Convex Hulls

Outline

I. Convex sets

II. Convex hull properties

III. Graham scan

IV. Correctness and running time

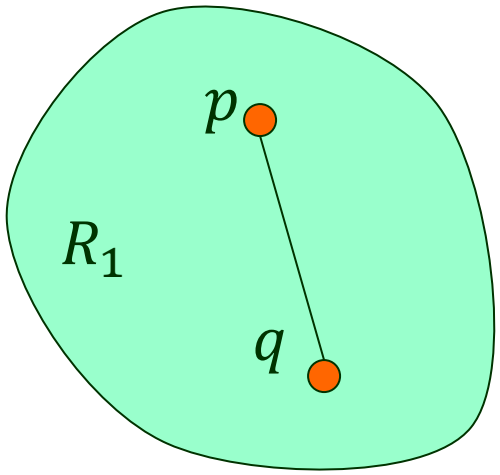
V. Javis' march

I. Convex Sets & Concave Sets

A planar region R is called *convex* if and only if for any pair of points p, q in R, the line segment $\overline{pq}$ lies *completely* in R.

I. Convex Sets & Concave Sets

A planar region R is called *convex* if and only if for any pair of points p, q in R, the line segment $\overline{pq}$ lies *completely* in R.

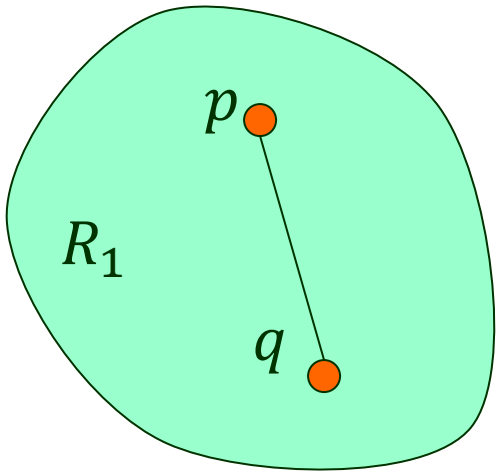


Convex

I. Convex Sets & Concave Sets

A planar region R is called *convex* if and only if for any pair of points p, q in R, the line segment $\overline{pq}$ lies *completely* in R.

Otherwise, it is called *concave*.

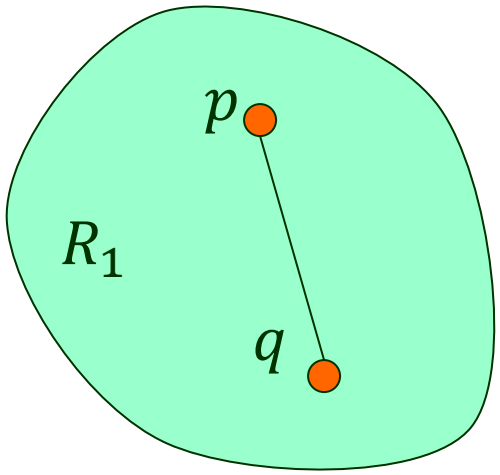


Convex

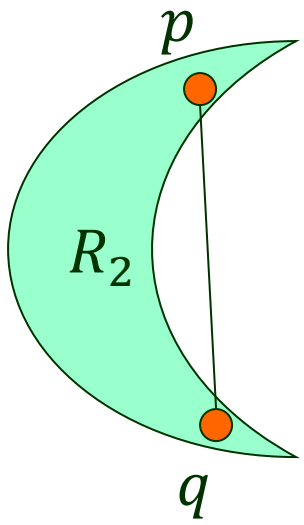
I. Convex Sets & Concave Sets

A planar region R is called *convex* if and only if for any pair of points p, q in R, the line segment $\overline{pq}$ lies *completely* in R.

Otherwise, it is called *concave*.

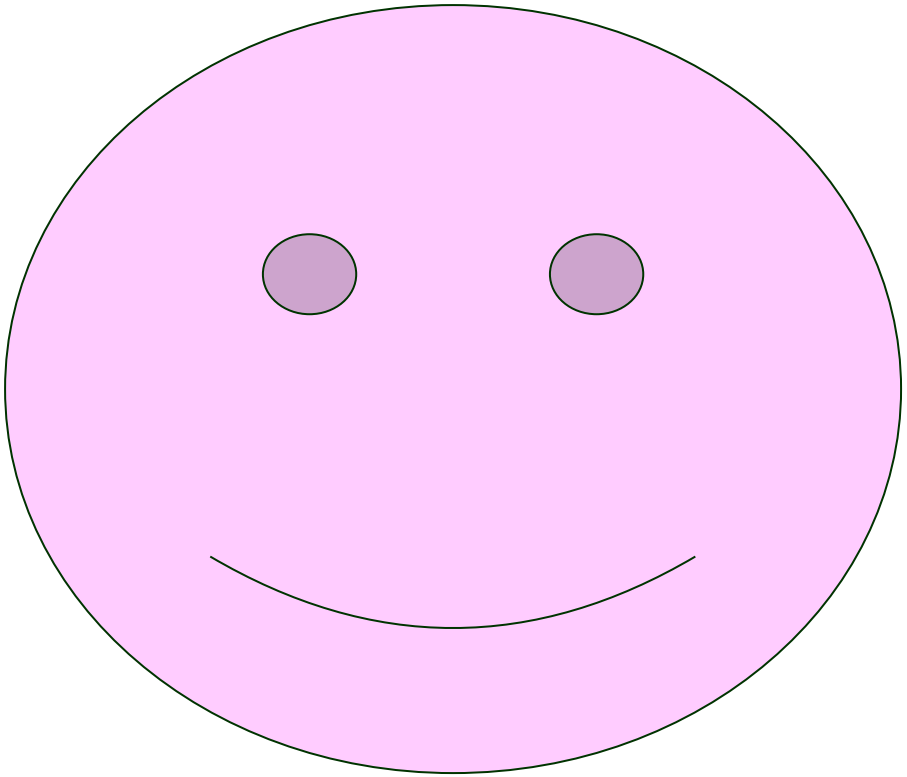


Convex

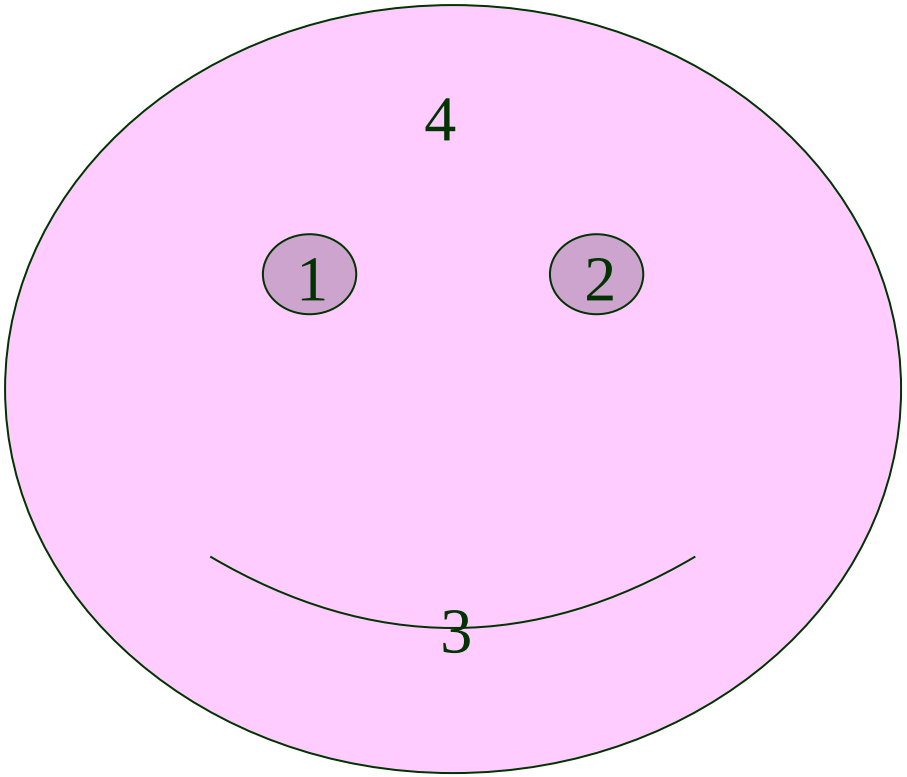


Concave

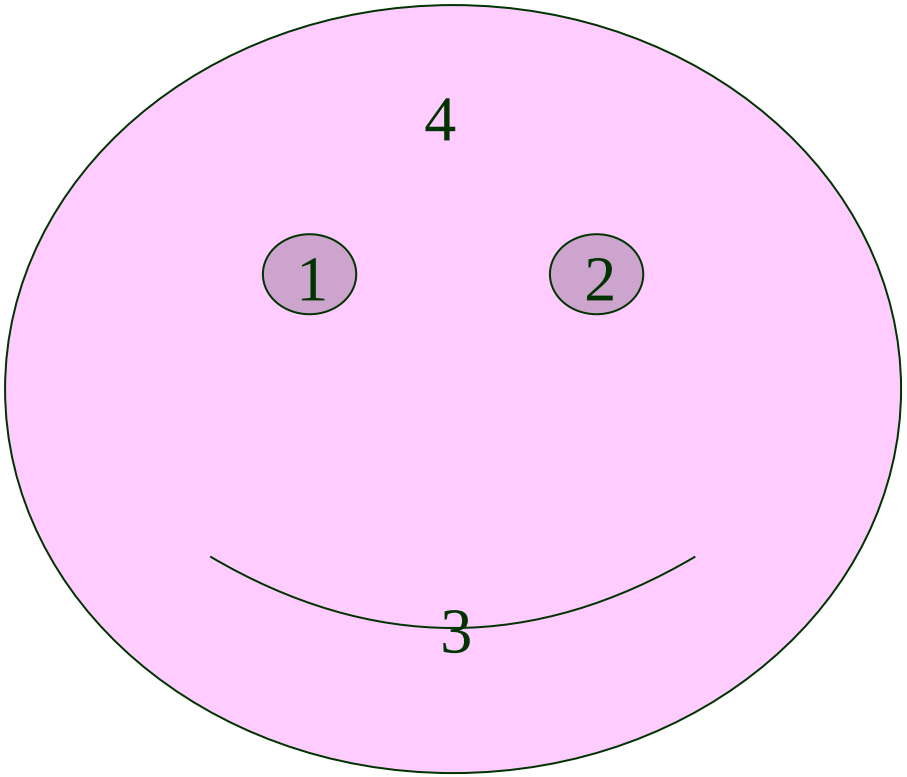
An Example



An Example



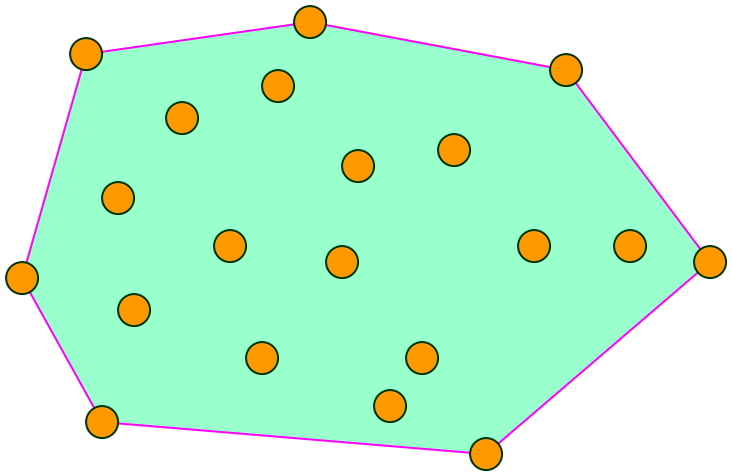
An Example



Regions 1 & 2: convex
Regions 3 & 4: concave

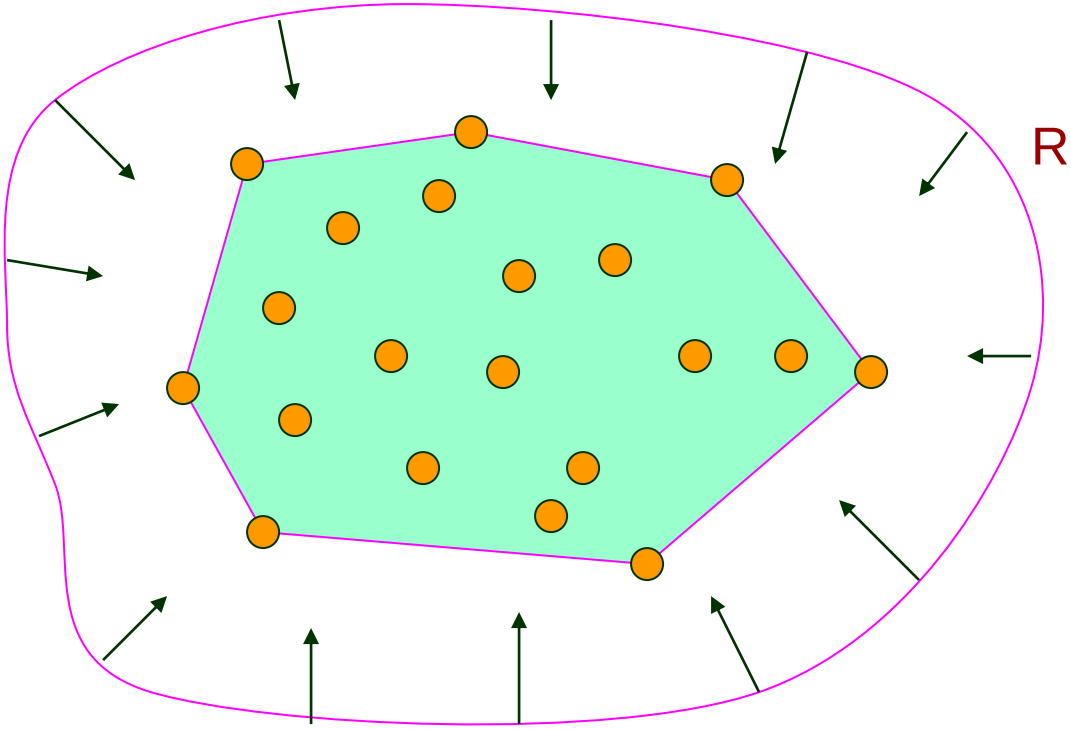
Convex Hull

The *convex hull* CH(Q) of a set Q is the *smallest* convex region that contains Q.



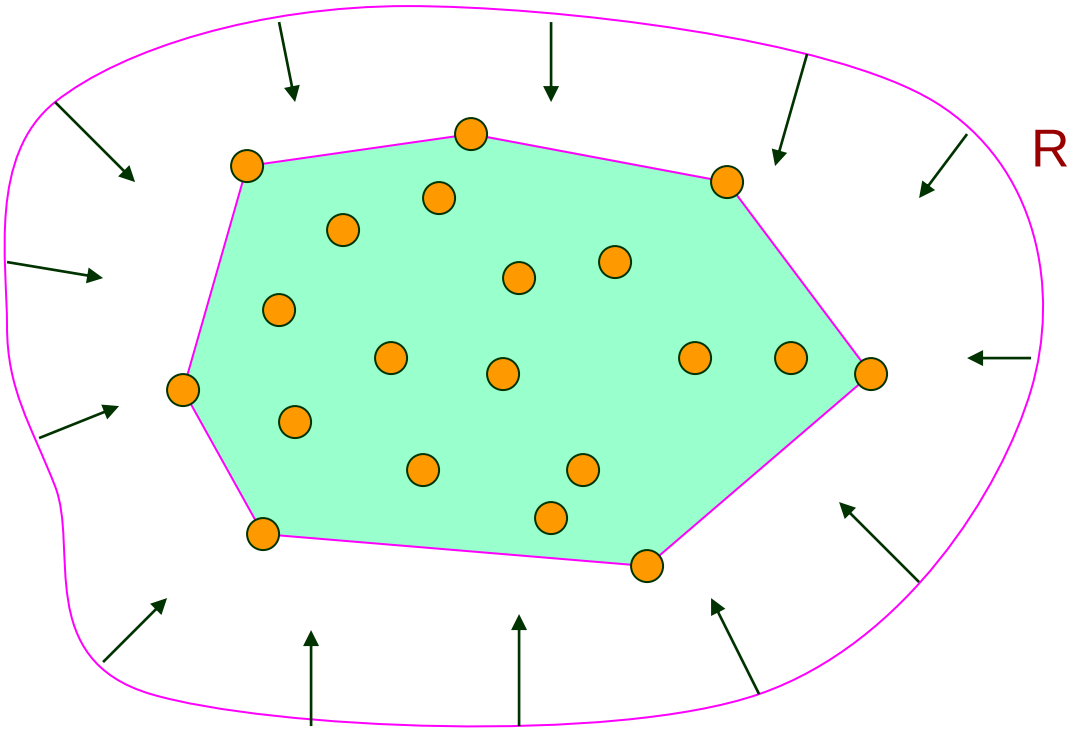
Convex Hull

The *convex hull* CH(Q) of a set Q is the *smallest* convex region that contains Q.



Convex Hull

The *convex hull* CH(Q) of a set Q is the *smallest* convex region that contains Q.



When Q is finite, its convex hull is the unique *convex polygon* whose vertices are from Q and that contains all points of Q.

Degenerate Cases

- The convex hull of a single point is itself.

- The convex hull of several collinear points is the line segment joining the two extreme points.

II. The Convex Hull Problem

Input: a set $P = \{ p_1 , p_2 , \ldots , p_n \}$ of points

II. The Convex Hull Problem

Input: a set $P = \{ p_1, p_2, \ldots, p_n \}$ of points

Output: a list of vertices of CH(P) in *counterclockwise* order.

II. The Convex Hull Problem

Input: a set $P = \{ p_1, p_2, \ldots, p_n \}$ of points

Output: a list of vertices of CH(P) in *counterclockwise* order.

(direction of traversal about the outward axis with the interior on the left)

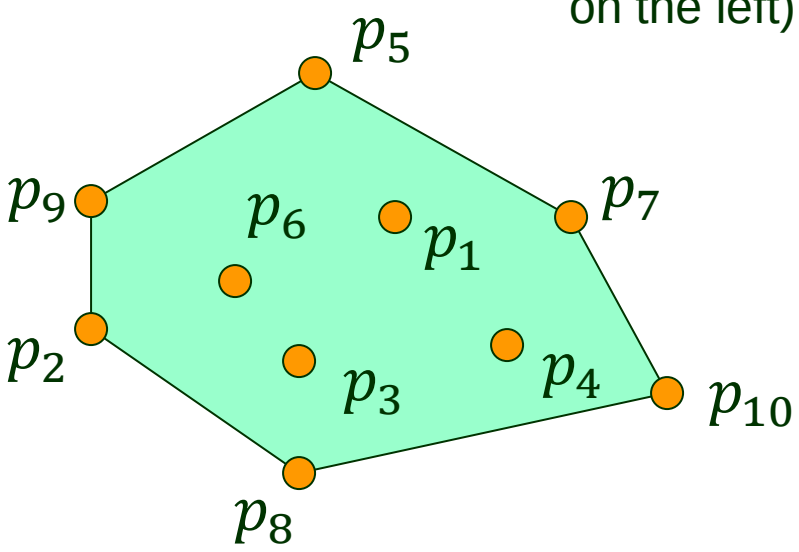
II. The Convex Hull Problem

Input: a set $P = \{ p_1, p_2, \ldots, p_n \}$ of points

Output: a list of vertices of CH(P) in *counterclockwise* order.

(direction of traversal about the outward axis with the interior on the left)

Example



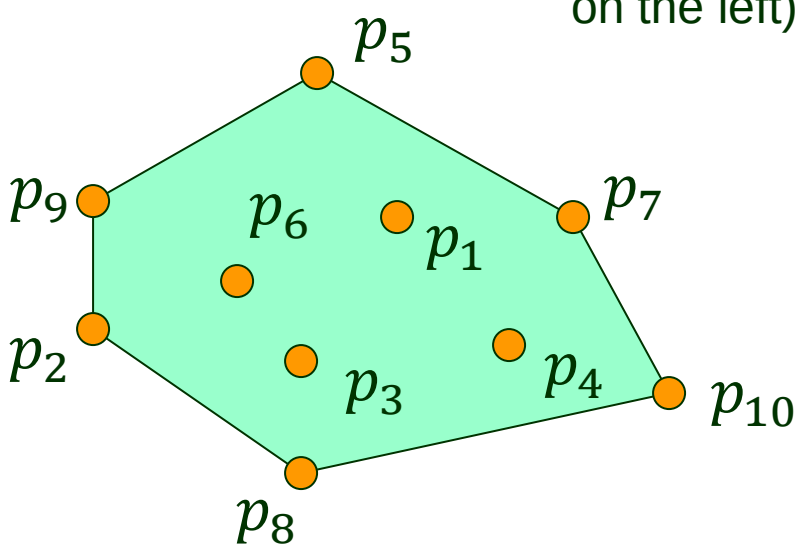
II. The Convex Hull Problem

Input: a set $P = \{ p_1, p_2, \ldots, p_n \}$ of points

Output: a list of vertices of CH(P) in *counterclockwise* order.

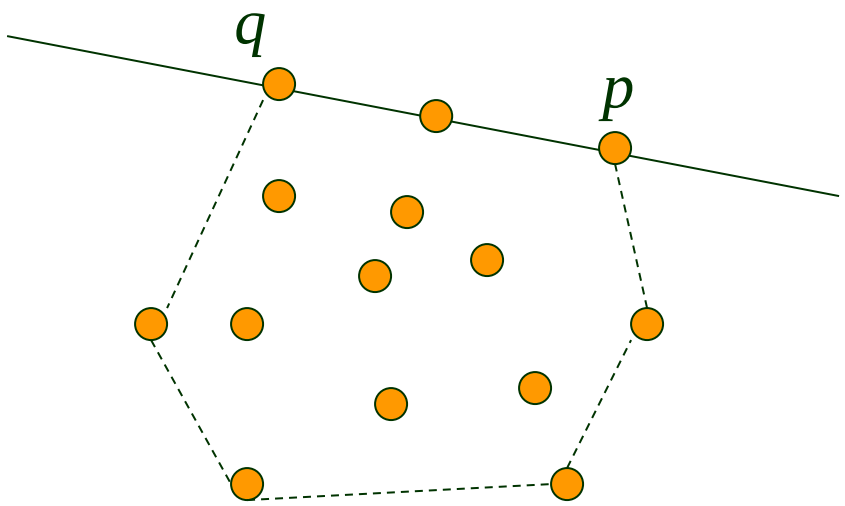
(direction of traversal about the outward axis with the interior on the left)

Example



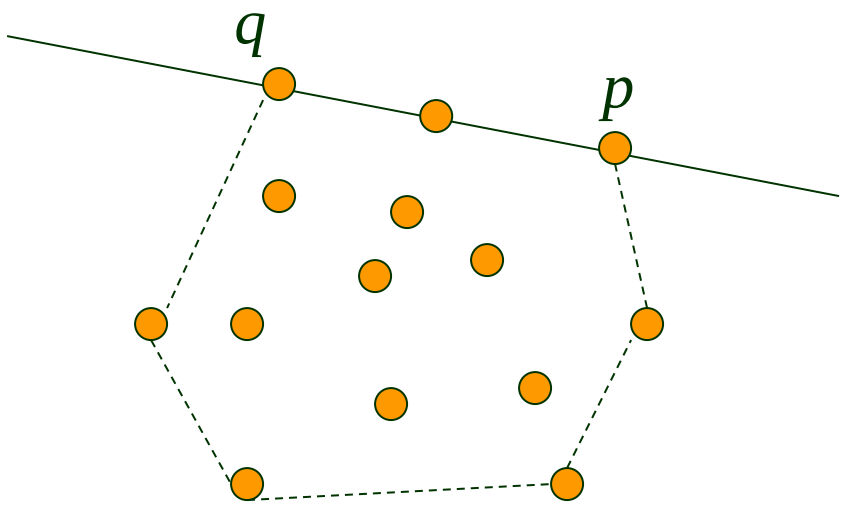
Output: $p_5, p_9, p_2, p_8, p_{10}, p_7.$

Edges of a Convex Hull



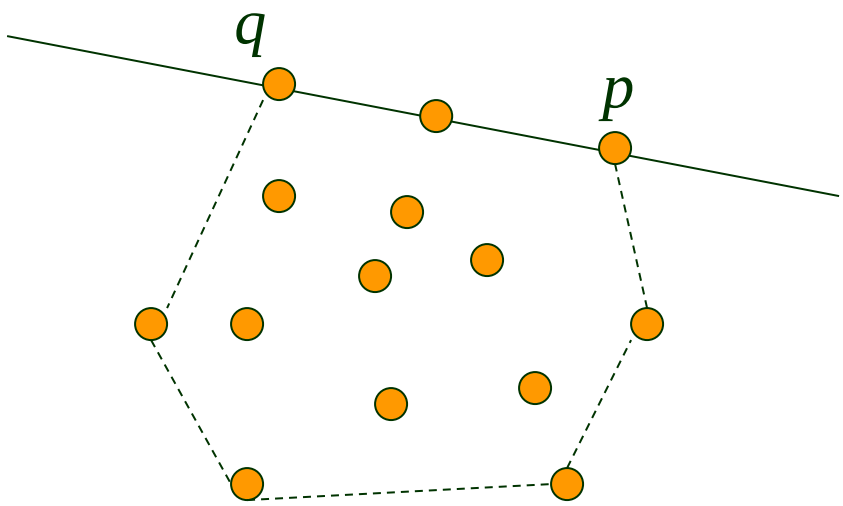
Edges of a Convex Hull

- For every edge with endpoints $p, q \in P$.



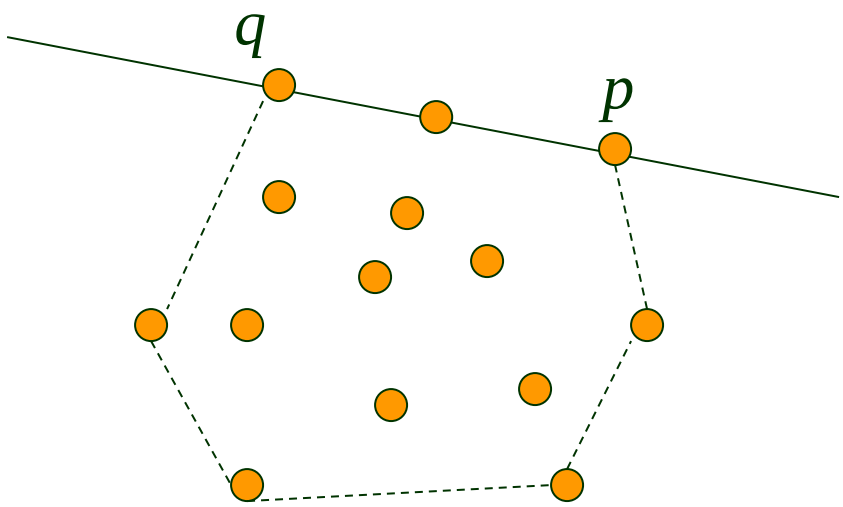
Edges of a Convex Hull

- For every edge with endpoints $p, q \in P$.
- All other points in P lie
  - to the same side of the line passing through p and q, or



Edges of a Convex Hull

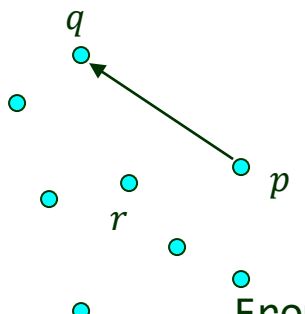
- For every edge with endpoints $p, q \in P$.
- All other points in P lie
  - to the same side of the line passing through p and q, or
  - between p and q if on the line.



