Computational Geometry

The systematic study of algorithms and data structures for geometric objects, with a focus on exact algorithms that are asymptotically fast.

Two key ingredients of a good algorithmic solution:

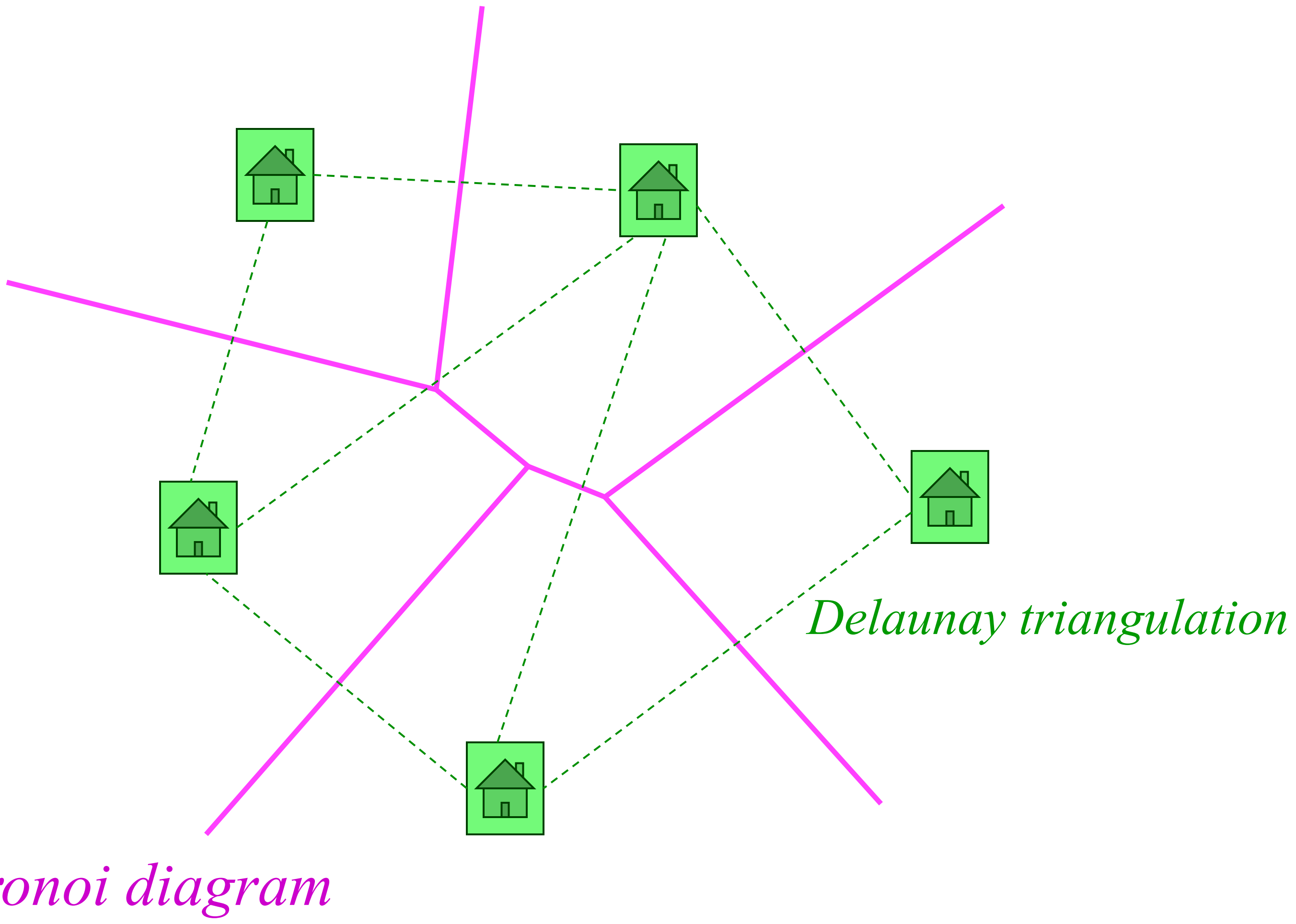
- Thorough understanding of the problem geometry.
- Proper application of algorithmic techniques and data structures.

Com S 418 (prereq. Com S 311)

`http://www.cs.iastate.edu/~cs518/`

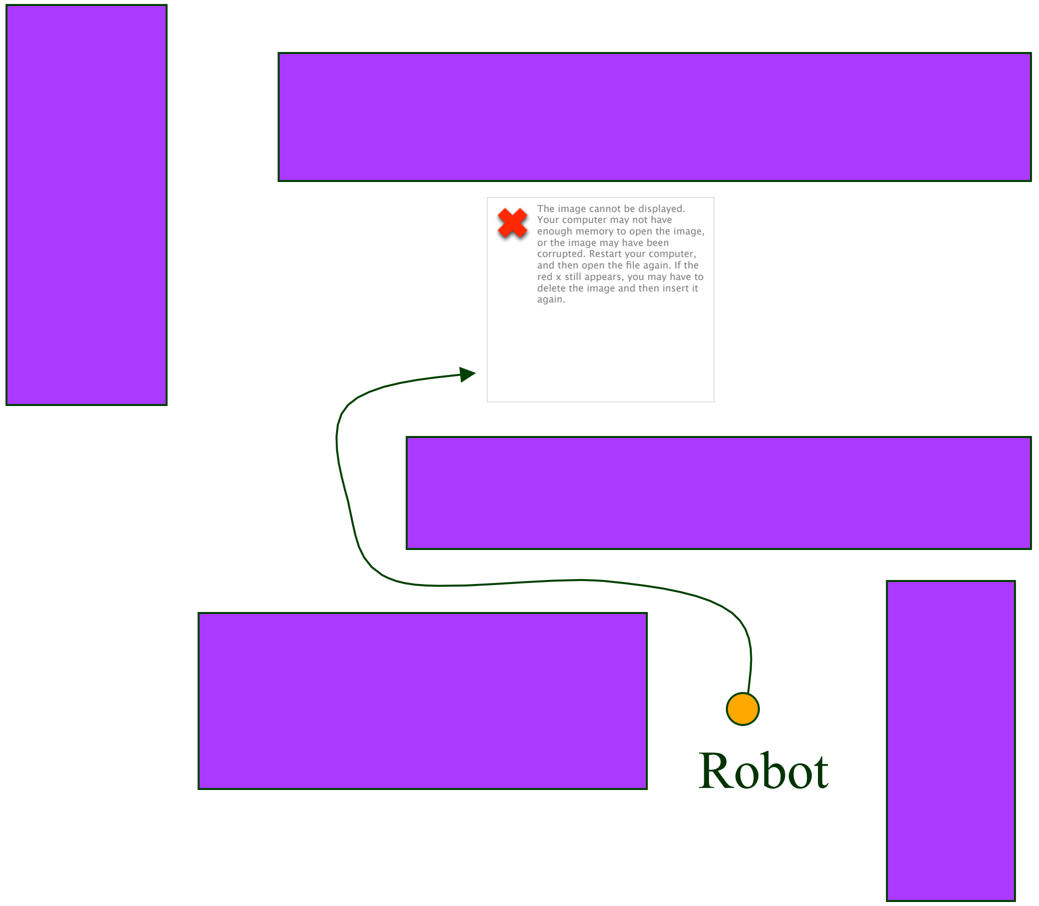
Example 1 - Proximity

Closest café on campus?

