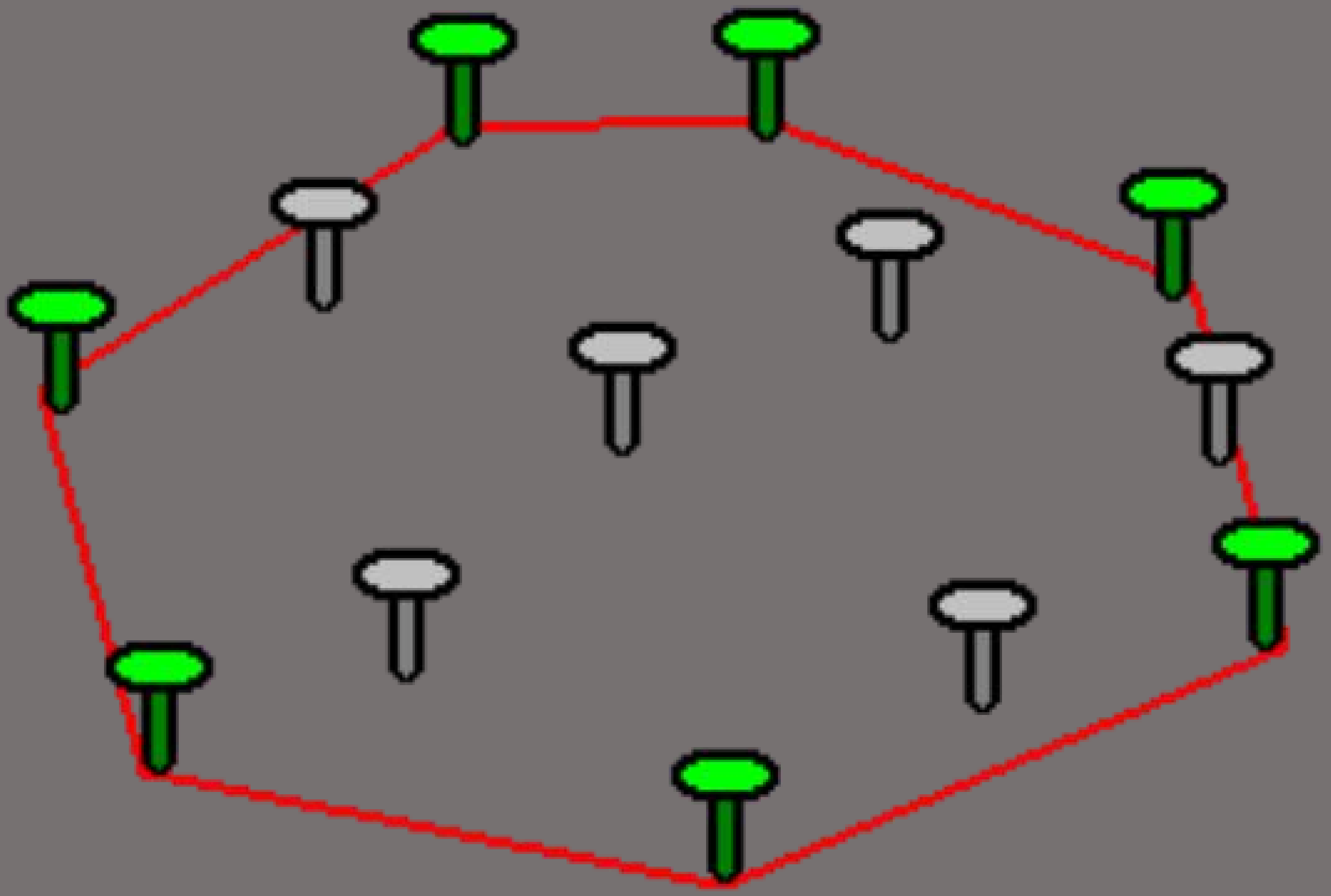


Figure 33.6 A set of points $Q = \{p_0, p_1, \ldots, p_{12}\}$ with its convex hull $\text{CH}(Q)$ in gray.

Convex Hull – Introduction



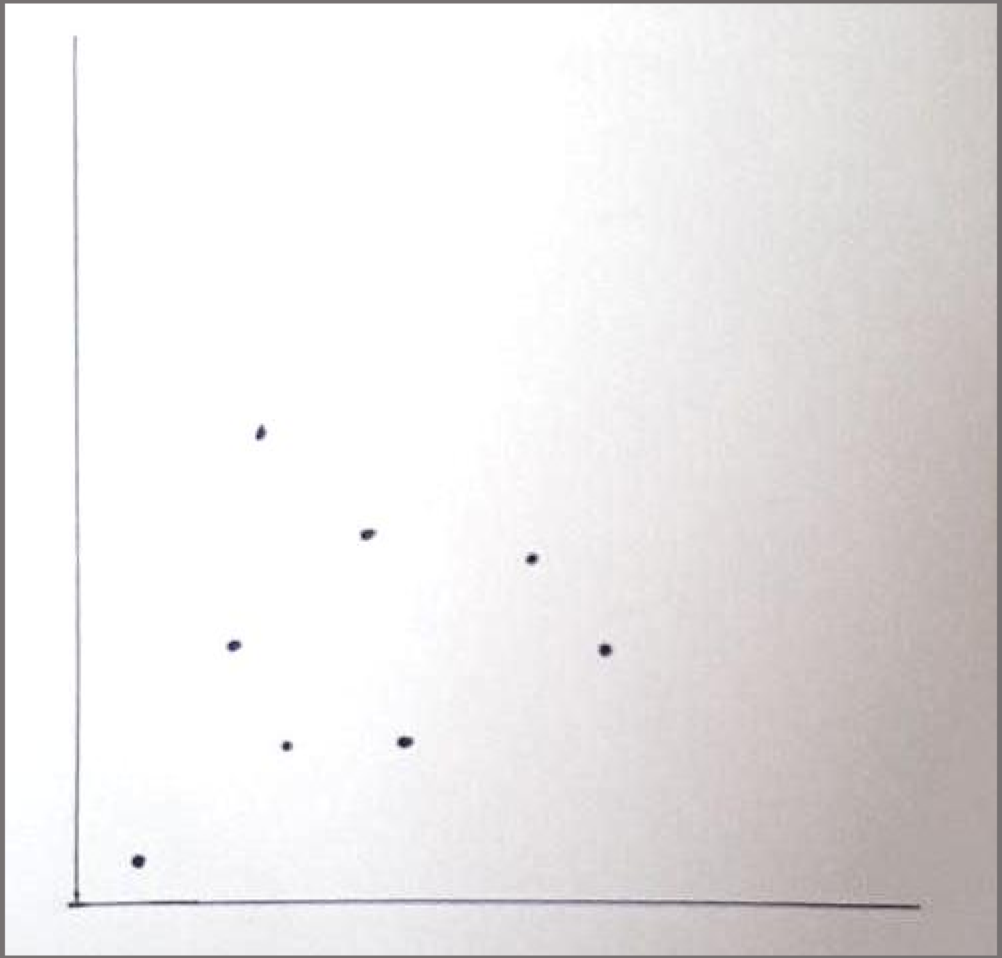
Convex Hull – Introduction

- First, known as Graham's scan, runs in $O(n \lg n)$ time
- second, called Jarvis's march, runs in $O(nh)$ time, where h is the number of vertices of the convex hull

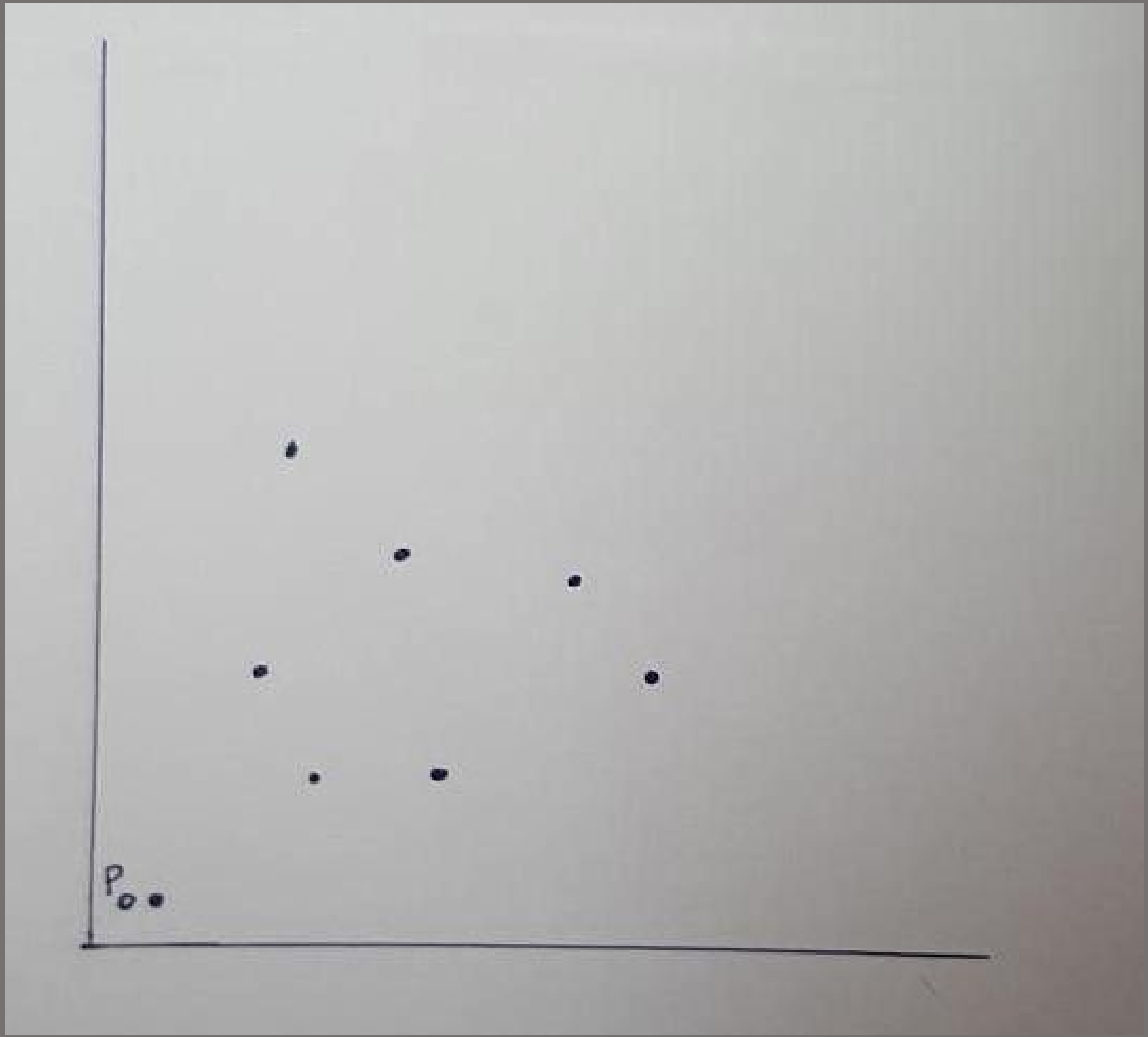
Convex Hull – Introduction

- Both Graham's scan and Jarvis's march use a technique called "rotational sweep," processing vertices in the order of the polar angles they form with a reference vertex

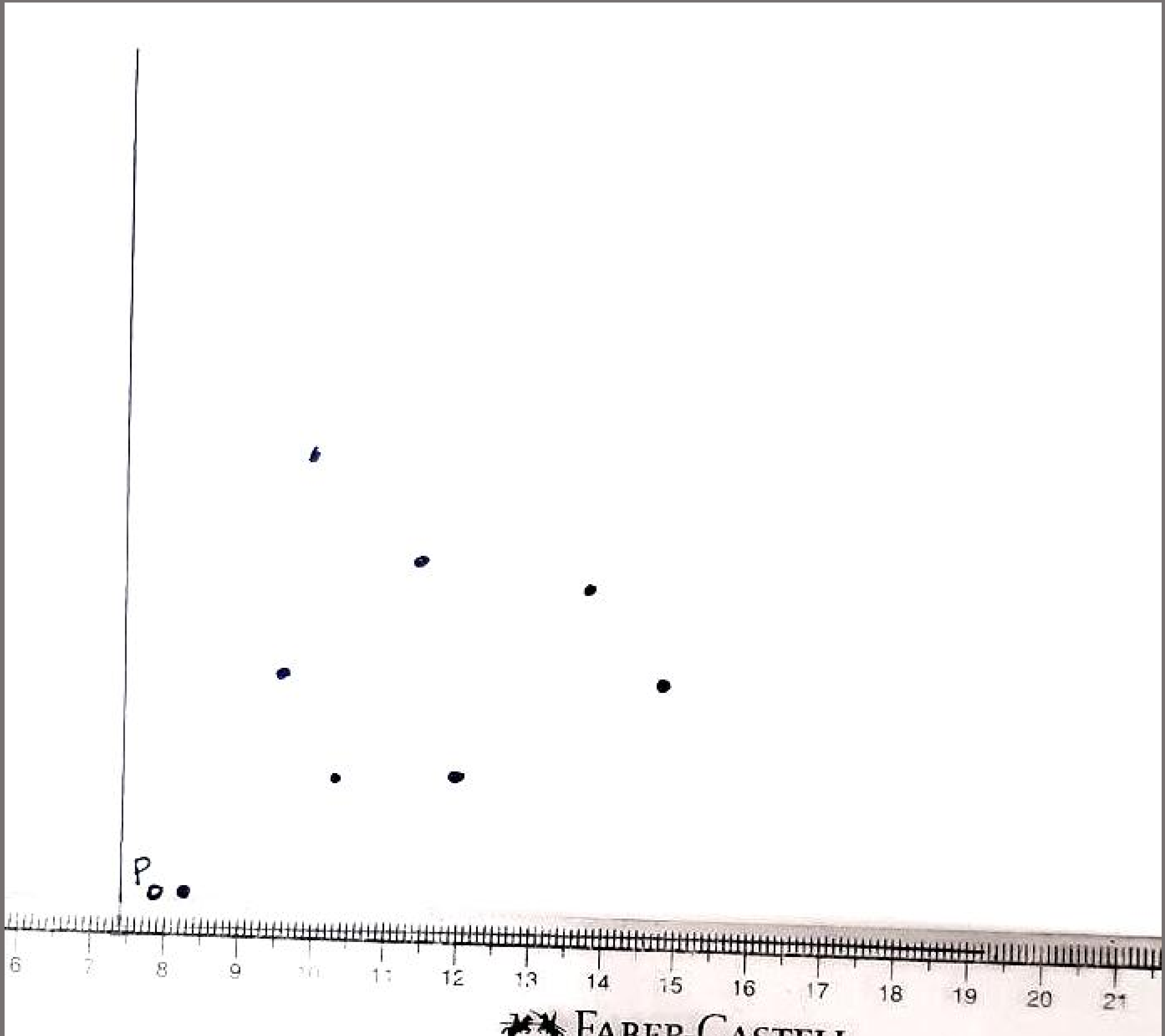
Input



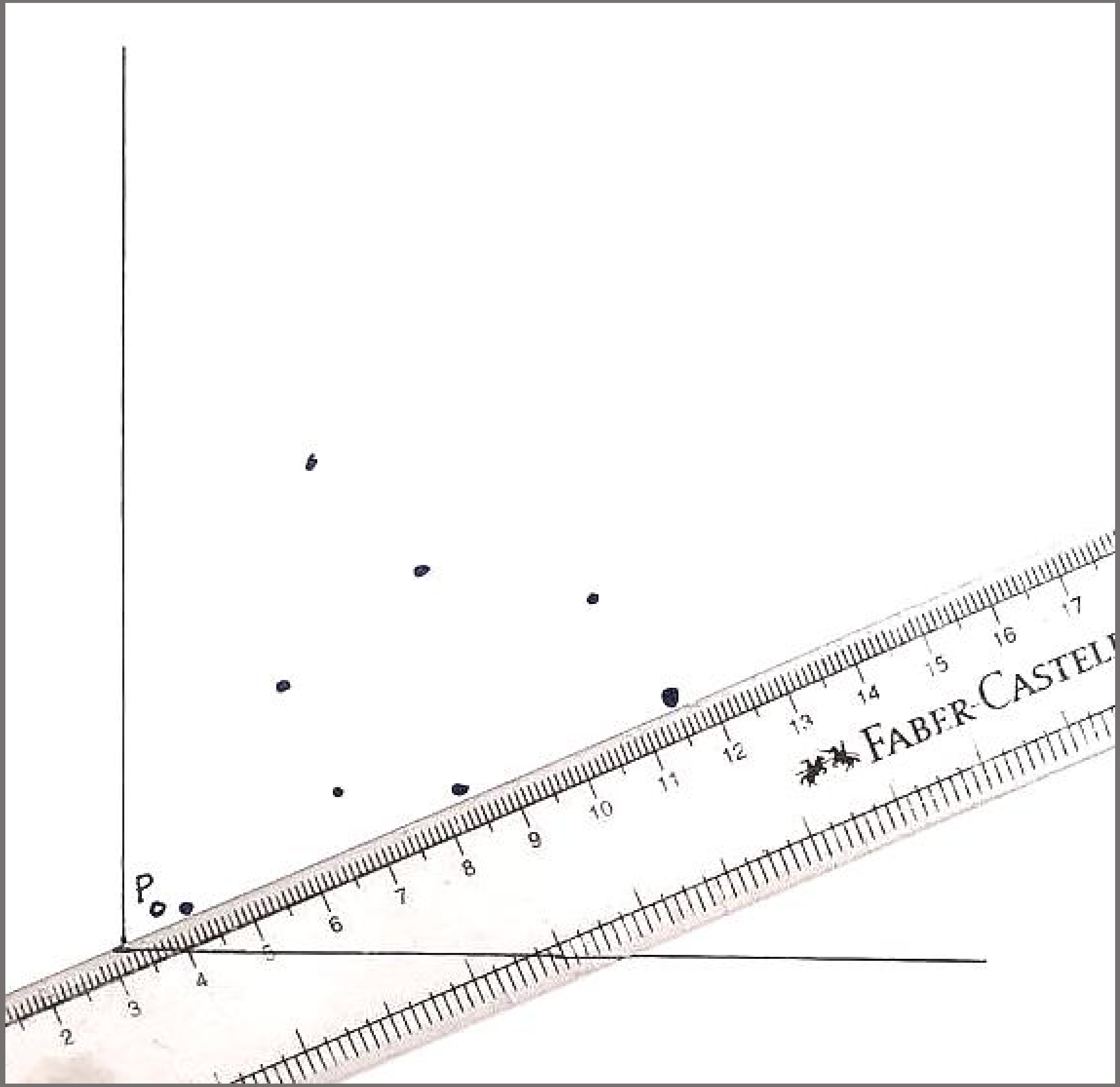
Identify the left and bottom most point and Name it as P_0