A Slow Convex Hull Algorithm

```
Slow-Convex-Hull(P)
      E ≠ {} // set of directed edges of CH(P) that bounds the
                  // points of P on the right.
      for every ordered pair (p,q), where p,q ∈ P and p ≠ q // Θ(n²) pairs
        do valid ← true
           for every point r ≠ p or q                    // n − 2 such points
              do if r lies to the right of  pq⃗ or
                    collinear with p and q but not on pq
                 then valid ← false
              if valid
              then E ≠ E ∪ {pq⃗}    // pq⃗ and qp⃗ cannot be both in E
      From E construct a list L of vertices of CH(P), sorted in
          counterclockwise order.   // O(n²) improvable to O(n log n)
      return L
```

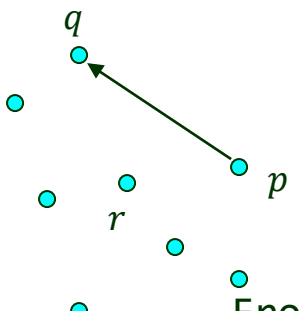
A Slow Convex Hull Algorithm

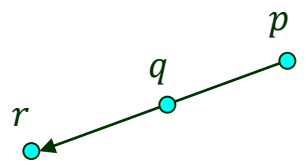



```
Slow-Convex-Hull(P)
    E ≠ {} // set of directed edges of CH(P) that bounds the
                // points of P on the right.
    for every ordered pair (p,q), where p,q ∈ P and p ≠ q // Θ(n²) pairs
      do valid ← true
         for every point r ≠ p or q                    // n − 2 such points
            do if r lies to the right of  pq⃗ or
                  collinear with p and q but not on pq
               then valid ← false
         if valid
         then E ≠ E ∪ {pq⃗}    // pq⃗ and qp⃗ cannot be both in E
    From E construct a list L of vertices of CH(P), sorted in
        counterclockwise order.   // O(n²) improvable to O(n log n)
    return L
```

A Slow Convex Hull Algorithm

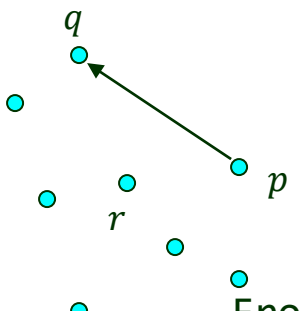
```
Slow-Convex-Hull(P)
    E ≠ {} // set of directed edges of CH(P) that bounds the
                // points of P on the right.
    for every ordered pair (p,q), where p,q ∈ P and p ≠ q // Θ(n²) pairs
      do valid ← true
         for every point r ≠ p or q                        // n − 2 such points
            do if r lies to the right of  pq or
                  collinear with p and q but not on pq
               then valid ← false
         if valid
         then E ≠ E ∪ {pq}    // pq and qp cannot be both in E
    From E construct a list L of vertices of CH(P), sorted in
        counterclockwise order.   // O(n²) improvable to O(n log n)
    return L
```

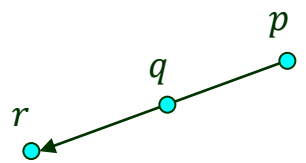




A Slow Convex Hull Algorithm

```
Slow-Convex-Hull(P)
    E ≠ {} // set of directed edges of CH(P) that bounds the
                // points of P on the right.
    for every ordered pair (p,q), where p,q ∈ P and p ≠ q // Θ(n²) pairs
      do valid ← true
         for every point r ≠ p or q                      // n − 2 such points
            do if r lies to the right of  pq or
                  collinear with p and q but not on pq
               then valid ← false
         if valid
         then E ≠ E ∪ {pq}    // pq and qp cannot be both in E
    From E construct a list L of vertices of CH(P), sorted in
        counterclockwise order.   // O(n²) improvable to O(n log n)
    return L
```

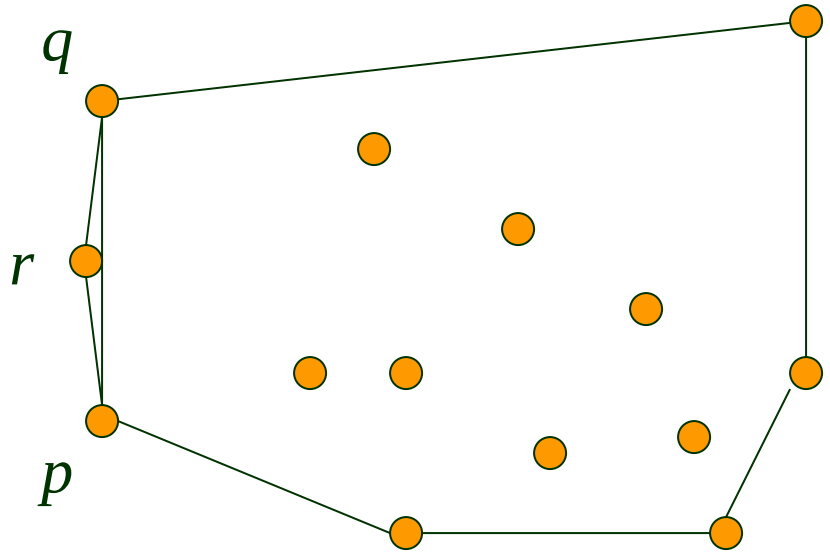




Running time $\Theta(n^3)$

Floating Arithmetic is not Exact!

Nearly colinear points p, q, r.



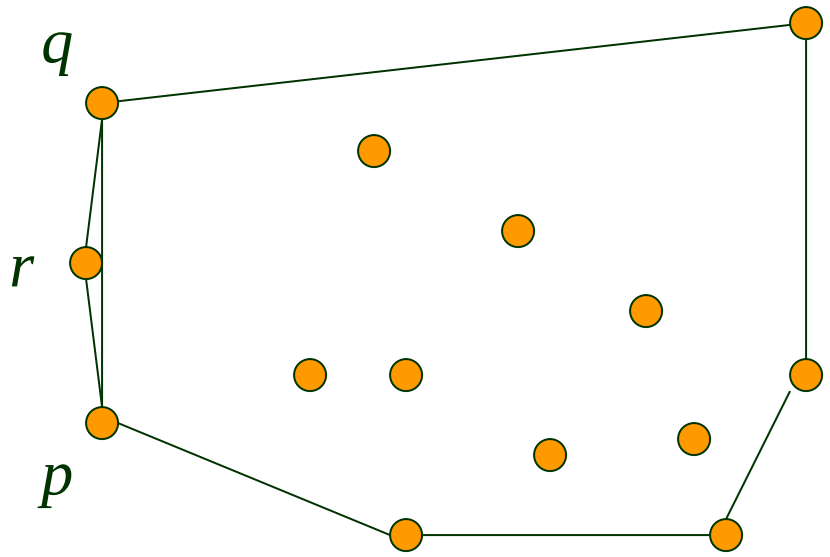
p to the left of $\overrightarrow{qr}$.

q to the left of $\overrightarrow{rp}$.

r to the left of $\overrightarrow{qp}$.

Floating Arithmetic is not Exact!

Nearly colinear points p, q, r.

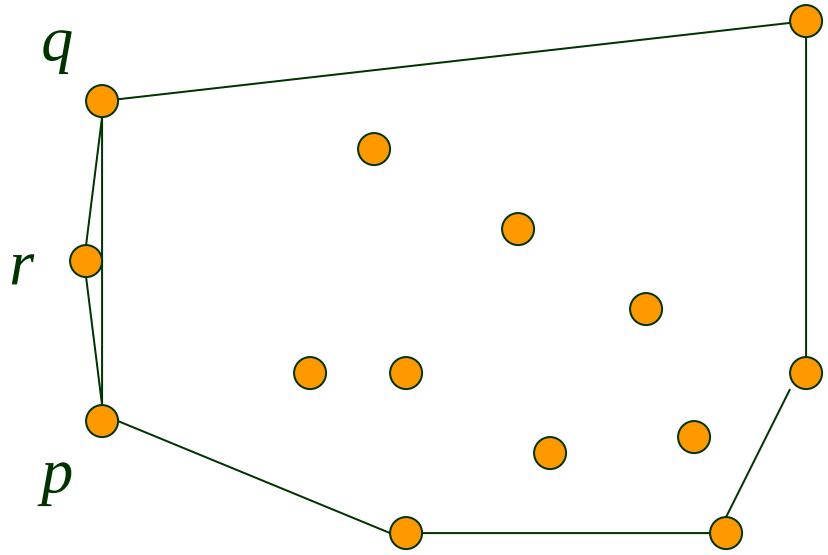


p to the left of $\overrightarrow{qr}$.
q to the left of $\overrightarrow{rp}$.
r to the left of $\overrightarrow{qp}$.

All three are accepted as edges!

Floating Arithmetic is not Exact!

Nearly colinear points p, q, r.

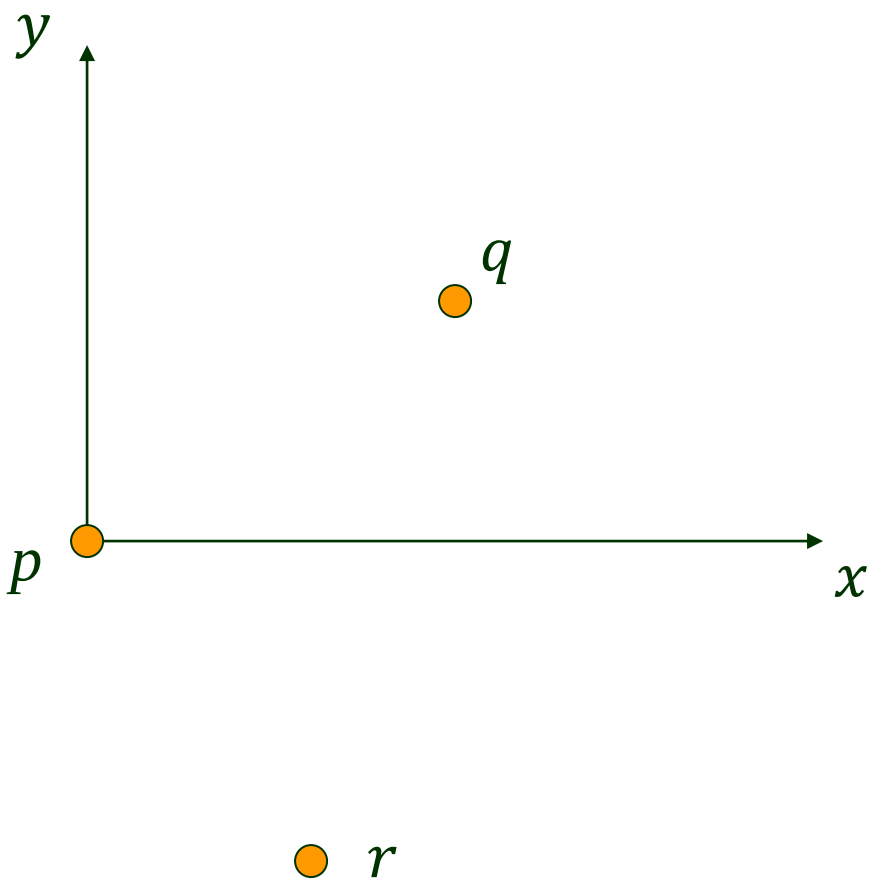


p to the left of $\overrightarrow{qr}$.
q to the left of $\overrightarrow{rp}$.
r to the left of $\overrightarrow{qp}$.

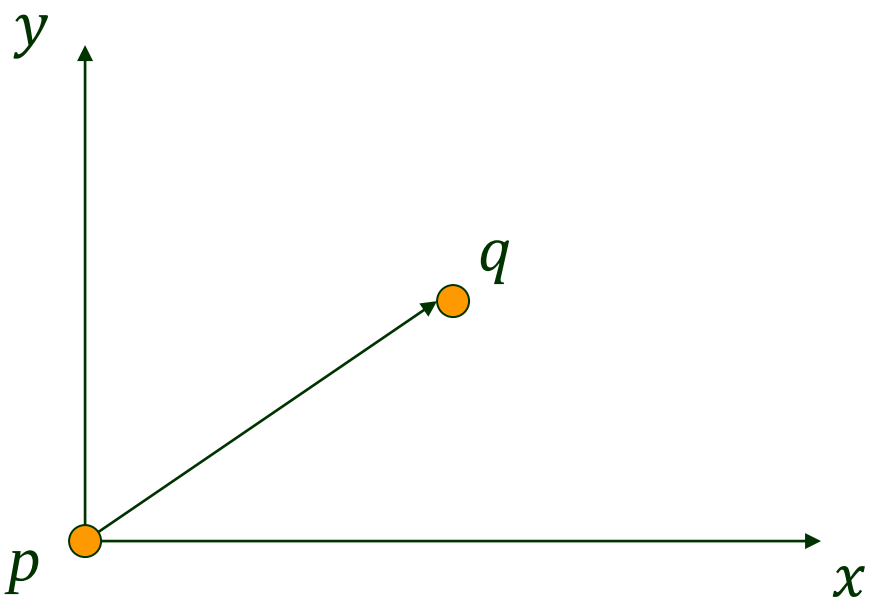
→ All three are accepted as edges!

Not *robust* – the algorithm could fail with small numerical error.

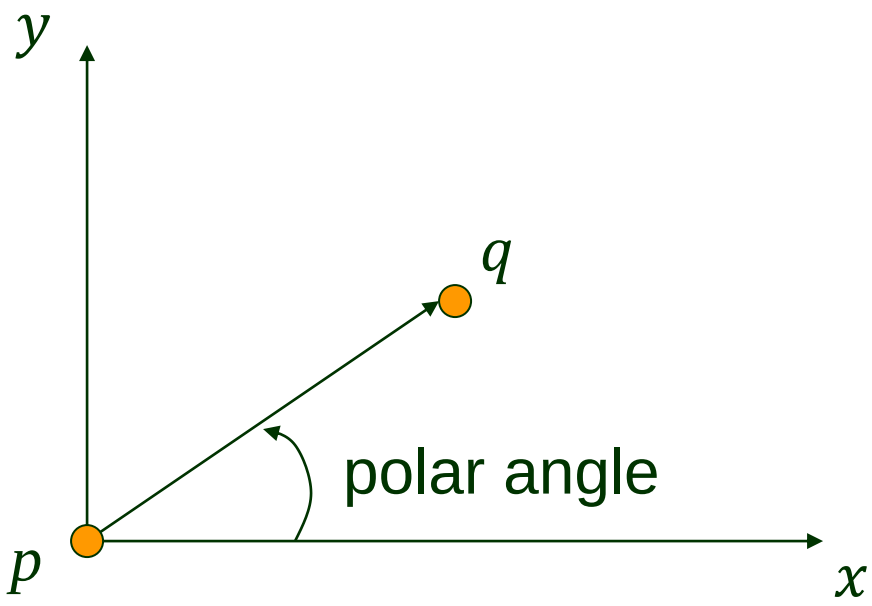
Polar Angle



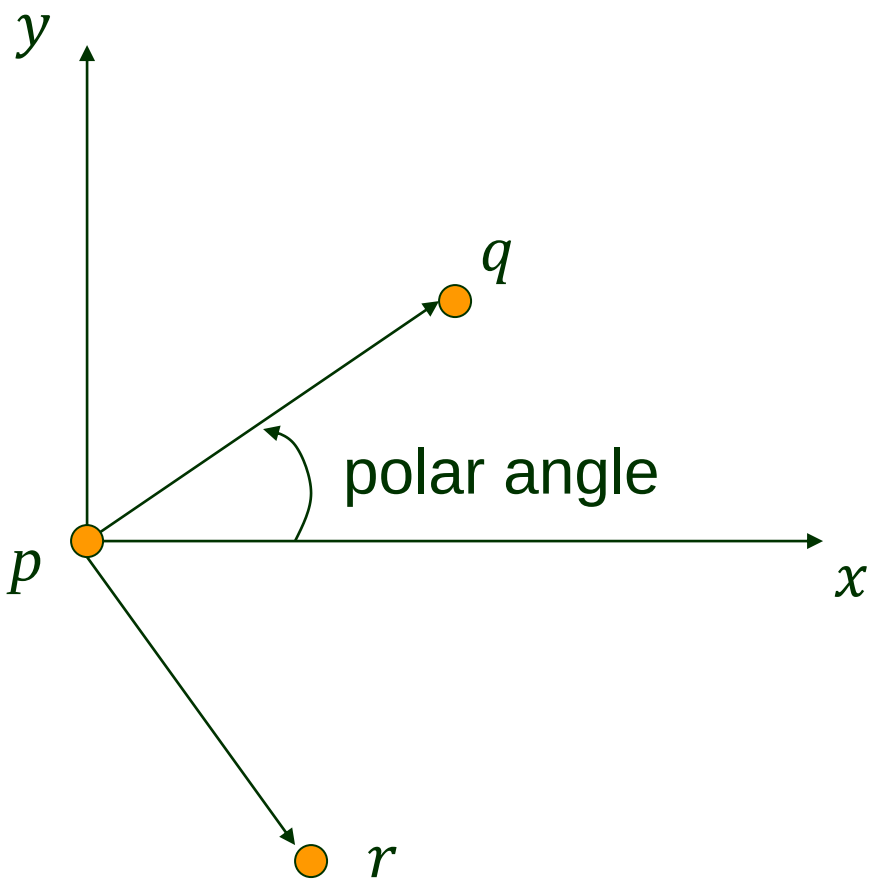
Polar Angle



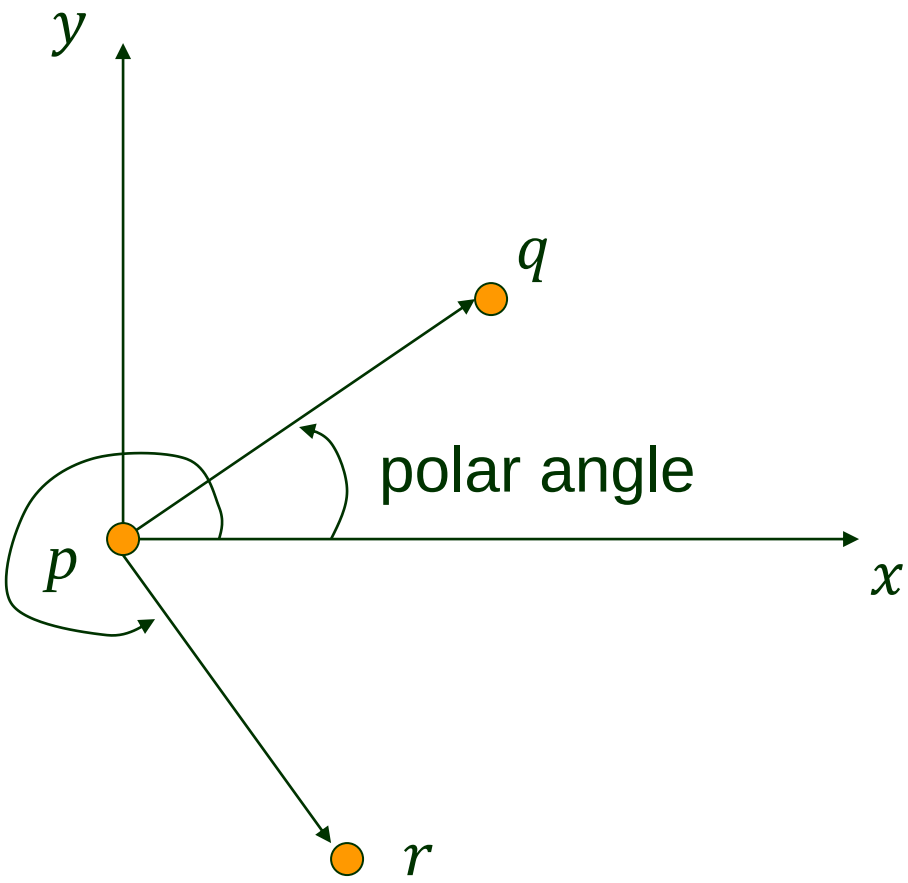
Polar Angle



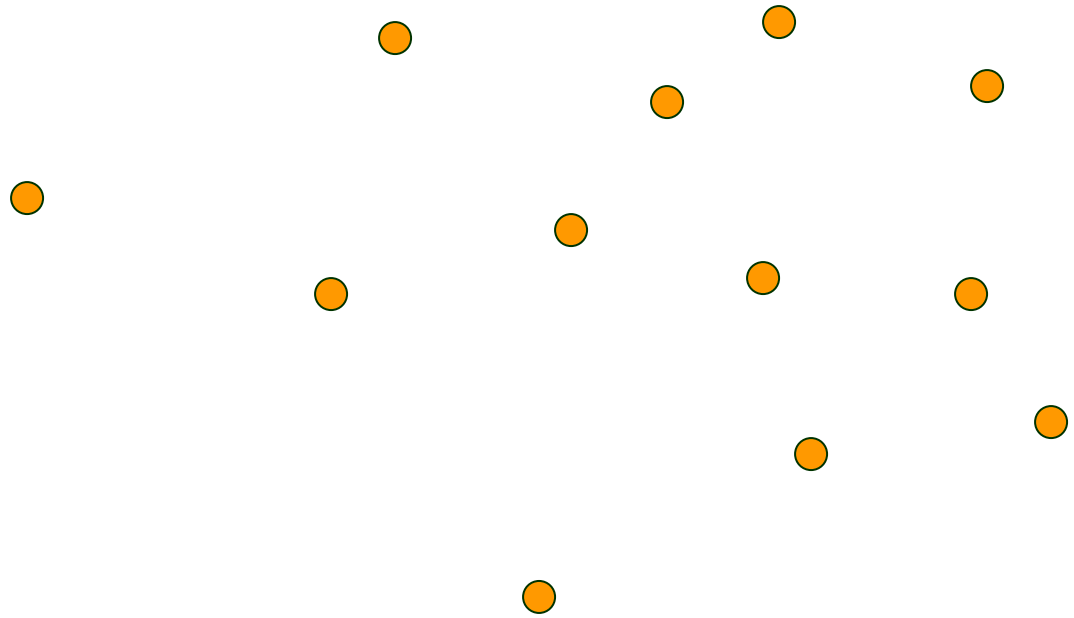
Polar Angle



Polar Angle

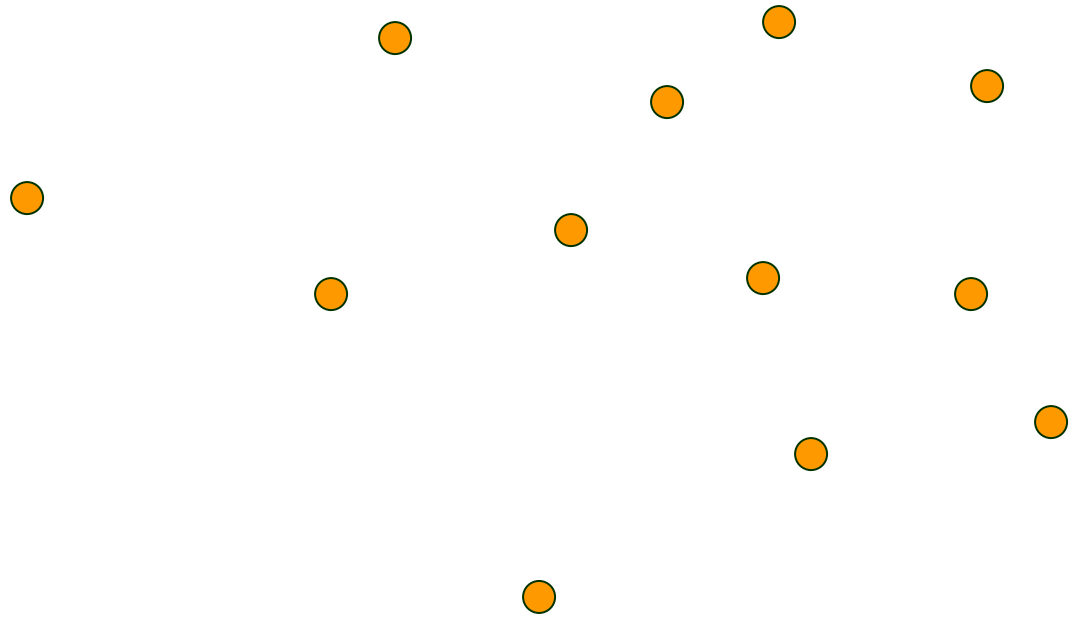


III. Graham Scan



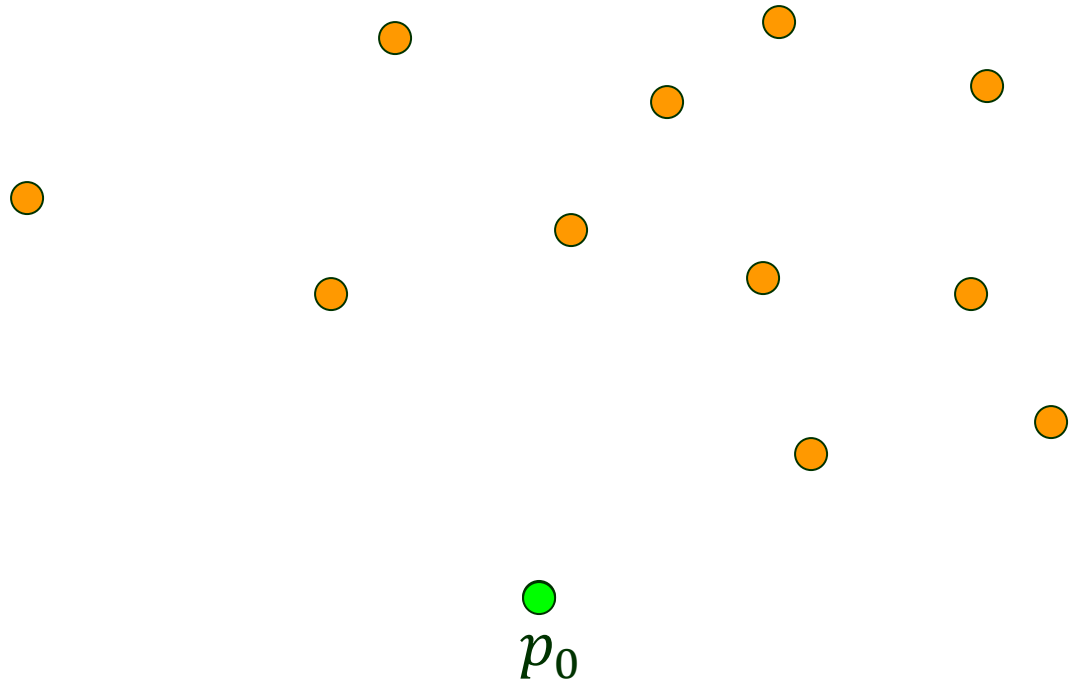
III. Graham Scan

1) Select the node with the smallest y coordinate.