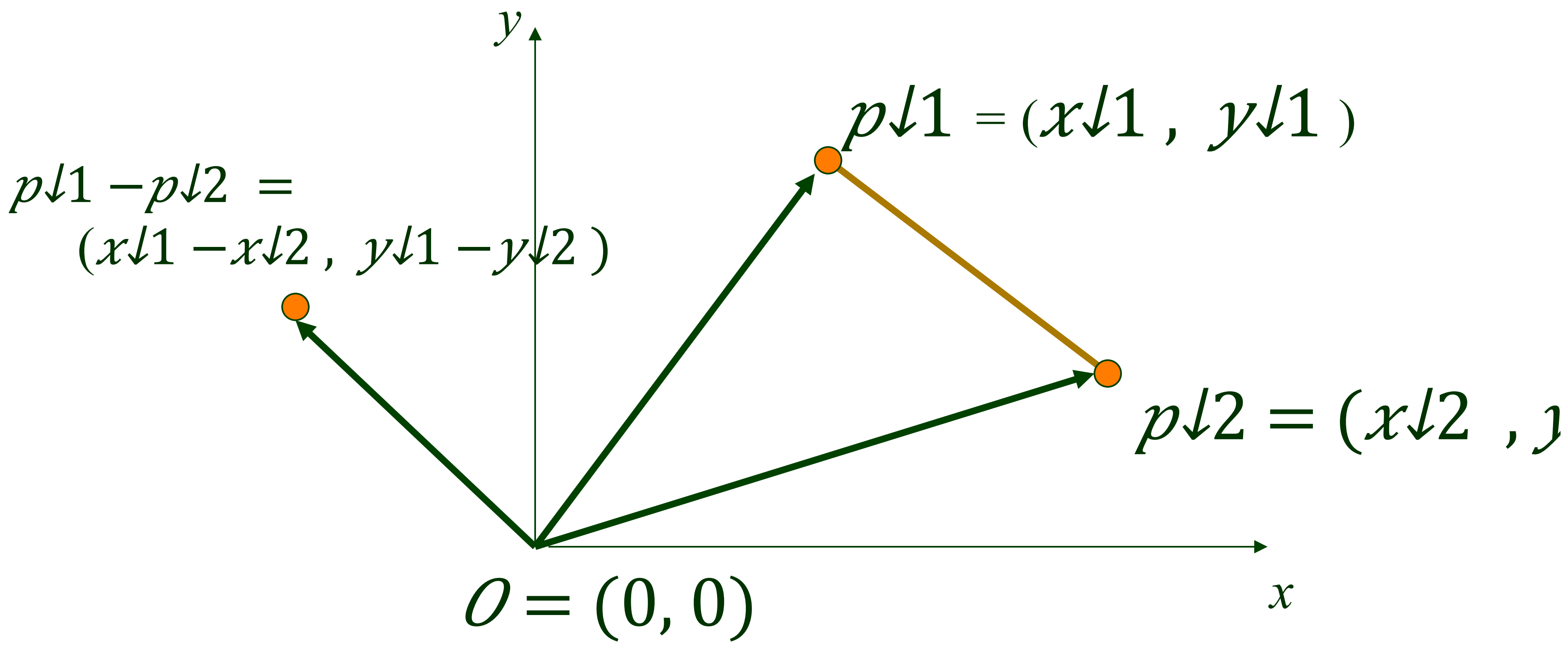


Example 2 - Path Planning

How can a robot find a *short* route to the destination that avoids all obstacles?