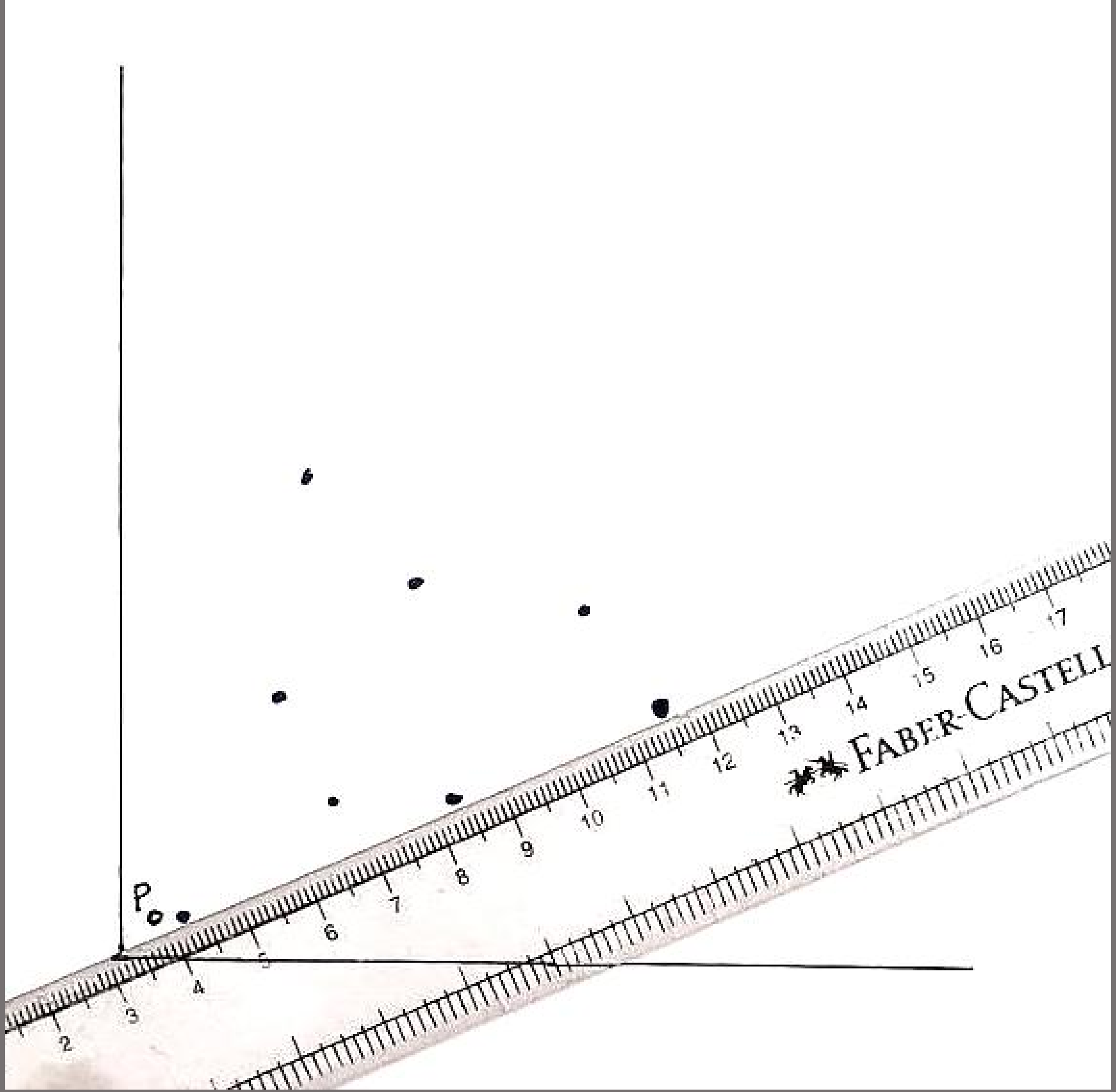
Sort rest of points based on Polar angle in Counter Clockwise Direction



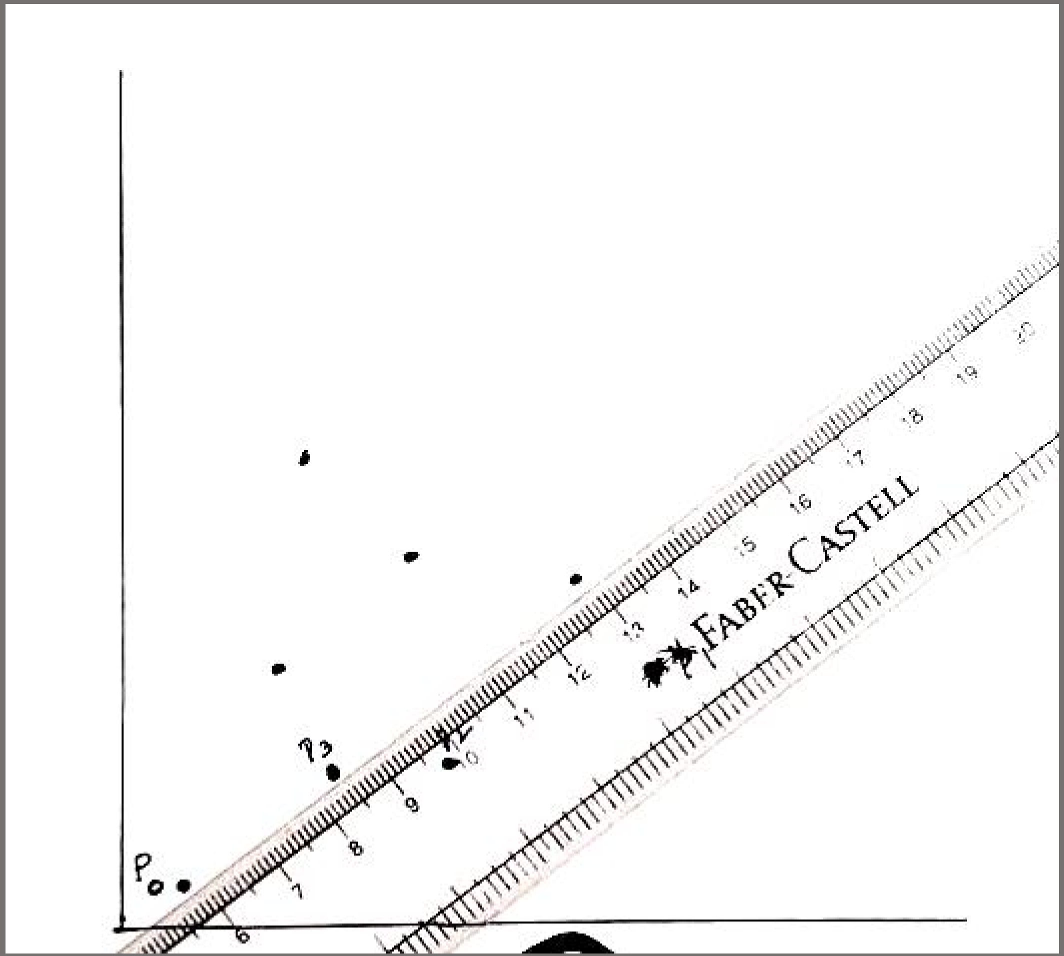
Sort rest of points based on Polar angle in Counter Clockwise Direction



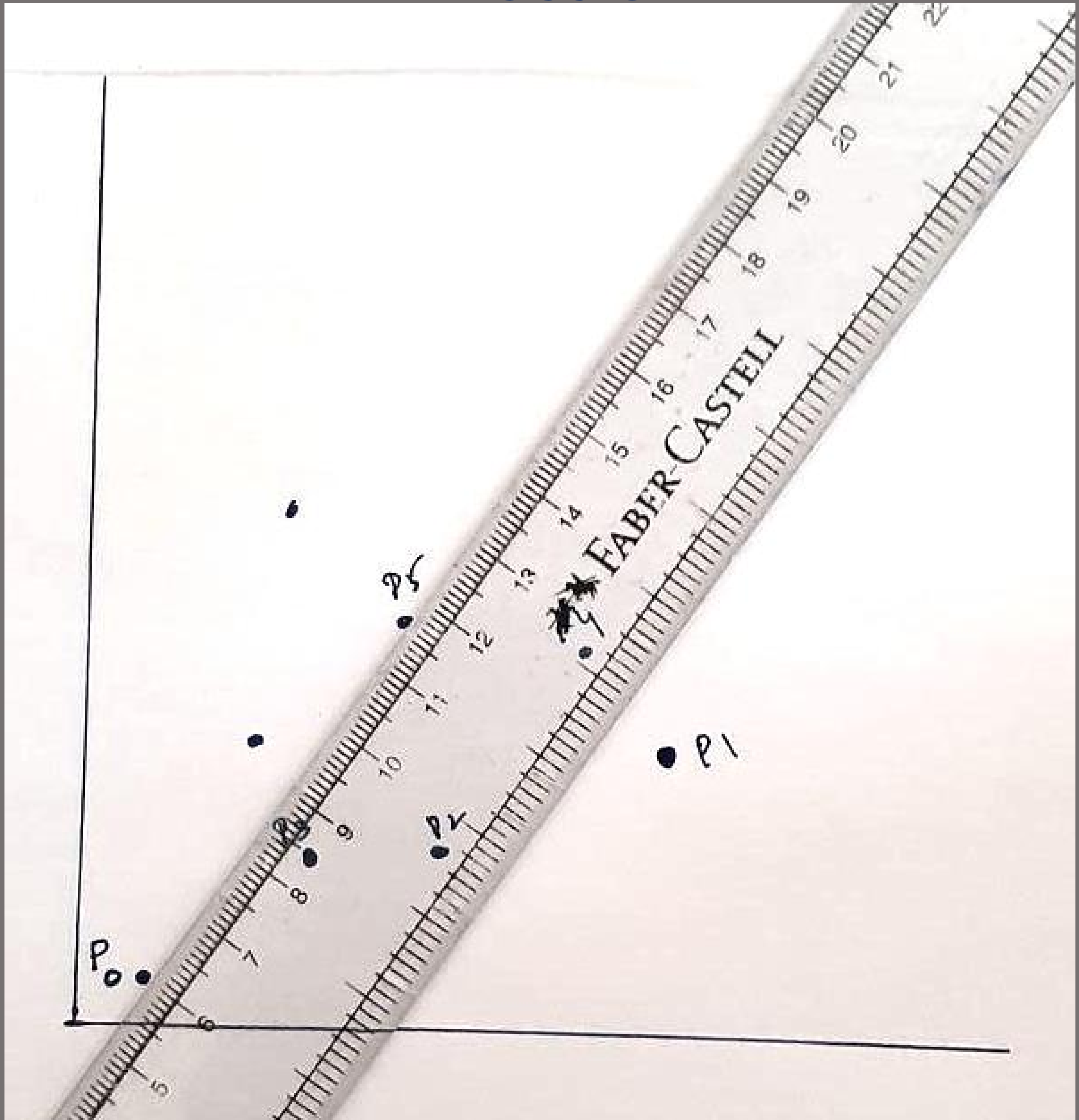
Sort rest of points based on Polar angle in Counter Clockwise Direction



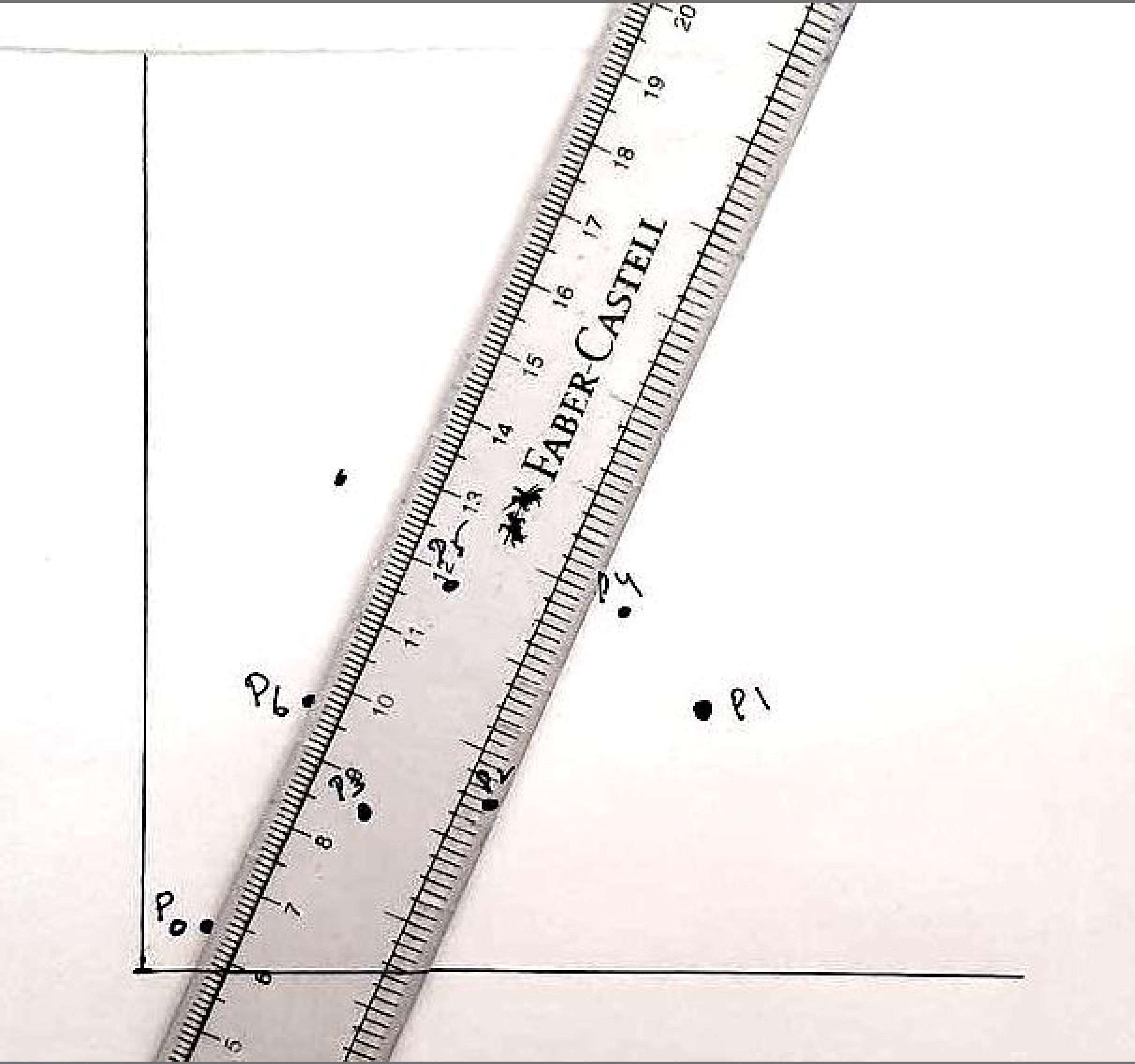
Sort rest of points based on Polar angle in Counter Clockwise Direction



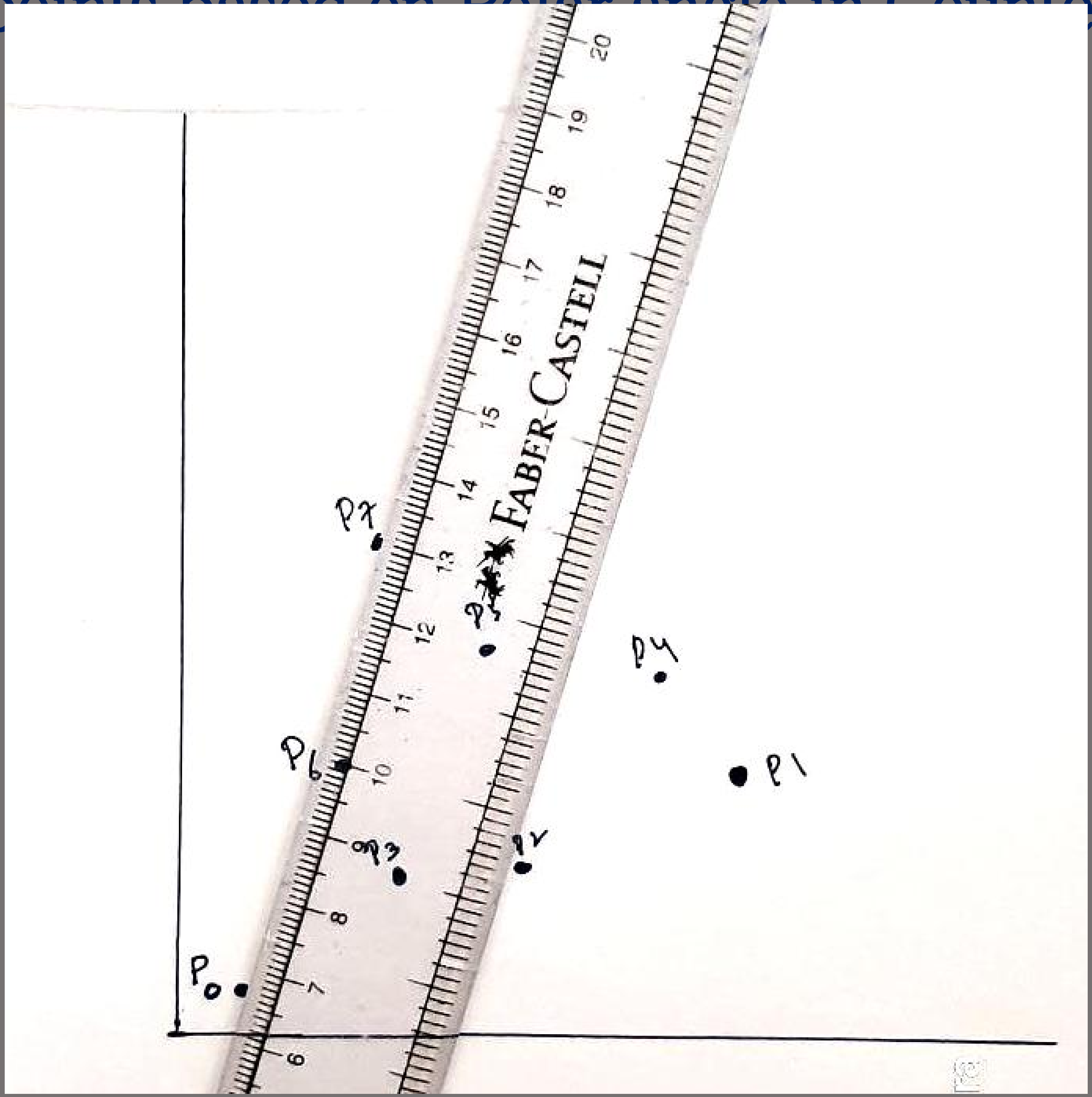
Sort rest of points based on Polar angle in Counter Clockwise Direction