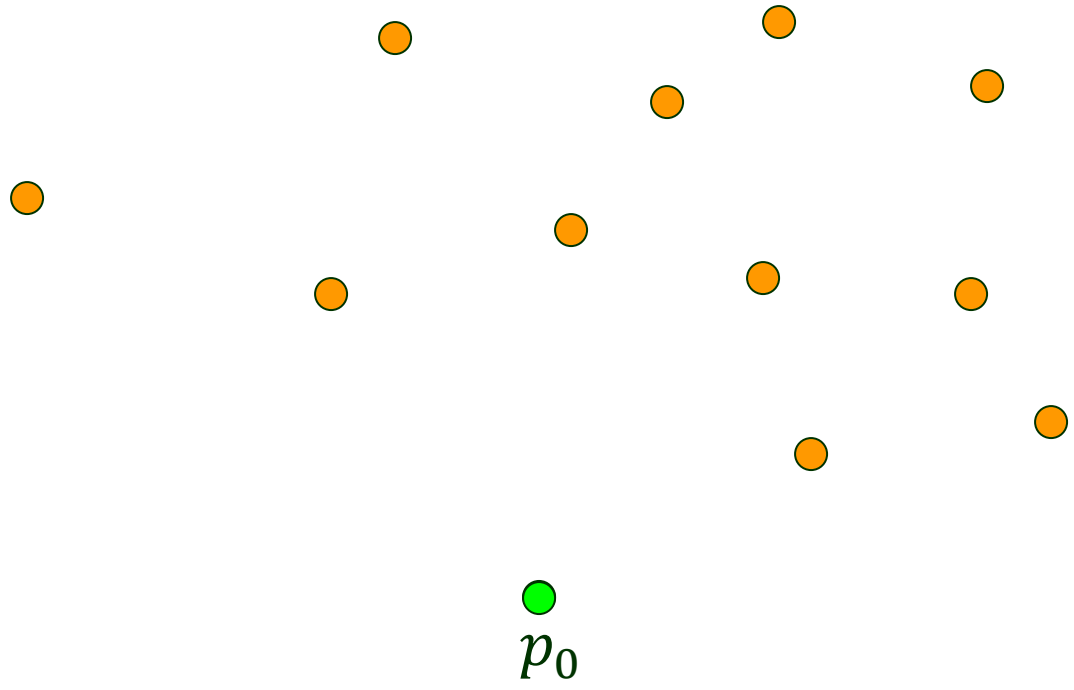
III. Graham Scan

1) Select the node with the smallest y coordinate.



III. Graham Scan

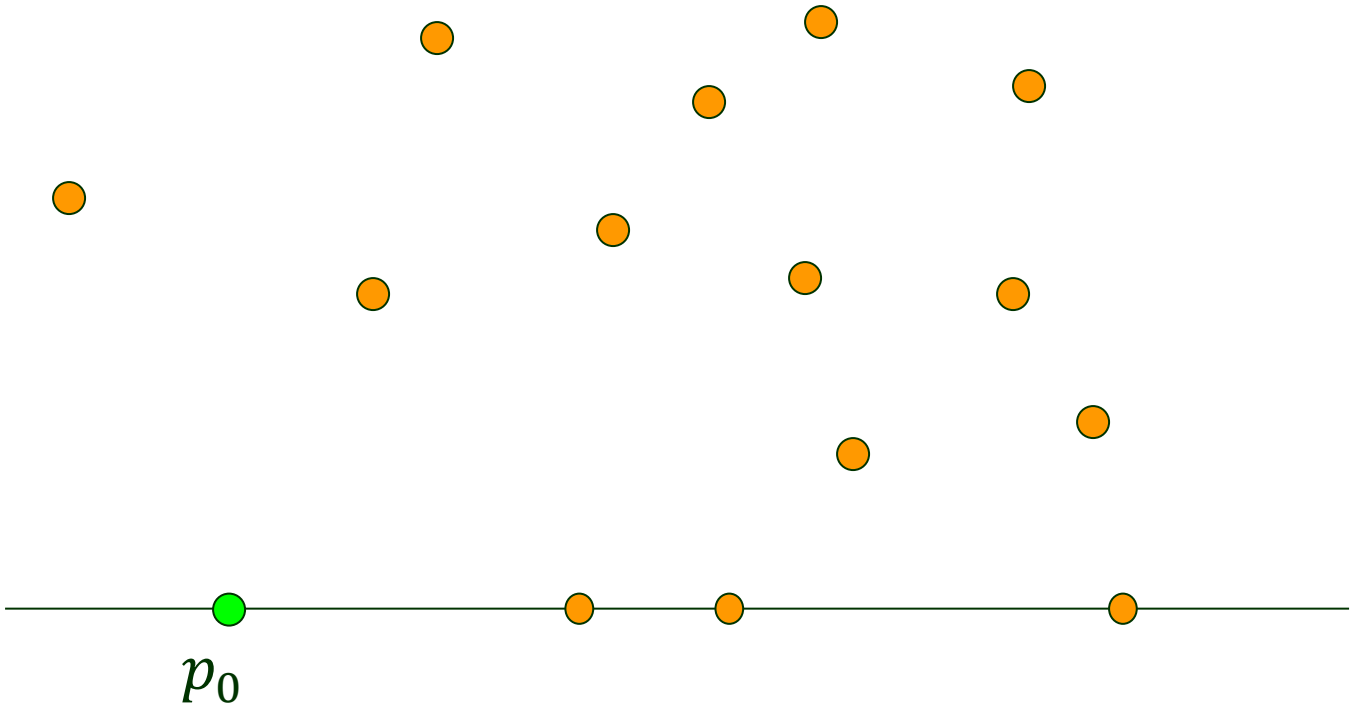
1) Select the node with the smallest y coordinate.



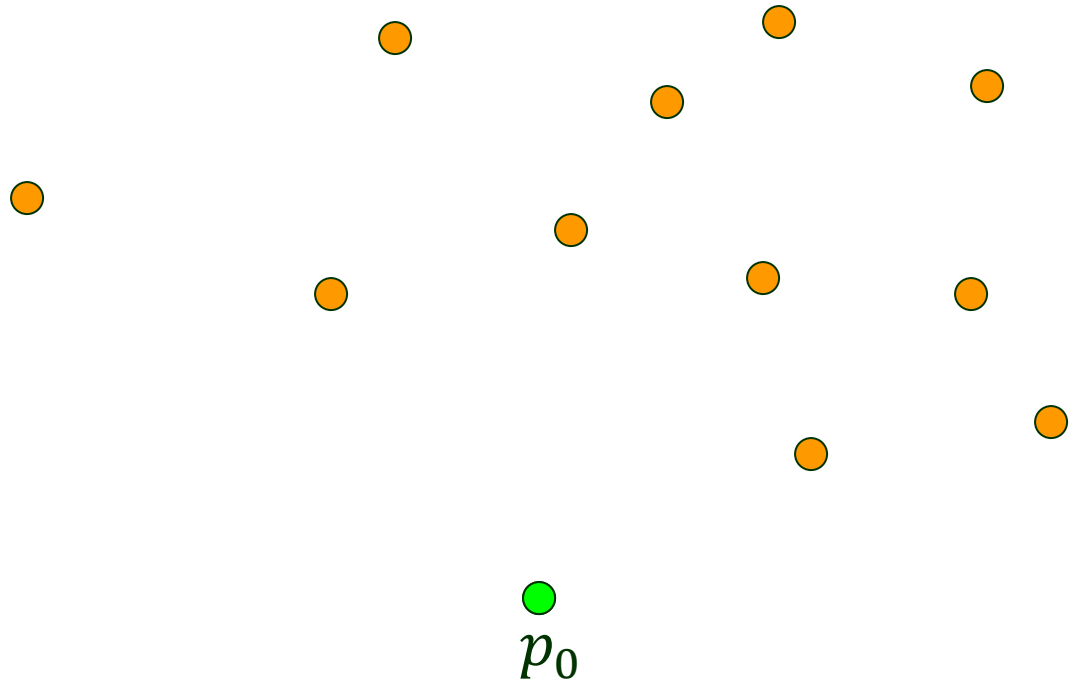
This node will be a vertex of the convex hull.

Tie Breaking (1)

When more than one point has the smallest y coordinate, pick the *leftmost* (or rightmost) one.

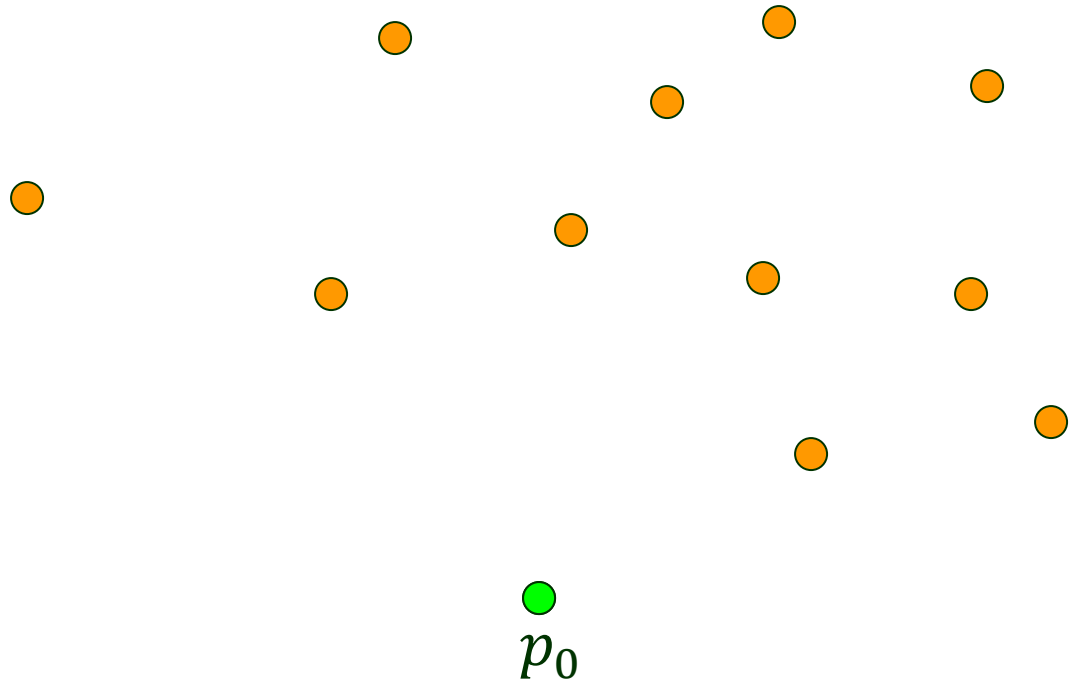


Sorting by Polar Angle



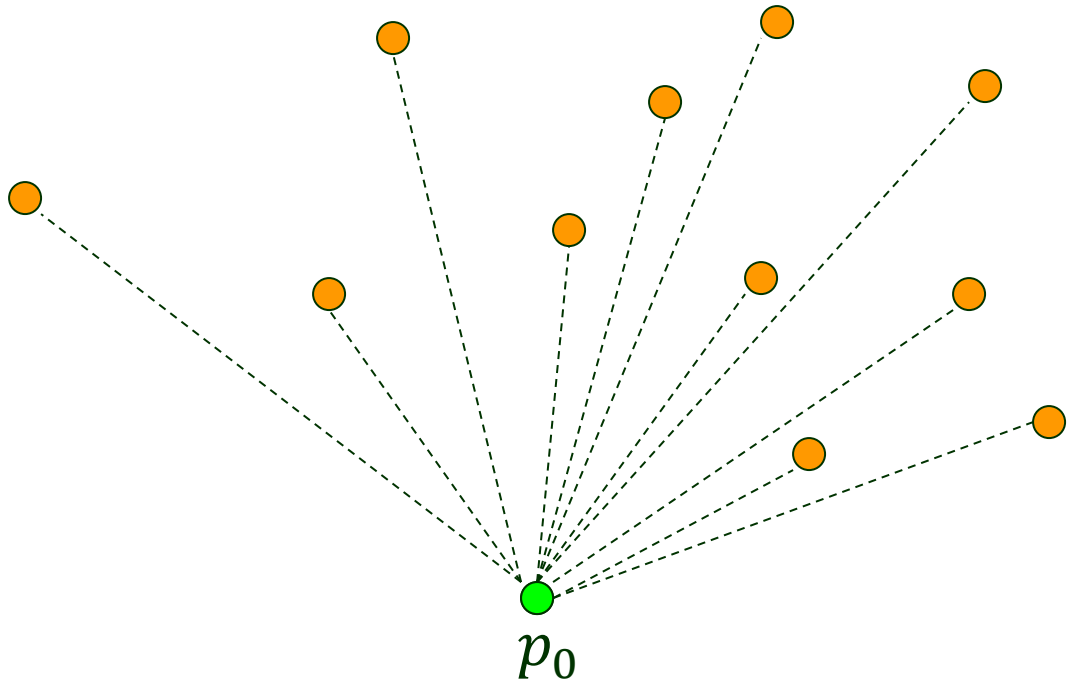
Sorting by Polar Angle

2) Sort by polar angle with respect to p_0.



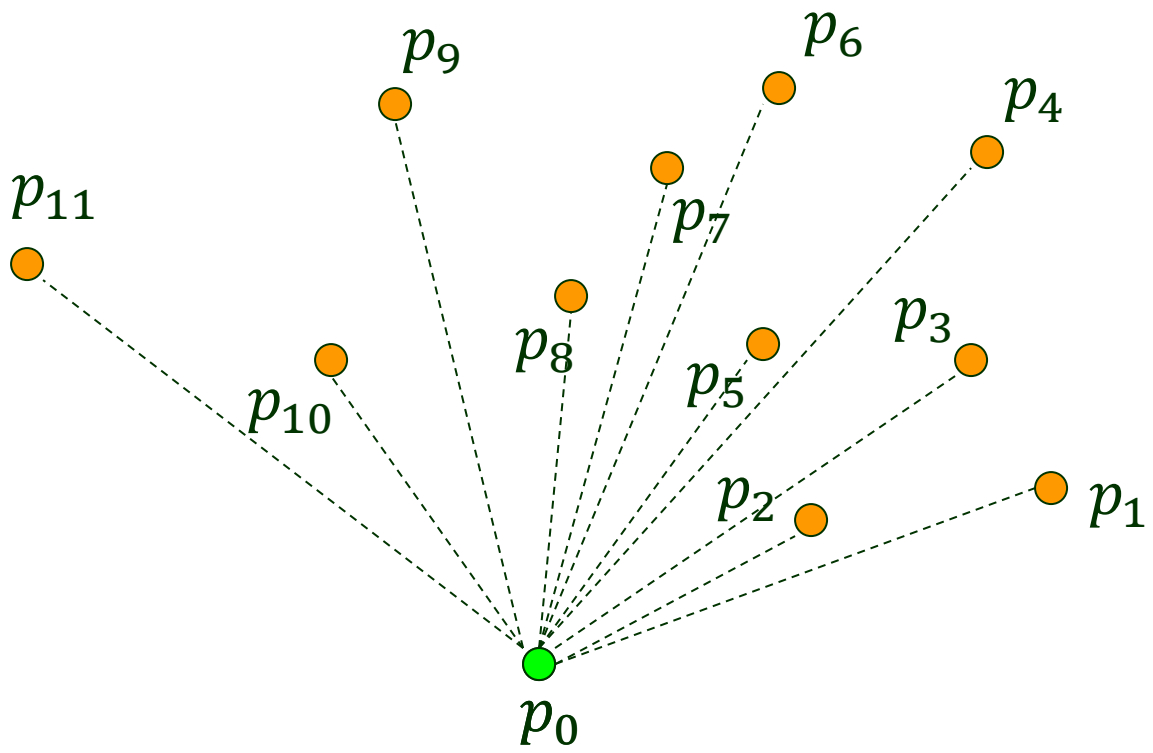
Sorting by Polar Angle

2) Sort by polar angle with respect to p_0.



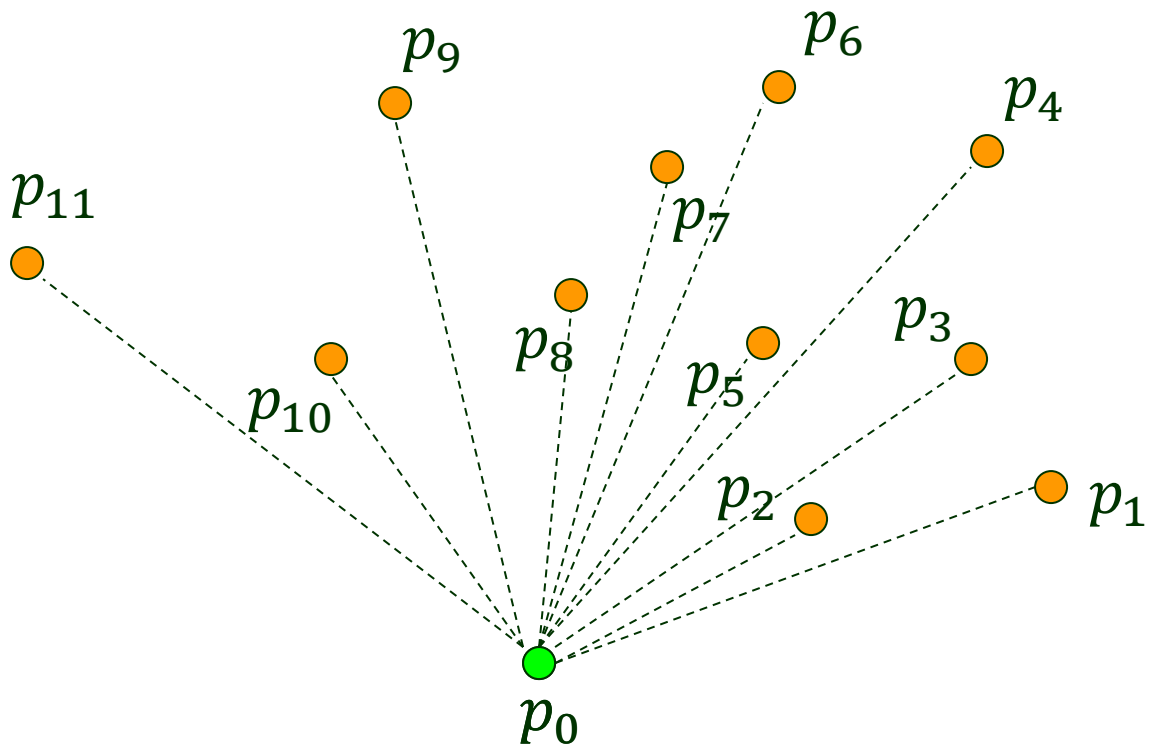
Sorting by Polar Angle

2) Sort by polar angle with respect to p_0.



Sorting by Polar Angle

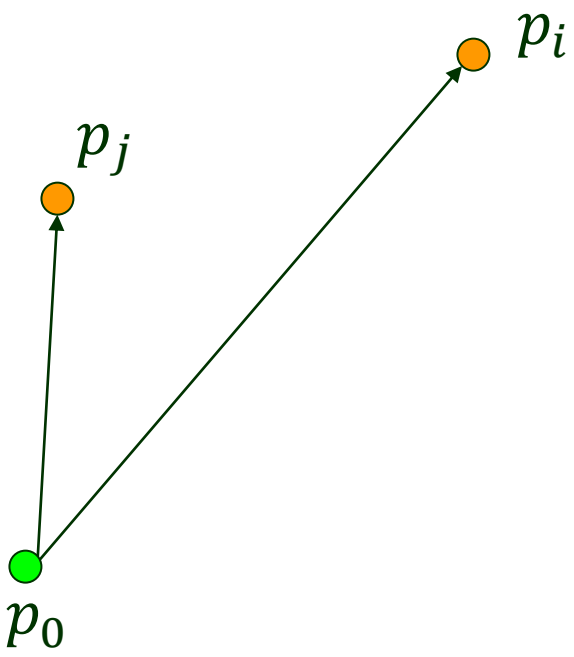
2) Sort by polar angle with respect to p_0.



Labels are in the polar angle order.

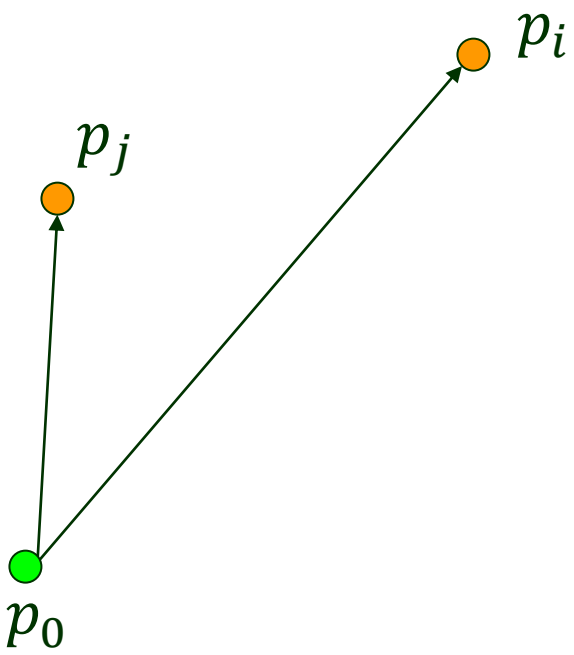
No Polar Angle Evaluation

p_0 is the lowest (and leftmost)



No Polar Angle Evaluation

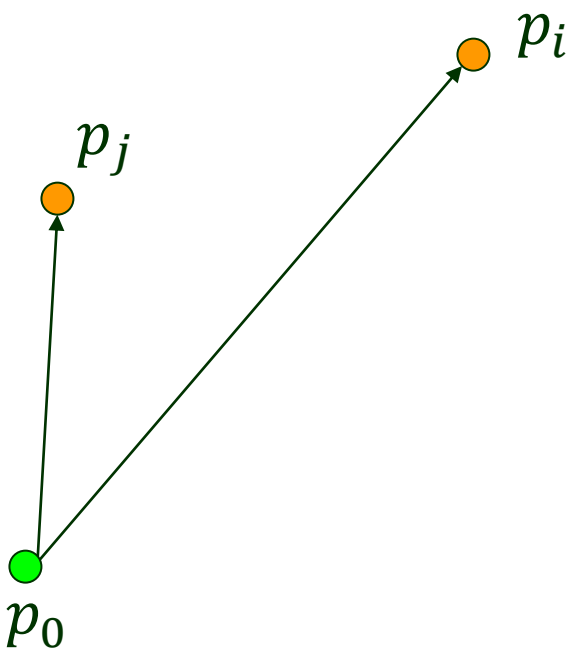
p_0 is the lowest (and leftmost) ⇒ all polar angles ∈ $[0, \pi)$.



No Polar Angle Evaluation

p_0 is the lowest (and leftmost) ⇒ all polar angles $\in [0, \pi)$.

Use cross product!

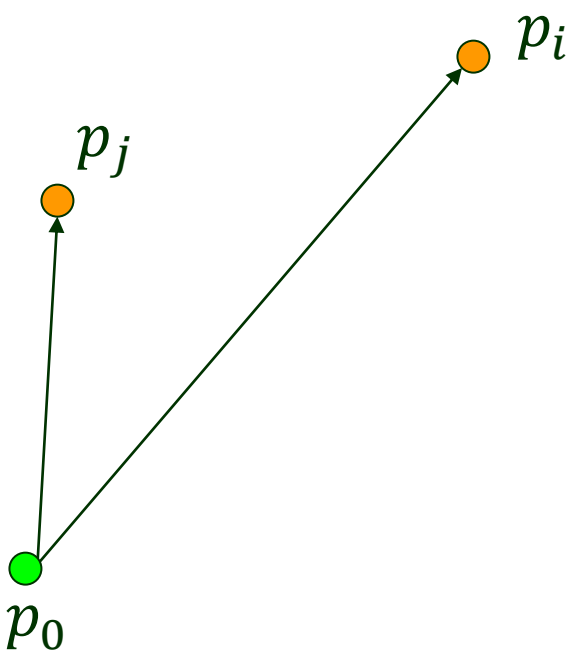


No Polar Angle Evaluation

p_0 is the lowest (and leftmost) ⇒ all polar angles $\in [0, \pi)$.

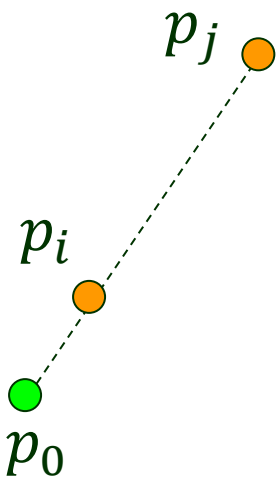
Use cross product!

$$p_i < p_j \text{ if } \overrightarrow{p_0 p_i} \times \overrightarrow{p_0 p_j} > 0$$



Tie Breaking (2)

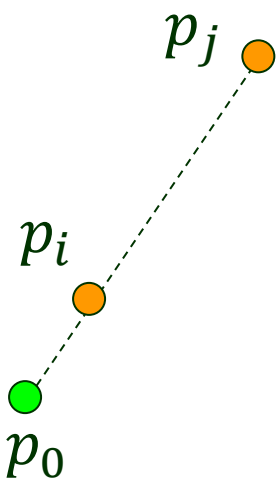
What if p_0, p_i, p_j are on the same line?



Tie Breaking (2)

What if p_0, p_i, p_j are on the same line?

Order them by distance from p_0.



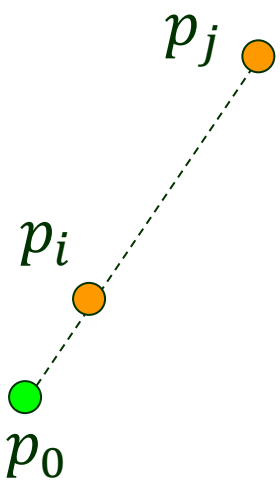
Tie Breaking (2)

What if p_0, p_i, p_j are on the same line?

Order them by distance from p_0.

$p_i < p_j$ if $\overrightarrow{p_0p_i} \times \overrightarrow{p_0p_j} = 0$ and

$|\overrightarrow{p_0p_i}| < |\overrightarrow{p_0p_j}|$



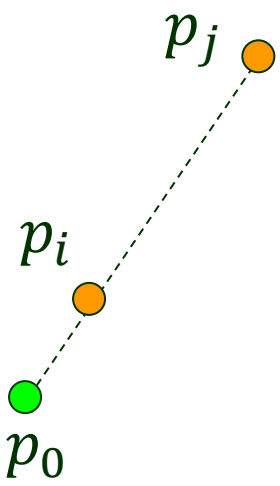
Tie Breaking (2)

What if p_0, p_i, p_j are on the same line?