The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.
Robot

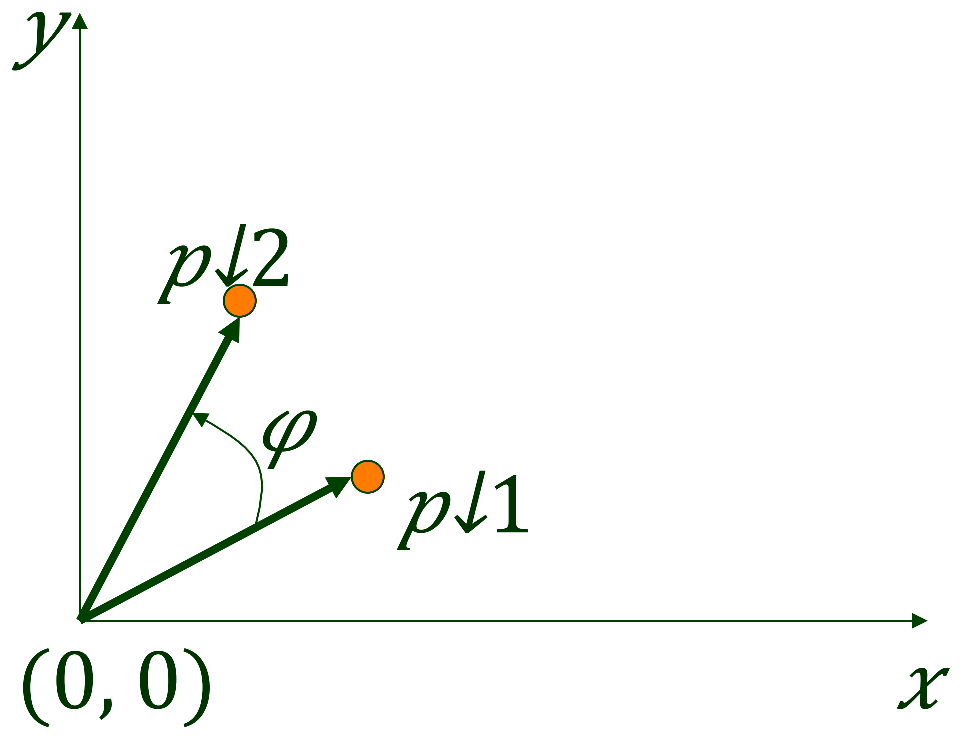
Line Segments & Vectors



Points (vectors): $p_1 , p_2 , p_1 - p_2 = p_2 p_1$

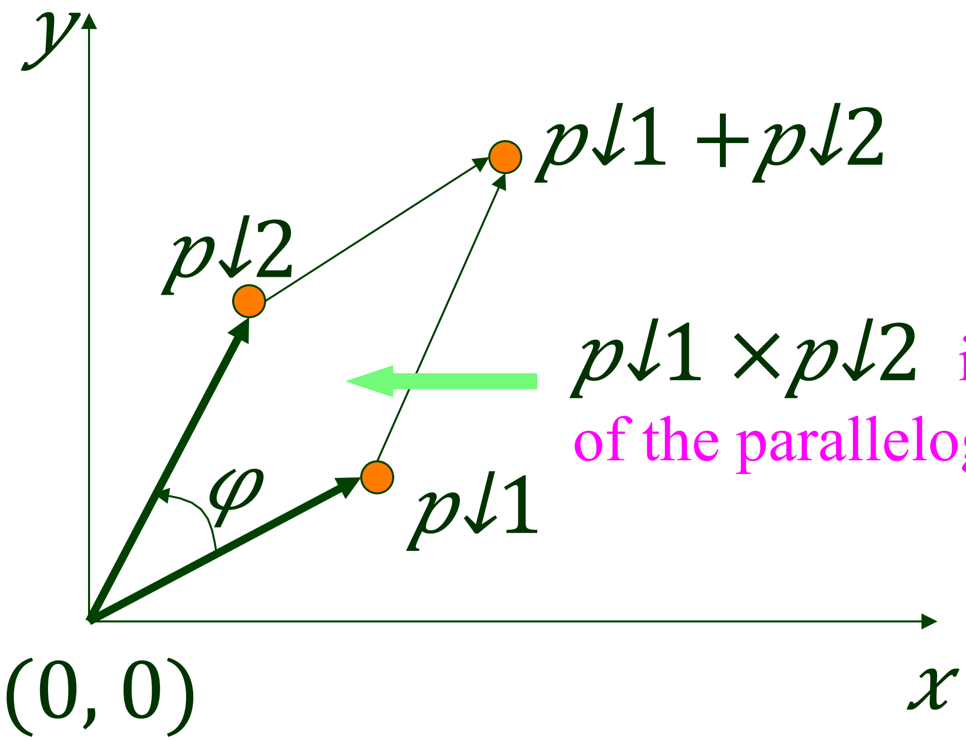
Line segment: $p_2 p_1 = p_1 p_2$

Dot (Inner) Product



$$p_1 \cdot p_2 = x_1 x_2 + y_1 y_2 = p_2 \cdot p_1 = |p_1| \, |p_2| \cos$$

Cross (Vector) Product



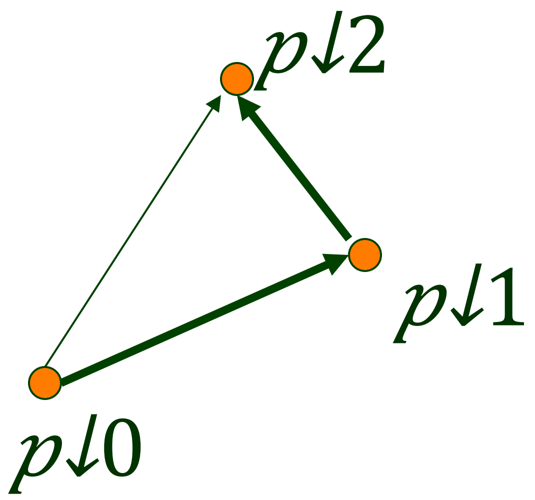
$p_1 \times p_2$ is the *signed* area of the parallelogram.

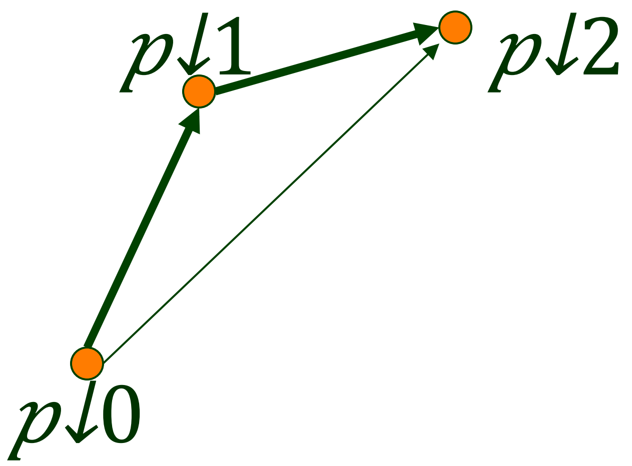
$$p_1 \times p_2 = x_1 y_2 - x_2 y_1 = -p_2 \times p_1 = |p_1|\,|p_2|\,\sin\varphi$$

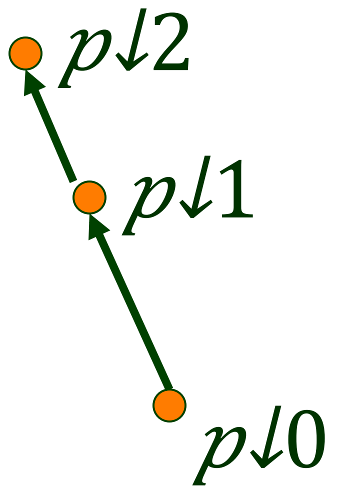
p_1 and p_2 are *collinear* with the origin iff $p_1 \times p_2 =$

Turning of Consecutive Segments

Segments $p_0 p_1$ and $p_1 p_2$. Move from p_0 to $p_{}$





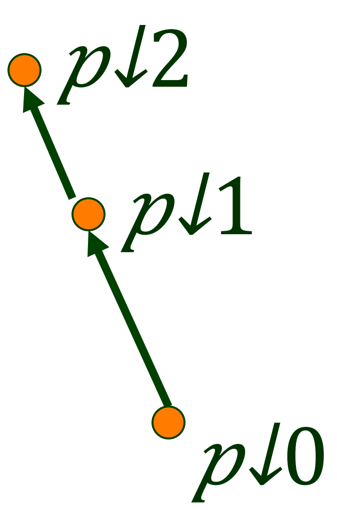


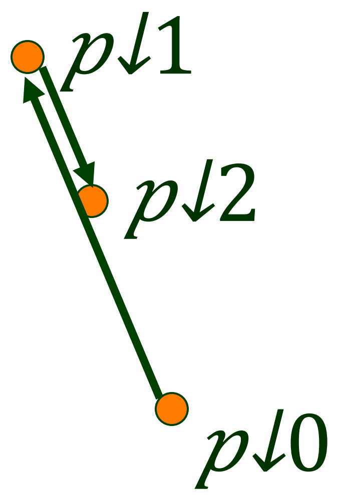
Counterclockwise — Clockwise — Turn of 0 or π

$p_0 p_1 \times p_1 p_2 > 0$ $\quad$ $p_0 p_1 \times p_1 p_2 < 0$ $\quad$ $p_0 p_1 \times p_1$

Collinear Points

$p_0 p_1 \times p_1 p_2 = 0 \Rightarrow p_0, p_1, p_2$ are collinear.