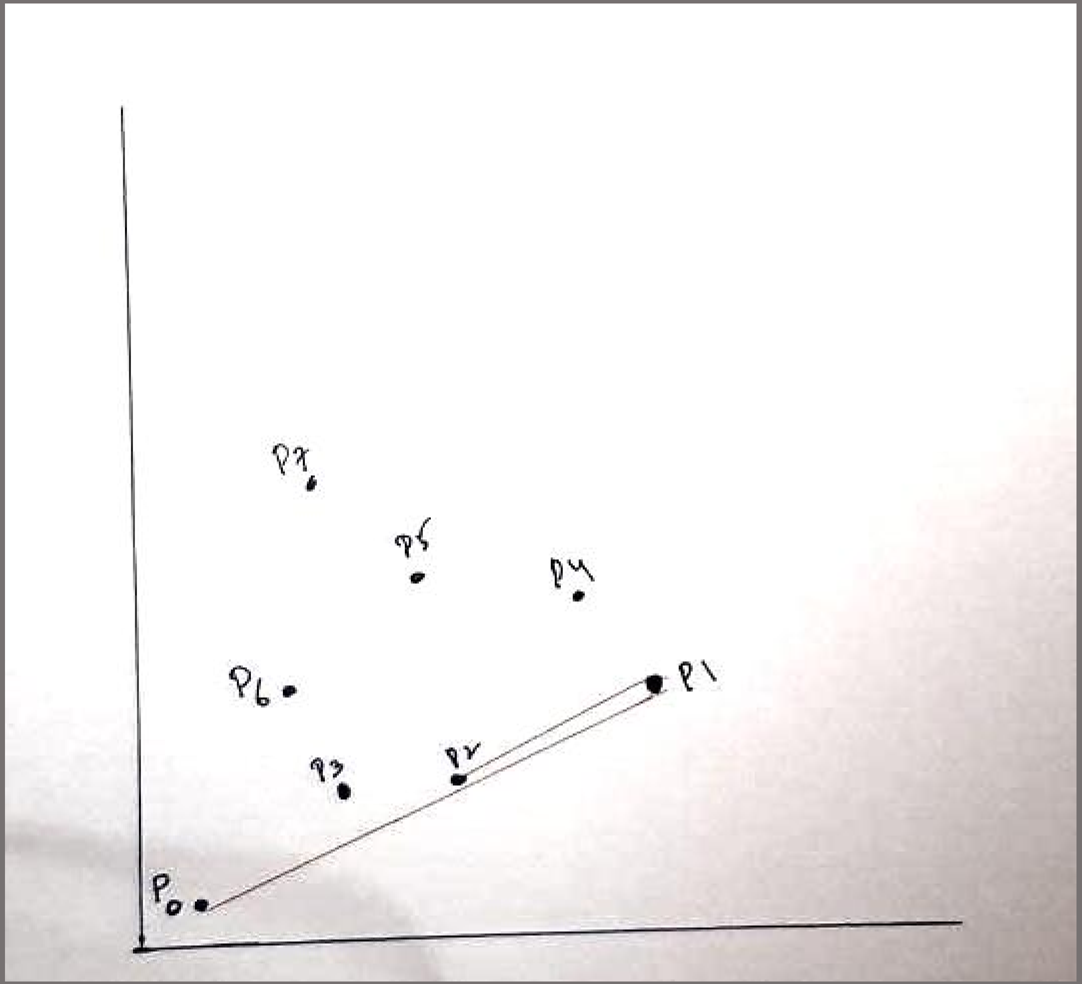
Sort rest of points based on Polar angle in Counter Clockwise Direction

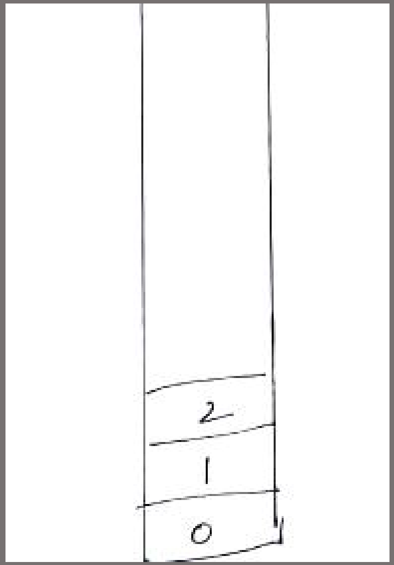


Sort rest of points based on Polar angle in Counter Clockwise



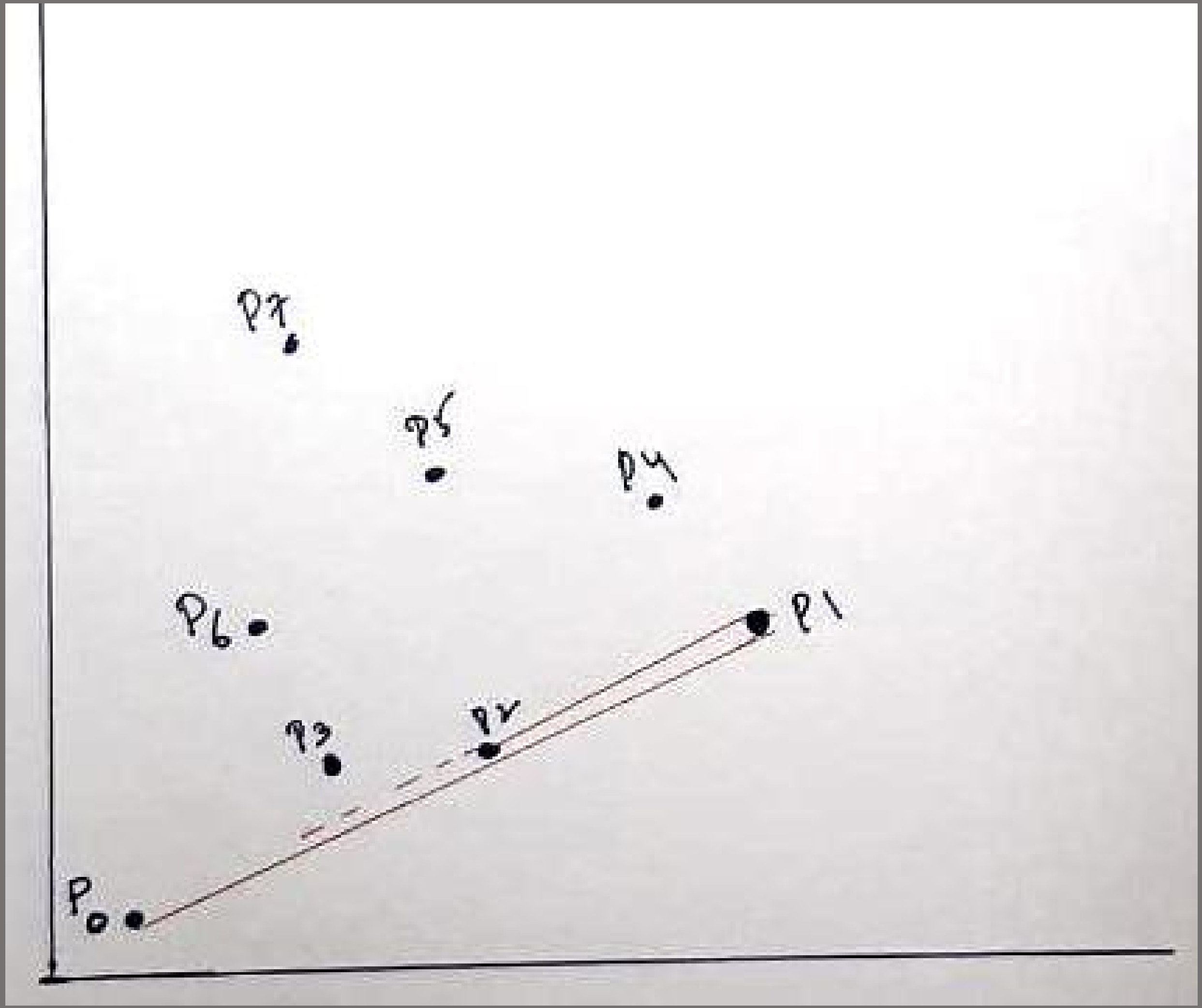
Push P0, P1 and P2 into the Stack

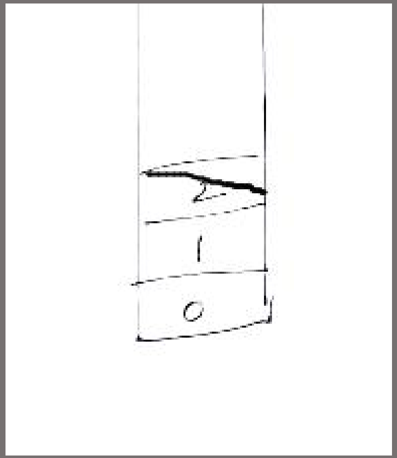




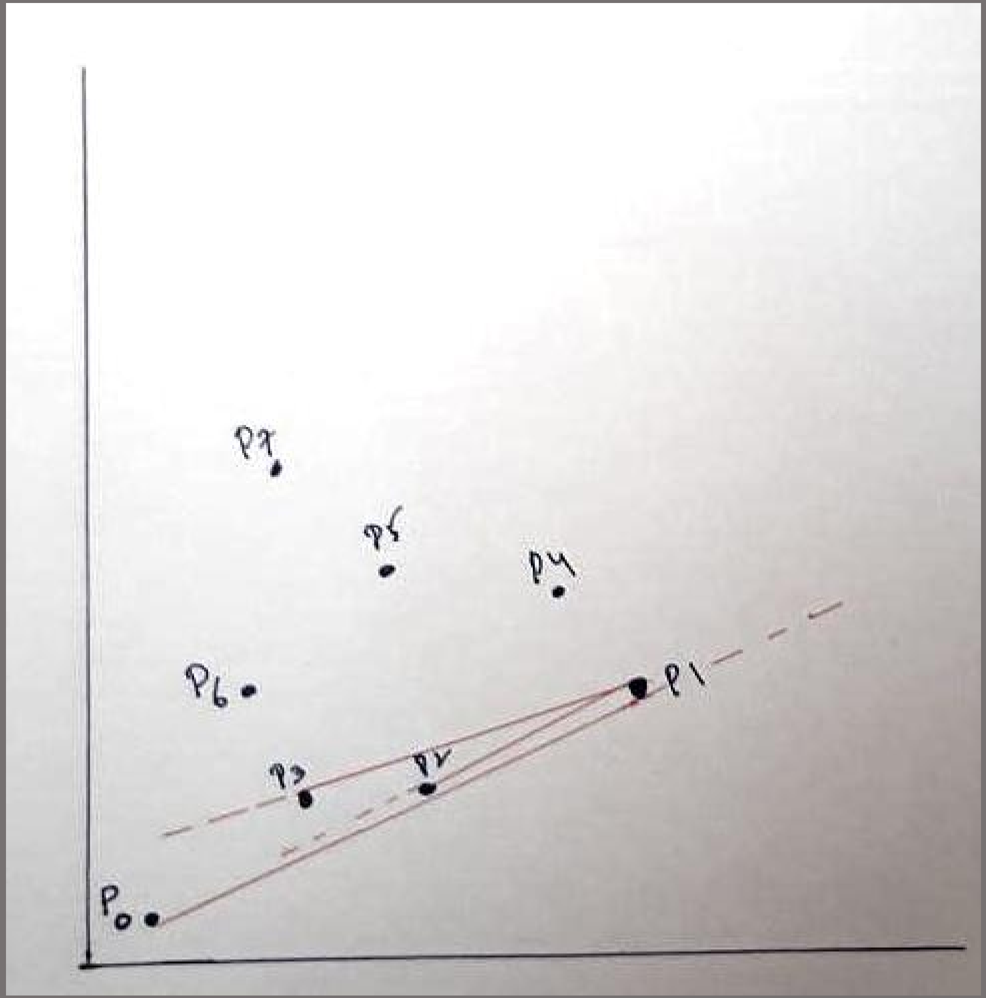
Extend P1 and P2 and Check Orientation of P3 – right

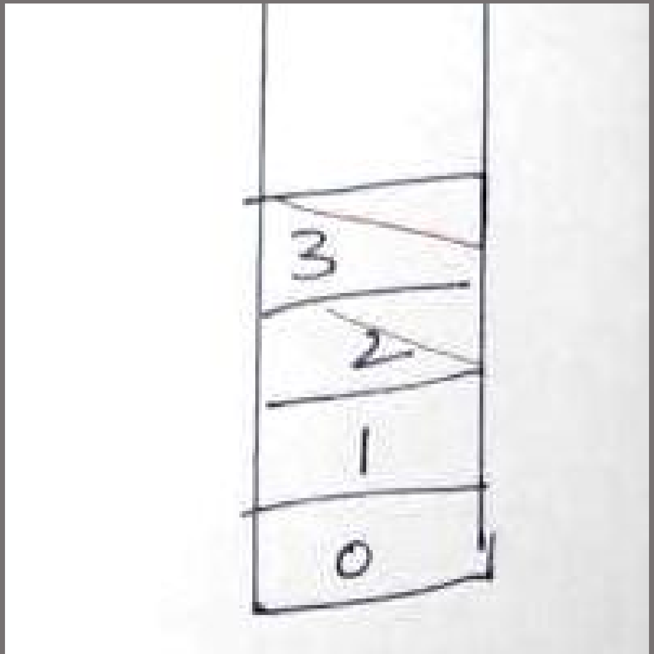
Pop Stack



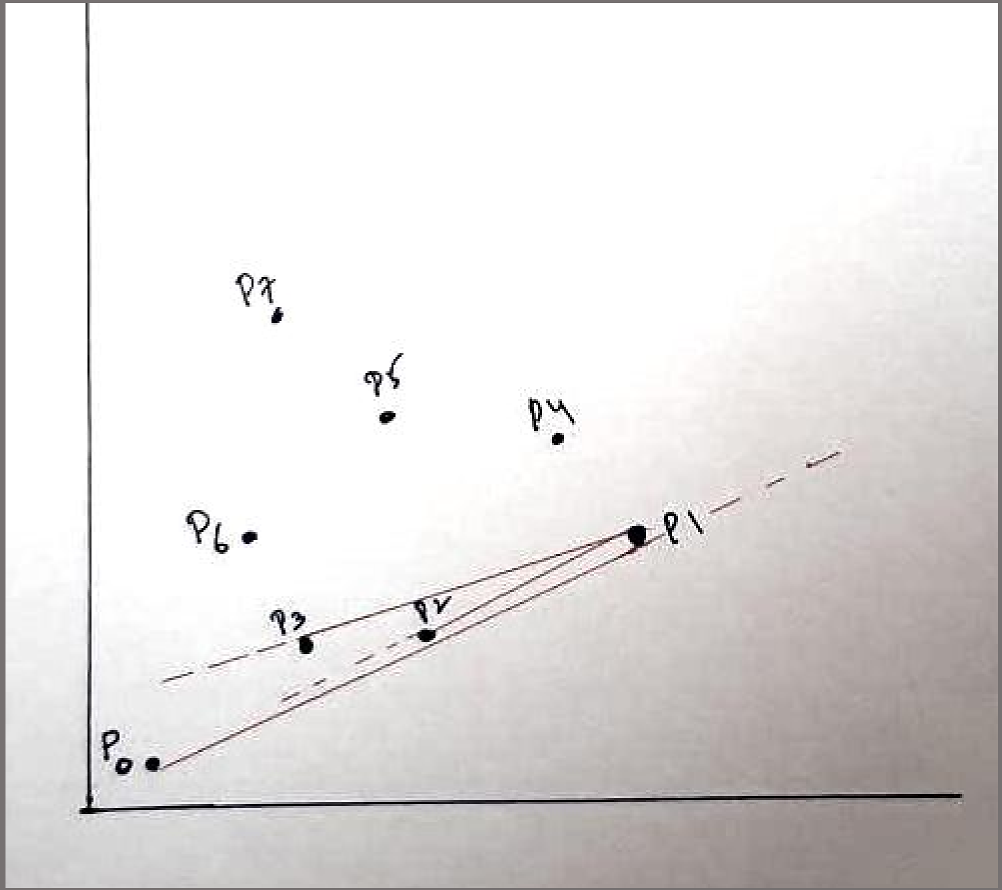


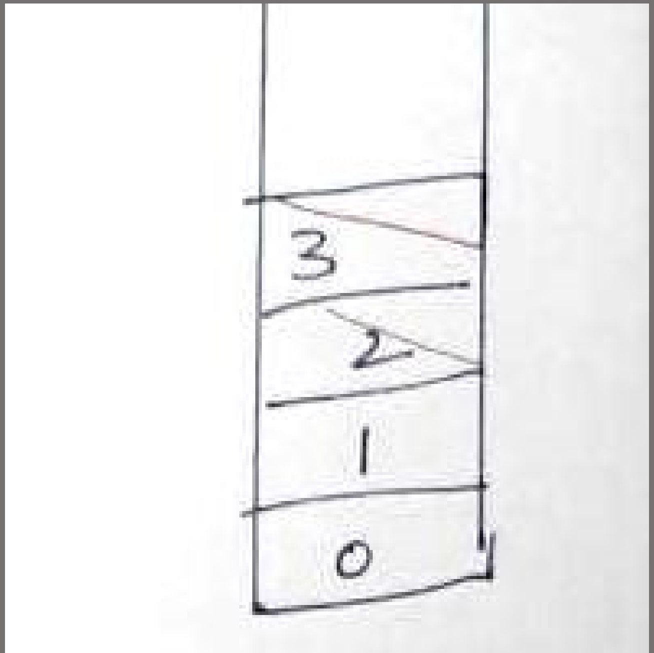
Extend P_0 and P_1 and Check Orientation of P_3 – counter clock wise Push into Stack