Order them by distance from p_0.

$p_i < p_j$ if $\overrightarrow{p_0p_i} \times \overrightarrow{p_0p_j} = 0$ and

$$|\overrightarrow{p_0p_i}| < |\overrightarrow{p_0p_j}|$$



Tie Breaking (2)

What if p_0, p_i, p_j are on the same line?

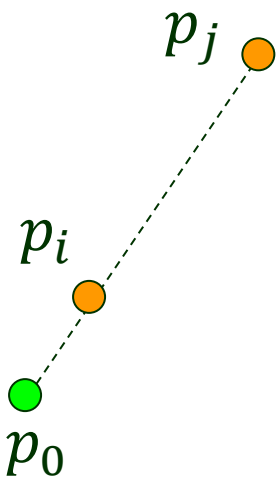
Order them by distance from p_0.

$p_i < p_j$ if $\overrightarrow{p_0p_i} \times \overrightarrow{p_0p_j} = 0$ and

$$|\overrightarrow{p_0p_i}| < |\overrightarrow{p_0p_j}|$$

No square roots.
Use dot product!

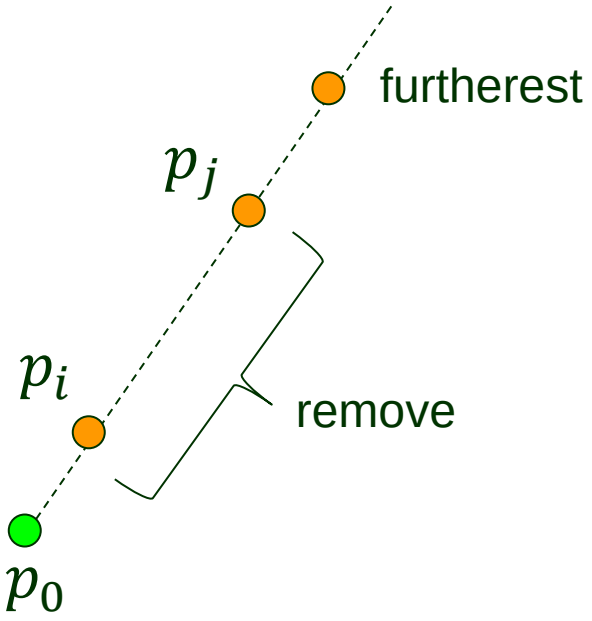
$$\overrightarrow{p_0p_i} \cdot \overrightarrow{p_0p_i} < \overrightarrow{p_0p_j} \cdot \overrightarrow{p_0p_j}$$



Point Elimination

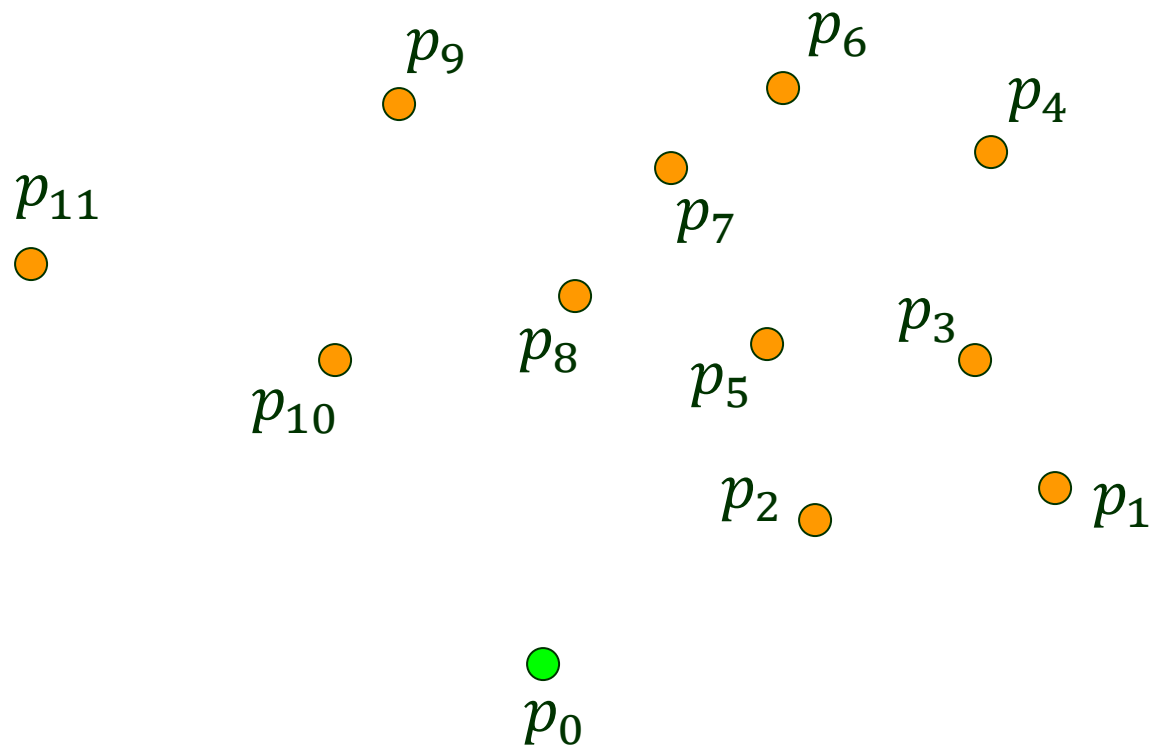
When multiple points have the same polar angle, keep the one *furthest* from p_0.

Remove the rest since they cannot possibly be the hull vertices.



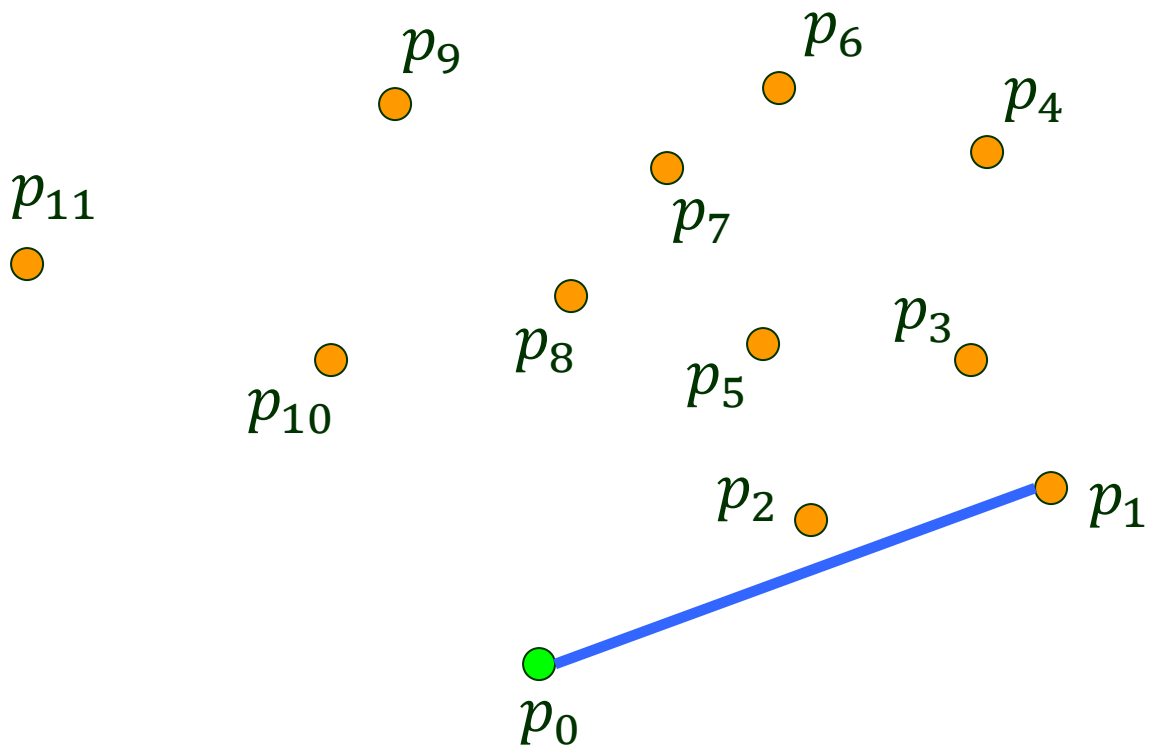
Stack Initialization

3) Scan points in the increasing order of polar angle, maintaining a stack.



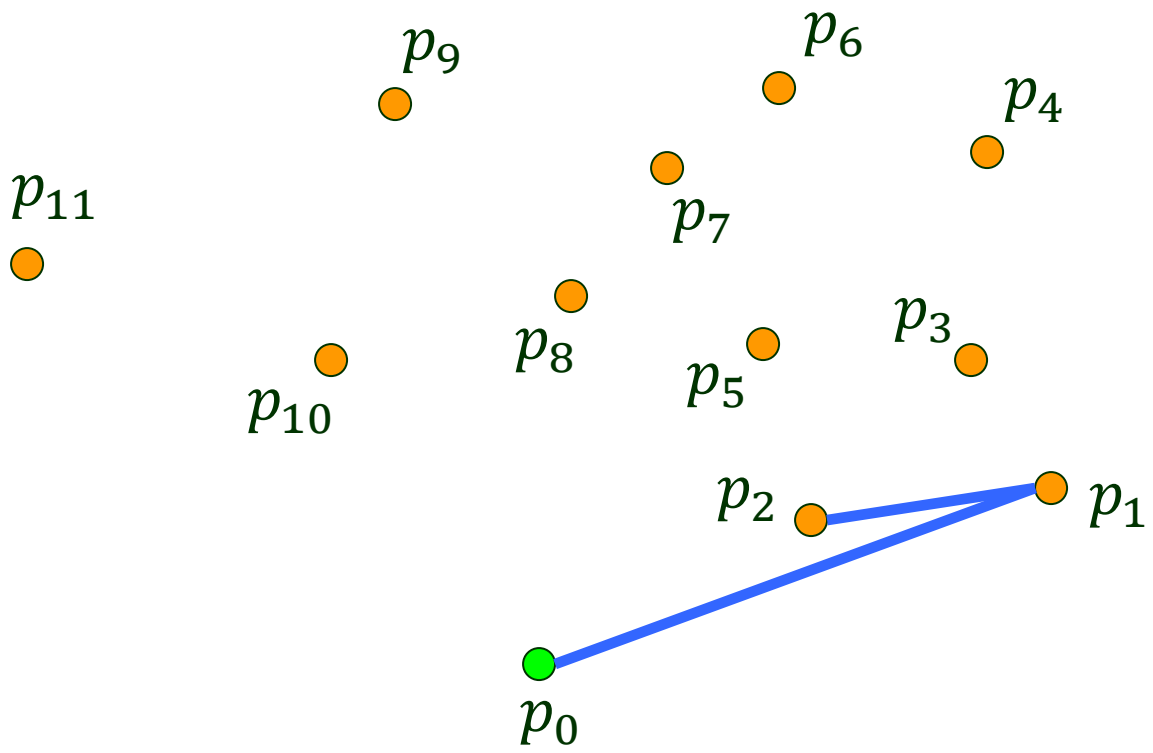
Stack Initialization

3) Scan points in the increasing order of polar angle, maintaining a stack.



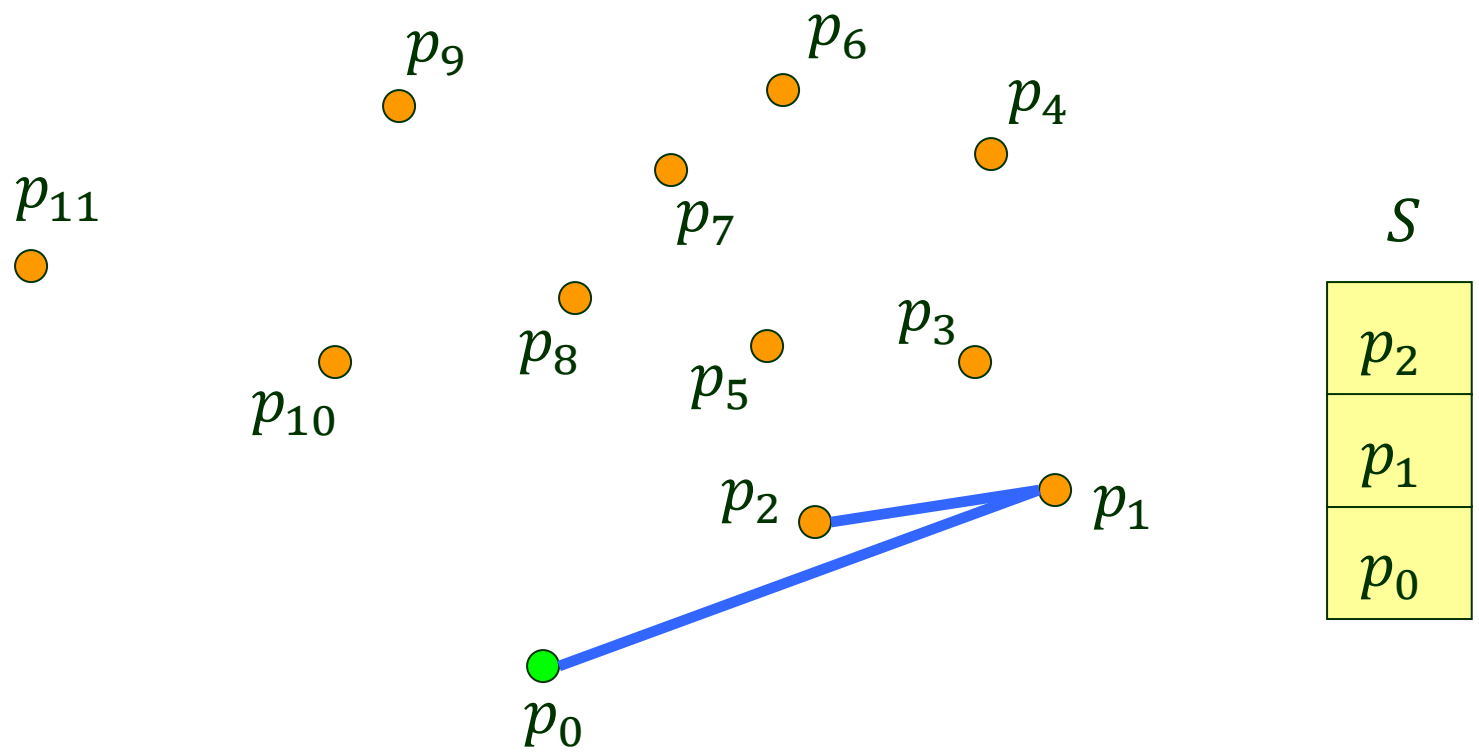
Stack Initialization

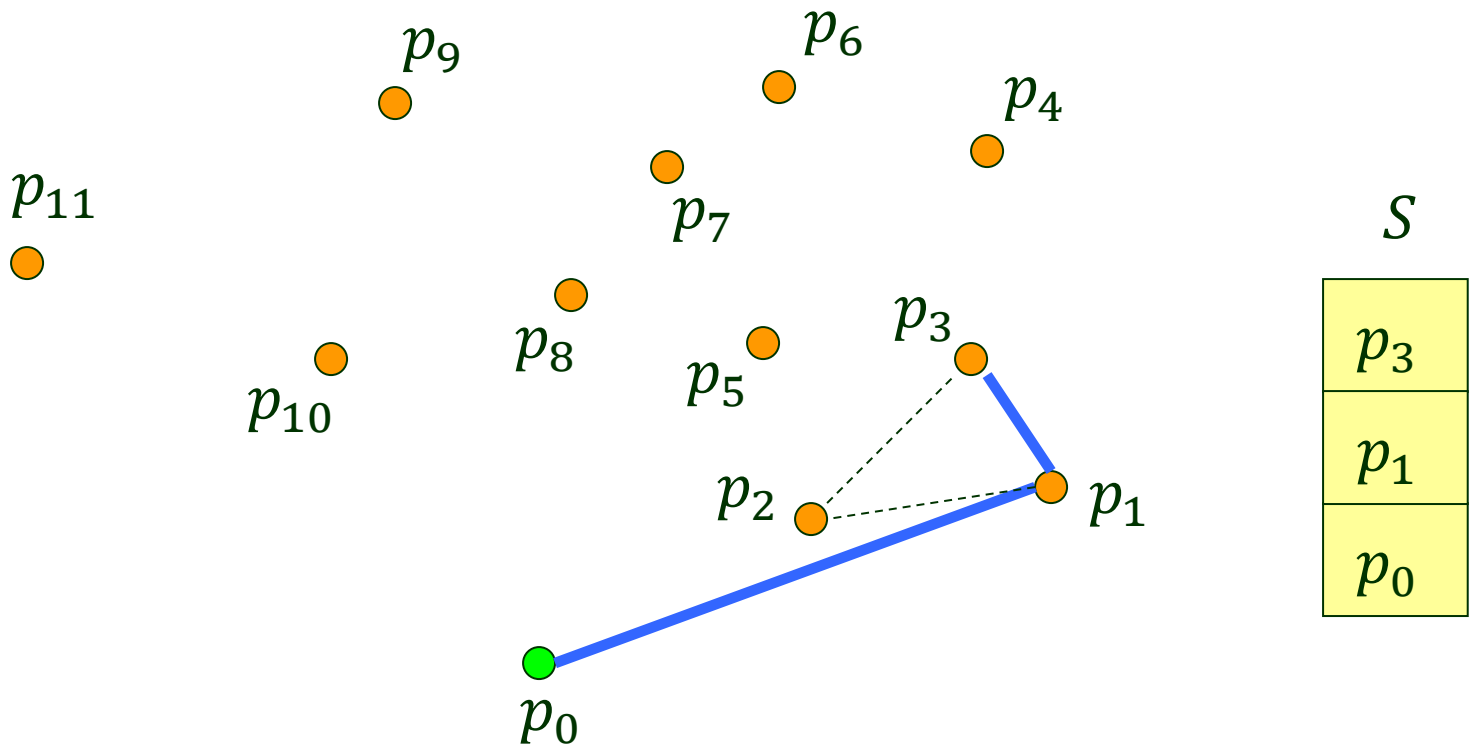
3) Scan points in the increasing order of polar angle, maintaining a stack.

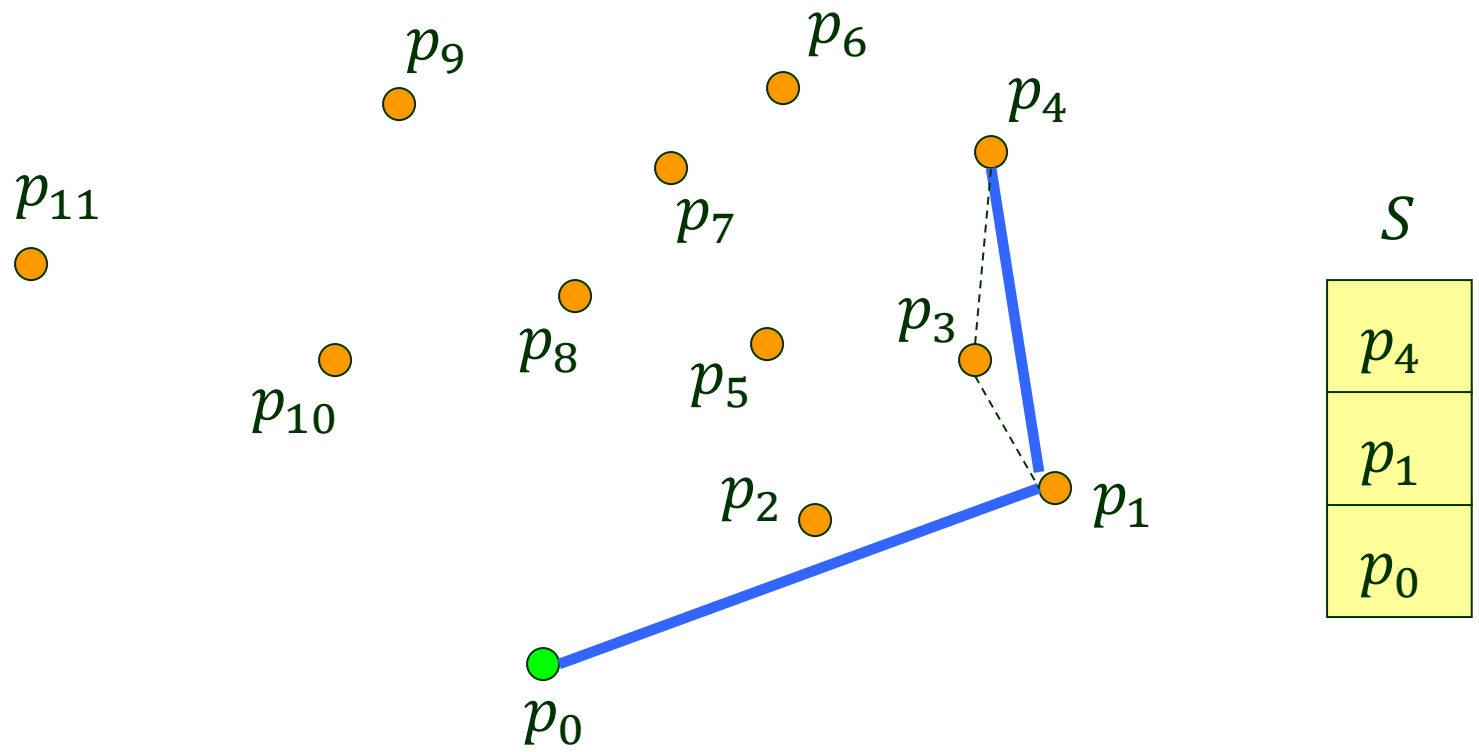


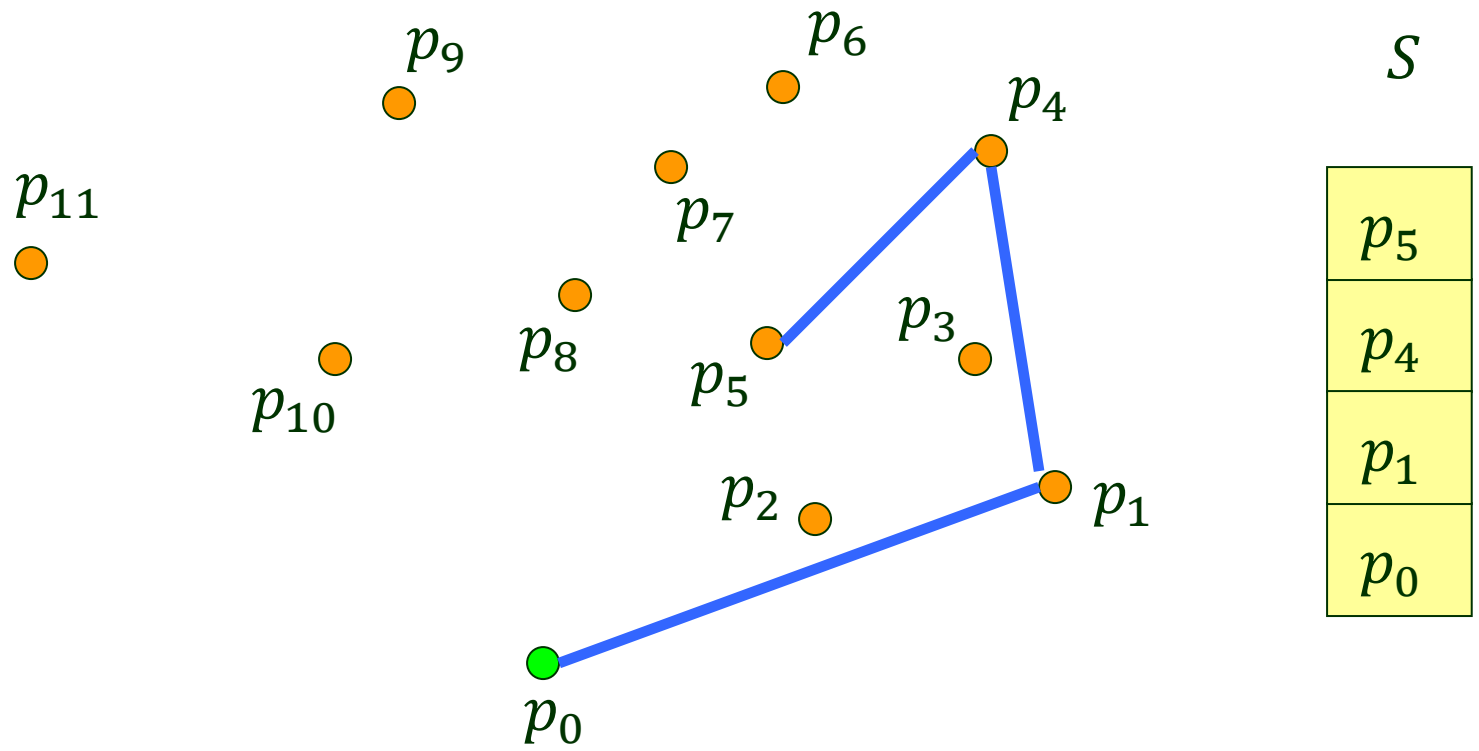
Stack Initialization

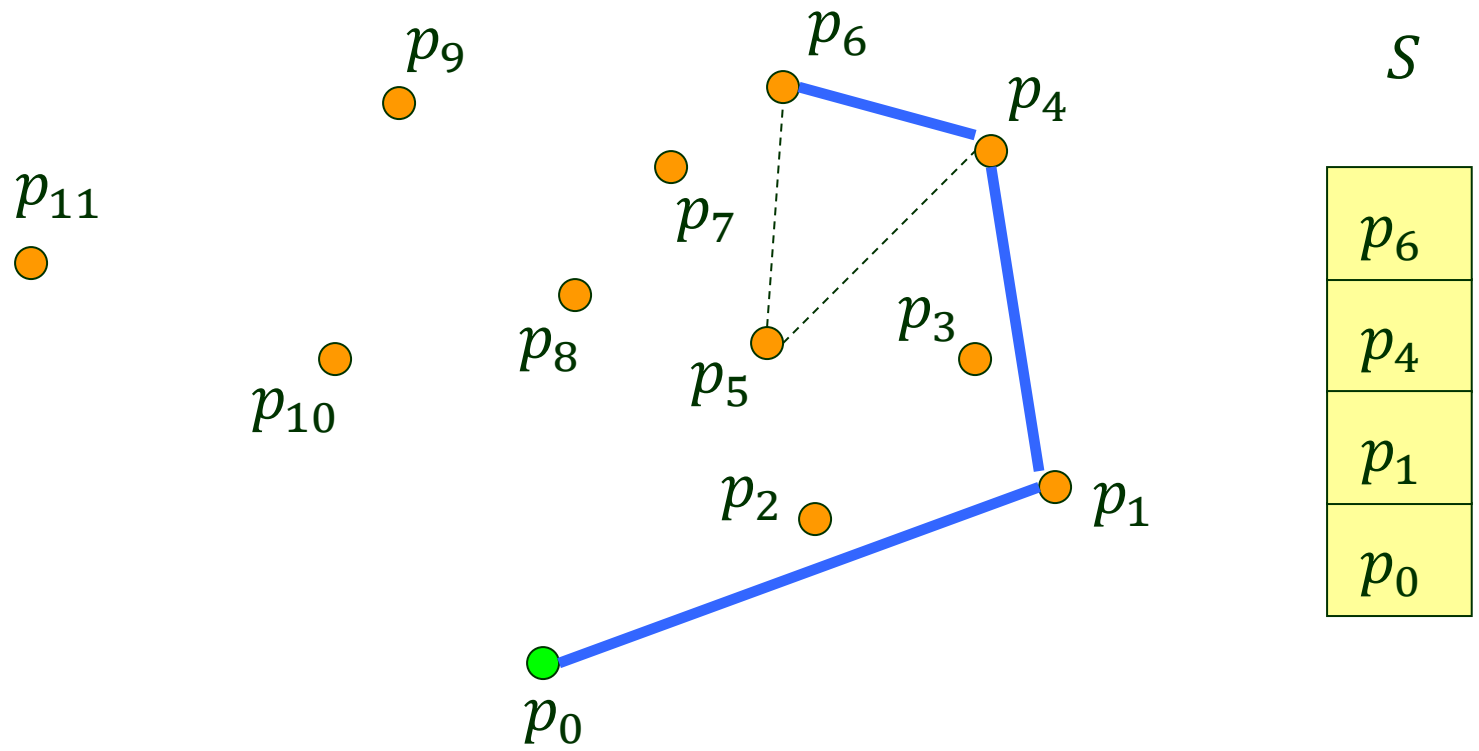
3) Scan points in the increasing order of polar angle, maintaining a stack.