No change of direction

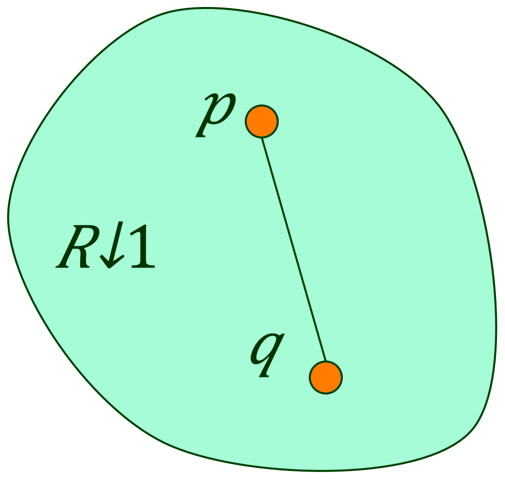
Direction reversal

$p_0 p_1 \cdot p_1 p_2 > 0$ $p_0 p_1 \cdot p_1 p_2 <$

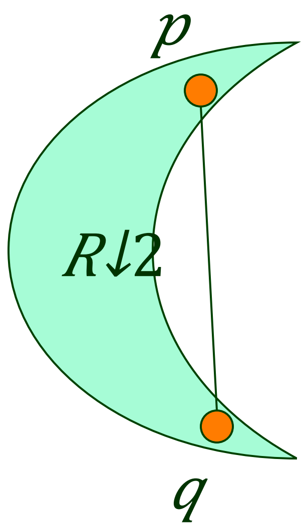
Convex Sets & Concave Sets

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

Otherwise, it is called *concave*.

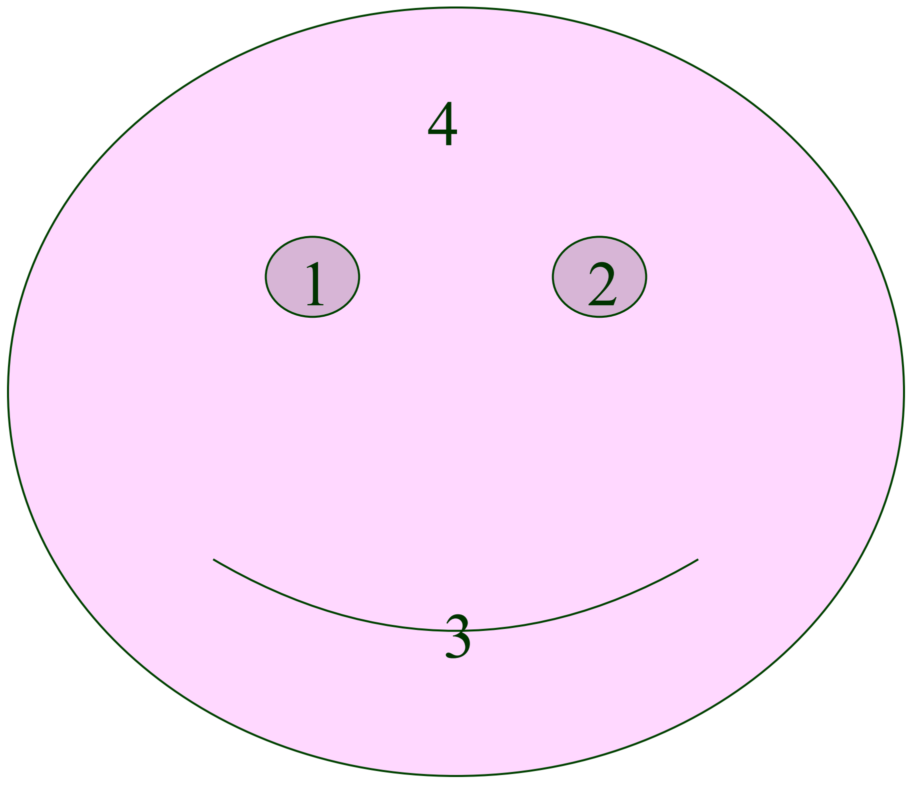


Convex



Concave

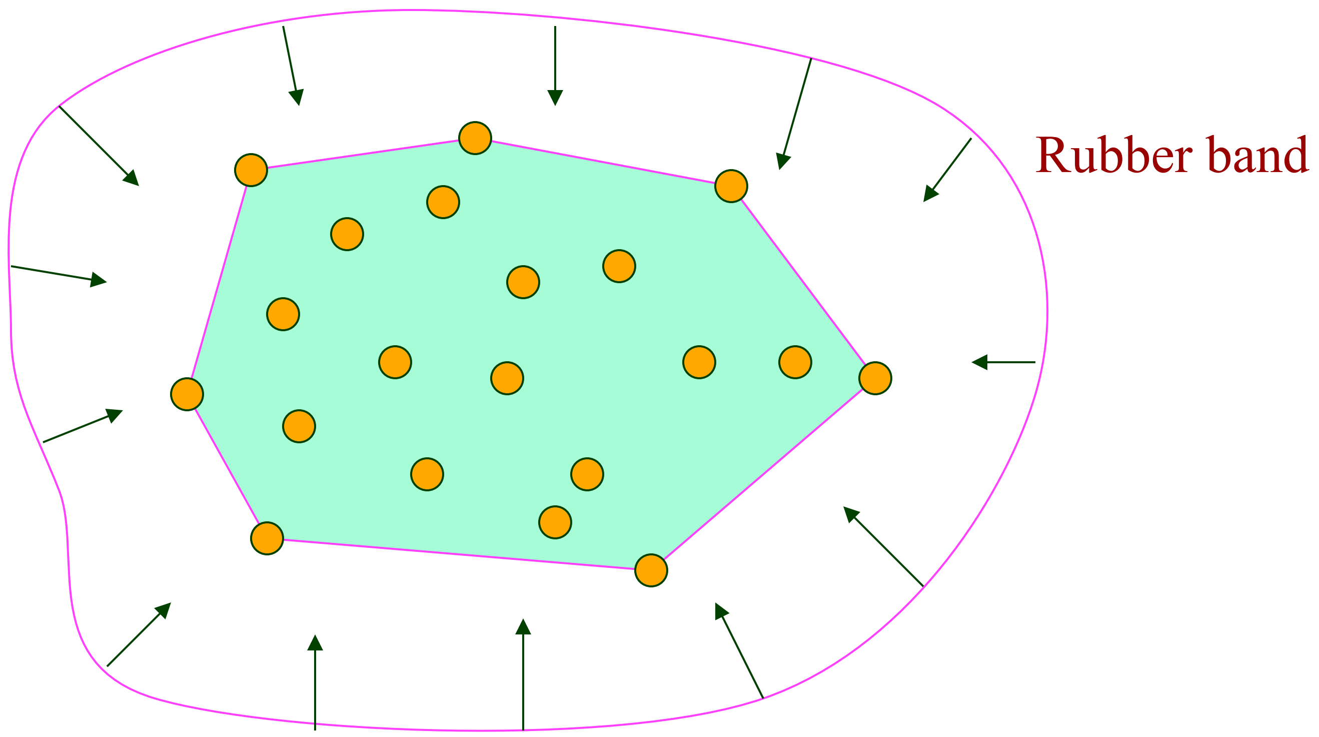
An Example



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.