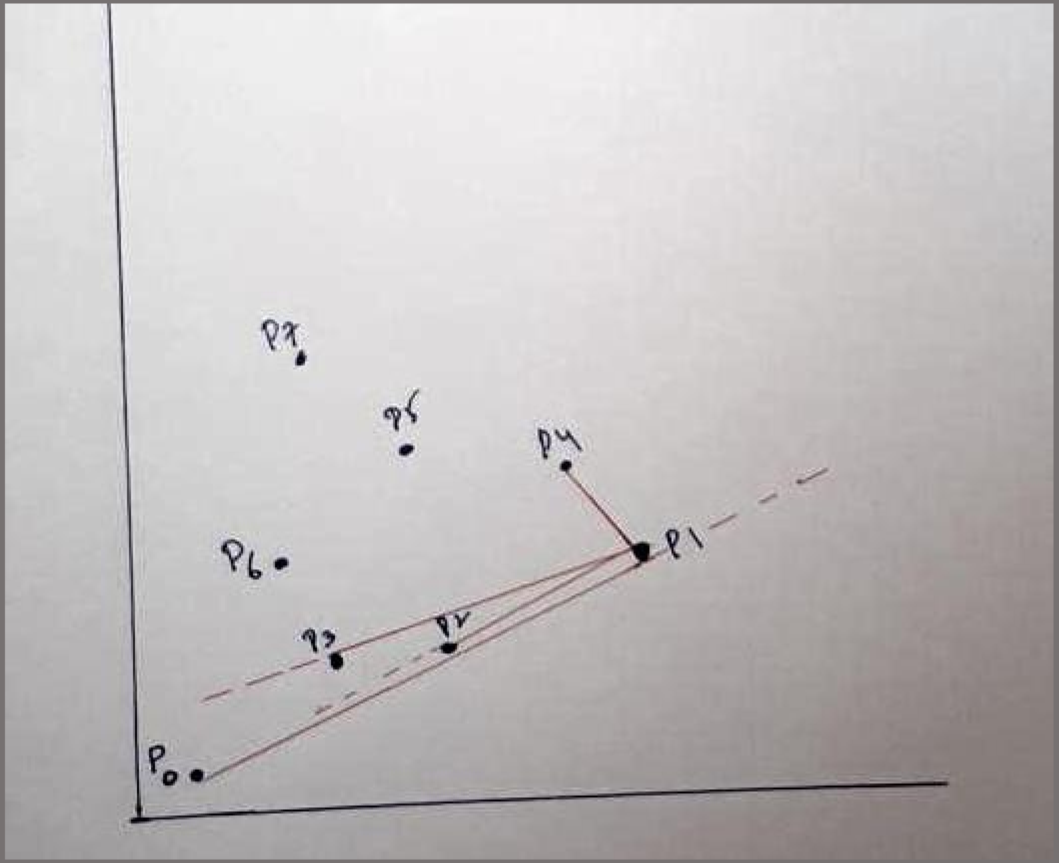


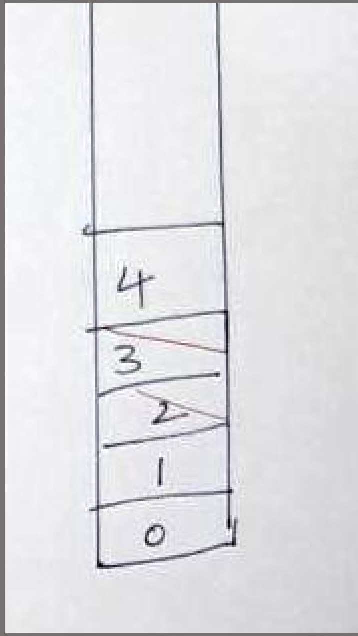
Extend P_2 and P_3 and Check Orientation of P_4 – clock wise Pop Stack



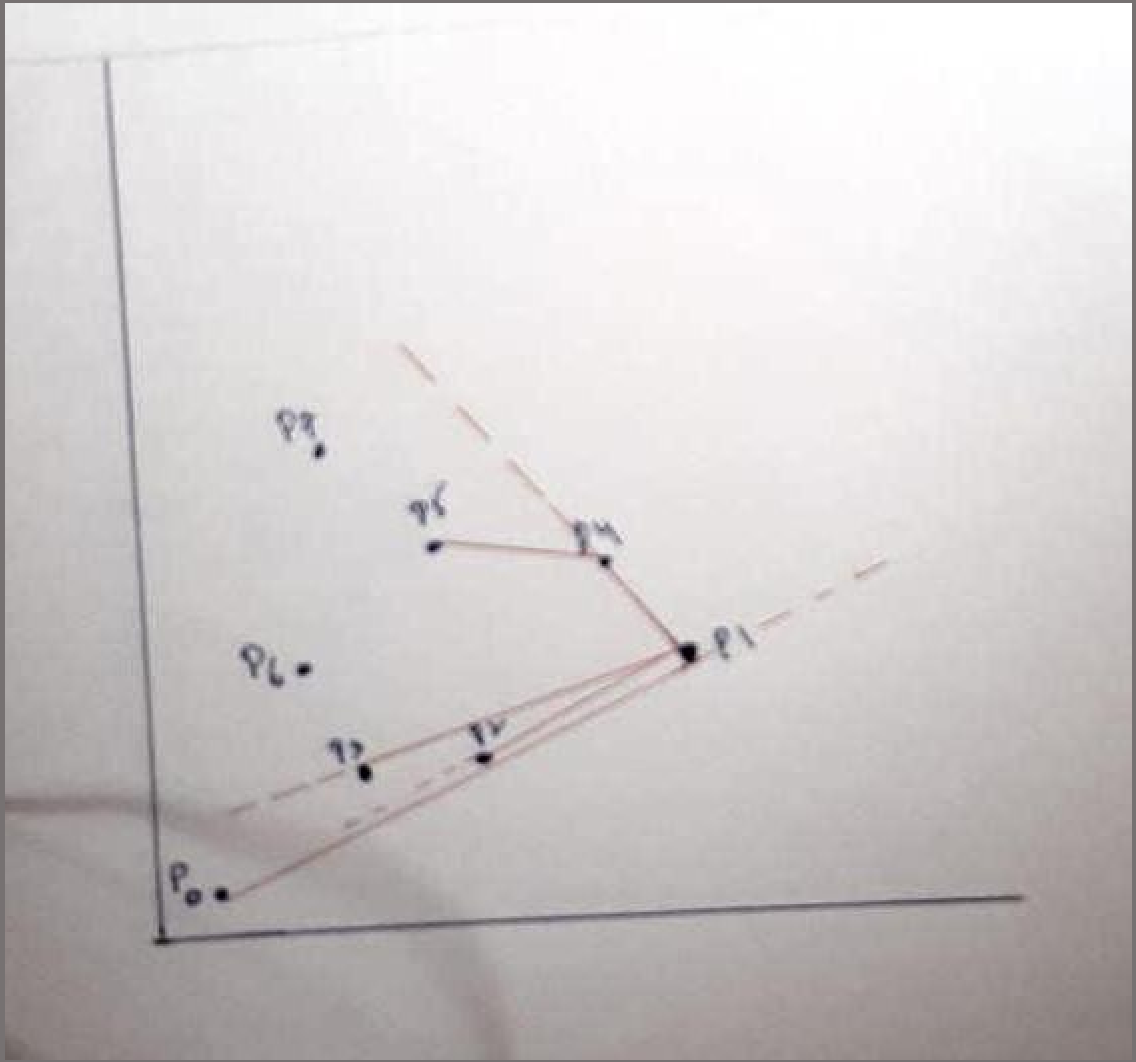


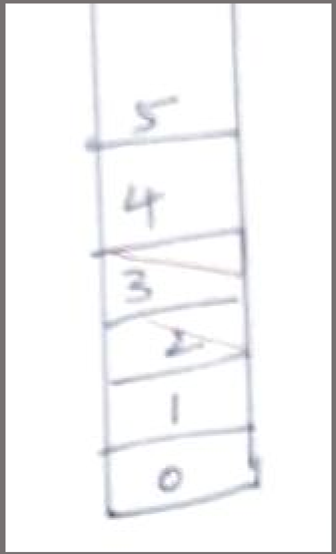
Extend P_0 and P_1 and Check Orientation of P_4 – Counter clock wise Push into Stack



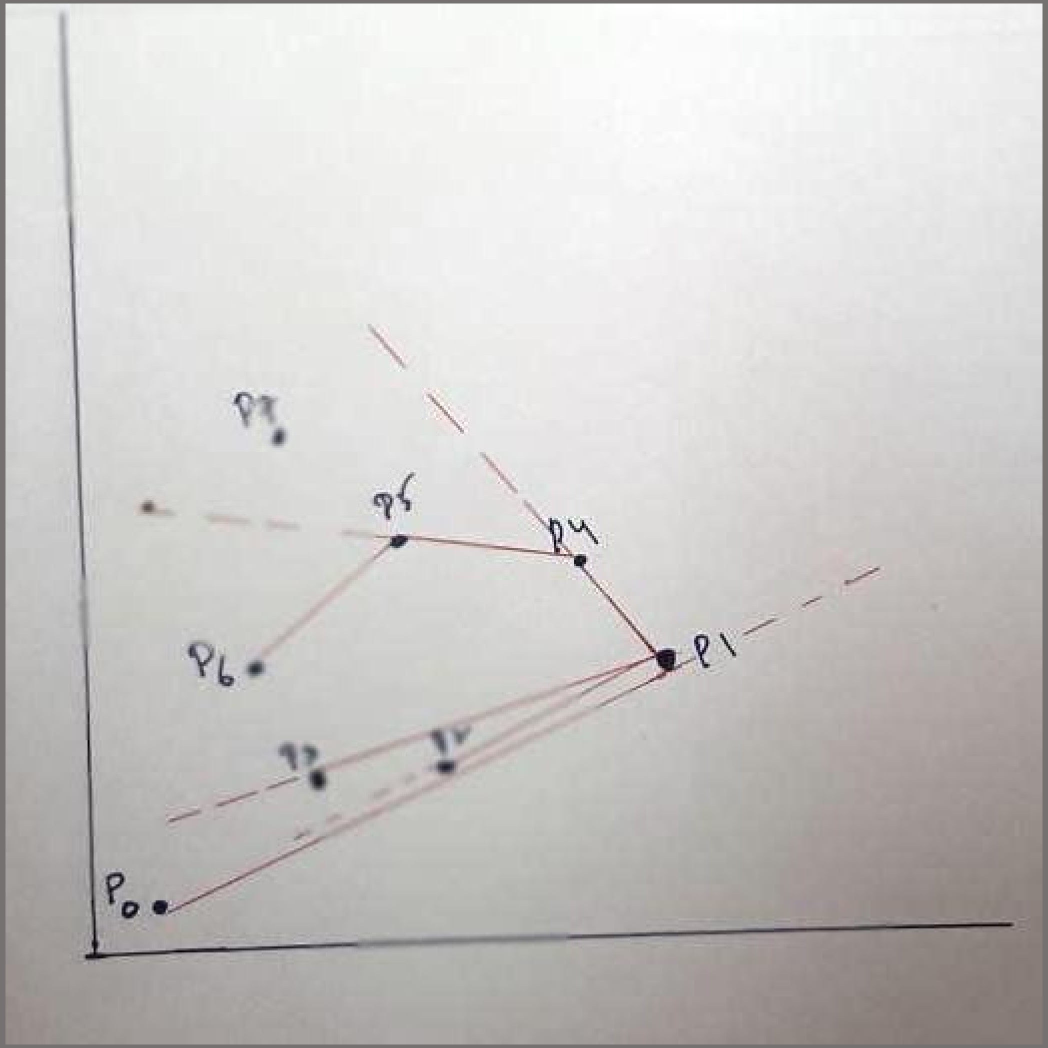


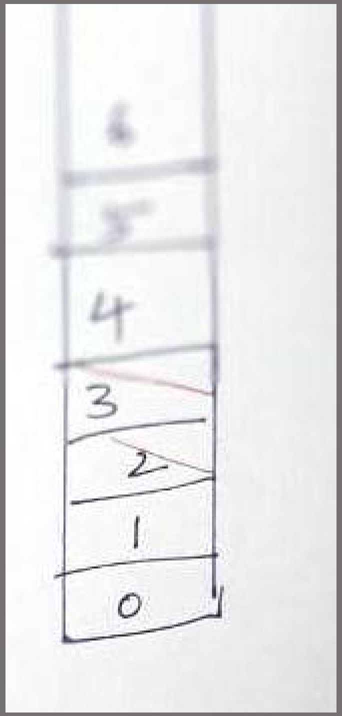
Extend P_1 and P_4 and Check Orientation of P_5 – Counter clock wise Push into Stack



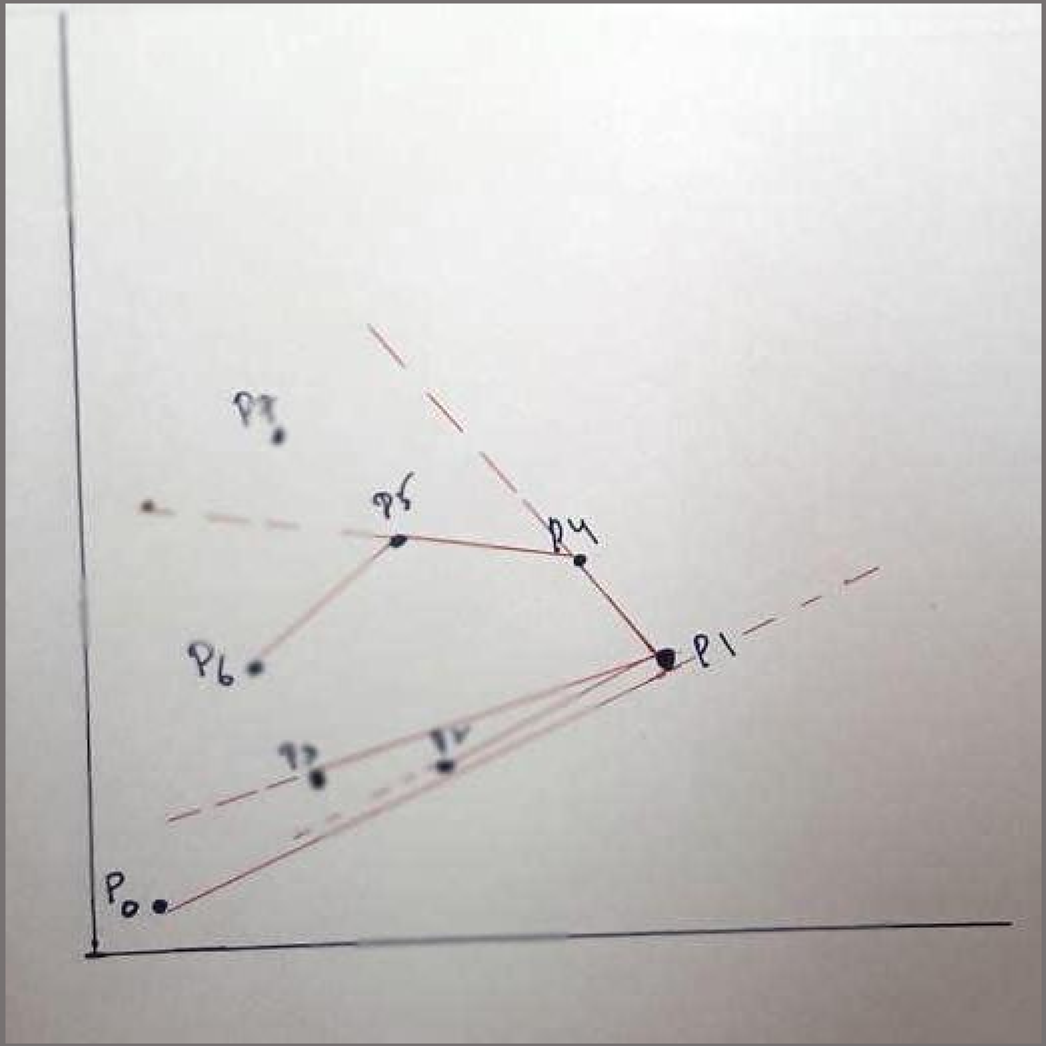


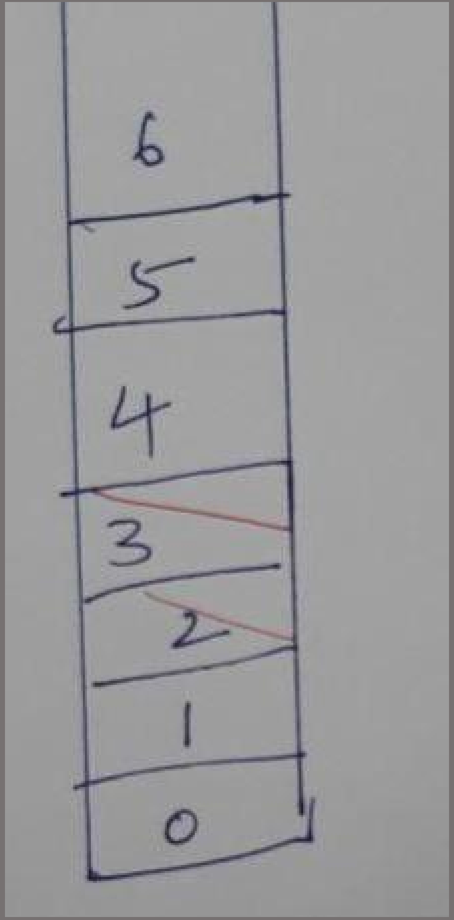
Extend P_4 and P_5 and Check Orientation of P_6 – Counter clock wise Push into Stack