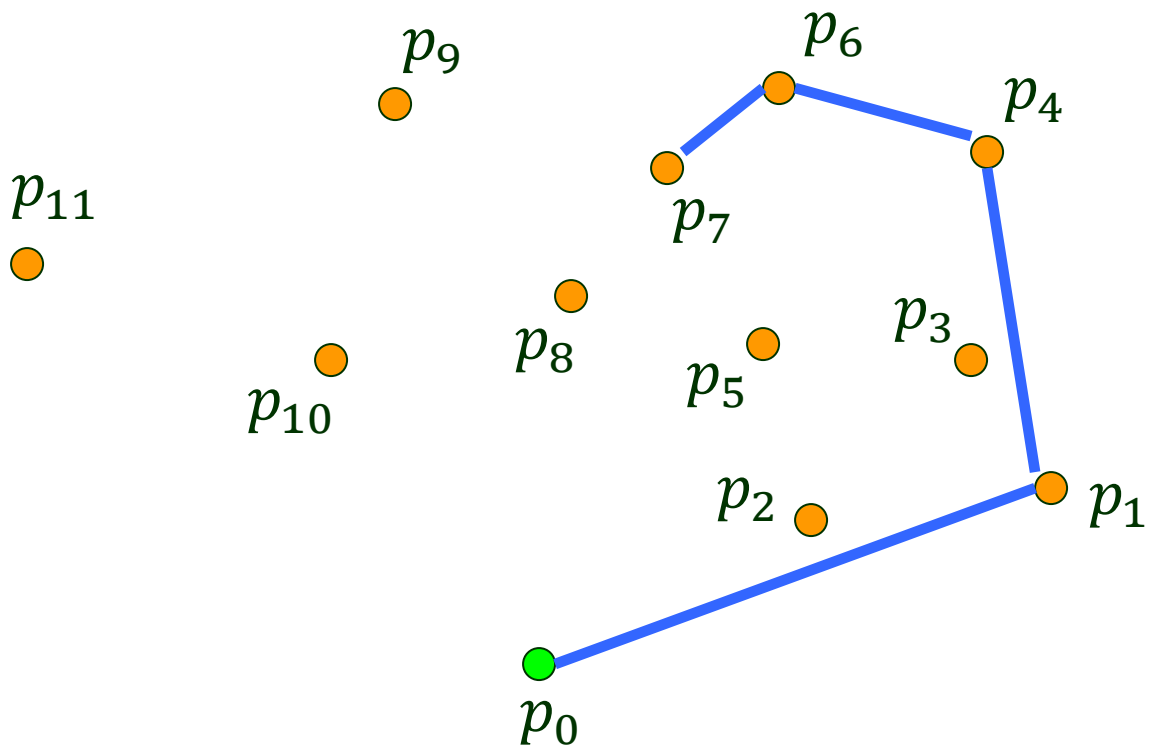



p_9
p_6
p_4
p_{11}
p_7
S
p_8
p_3
p_3
p_5
p_{10}
p_1
p_2
p_1
p_0
p_0


p_9
p_6
p_4
p_{11}
p_7
S
p_3
p_8
p_4
p_{10}
p_5
p_1
p_2
p_1
p_0
p_0

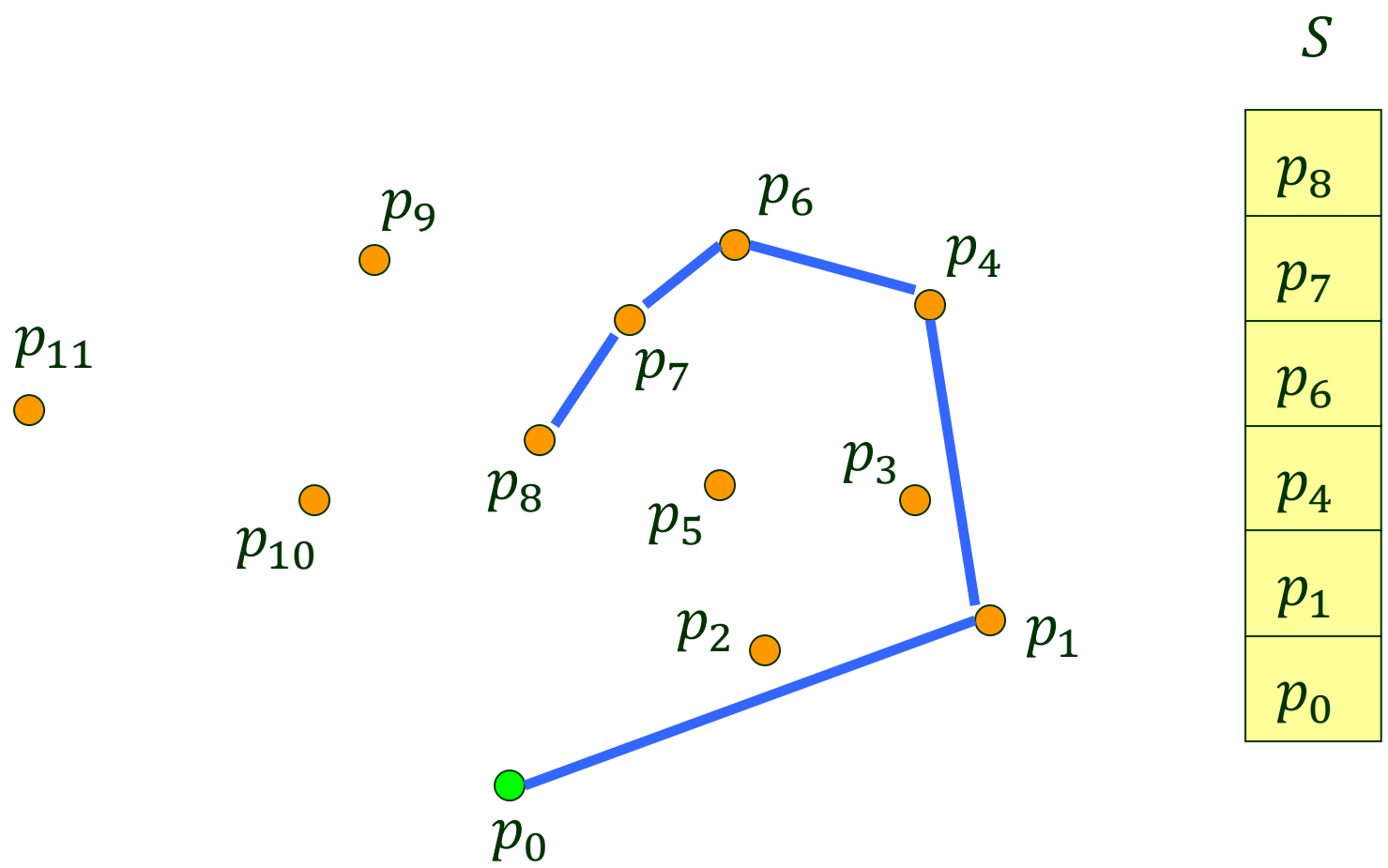

p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_{10}
p_5
p_2
p_1
p_0
S
p_5
p_4
p_1
p_0

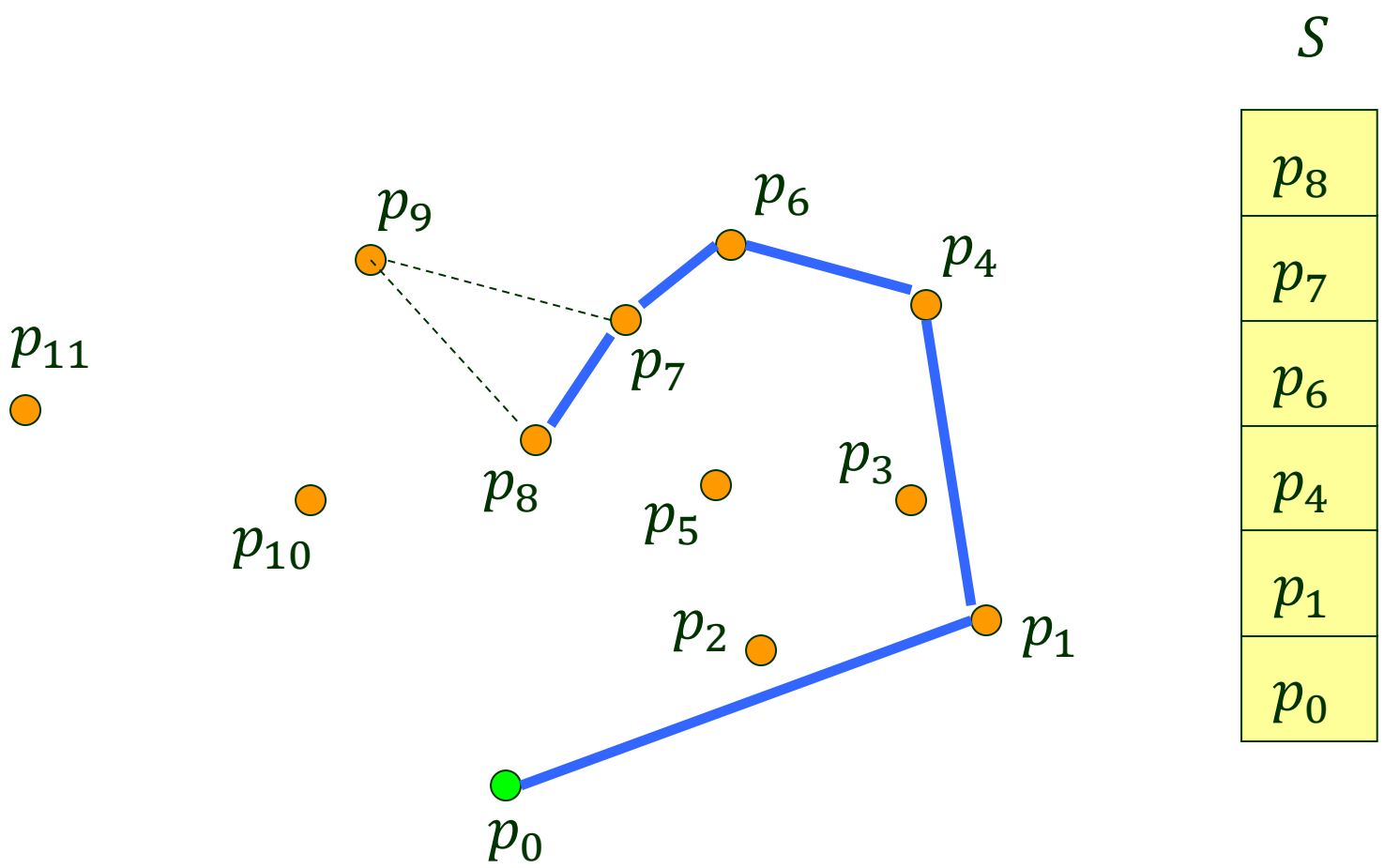

p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_{10}
p_5
p_2
p_1
p_0
S
p_6
p_4
p_1
p_0

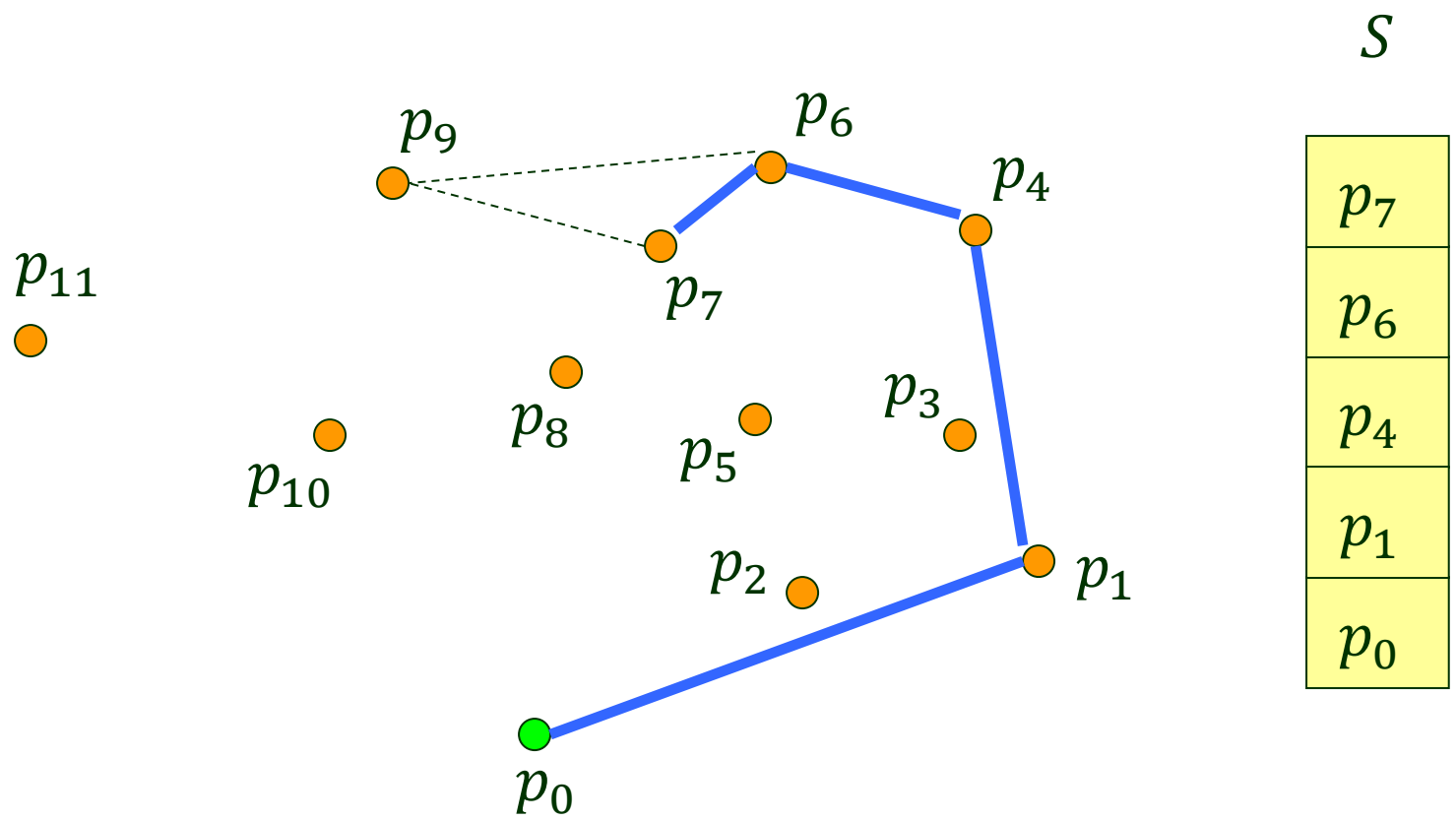


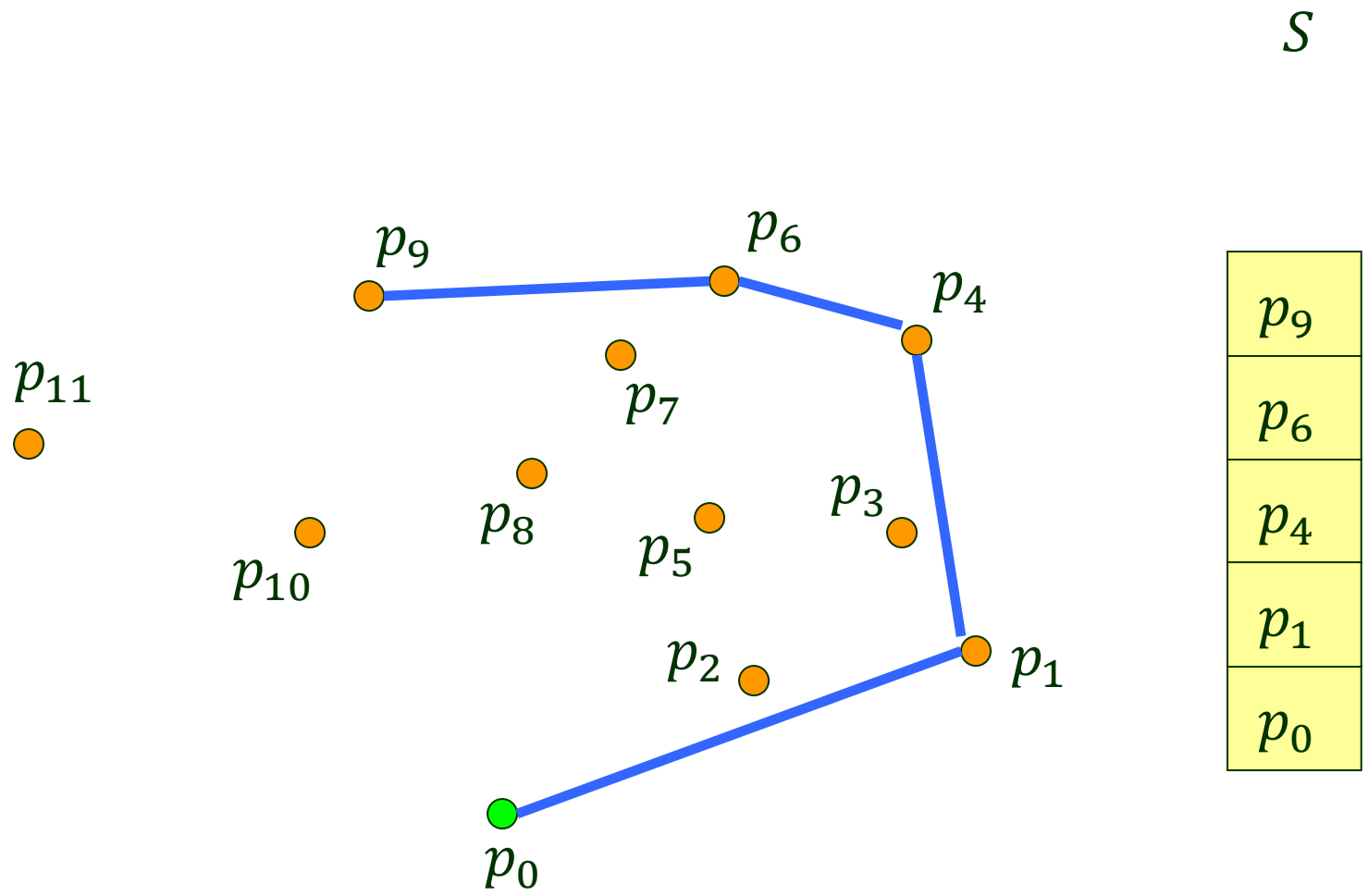
S

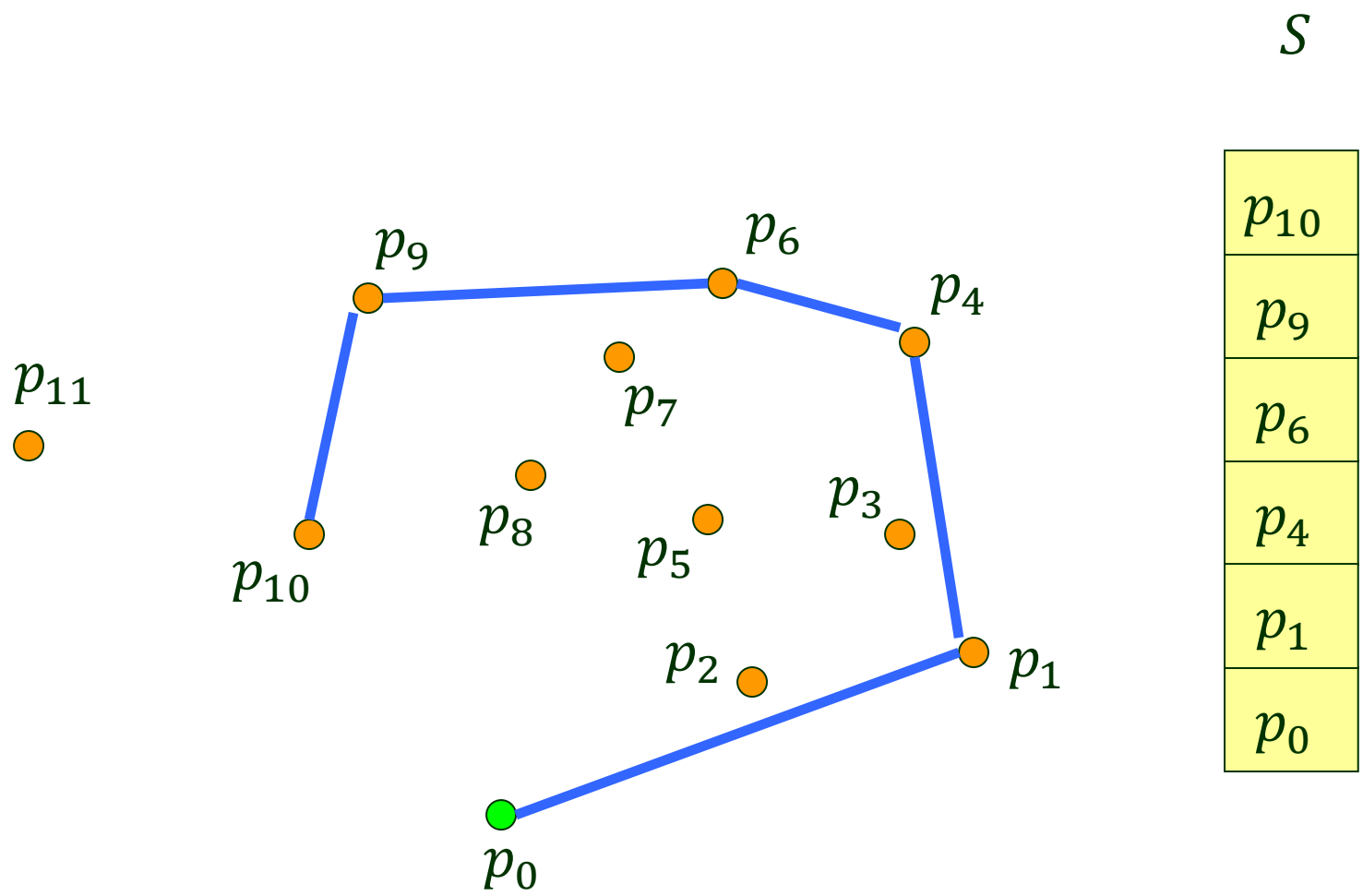
p_7
p_6
p_4
p_1
p_0

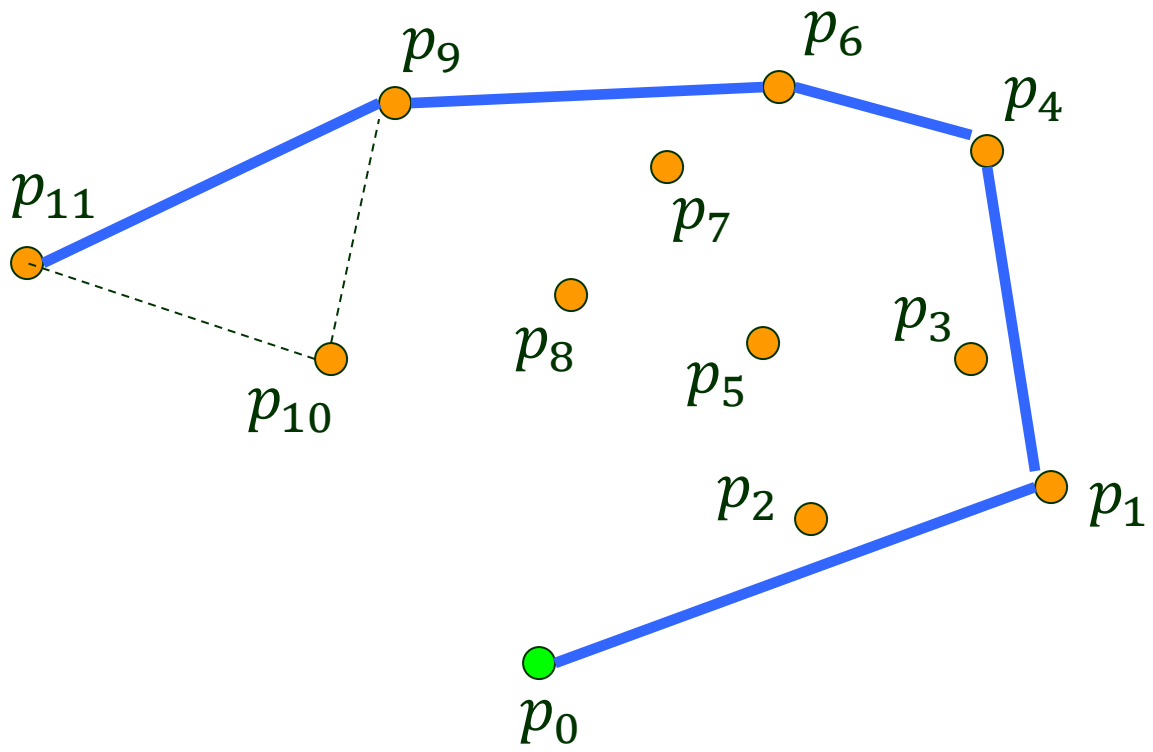

p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_{10}
p_5
p_2
p_1
p_0
S
p_8
p_7
p_6
p_4
p_1
p_0


p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_{10}
p_5
p_2
p_1
p_0
S
p_8
p_7
p_6
p_4
p_1
p_0


p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_5
p_{10}
p_2
p_1
p_0
S
p_7
p_6
p_4
p_1
p_0