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 Hulls

- The convex hull of a single point is itself.

- The convex hull of several collinear points is the line segment joining the leftmost and rightmost ones of them.

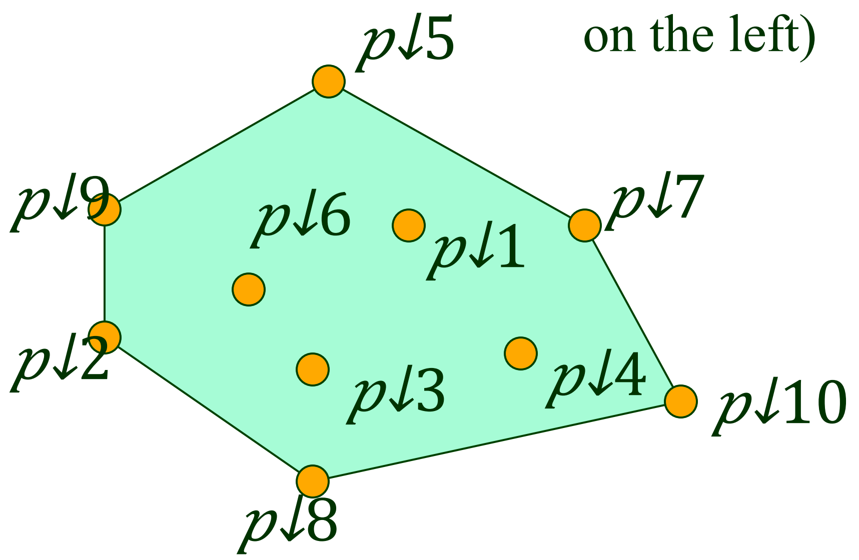
The Convex Hull Problem

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)

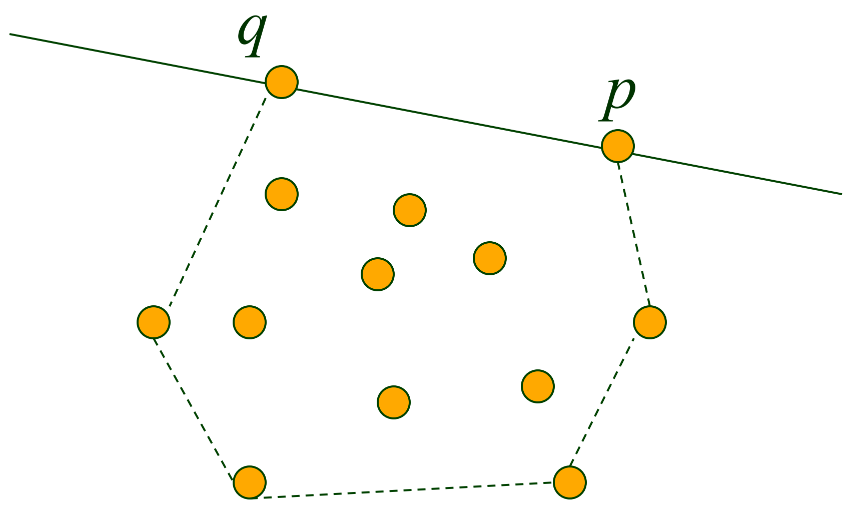
Example



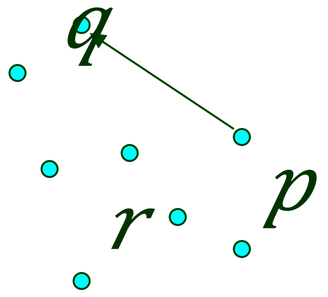
Output: p_8, p_{10}, p_7, p_5, p_9, p_2.

Edges of a Convex Hull

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



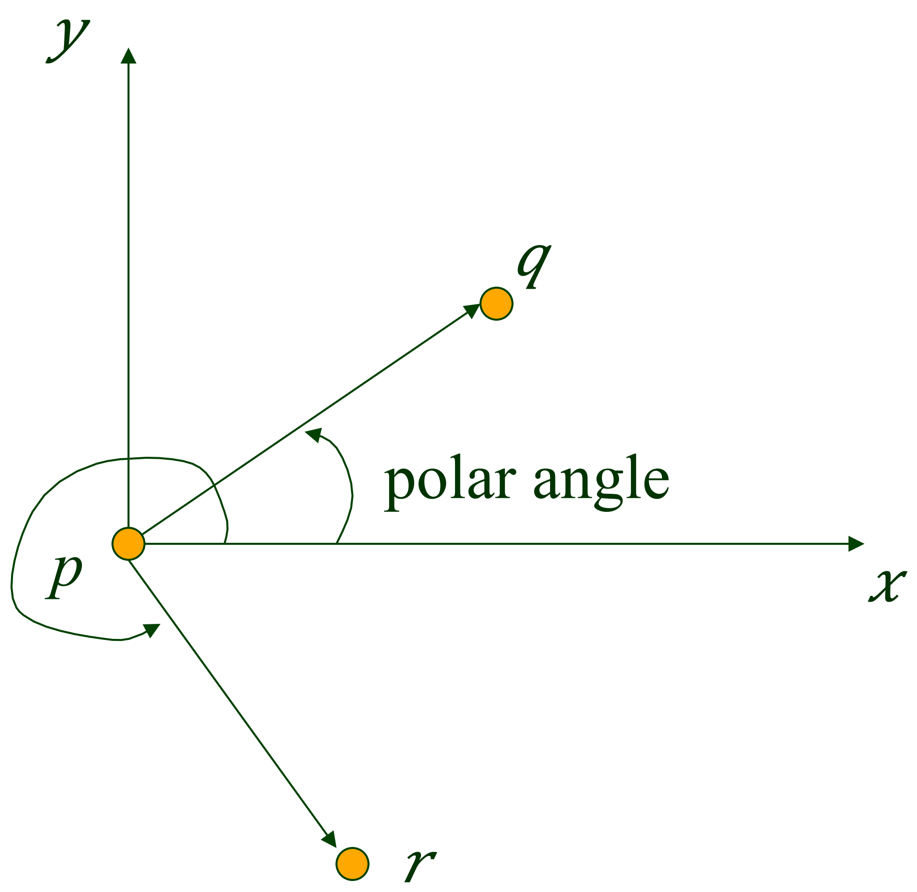
A Slow Convex Hull Algorithm



```
Slow-Convex-Hull($P$)
    $E \leftarrow \{\}$    // set of directed edges of CH($P$) that bounds the
                           // points of $P$ on the right.
    for every ordered pair $(p, q)$, where $p, q \in P$ and $p \neq q$   // [illegible]
        do valid ← true
            for every point $r \neq p$ or $q$          // $n$ [illegible] 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 $E \leftarrow E \cup \{ pq \}$       // $pq$ and $qp$ cannot be
    From $E$ construct a list $L$ of vertices of CH($P$), sorted in
```