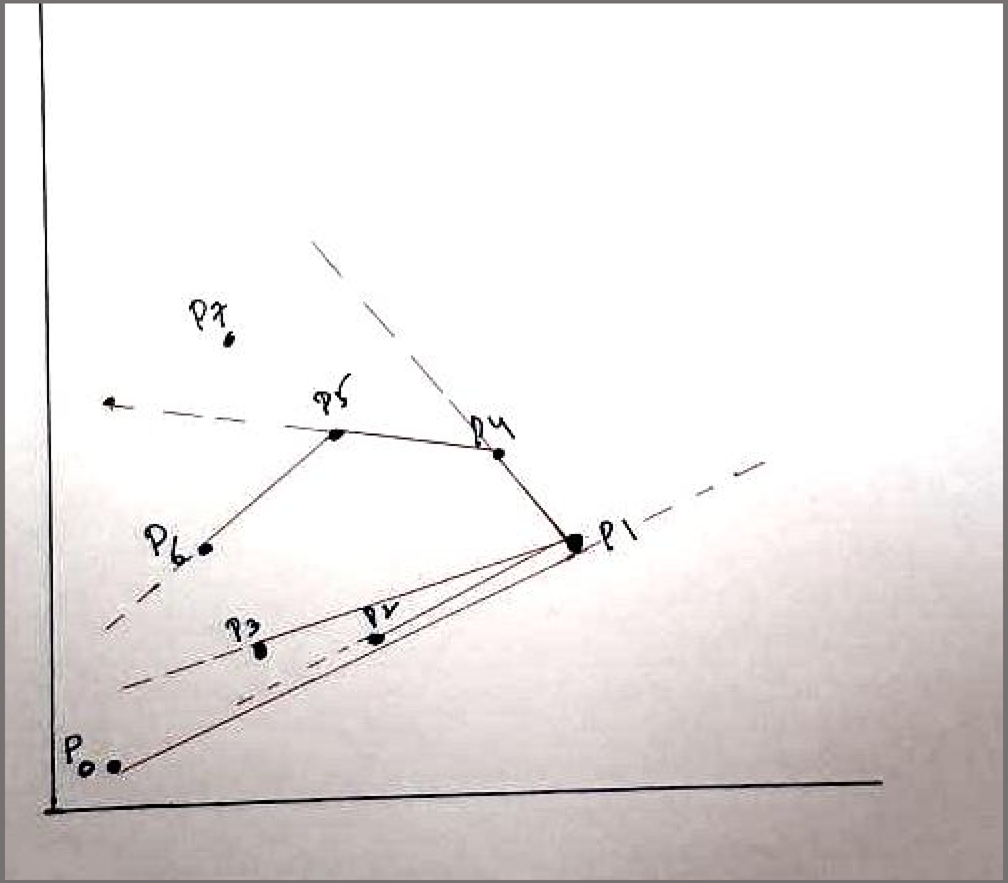


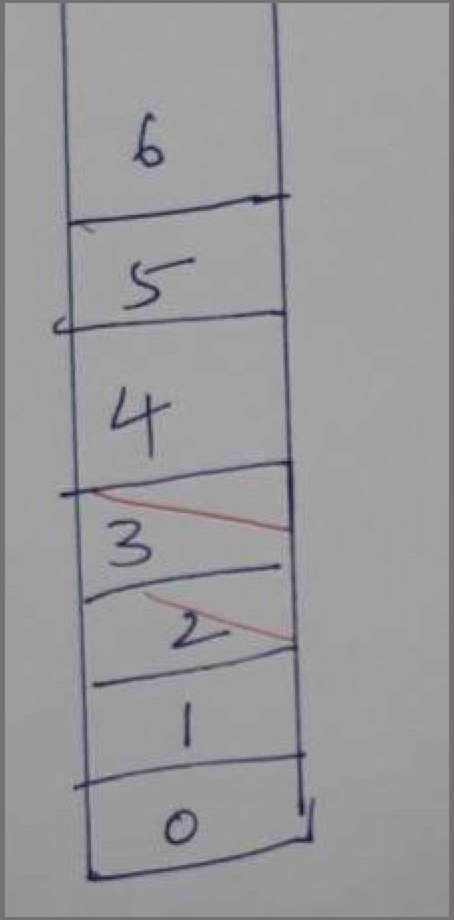
Extend P_5 and P_6 and Check Orientation of P_7 – Clock wise Pop Stack



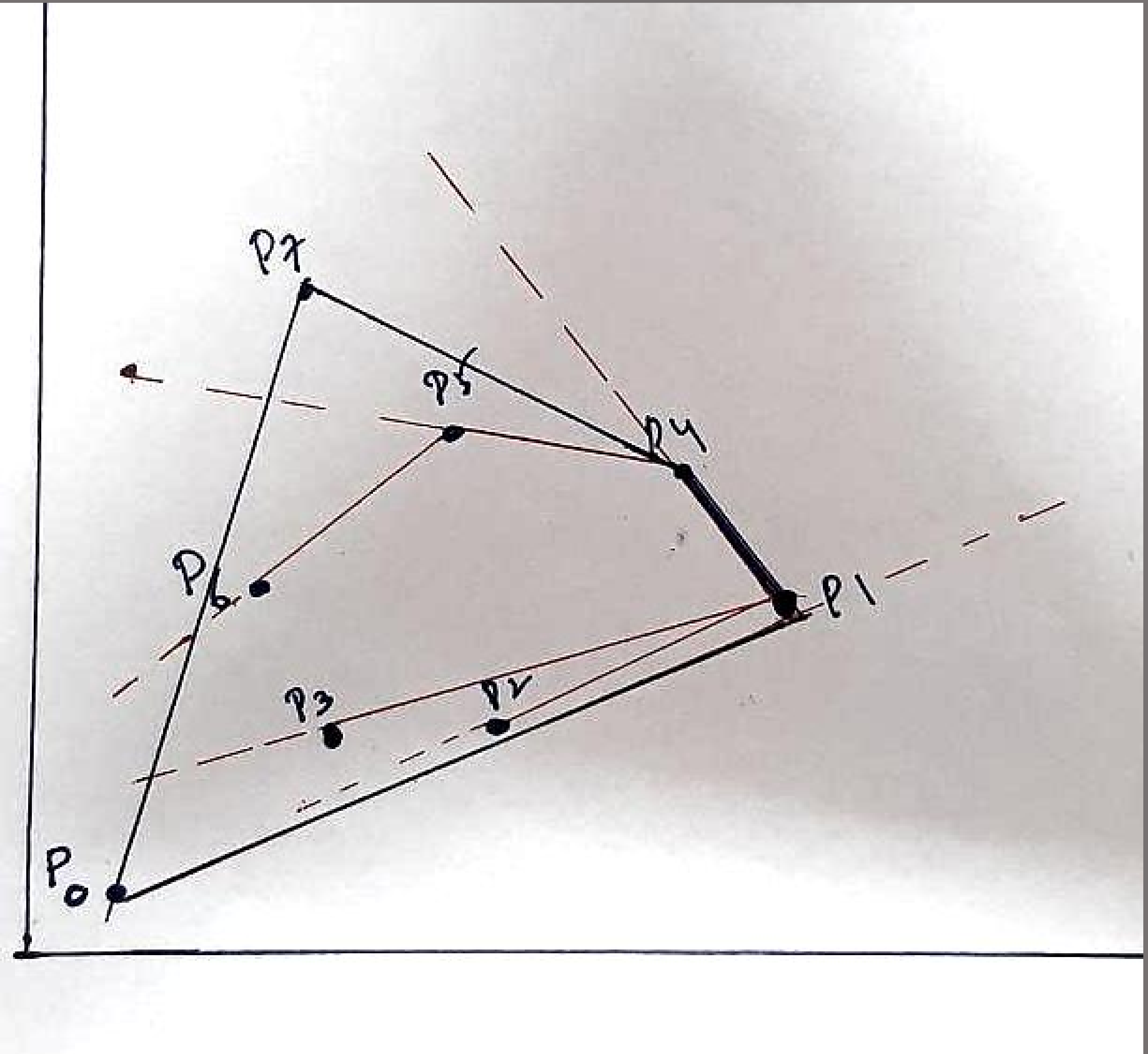


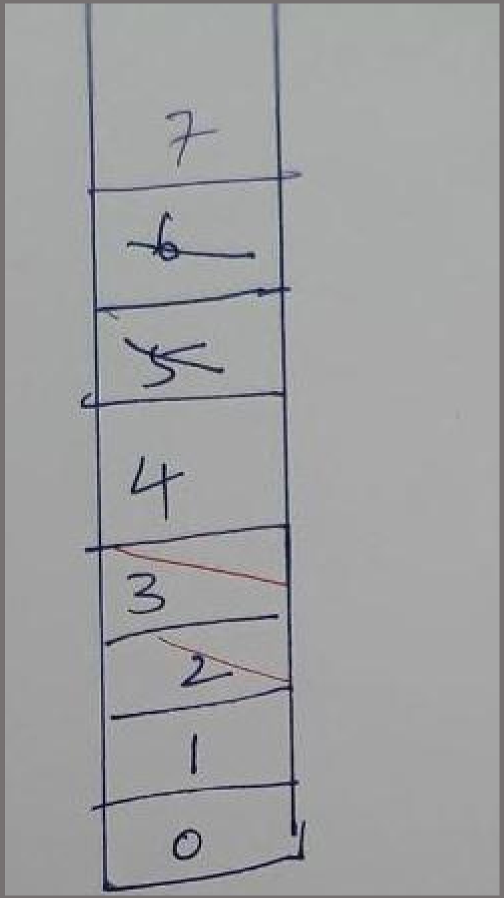
Check P7 with Points in Stack





Convex Hull Formed





Graham’s scan

- Solves convex–hull problem by maintaining a stack S of candidate points
- Pushes each point of the input set Q onto the stack one time
- eventually pops from the stack each point that is not a vertex of CH(Q)
- When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary

Graham’s scan

- Procedure G RAHAM –S CAN takes as input a set Q of points, where $|Q| \geq 3$
- It calls the functions TOP(S), which returns the point on top of stack S without changing S
- N EXT –TO–TOP(S), which returns the point one entry below the top of stack S without changing S
- As we shall prove in a moment, the stack S returned by G RAHAM –S CAN contains, from bottom to top, exactly the vertices of CH(Q) in counterclockwise order.

```
GRAHAM-SCAN(Q)
let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
let ⟨p1, p2, ..., pm⟩ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p0
        (if more than one point has the same angle, remove all but
        the one that is farthest from p0)
let S be an empty stack
PUSH(p0, S)
PUSH(p1, S)
PUSH(p2, S)
for i = 3 to m
        while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                        and pi makes a nonleft turn
                POP(S)
        PUSH(pi, S)
return S
```

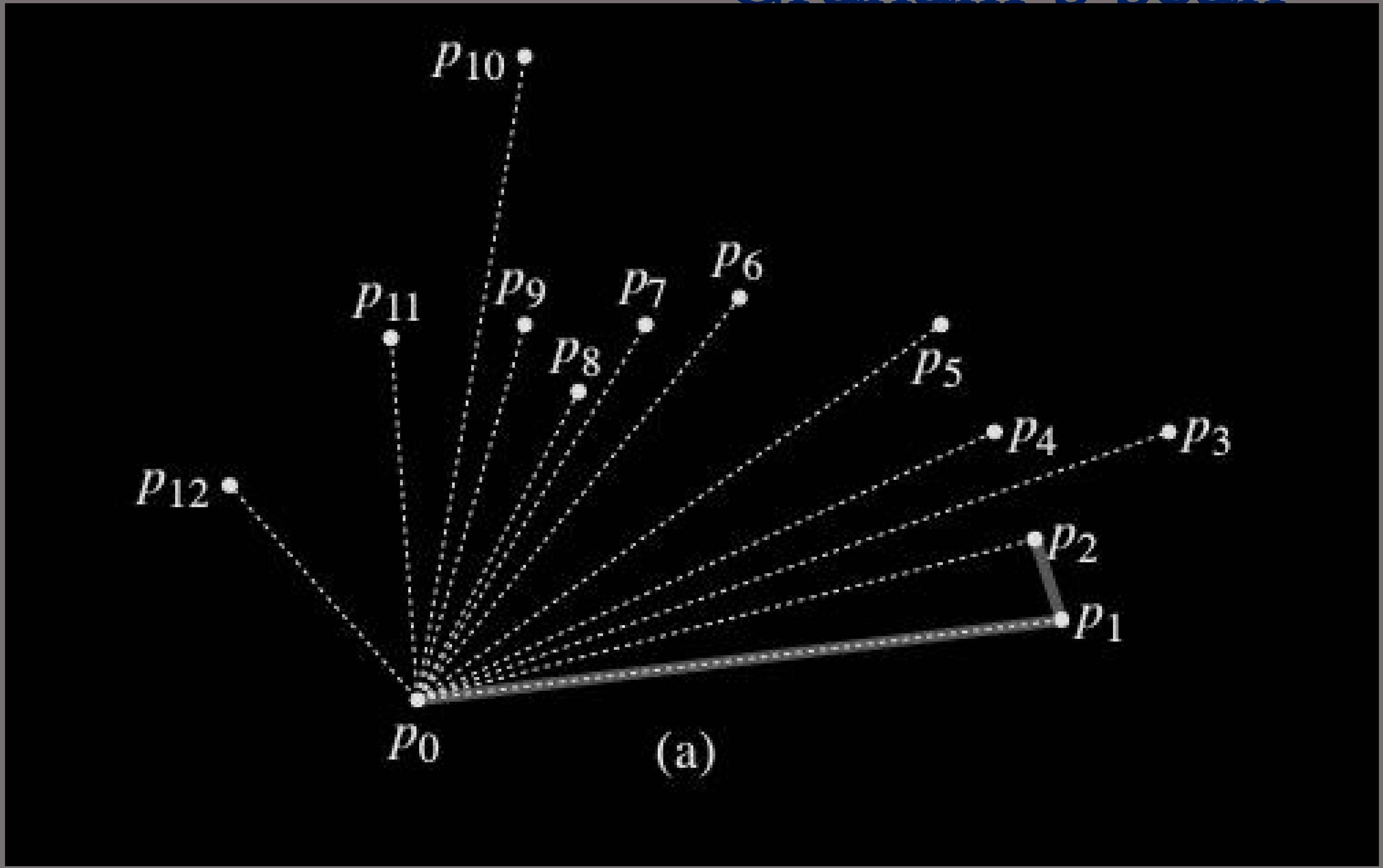
Graham's scan

- Figure 33.7 illustrates the progress of GRAHAM –S CAN
- Line 1 chooses point p0 as the point with the lowest y–coordinate, picking the leftmost such point in case of a tie.
- Since there is no point in Q that is below p0 and any other points with the same y–coordinate are to its right, p0 must be a vertex of CH(Q)
- Line 2 sorts the remaining points of Q by polar angle relative to p0 , using the same method—comparing cross products—as in Exercise 33.1–3

Graham's scan

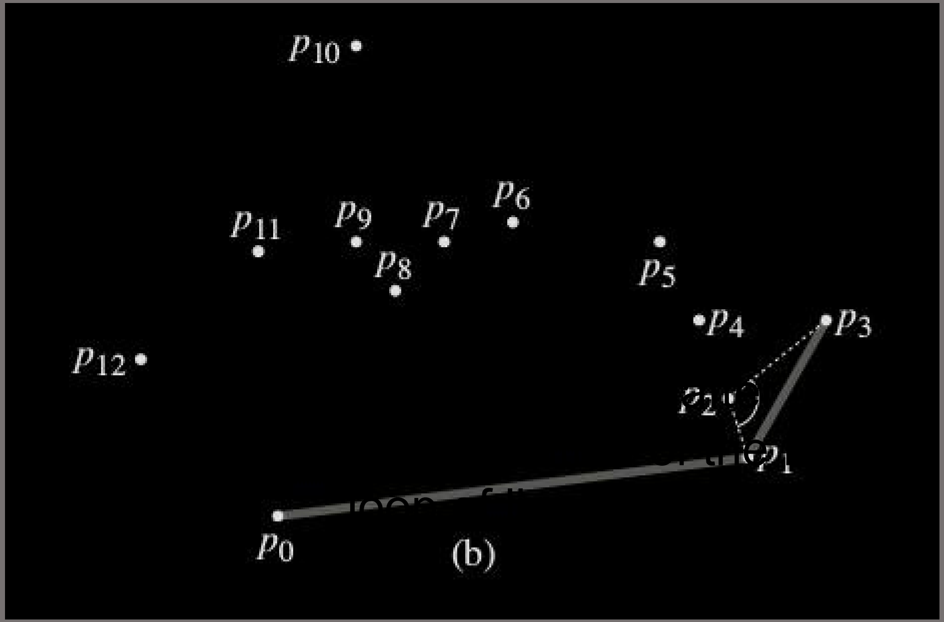
- If two or more points have the same polar angle relative to p0 , all but the farthest such point are convex combinations of p0 and the farthest point, and so we remove them entirely from consideration
- We let m denote the number of points other than p 0 that remain
- The polar angle, measured in radians, of each point in Q relative to p 0 is in the half-open interval [0, Π)
- Since the points are sorted according to polar angles, they are sorted in counterclockwise order relative to p0

Graham's scan



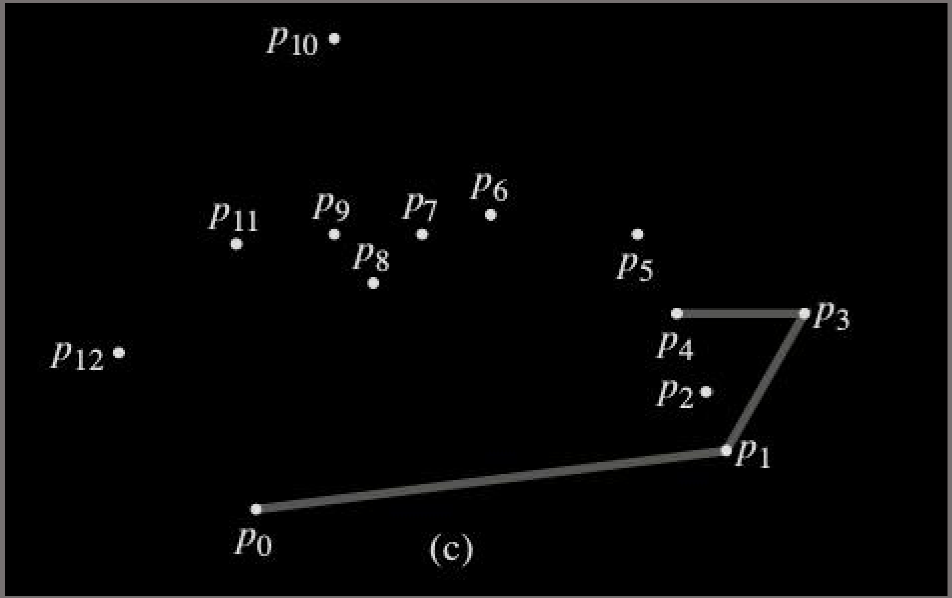
Sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points numbered in order of increasing polar angle relative to p_0 , and the initial stack S containing p_0, p_1, and p_2 .

Graham's scan

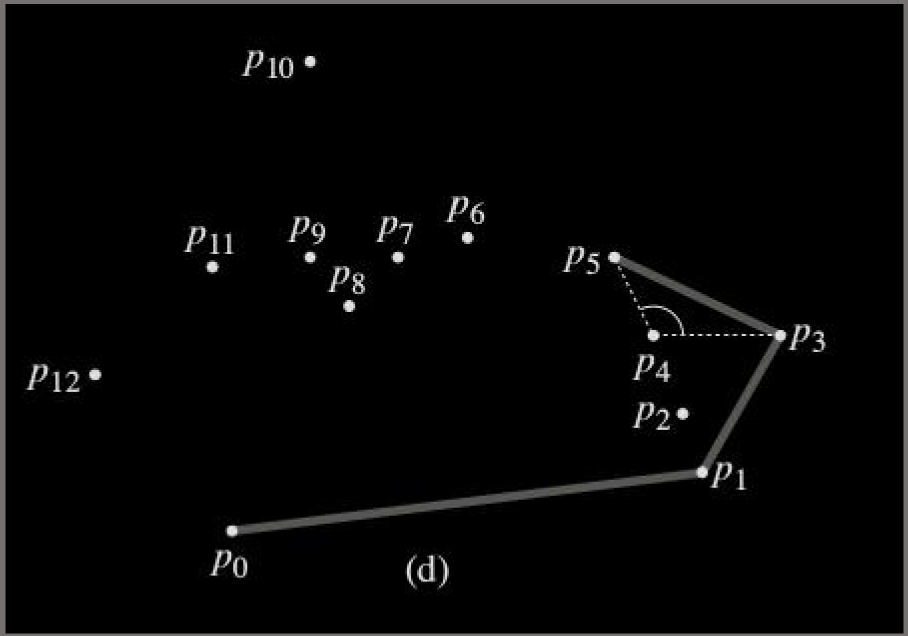


(b) – (k) Stack S after each iteration of the for loop of lines 7 – 10.