S
p_9
p_6
p_4
p_{11}
p_7
p_8
p_3
p_{10}
p_5
p_2
p_1
p_0
p_9
p_6
p_4
p_1
p_0

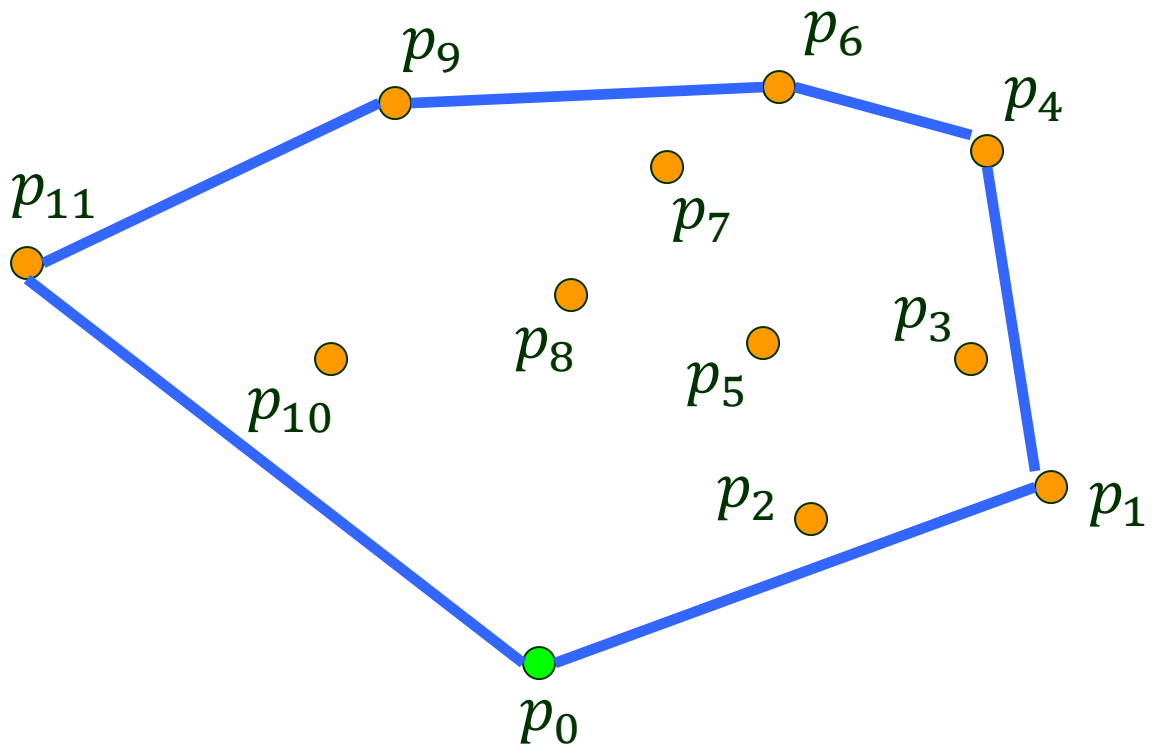

S
p_{10}
p_9
p_6
p_4
p_1
p_0
p_{11}
p_9
p_6
p_4
p_7
p_8
p_3
p_5
p_{10}
p_2
p_1
p_0



S

p_{11}
p_9
p_6
p_4
p_1
p_0

Finish



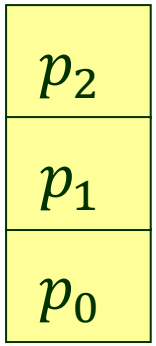
S

p_{11}
p_9
p_6
p_4
p_1
p_0

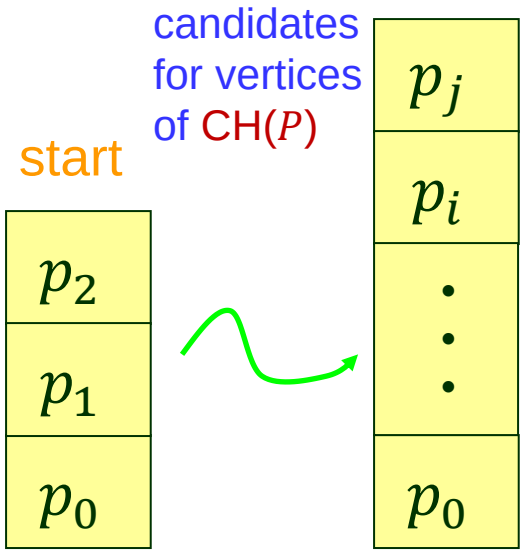
Graham’s Scan

Graham's Scan

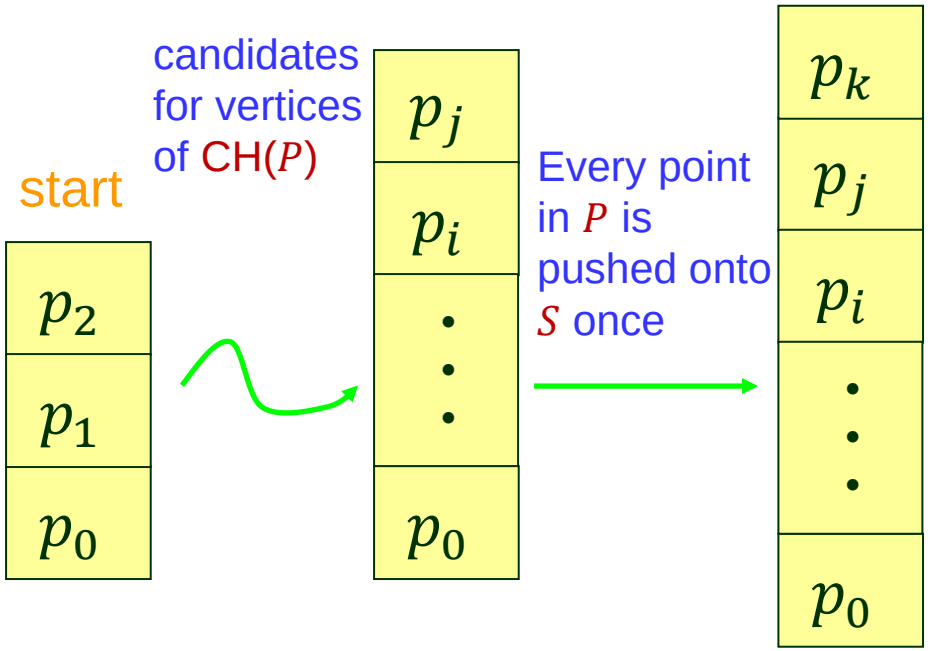
start



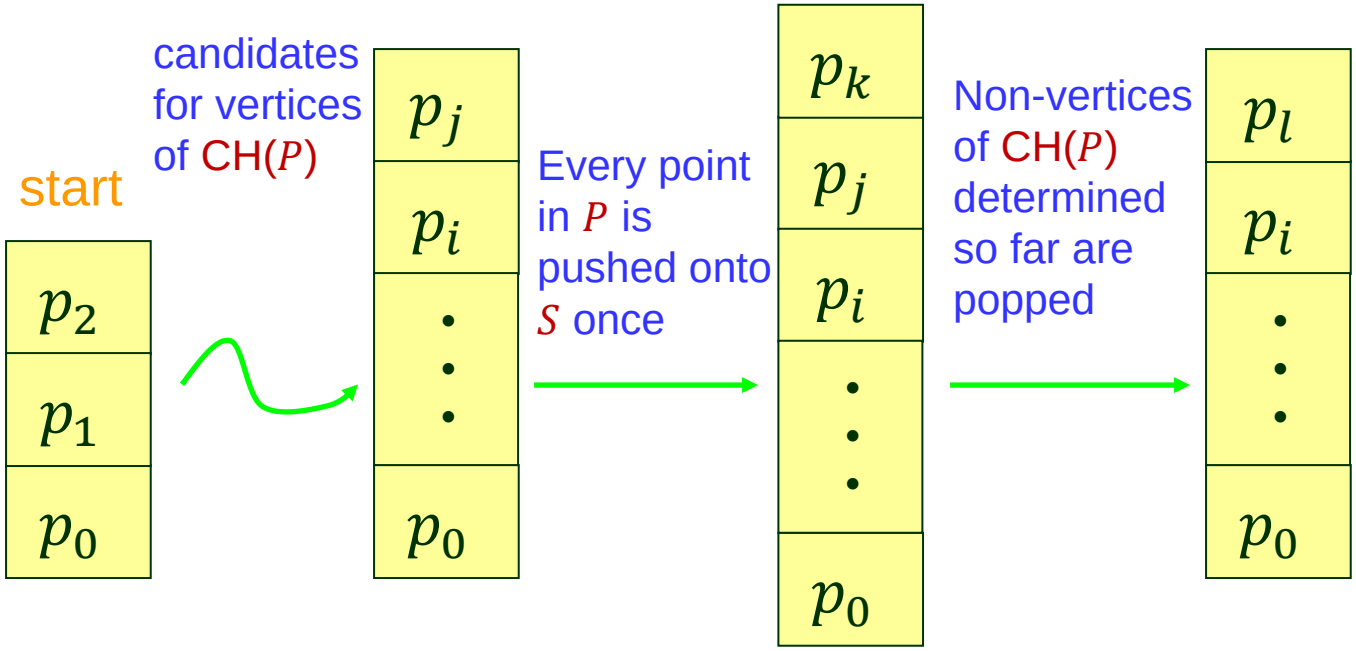
Graham's Scan



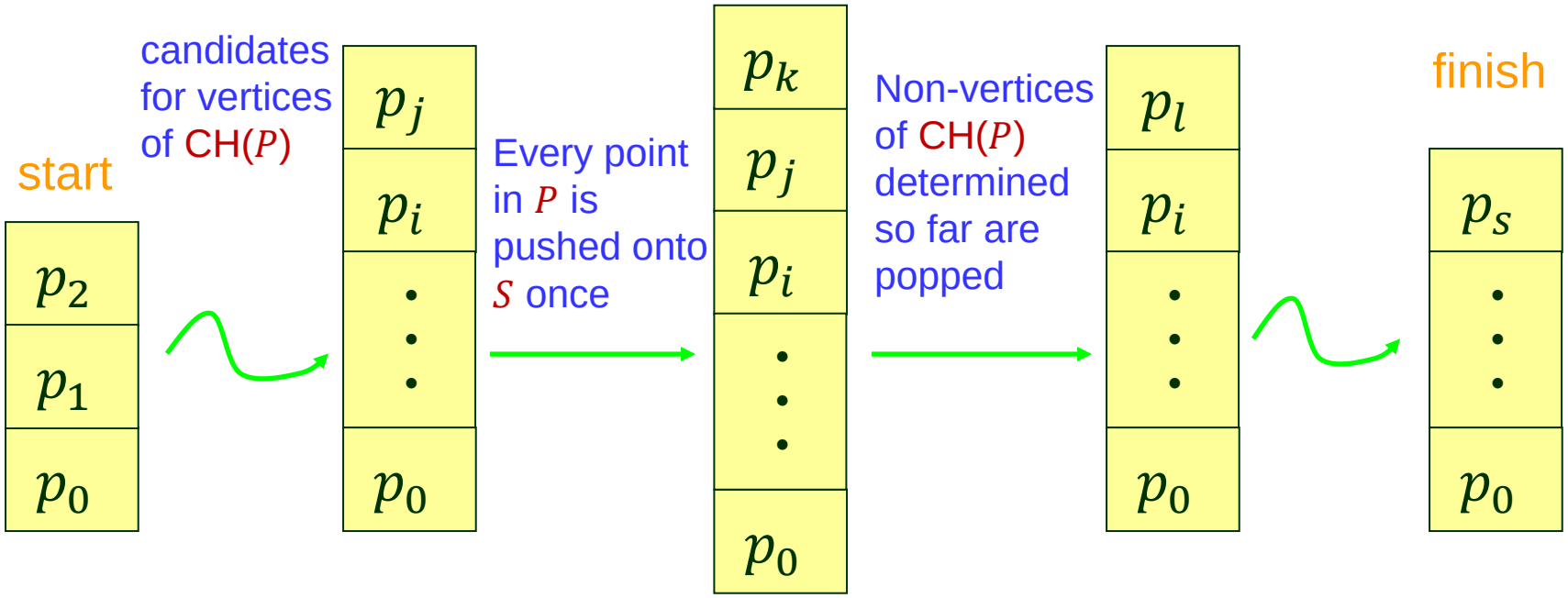
Graham's Scan



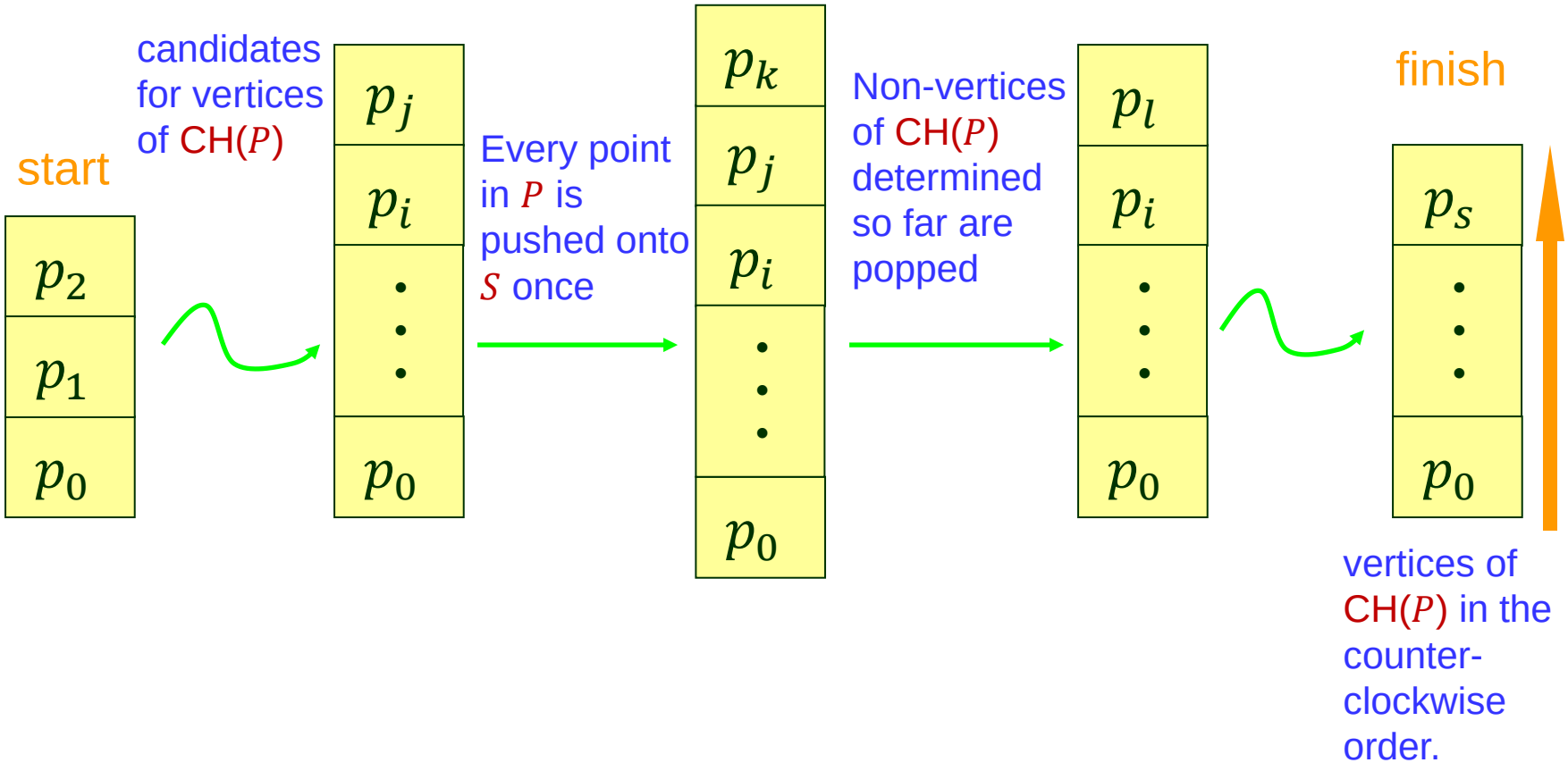
Graham's Scan



Graham's Scan



Graham's Scan



IV. The Graham Scan Algorithm

```
Graham-Scan(P)
    let p_0 be the point in P with minimum y-coordinate
    let ⟨p_1, p_2, …, p_{n−1}⟩ be the remaining points in P
        sorted in counterclockwise order by polar angle around p_0.
    Top[S] ← 0
    Push(p_0, S)
    Push(p_1, S)
    Push(p_2 , S)
    for i = 3 to n − 1
        do while p_i makes a nonleft turn from the line segment
                    determined by Top(S) and Next-to-Top(S)
                do Pop(S)
            Push(S, p_i)
    return S
```

Running time

#operations	time / operation	total

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$
Sorting	1	$O(n \log n)$	$O(n \log n)$

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$
Sorting	1	$O(n \log n)$	$O(n \log n)$
Push	n	$O(1)$	$\Theta(n)$

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$
Sorting	1	$O(n \log n)$	$O(n \log n)$
Push	n	$O(1)$	$\Theta(n)$
Pop	$\leq n - 2$	$O(1)$	$O(n)$

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$
Sorting	1	$O(n \log n)$	$O(n \log n)$
Push	n	$O(1)$	$\Theta(n)$
Pop	$\leq n - 2$ Why?	$O(1)$	$O(n)$

Running time

	#operations	time / operation	total
Finding p_0	1	$\Theta(n)$	$\Theta(n)$
Sorting	1	$O(n \log n)$	$O(n \log n)$
Push	n	$O(1)$	$\Theta(n)$
Pop	$\leq n - 2$ Why?	$O(1)$	$O(n)$

The running time of Graham's Scan is $\boldsymbol{O(n \log n)}$.

Proof of Correctness

Claim 1 Each point popped from stack S is not a vertex of CH(P).

Proof

Proof of Correctness

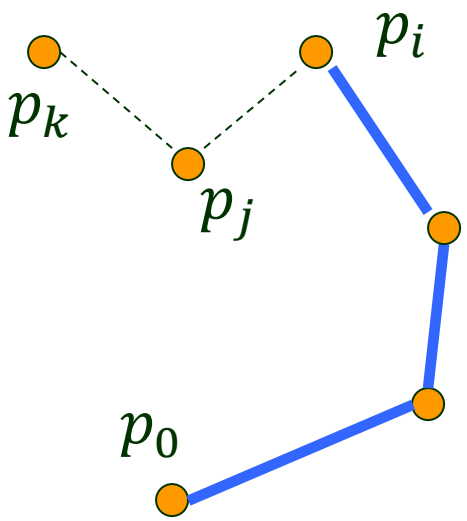
Claim 1 Each point popped from stack *S* is not a vertex of CH(*P*).

Proof Two cases when p_j is popped:

Proof of Correctness

Claim 1 Each point popped from stack *S* is not a vertex of CH(*P*).

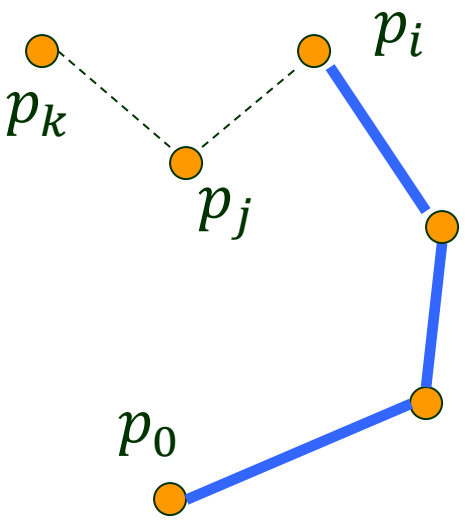
Proof Two cases when p_j is popped:

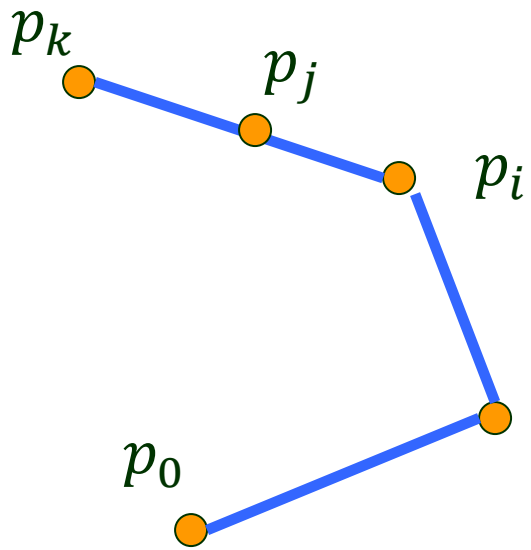


Proof of Correctness

Claim 1 Each point popped from stack *S* is not a vertex of CH(*P*).

Proof Two cases when p_j is popped:

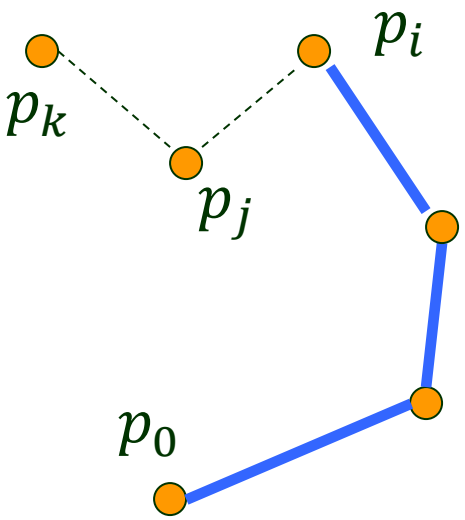


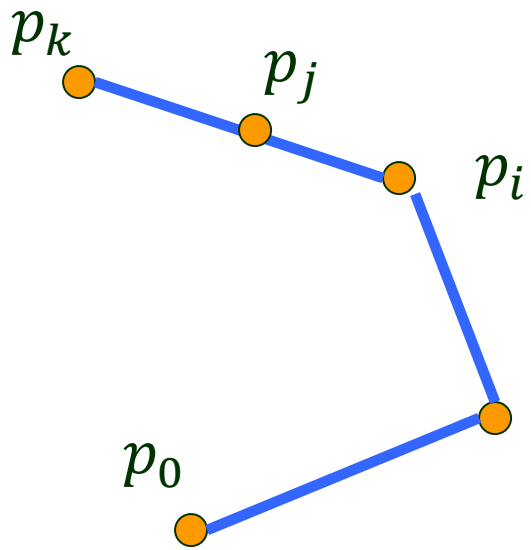


Proof of Correctness

Claim 1 Each point popped from stack *S* is not a vertex of CH(*P*).

Proof Two cases when p_j is popped:





In neither case can p_j become a vertex of CH(*P*).

Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack *S* always form the vertices of a convex polygon in counterclockwise order.

Proof

Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).

Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof

- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.

Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack *S* always form the vertices of a convex polygon in counterclockwise order.

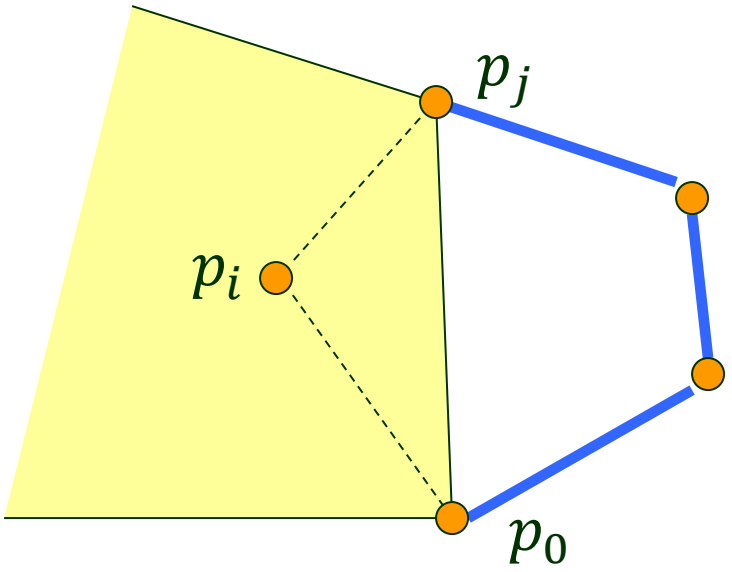
Proof

- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.
- Consider a point p_i being pushed onto S.

Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof

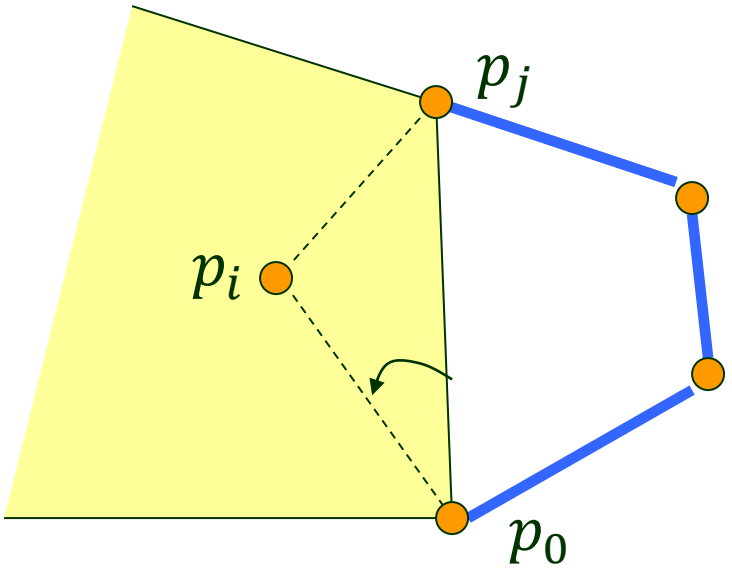
- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.
- Consider a point p_i being pushed onto S.



Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof

- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.
- Consider a point p_i being pushed onto S.

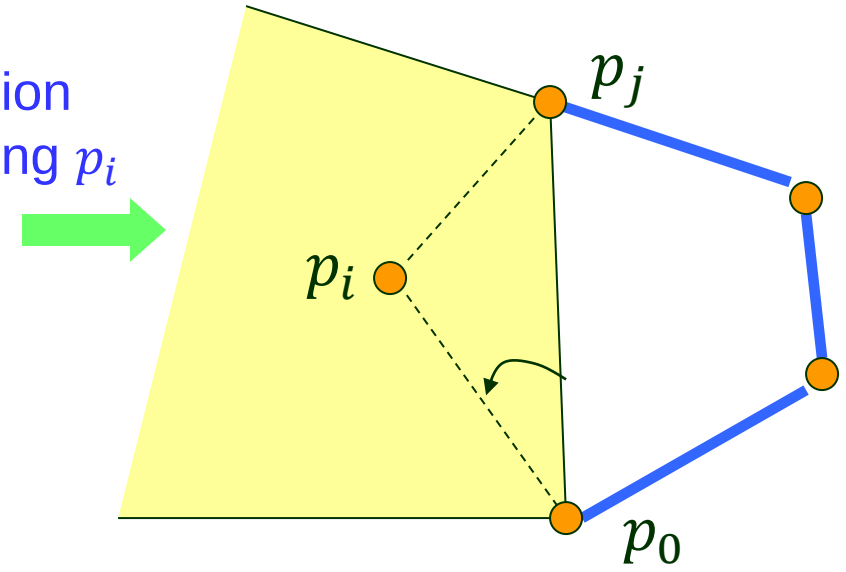


Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof

- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.
- Consider a point p_i being pushed onto S.