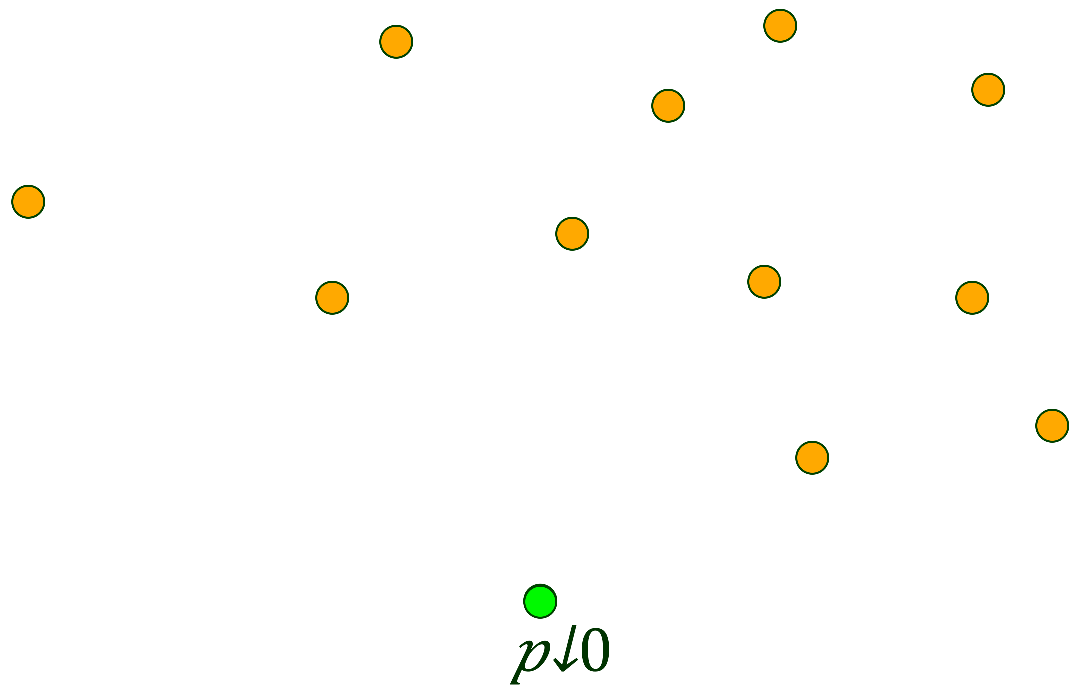
Running time [illegible] (n^3)

Polar Angle



Graham's Scan (1972)

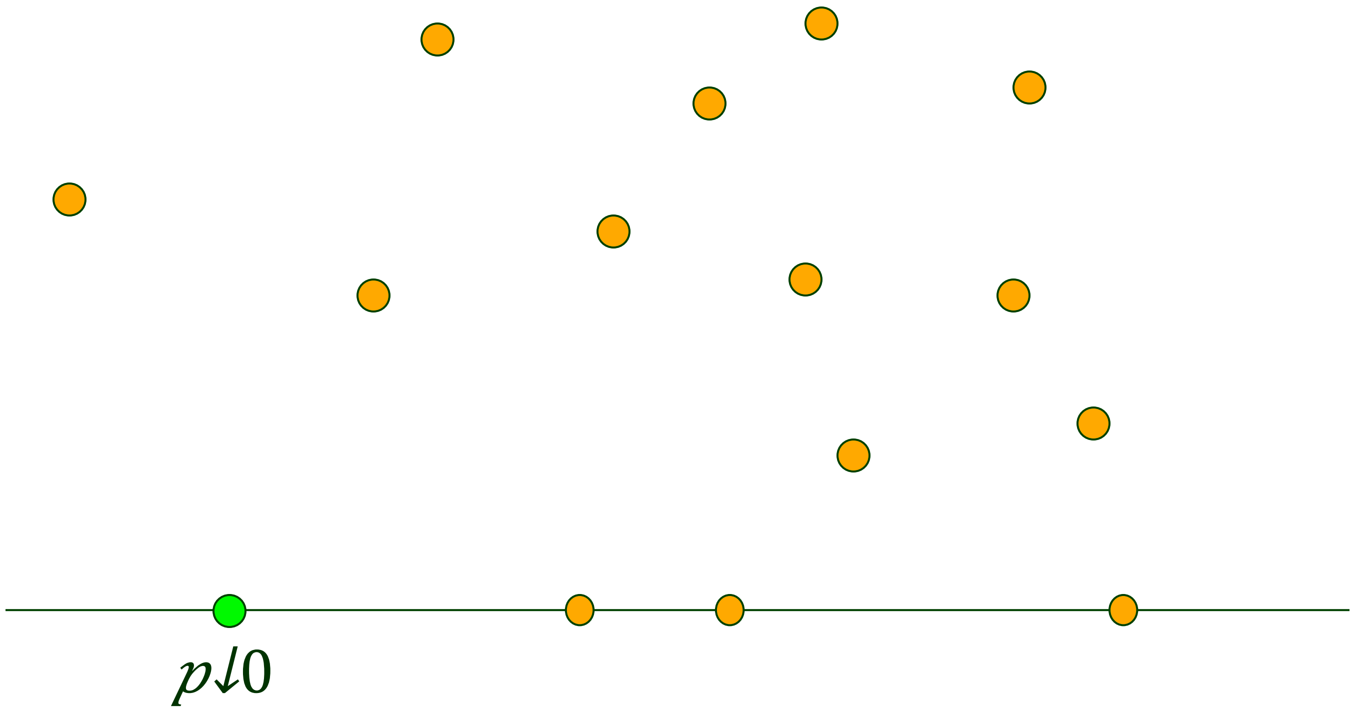
1) Select the node with the smallest y coordinate.



This node will be a vertex of the convex hull.