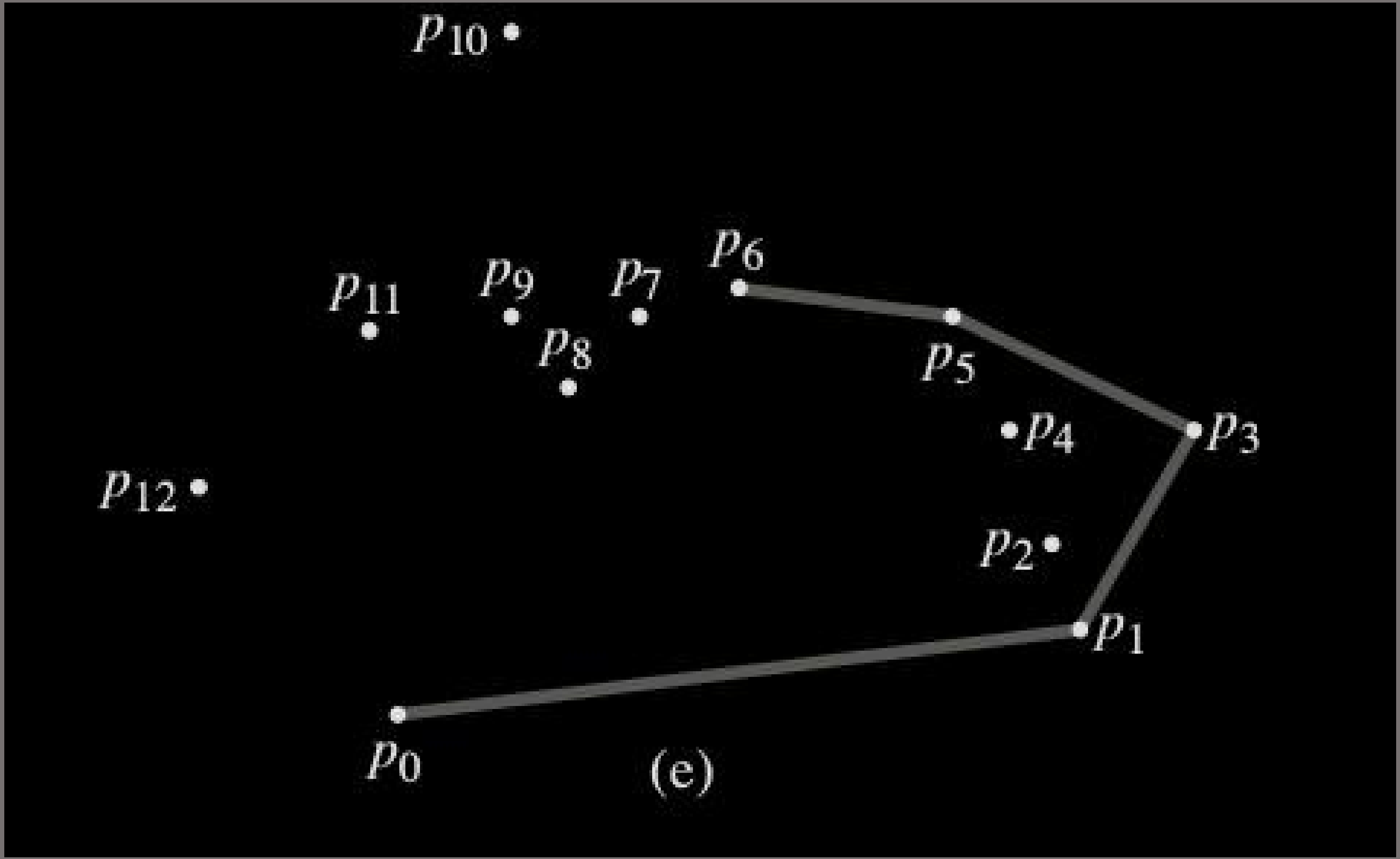
Graham’s scan



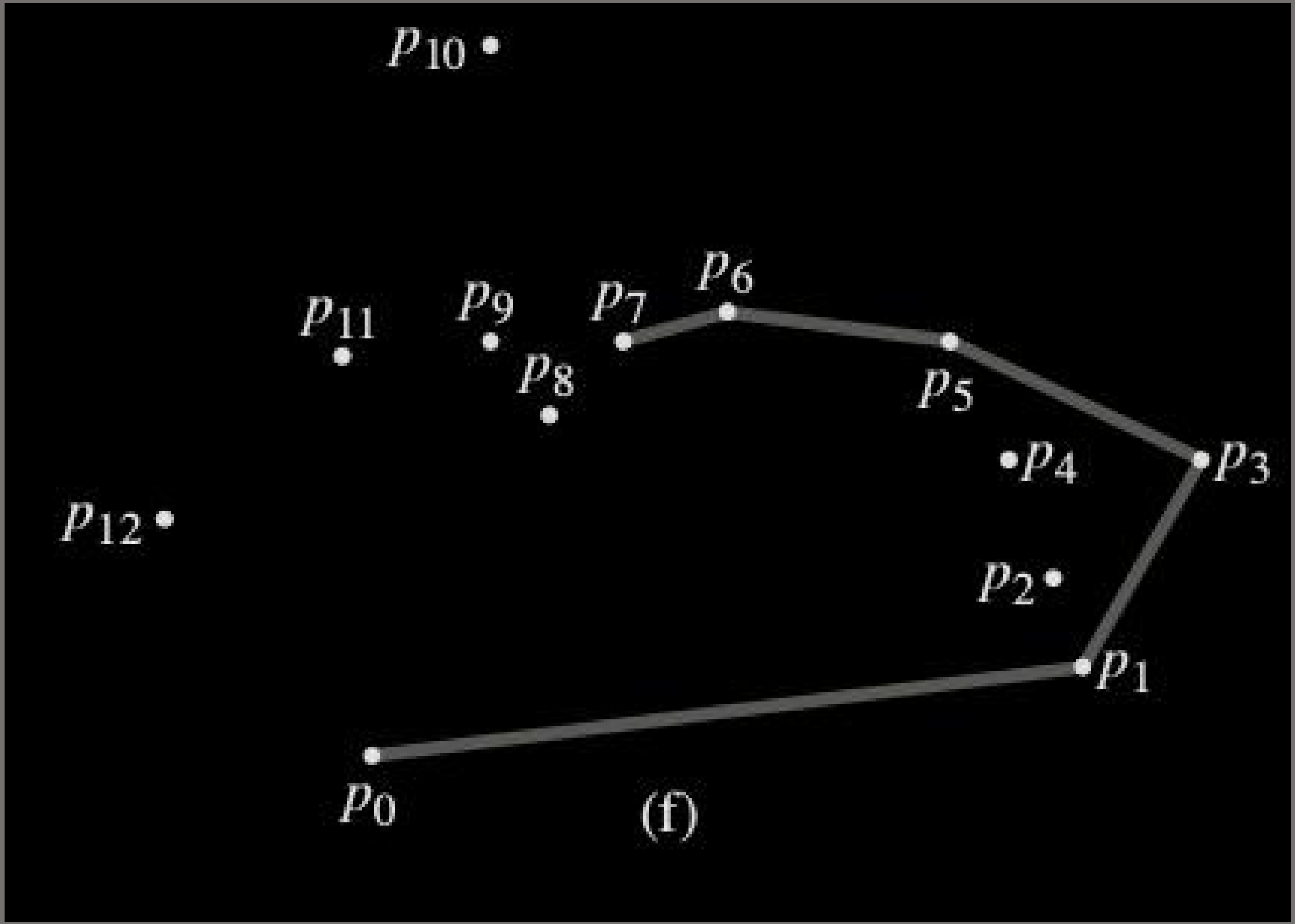
Graham's scan



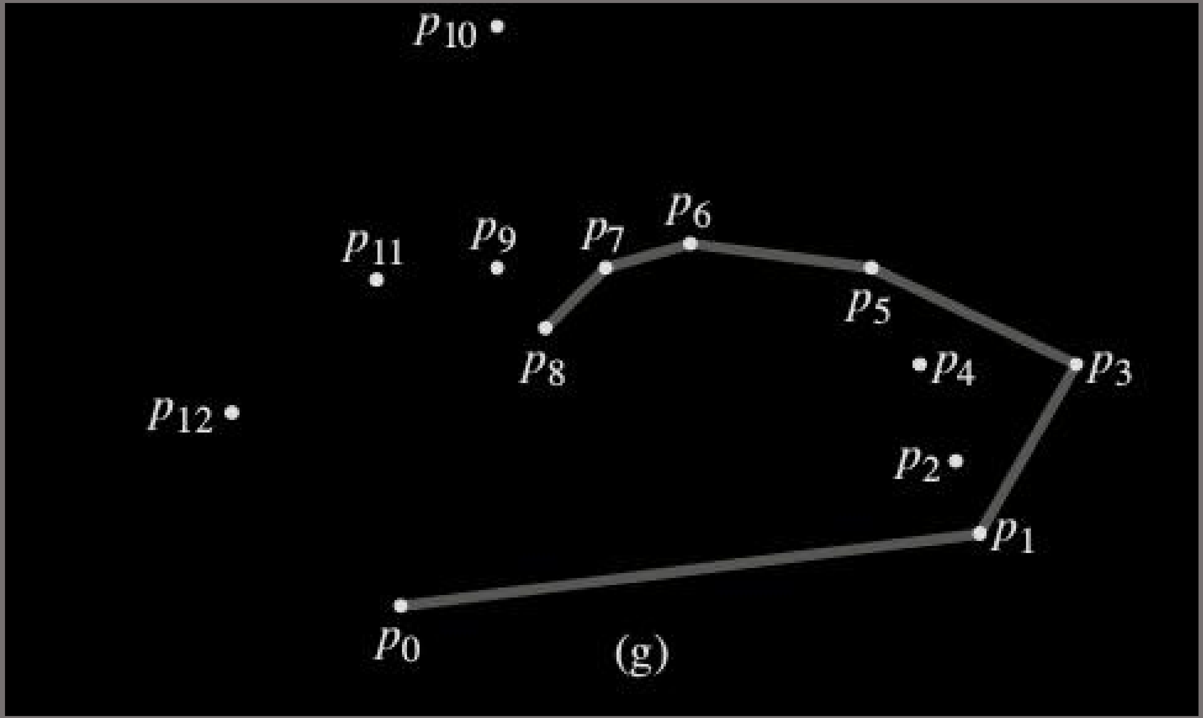
Graham's scan



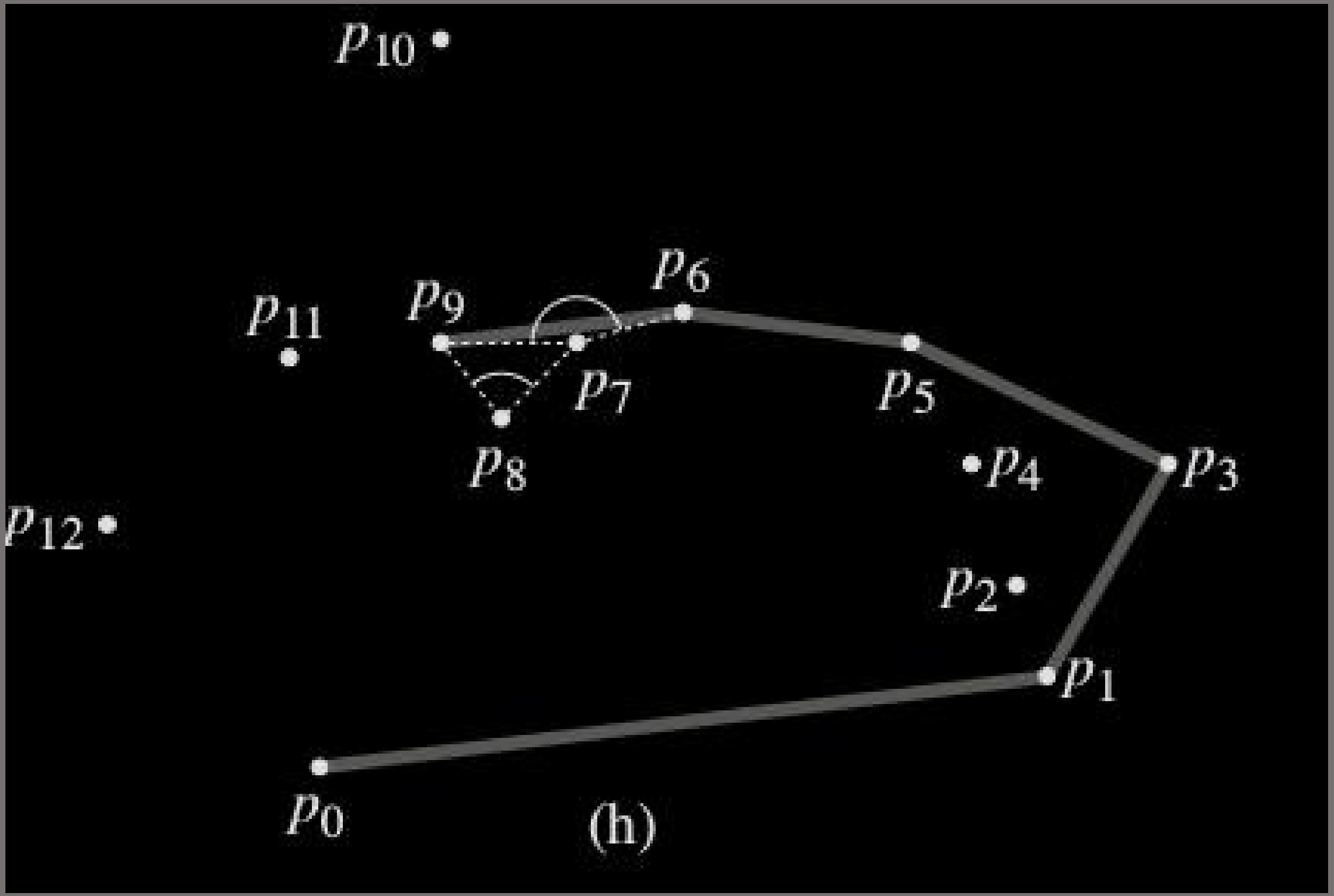
Graham's scan



Graham’s scan

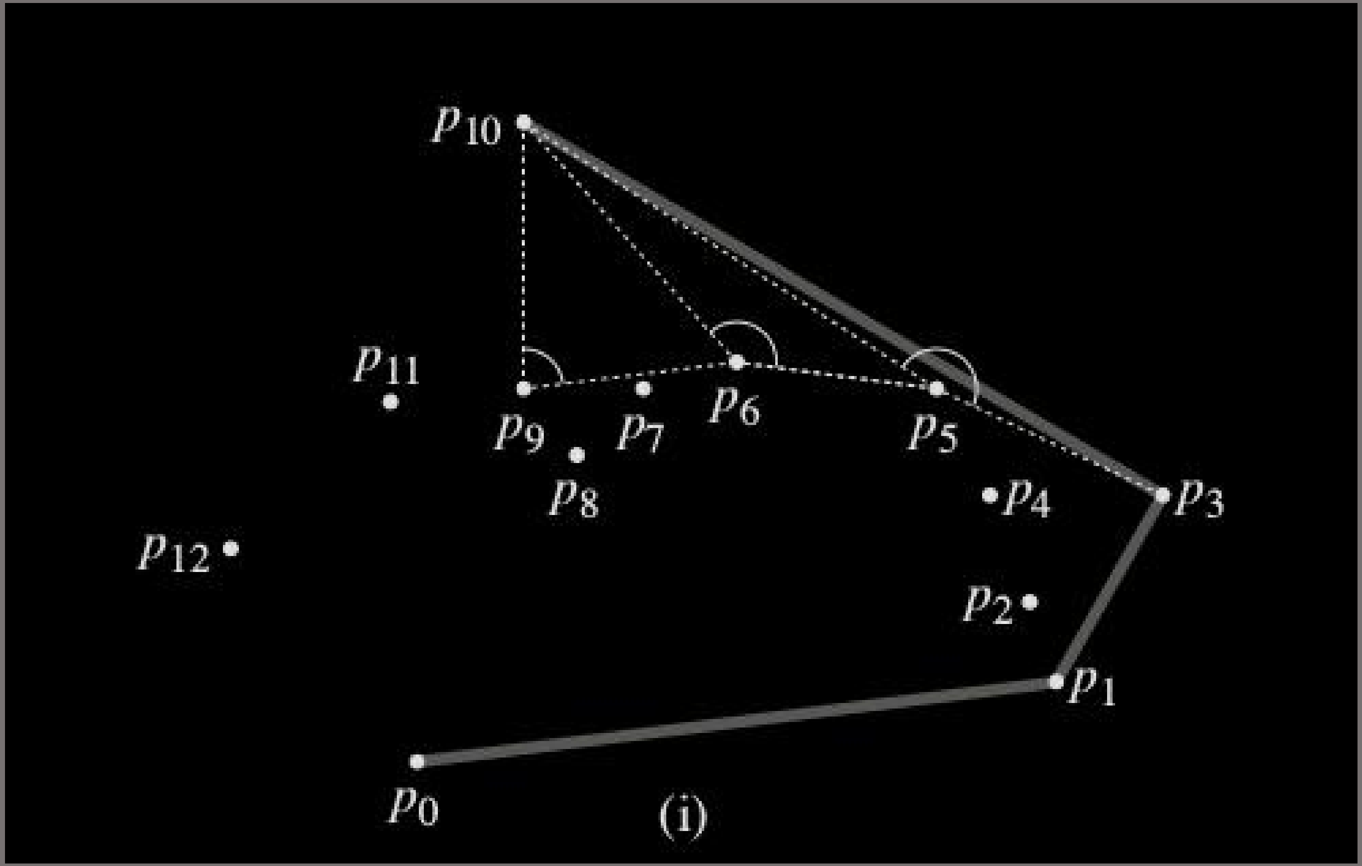


Graham's scan

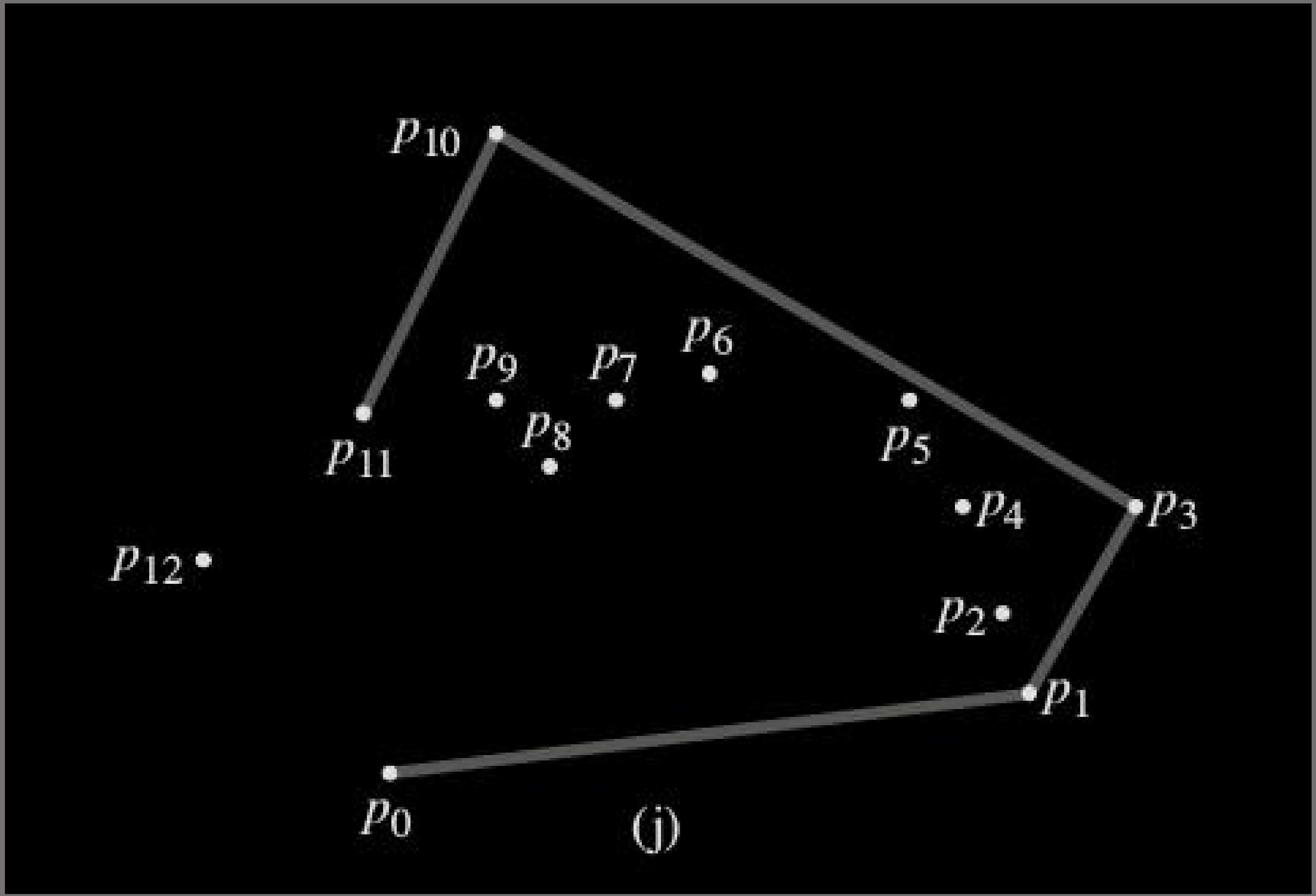


Right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped

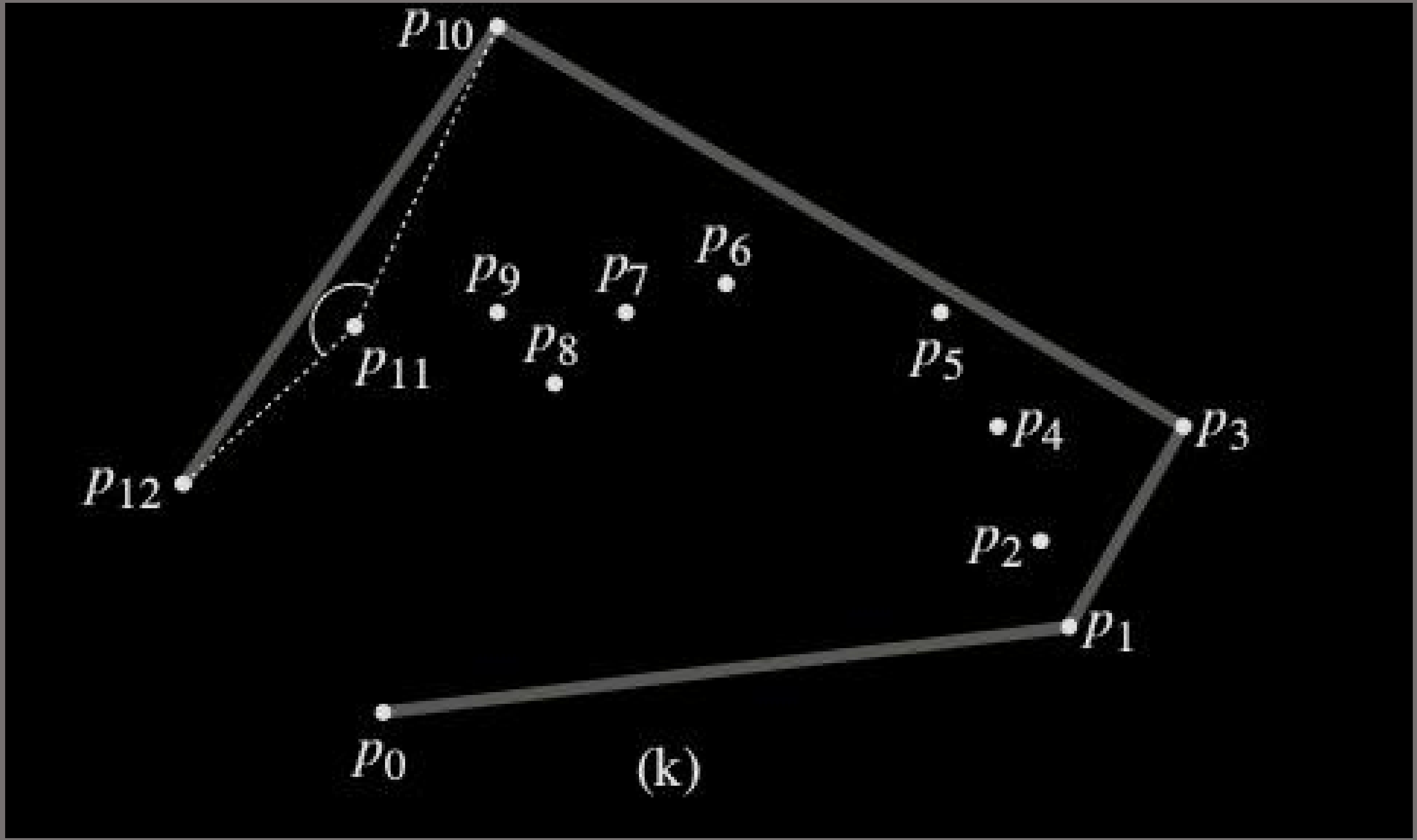
Graham's scan



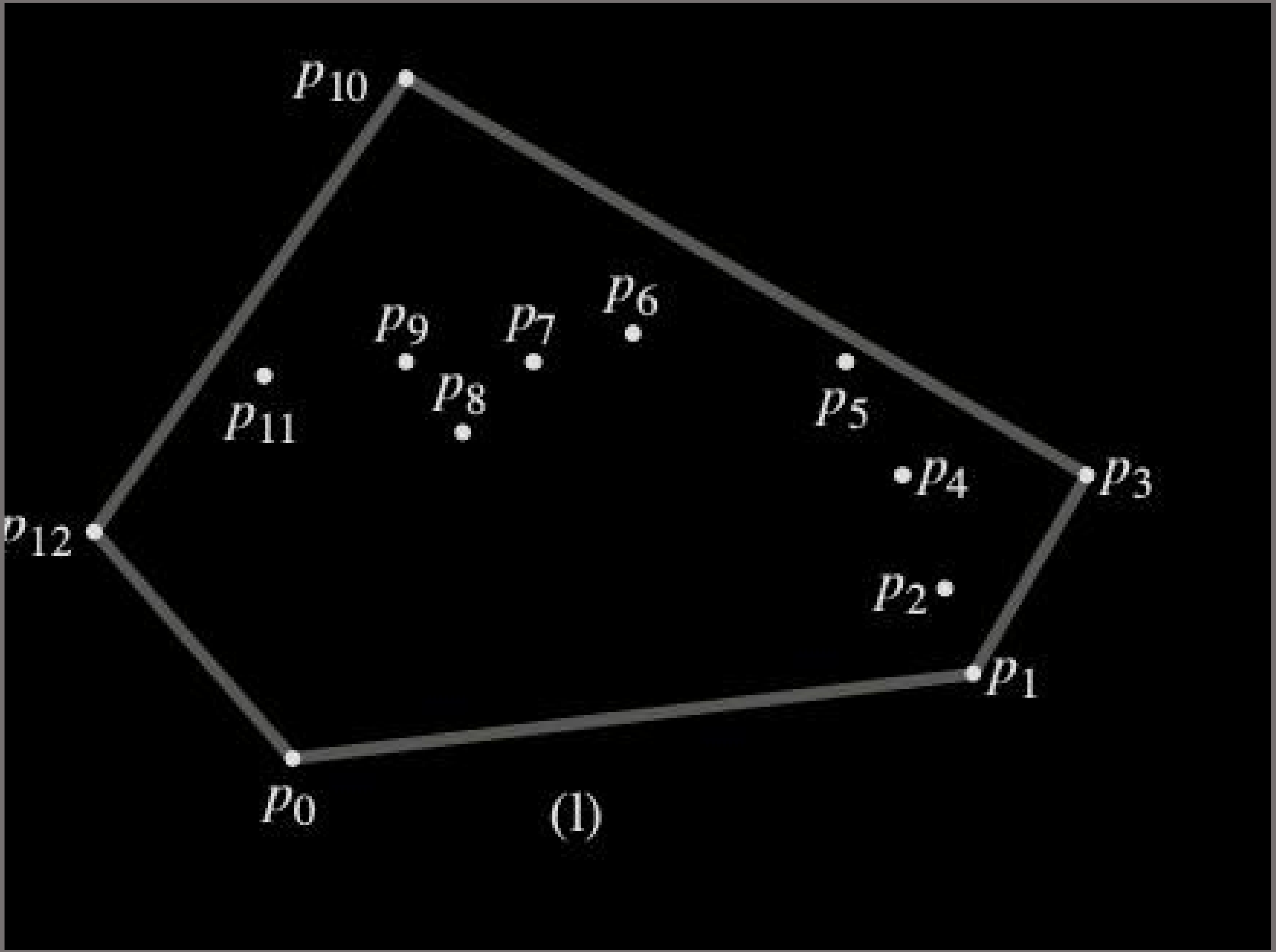
Graham’s scan



Graham's scan



Graham’s scan



Graham's scan

- We designate this sorted sequence of points by $<p_1, p_2, \ldots, p_m>$
- Note that points p_1 and p_m are vertices of CH(Q)
- Figure 33.7(a) shows the points of Figure 33.6 sequentially numbered in order of increasing polar angle relative to p_0
- remainder of the procedure uses the stack S
- Lines 3 – 6 initialize the stack to contain, from bottom to top, the first three points p_0, p_1, and p_2

Graham's scan

- Figure 33.7(a) shows the initial stack S
- The for loop of lines 7 – 10 iterates once for each point in the subsequence $<p_3,p_4,\ldots,p_m>$
- We shall see that after processing point p_i, stack S contains, from bottom to top, the vertices of CH($\{p_0, p_1,\ldots, p_i\}$) in counterclockwise order

Graham's scan

- The while loop of lines 8 – 9 removes points from the stack if we find them not to be vertices of the convex hull.
- When we traverse the convex hull counterclockwise, we should make a left turn at each vertex.
- Thus, each time the while loop finds a vertex at which we make a nonleft turn, we pop the vertex from the stack
- By checking for a nonleft turn, rather than just a right turn, this test precludes the possibility of a straight angle at a vertex of the resulting convex hull.

Graham's scan