The region containing p_i

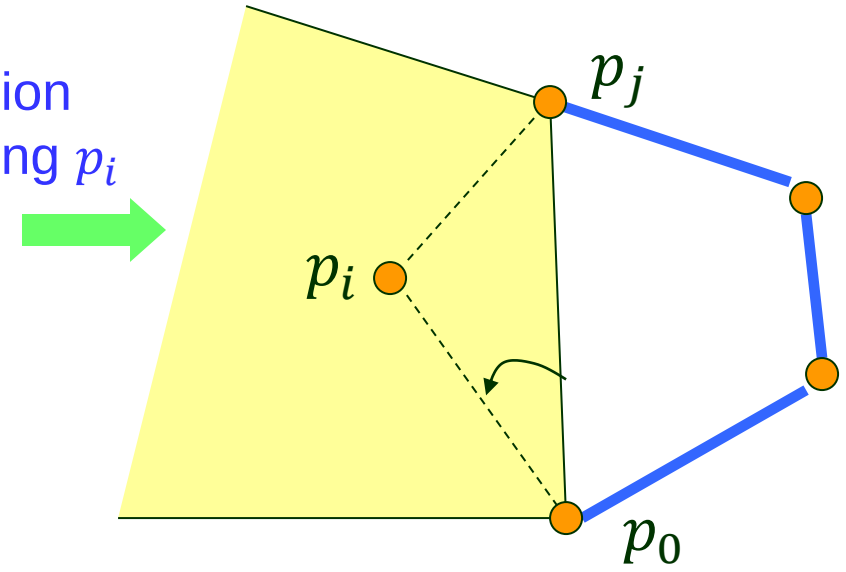


Claim 2 Graham-Scan maintains the ***invariant*** that the points on stack S always form the vertices of a convex polygon in counterclockwise order.

Proof

- The claim holds right after initialization of S when p_0, p_1, p_2 form a triangle (which is obviously convex).
- Popping a point from S preserves the invariant.
- Consider a point p_i being pushed onto S.

The region containing p_i



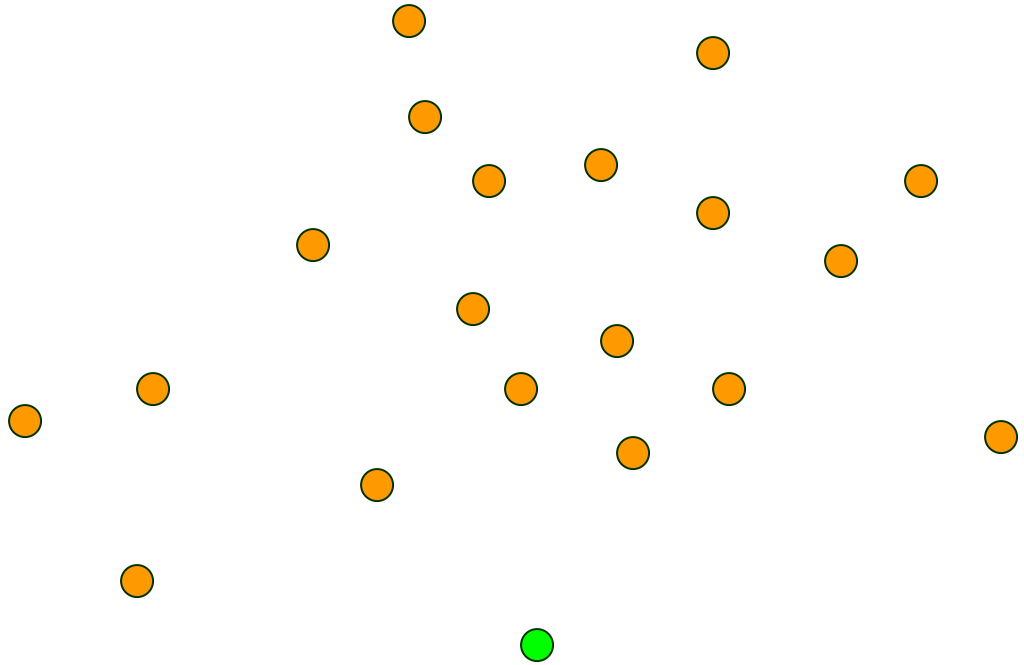
The invariant still holds.

Correctness of Graham's Scan

Theorem If Graham-Scan is run on a set P of at least three points, then a point of P is on the stack S at termination if and only if it is a vertex of CH(P).