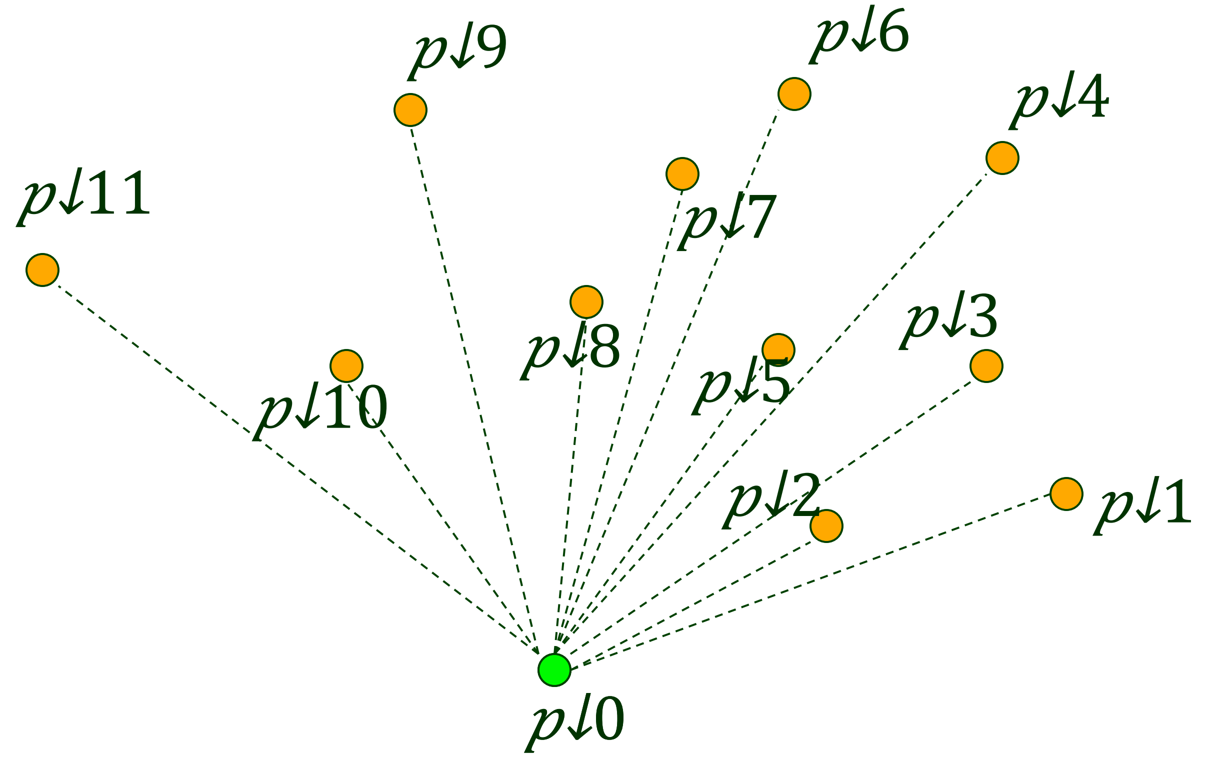
Tie Breaking (1)

When more than one point has the smallest y coordinate, pick the *leftmost* one.



Sorting by Polar Angle

2) Sort by polar angle with respect to p_0 .



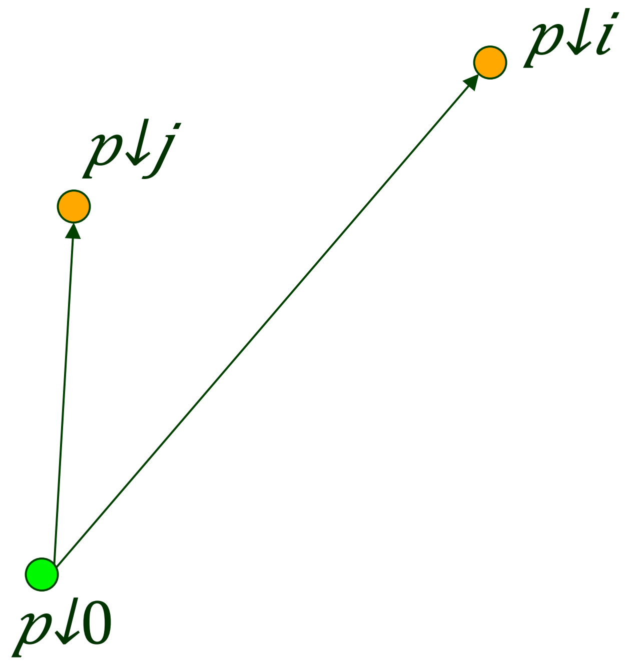
Labels are in the polar angle order.

No Polar Angle Evaluation

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

Use cross product!

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



Tie Breaking (2)

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

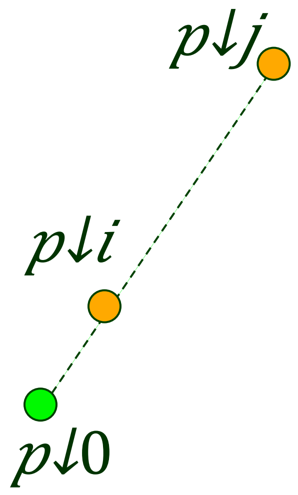
Order them by distance from p_0.

$$p_i < p_j \text{ if } p_0 p_i \times p_0 p_j = 0 \text{ and } |p_0 p_i| < |p_0 p_j|$$

No square roots.
Use dot product!

$$\Updownarrow$$

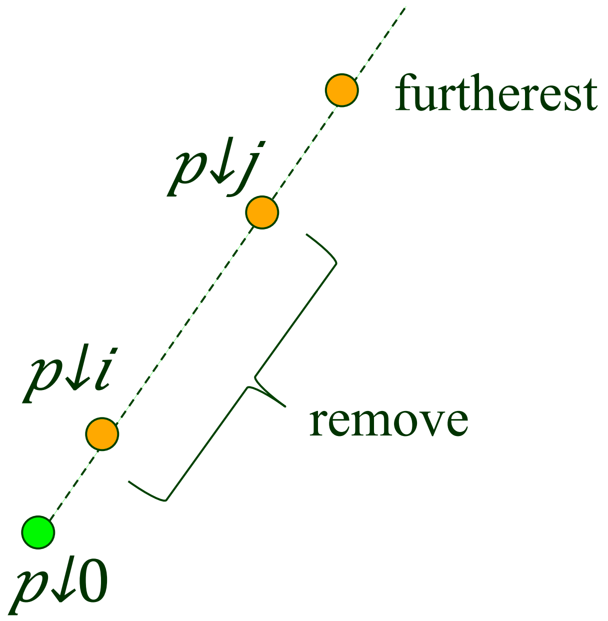
$$p_0 p_i \cdot p_0 p_i < p_0 p_j \cdot p_0 p_j$$