- We want no straight angles, since no vertex of a convex polygon may be a convex combination of other vertices of the polygon
- After we pop all vertices that have nonleft turns when heading toward point p_i , we push p i onto the stack.
- Figures 33.7(b) – (k) show the state of the stack S after each iteration of the for loop.
- Finally, G RAHAM –S CAN returns the stack S in line 11
- Figure 33.7(l) shows the corresponding convex hull.

Jarvis's march

- Jarvis's march computes the convex hull of a set Q of points by a technique known as package wrapping (or gift wrapping)
- algorithm runs in time O(nh), where h is the number of vertices of CH(Q)
- When h is o(lg n), Jarvis's march is asymptotically faster than Graham's scan
- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set Q.

Idea of Jarvis's march

- Similar to selection sort – find the leftmost point and add it to the CH
- Till a point in CH is reached
  - find the greatest left turn
  - in case of collinearity, consider the farthest point

Jarvis’s march Algorithm – Step 1

- From the given set of points P, we find a point with minimum x–coordinates (or leftmost point with reference to the x–axis).
- Let’s call this point l.
- Since this point is guaranteed to be in the convex hull, we add this point to the list of convex hull vertices.

Jarvis's march Algorithm – Step 2

- From l, find the leftmost point
- We select the vertex following l and call it q.
- We check if q is turning right from the line joining l and every other point one at a time.
- If q is turning right, we move q to the point from where it was turning right.

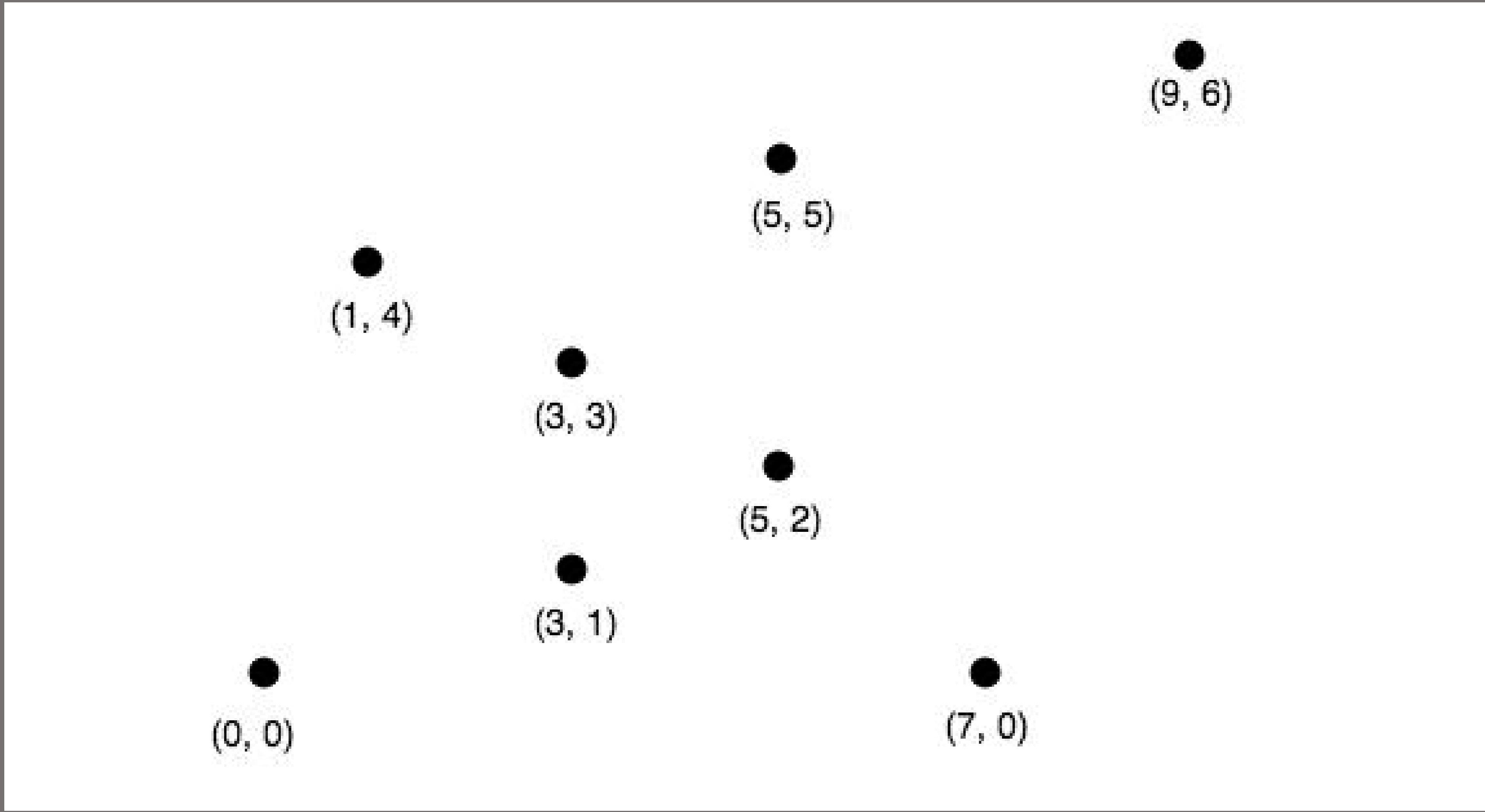
Jarvis’s march Algorithm – Step 2

- This way we move q towards left in each iteration and finally stop when q is in the leftmost position from l.
- We add q to the list of convex hull vertices.