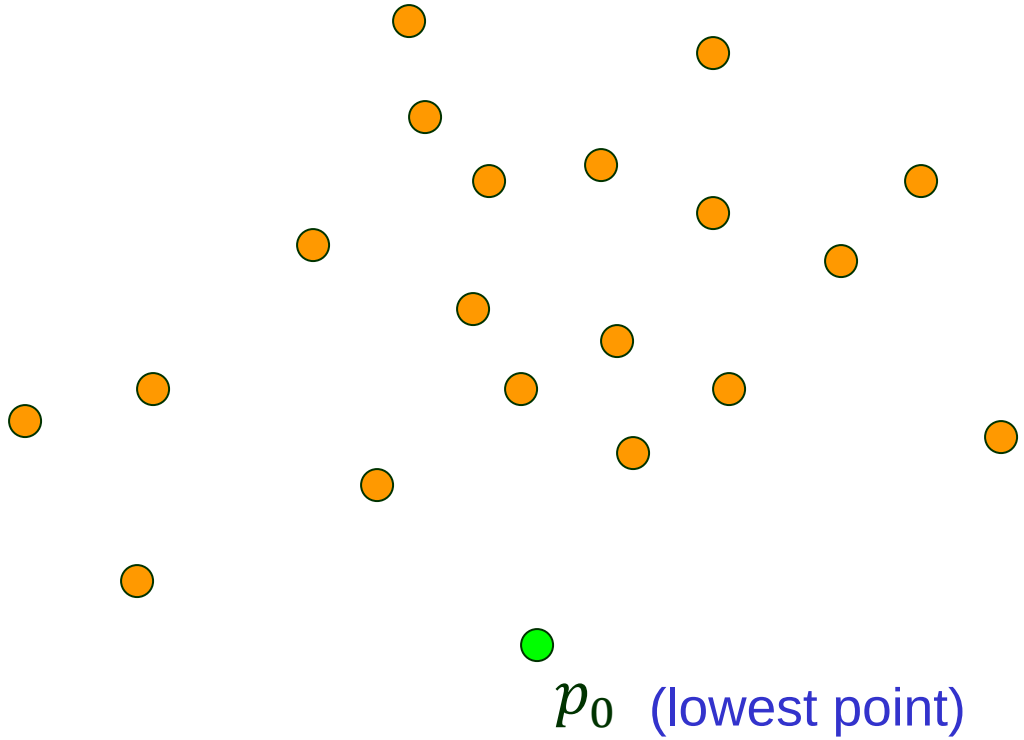
V. Jarvis' March

A "package wrapping" technique



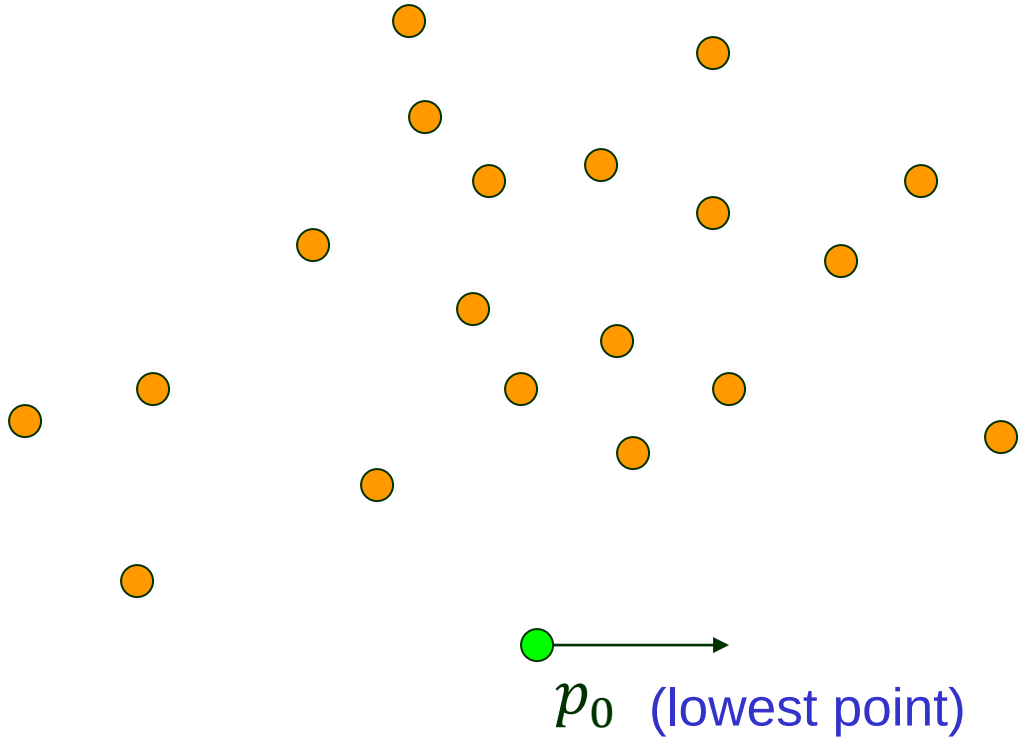
V. Jarvis' March

A "package wrapping" technique



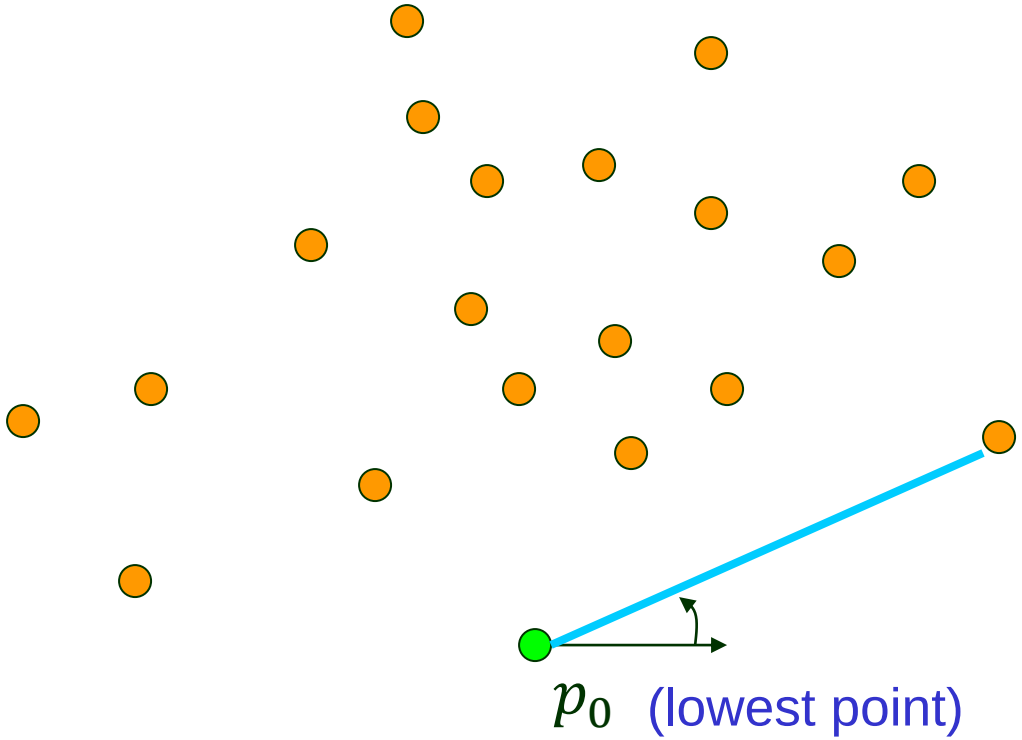
V. Jarvis' March

A "package wrapping" technique



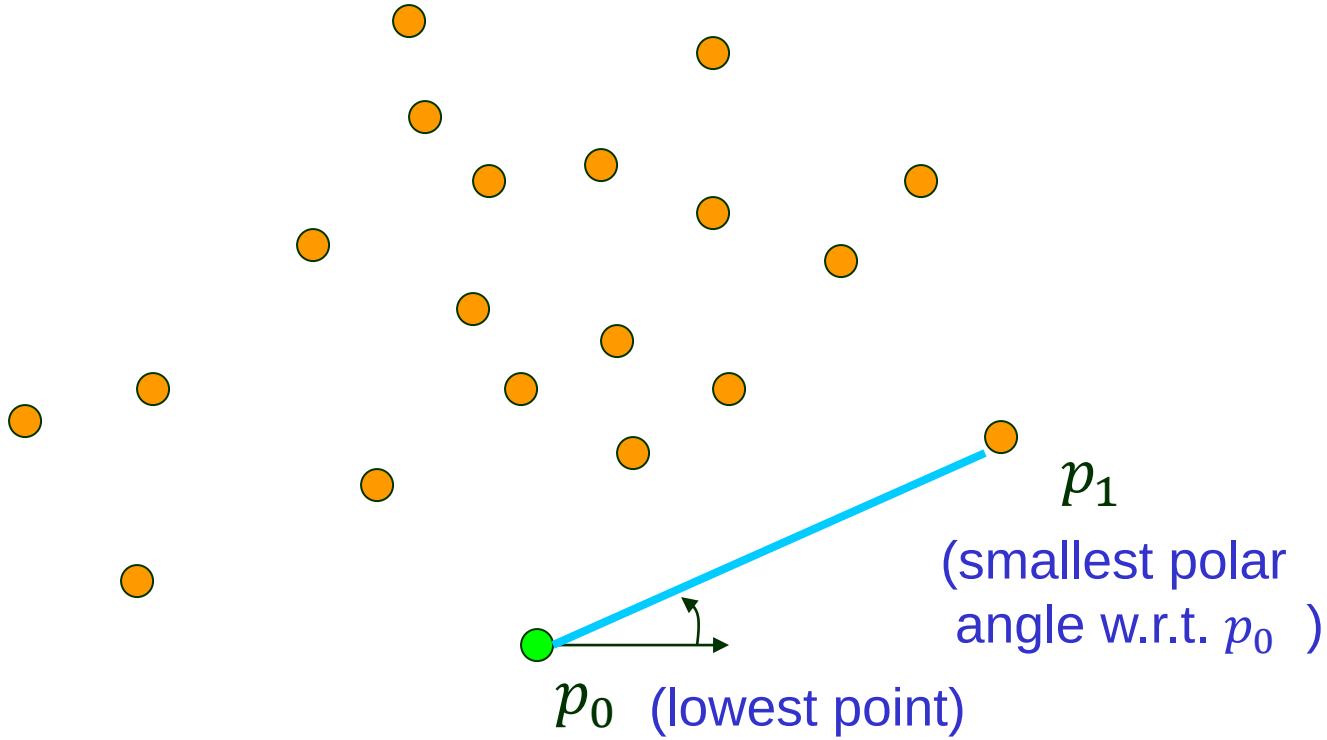
V. Jarvis' March

A "package wrapping" technique



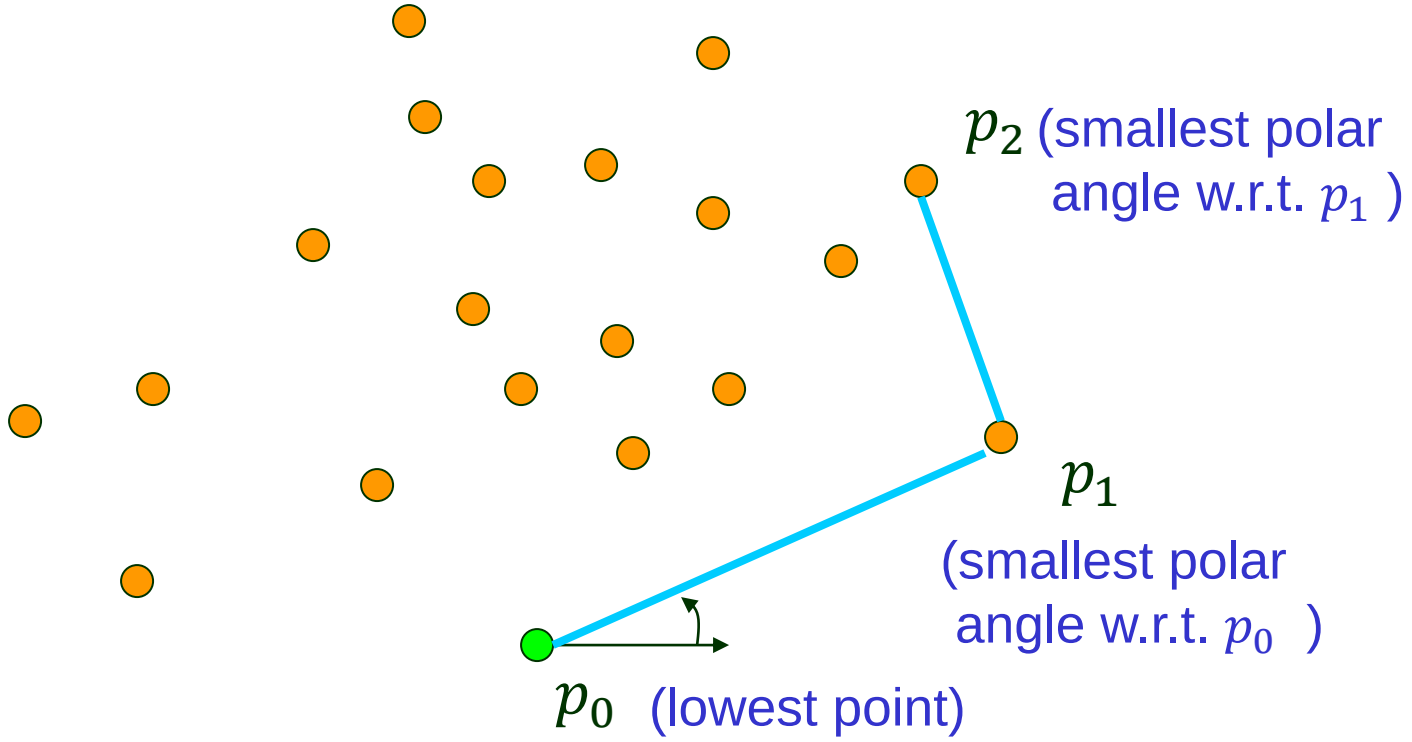
V. Jarvis' March

A "package wrapping" technique



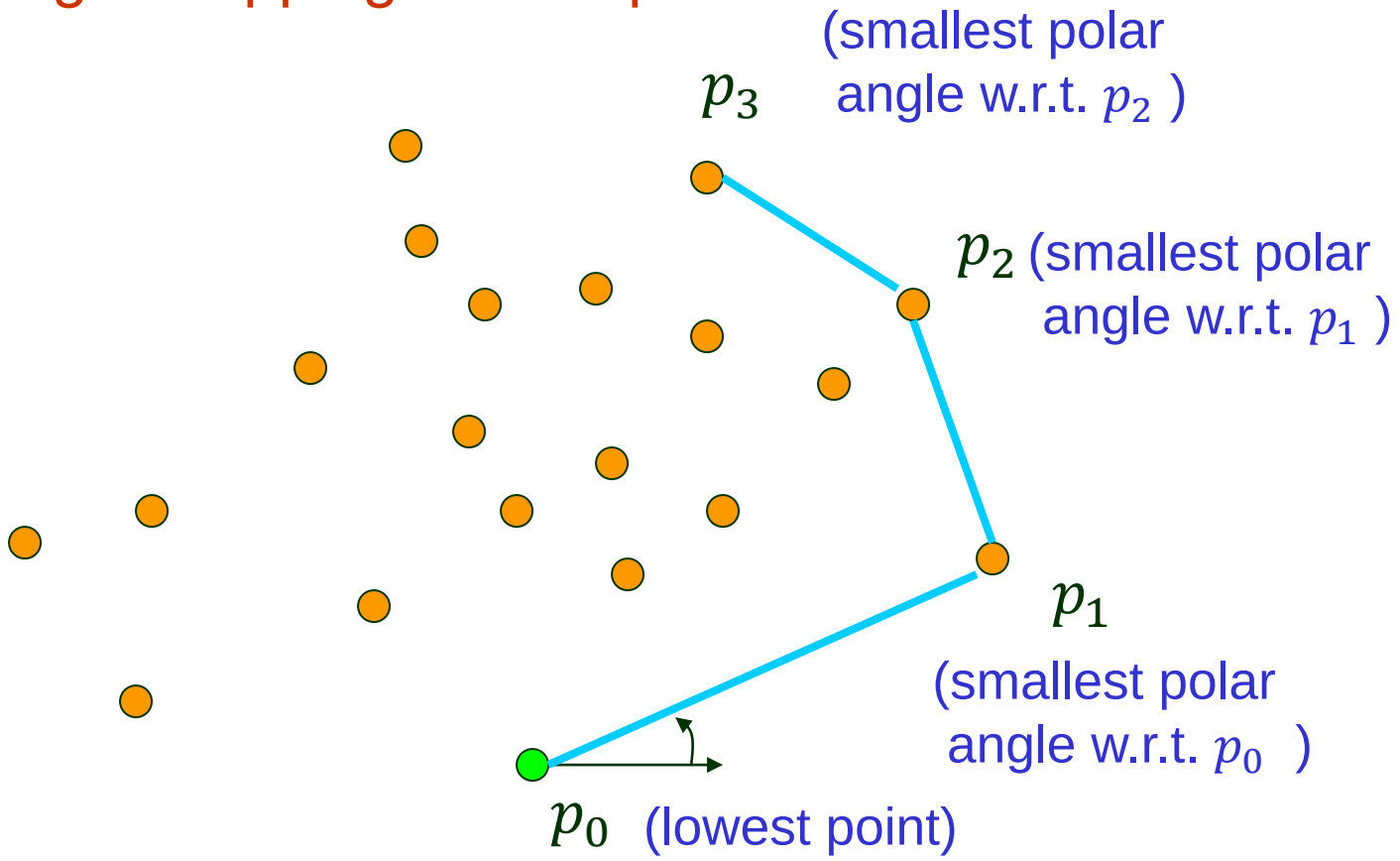
V. Jarvis' March

A "package wrapping" technique



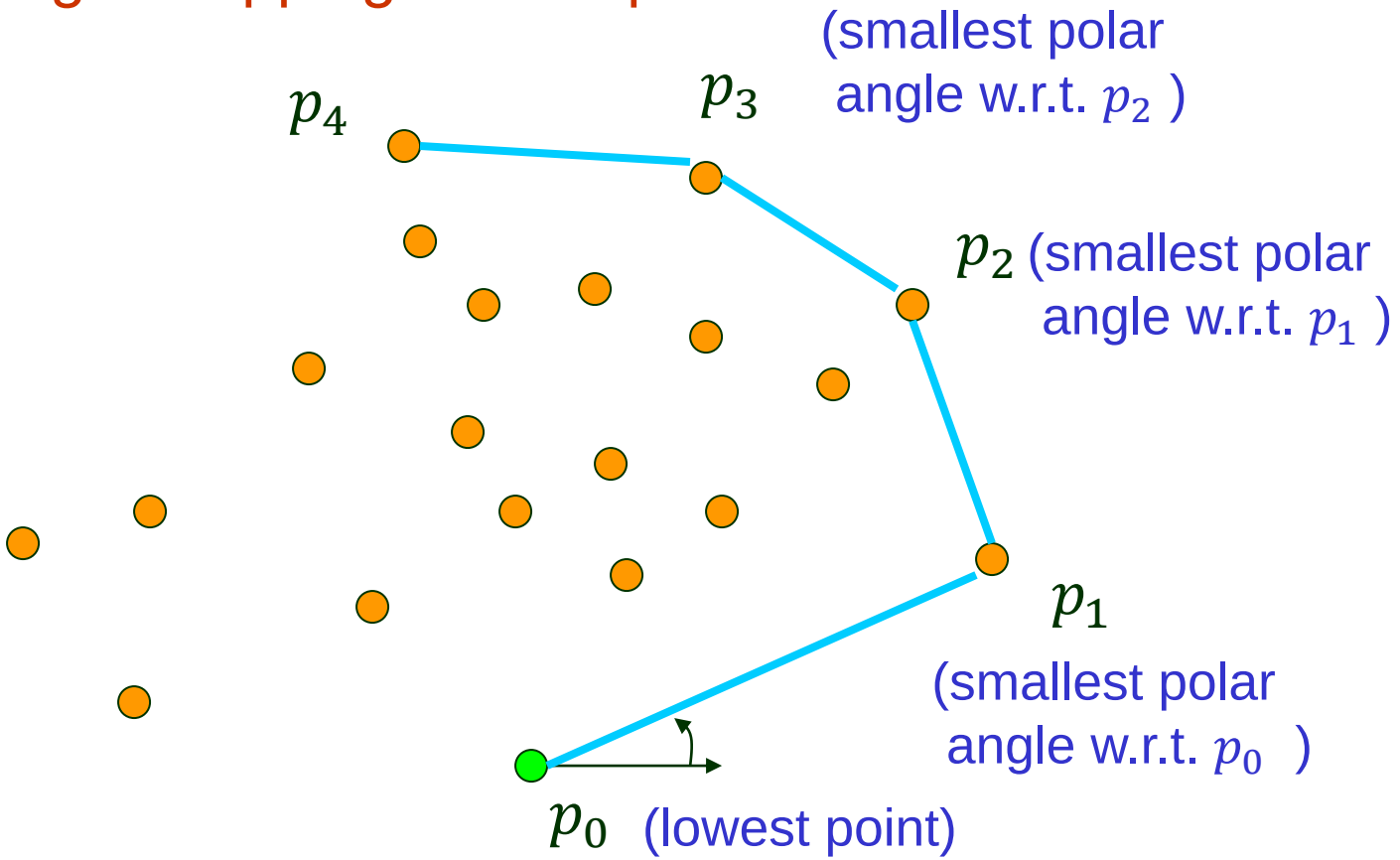
V. Jarvis' March

A "package wrapping" technique



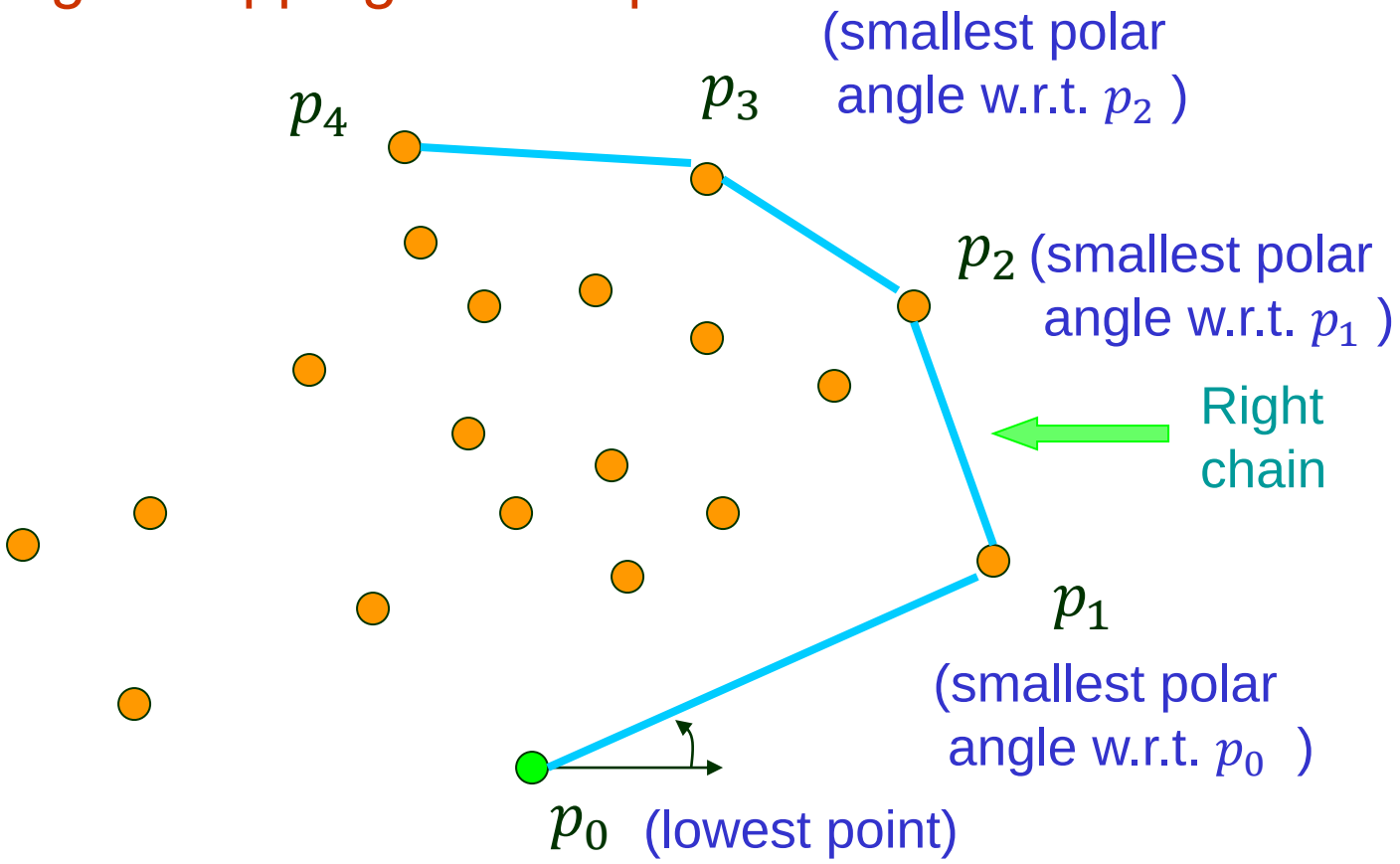
V. Jarvis' March

A "package wrapping" technique



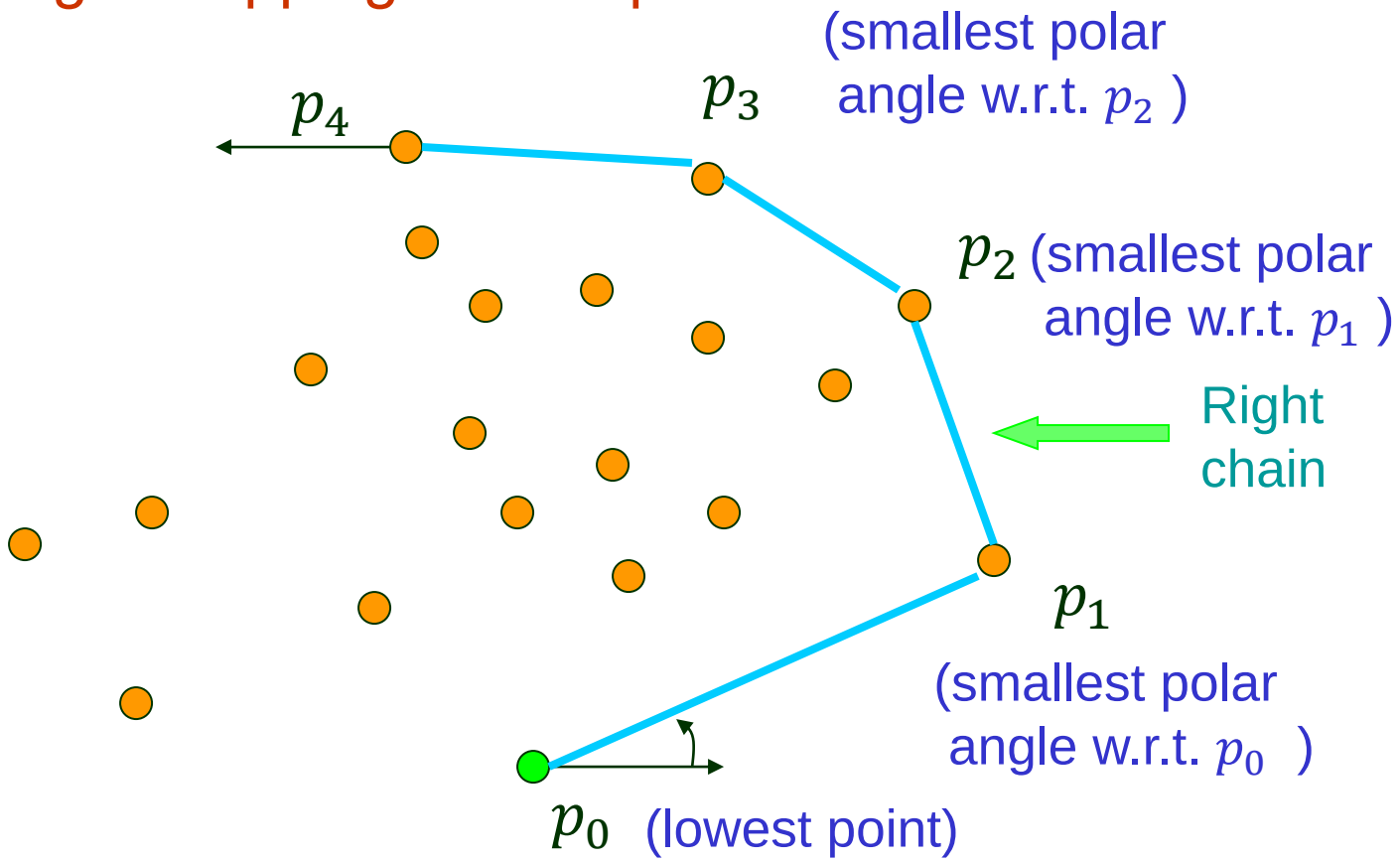
V. Jarvis' March

A "package wrapping" technique



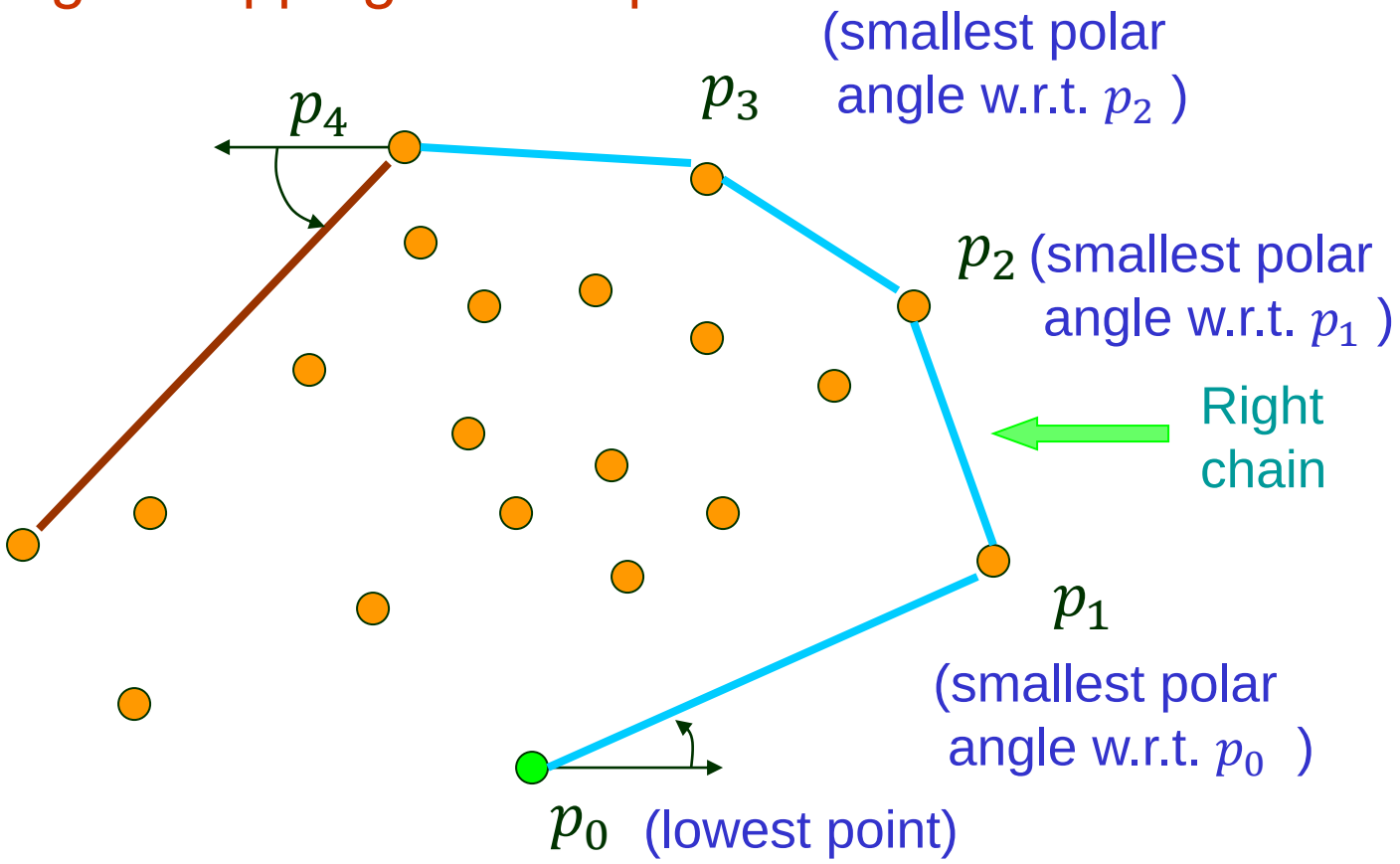
V. Jarvis' March

A "package wrapping" technique



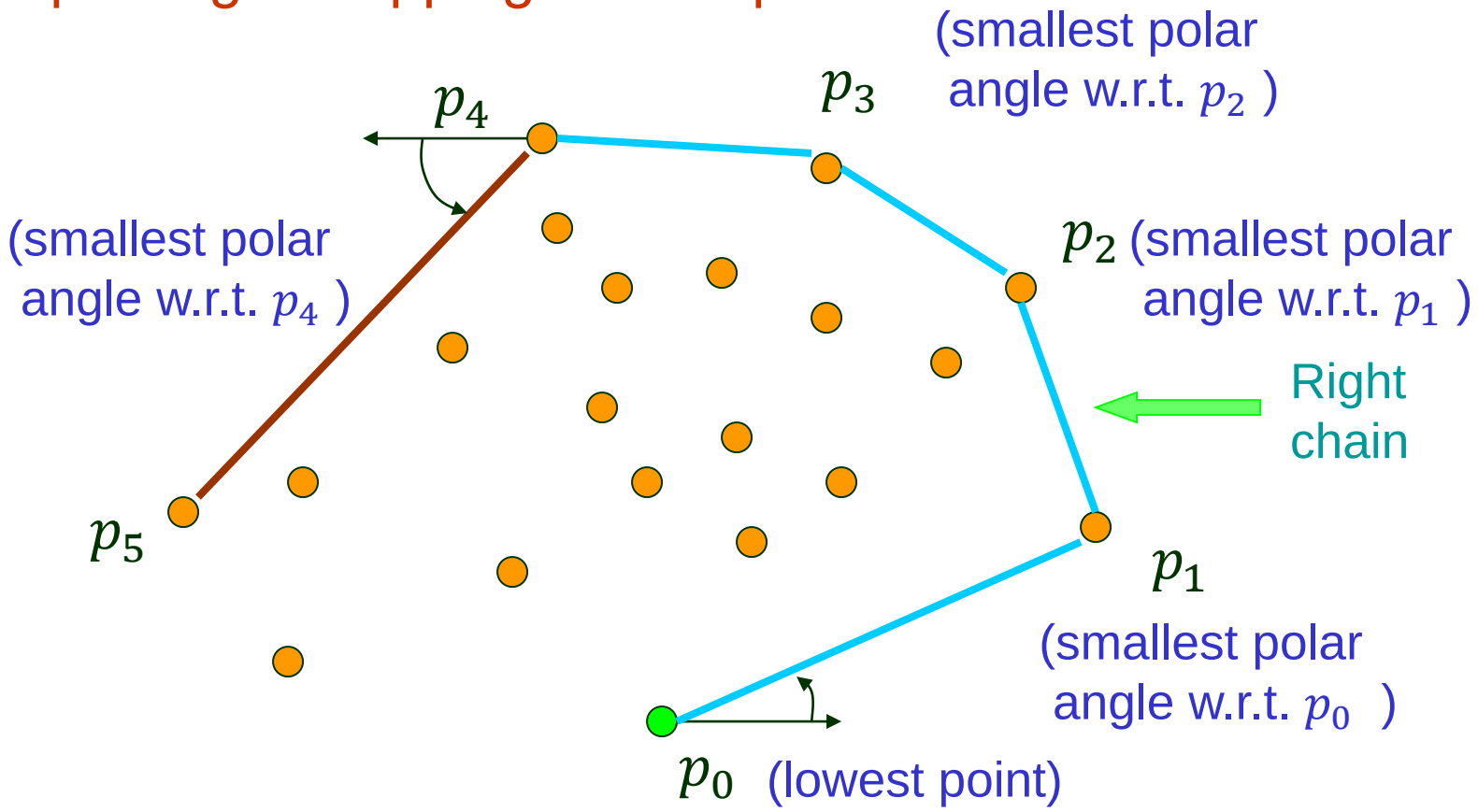
V. Jarvis' March

A "package wrapping" technique



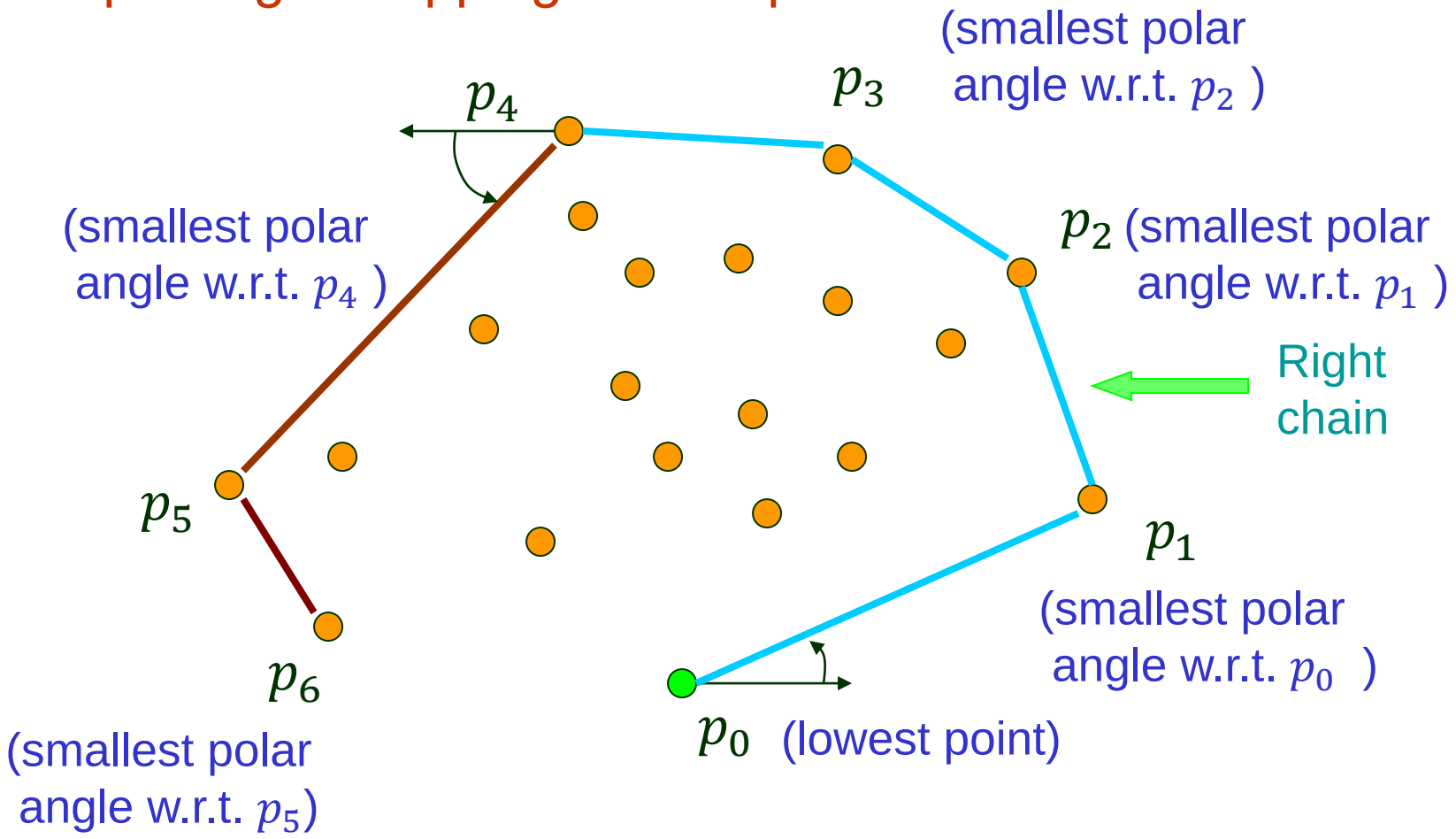
V. Jarvis' March

A "package wrapping" technique



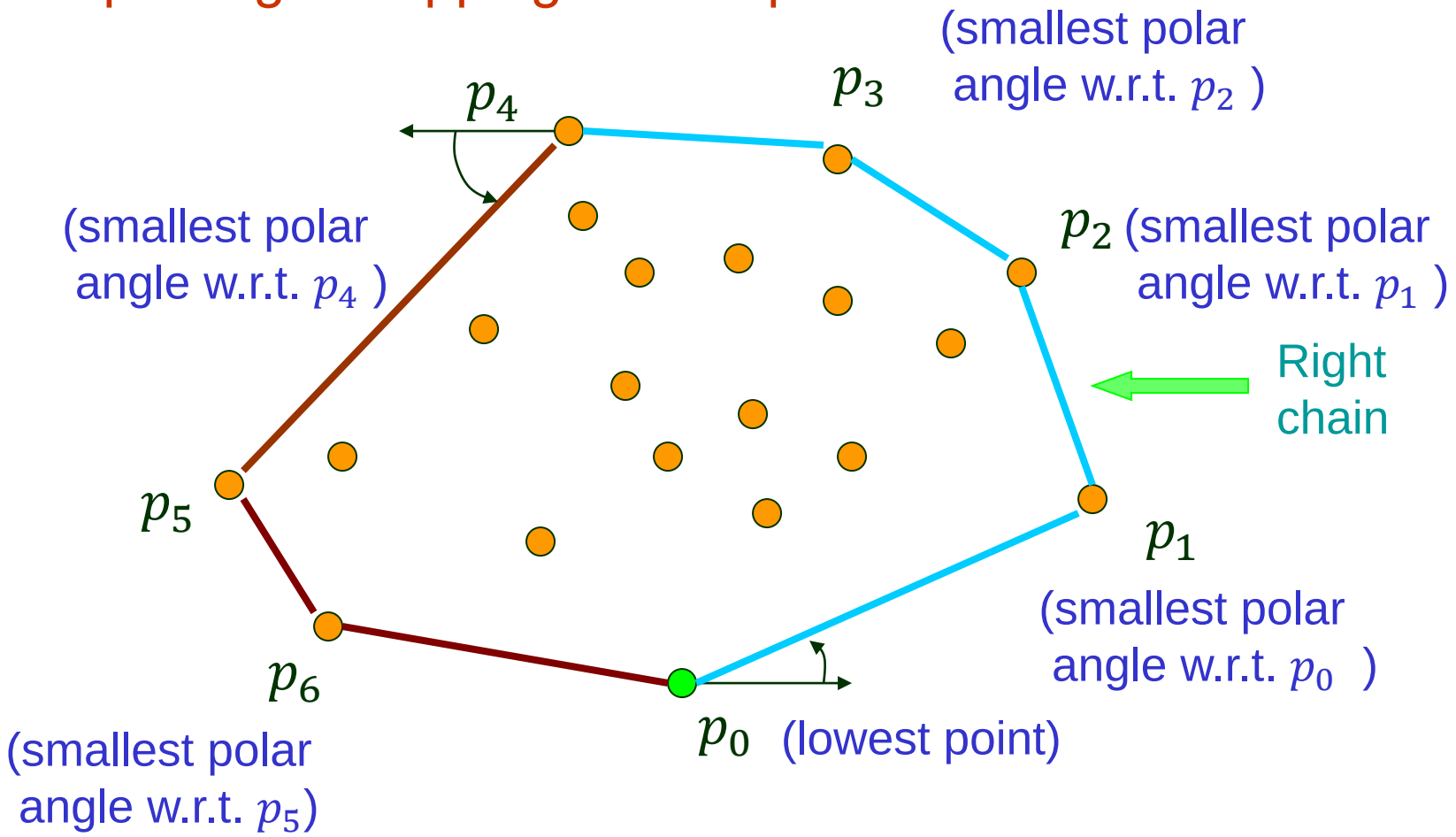
V. Jarvis' March

A "package wrapping" technique



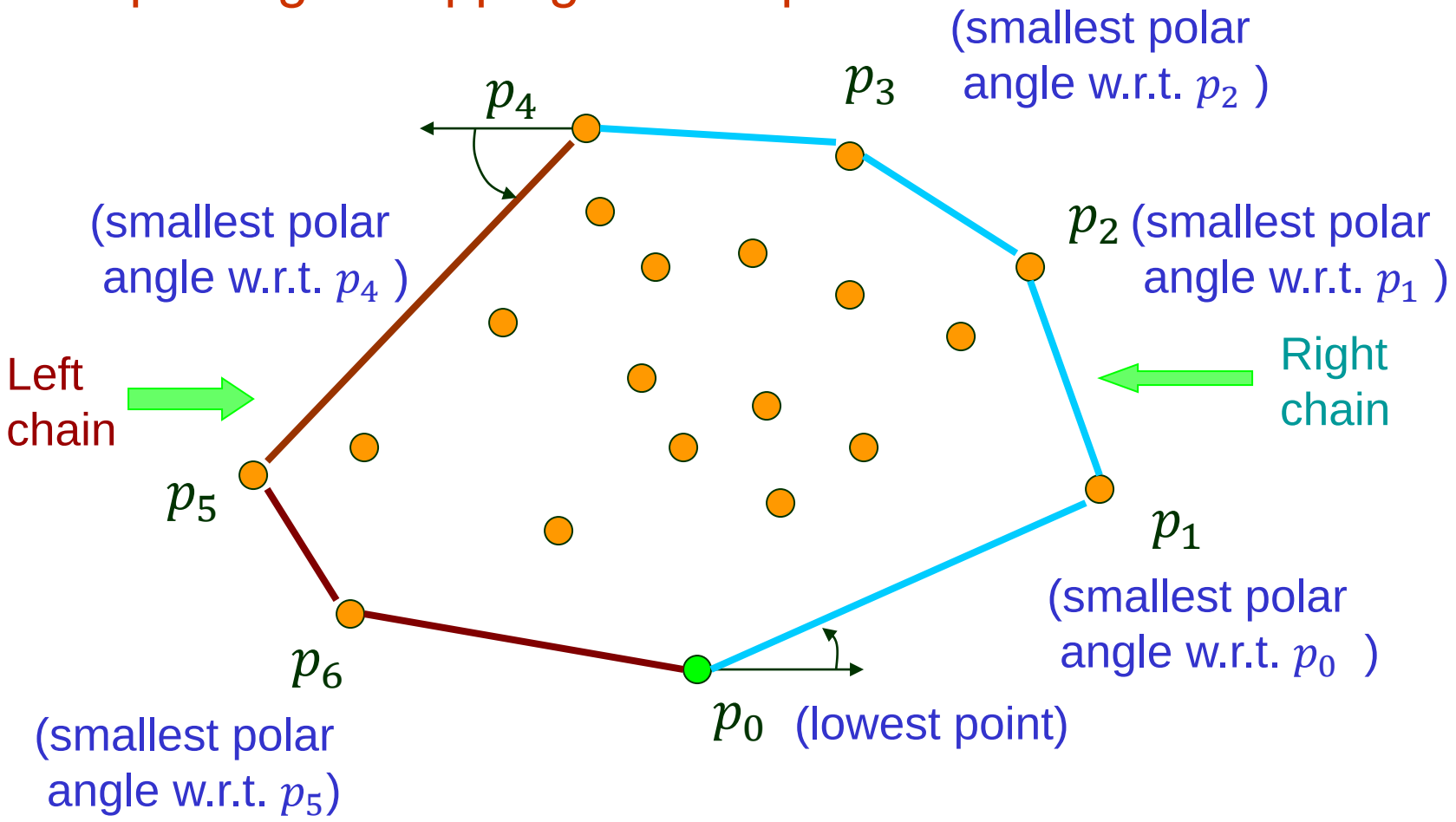
V. Jarvis' March

A "package wrapping" technique



V. Jarvis' March

A "package wrapping" technique



Running Time of Jarvis's March

Let h be the number of vertices of the convex hull.

Running Time of Jarvis's March

Let h be the number of vertices of the convex hull.

- For each vertex, finding the point with the minimum Polar angle, that is, the next vertex, takes time $O(n)$.

Running Time of Jarvis's March

Let h be the number of vertices of the convex hull.

- For each vertex, finding the point with the minimum Polar angle, that is, the next vertex, takes time $O(n)$.

  Comparison between two polar angles can be done using cross product.

Running Time of Jarvis's March

Let h be the number of vertices of the convex hull.

- For each vertex, finding the point with the minimum Polar angle, that is, the next vertex, takes time $O(n)$.

  Comparison between two polar angles can be done using cross product.

Thus $\boldsymbol{O(nh)}$ time in total.