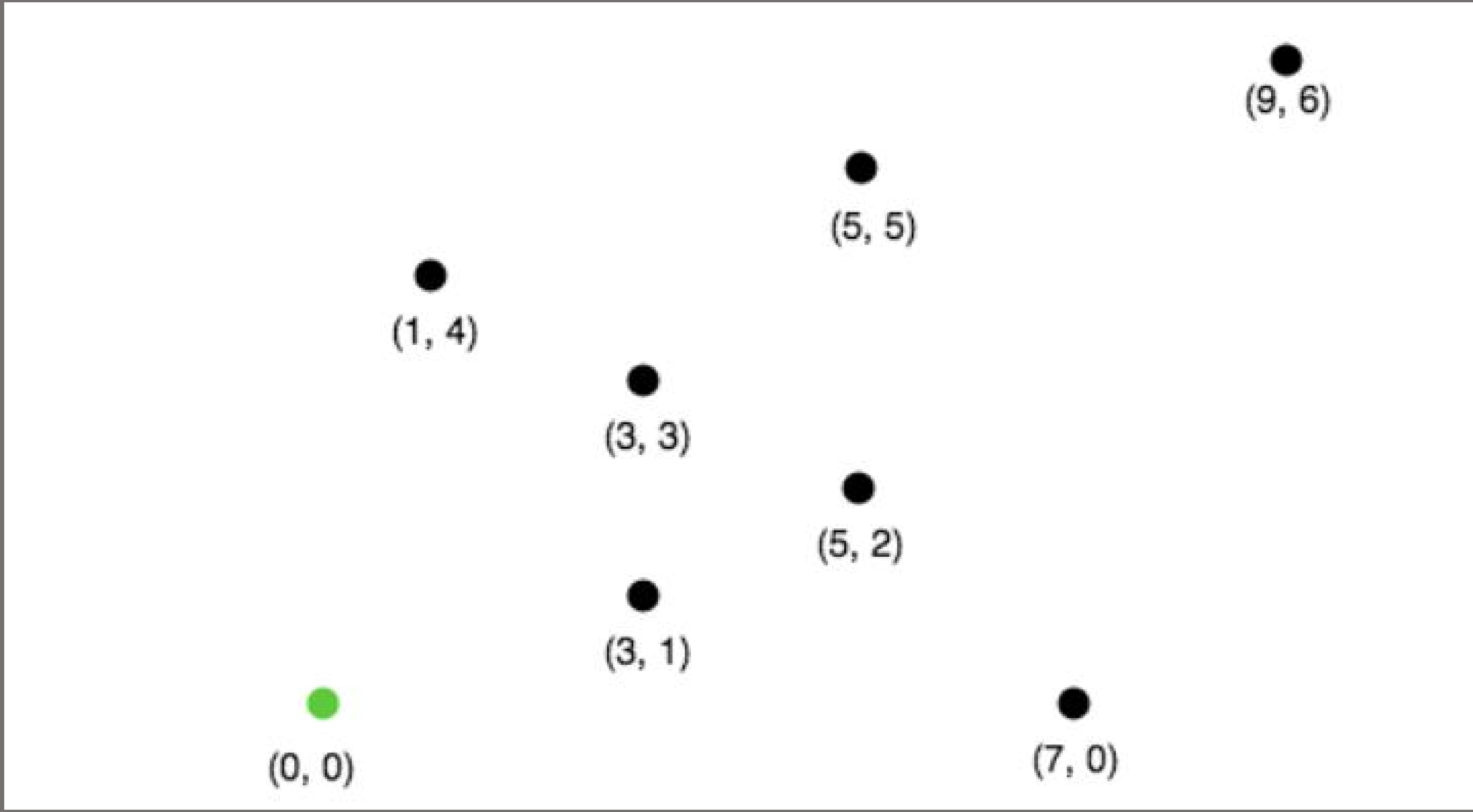
Jarvis’s march Algorithm – Step 3 and 4

- Now q becomes l and we repeat the step (2).
- Repeat step (2) and (3) until we reach the point where we started.

Execution Trace



Find Leftmost Point



- Pick a point following l and call it q.
  Let q be the point (3,3) (You can pick any point, generally we pick next of l in array of points)
  Now we check whether the sequence of points (l,i,q) turns right.
  If it turns right, we replace q by i and repeat the same process for remaining points.

- 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

- Let next i=(3,1).

  The sequence ((0, 0), (3, 1), (3, 3)) turns left and we

  move on without doing anything

- Let next i=(5,2).

  The sequence ((0, 0), (5, 2), (3, 3)) again turns left and

  we move on.

- Next i=(5,5).

  The sequence ((0, 0), (5, 5), (3, 3)) is collinear.

  In the case of collinear, we replace q with i only if distance between l and i is greater than distance between q and l.

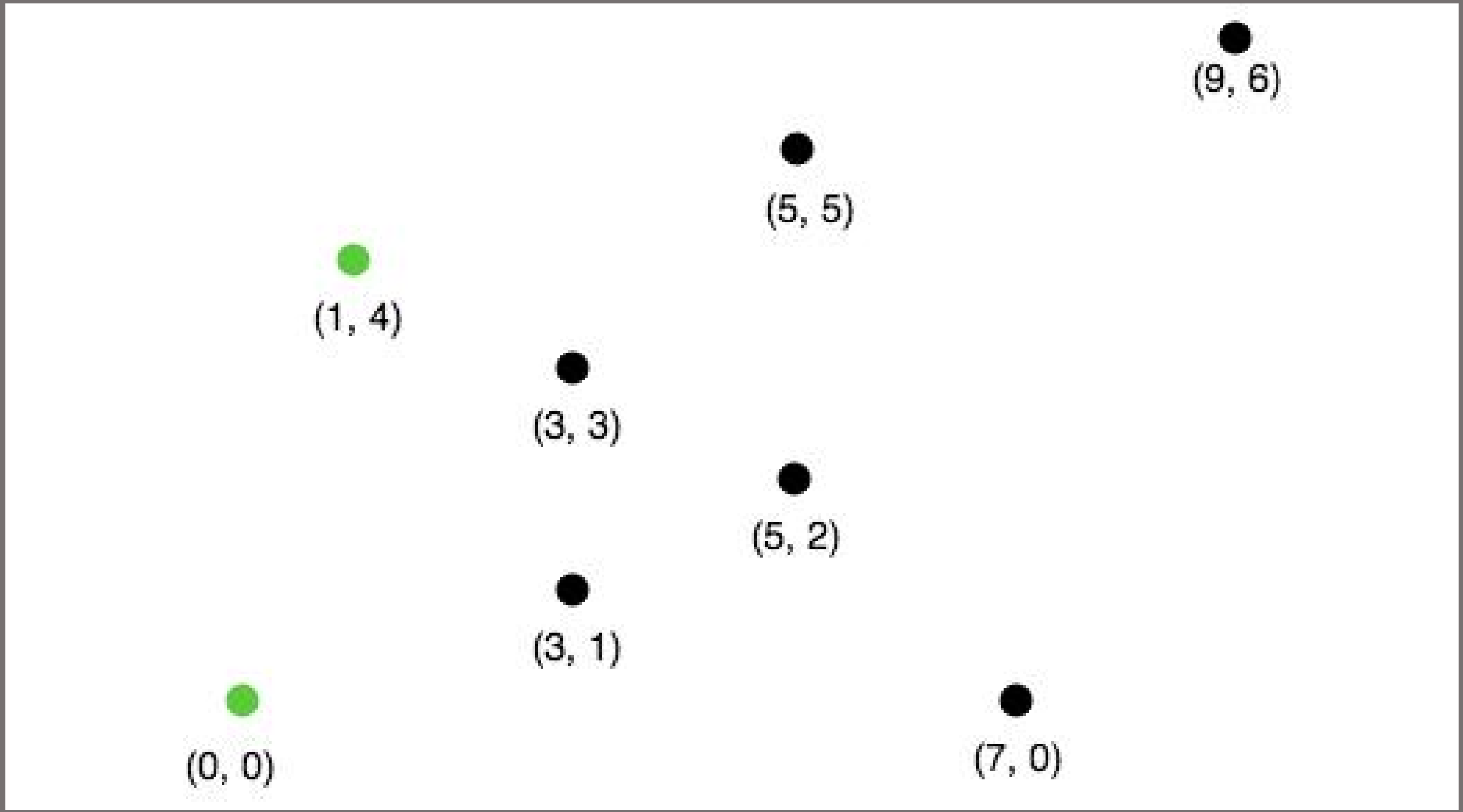
  In this case the distance between (0,0) and (5,5) is greater than the distance between (0,0) and (3,3) we replace q with point (5,5).

- Let next i=(1,4).

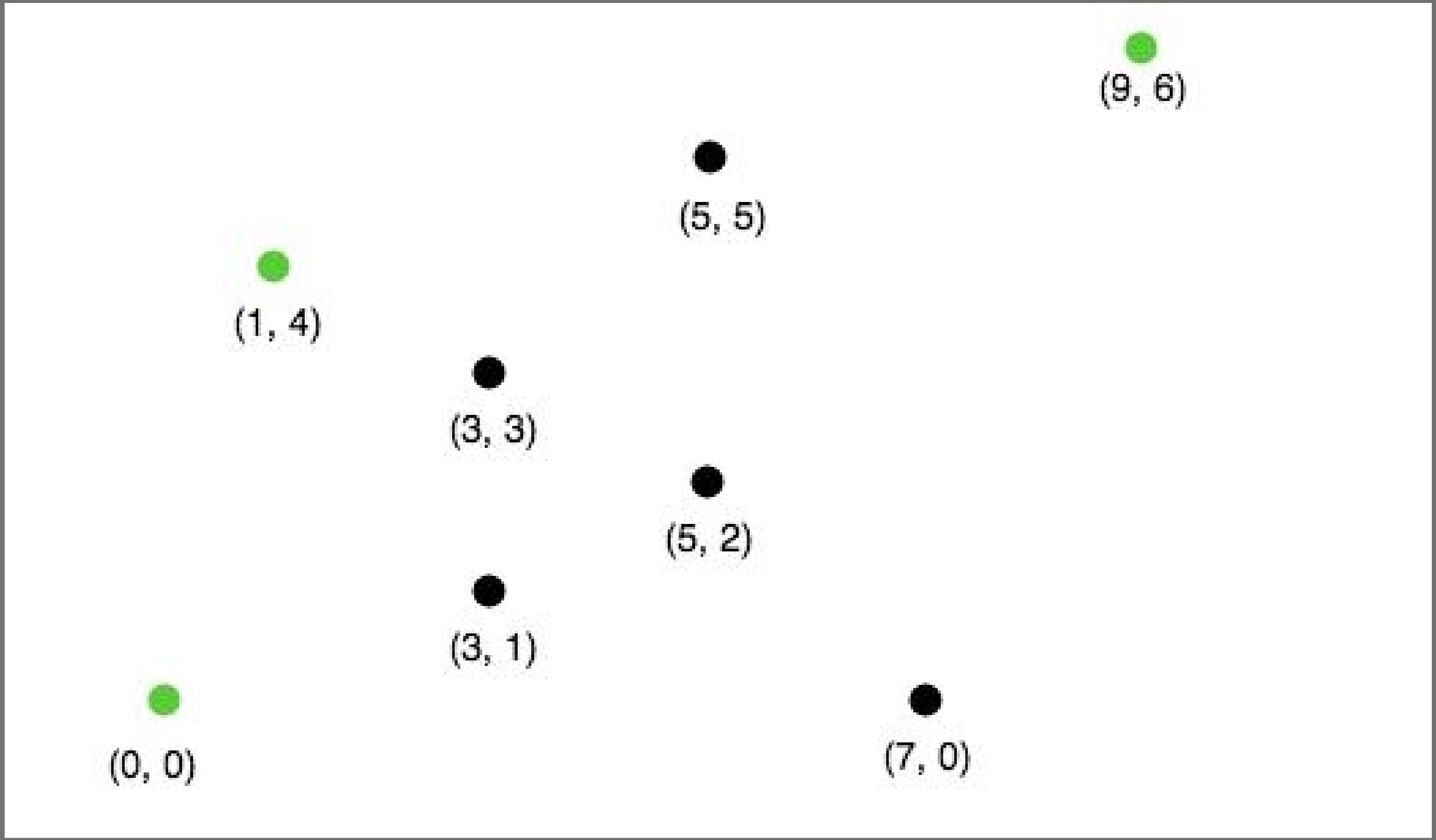
  The sequence ((0, 0), (1, 4), (5, 5)) turns right.

  We replace q by point (1,4).

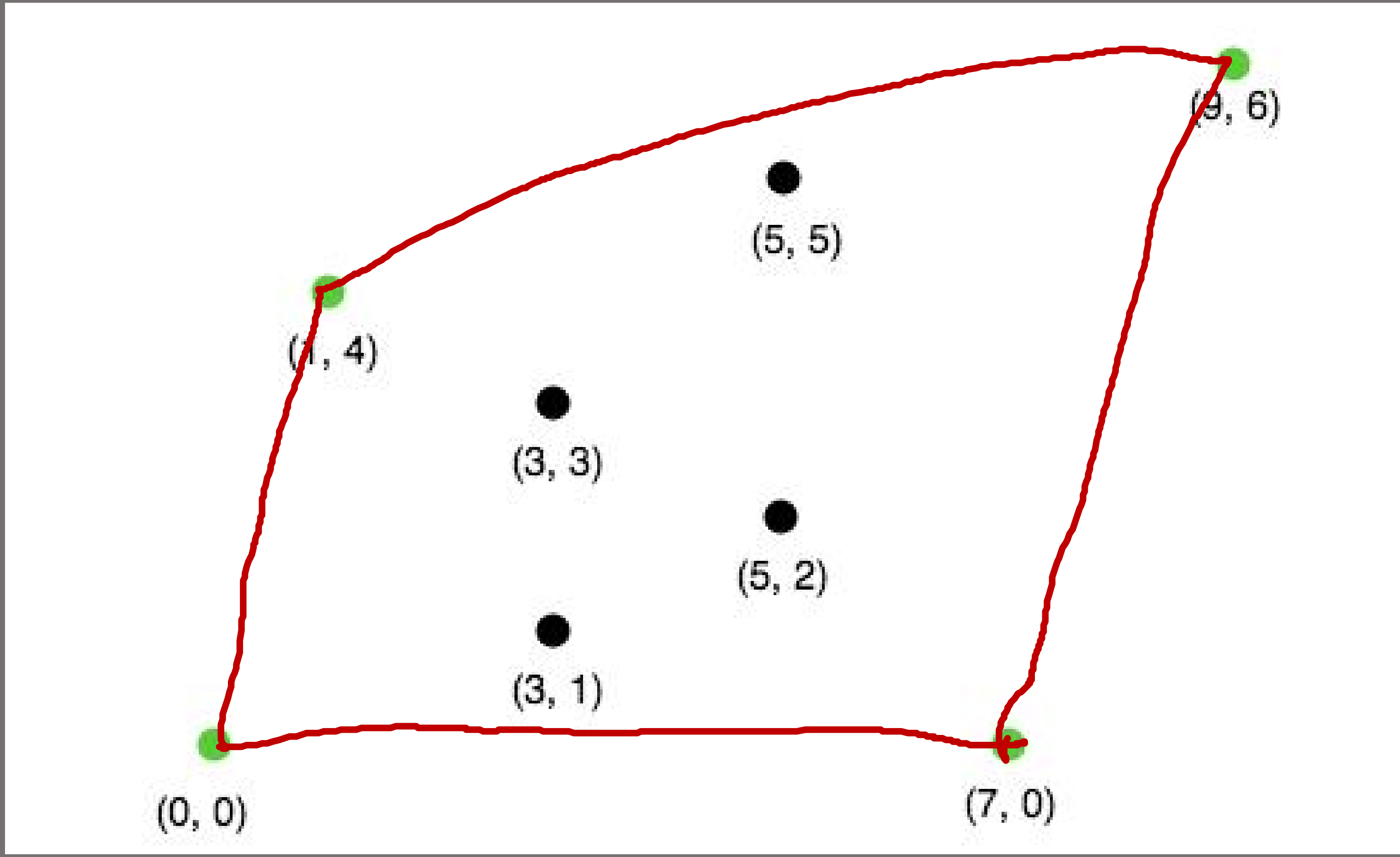
- Finally the only choice for i is (9,6).
  The sequence ((0, 0), (9, 6), (1, 4)) turns left.
  So we do nothing.
  We went through all the points and now q=(1,4) is the left most point.
  add point (1,4) to the convex hull.


(9, 6)
(5, 5)
(1, 4)
(3, 3)
(5, 2)
(3, 1)
(0, 0)
(7, 0)

- Next, we find the leftmost point from the point (1,4) following the steps 1 – 8 mentioned above.
  If we follow all the steps, the leftmost point will be (9,6).


(9, 6)
(5, 5)
(1, 4)
(3, 3)
(5, 2)
(3, 1)
(0, 0)
(7, 0)

Using the same process, the leftmost point from (9,6) will be the point (7,0).

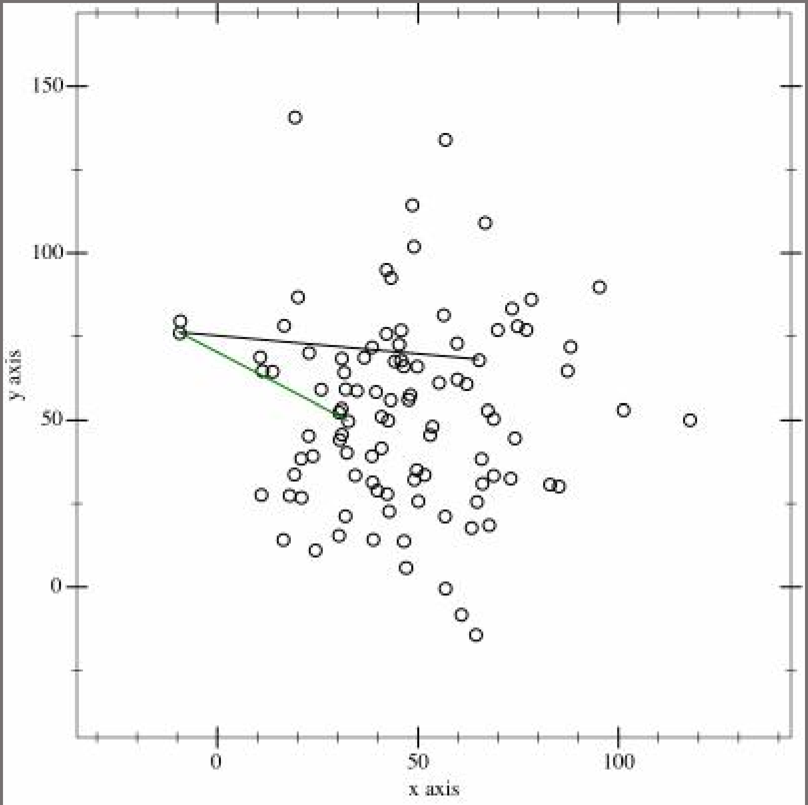


Finally from (7,0) we compute the leftmost point.

The leftmost point from (7,0) will be the point (0, 0).

Since (0,0) is already in the convex hull, the algorithm stops.

- Complexity
- algorithm spends O(n) time on each convex hull vertex.
- If there are h convex hull vertices, the total time complexity of the algorithm would be O(nh).
- Since h is the number of output of the algorithm, this algorithm is also called output sensitive algorithm since the complexity also depends on the number of output.


150
100
50
0
y axis
0
50
100
x axis