Point Elimination

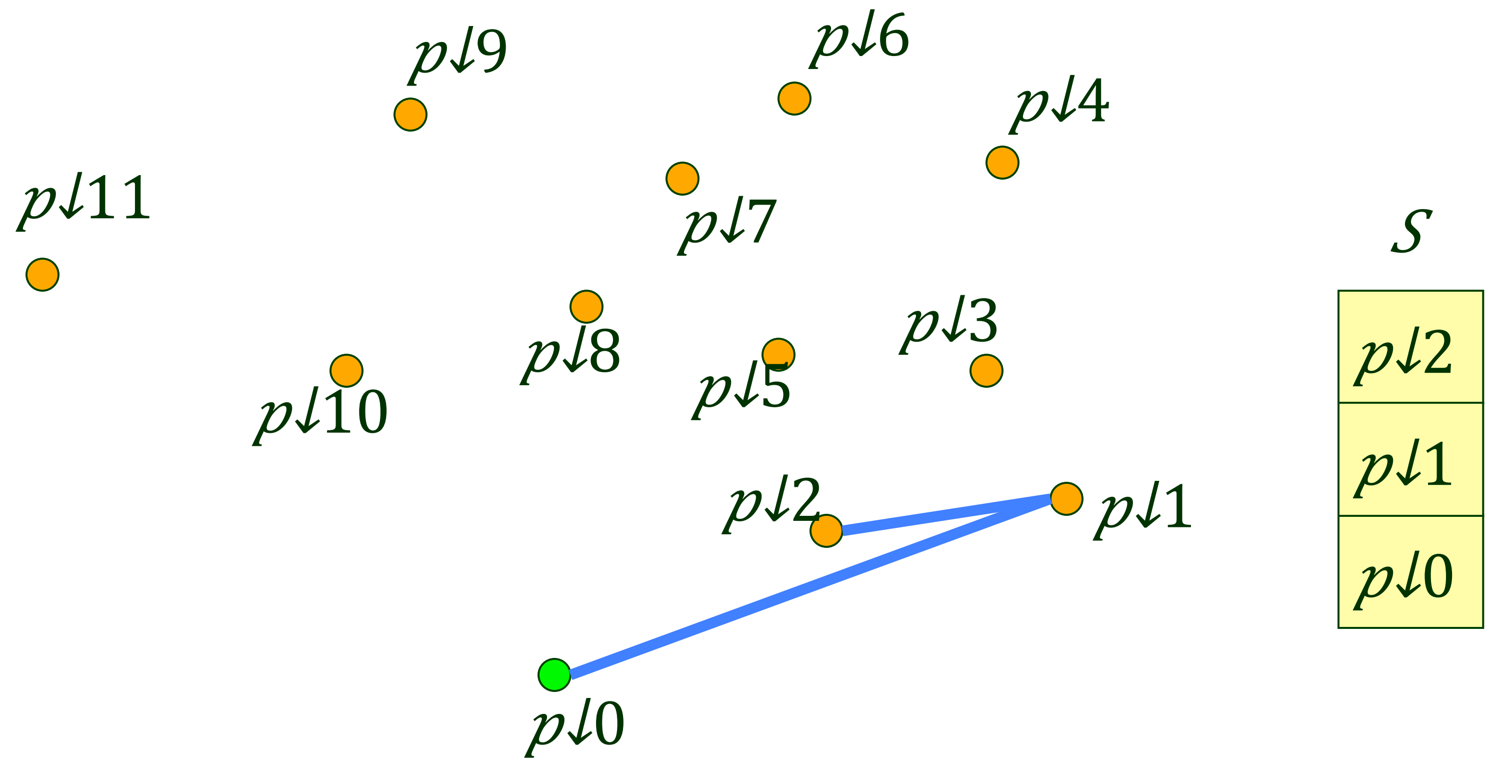
When multiple points have the same polar angle, keep the one *furthest* from p_0 .

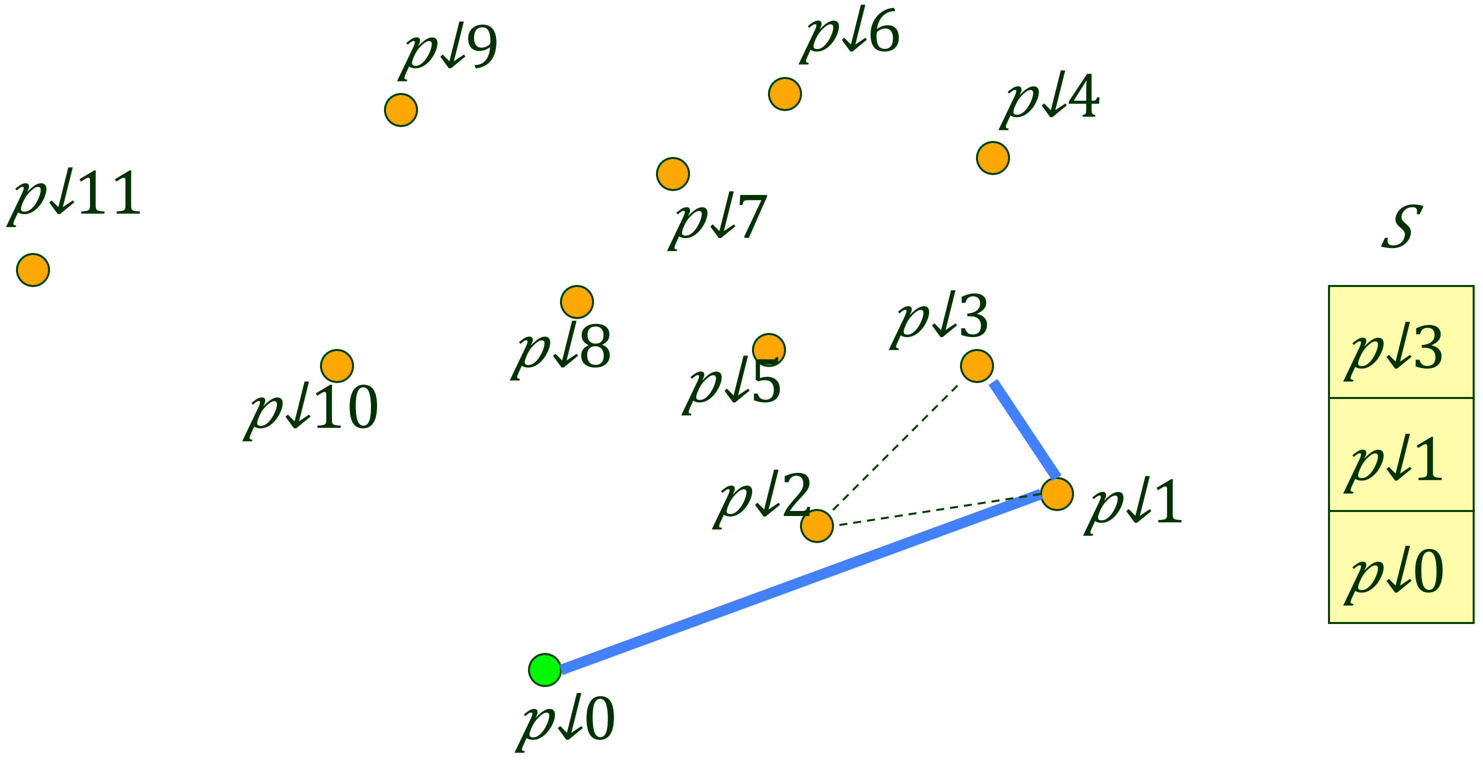
Remove the rest since they cannot possibly be the hull vertices.

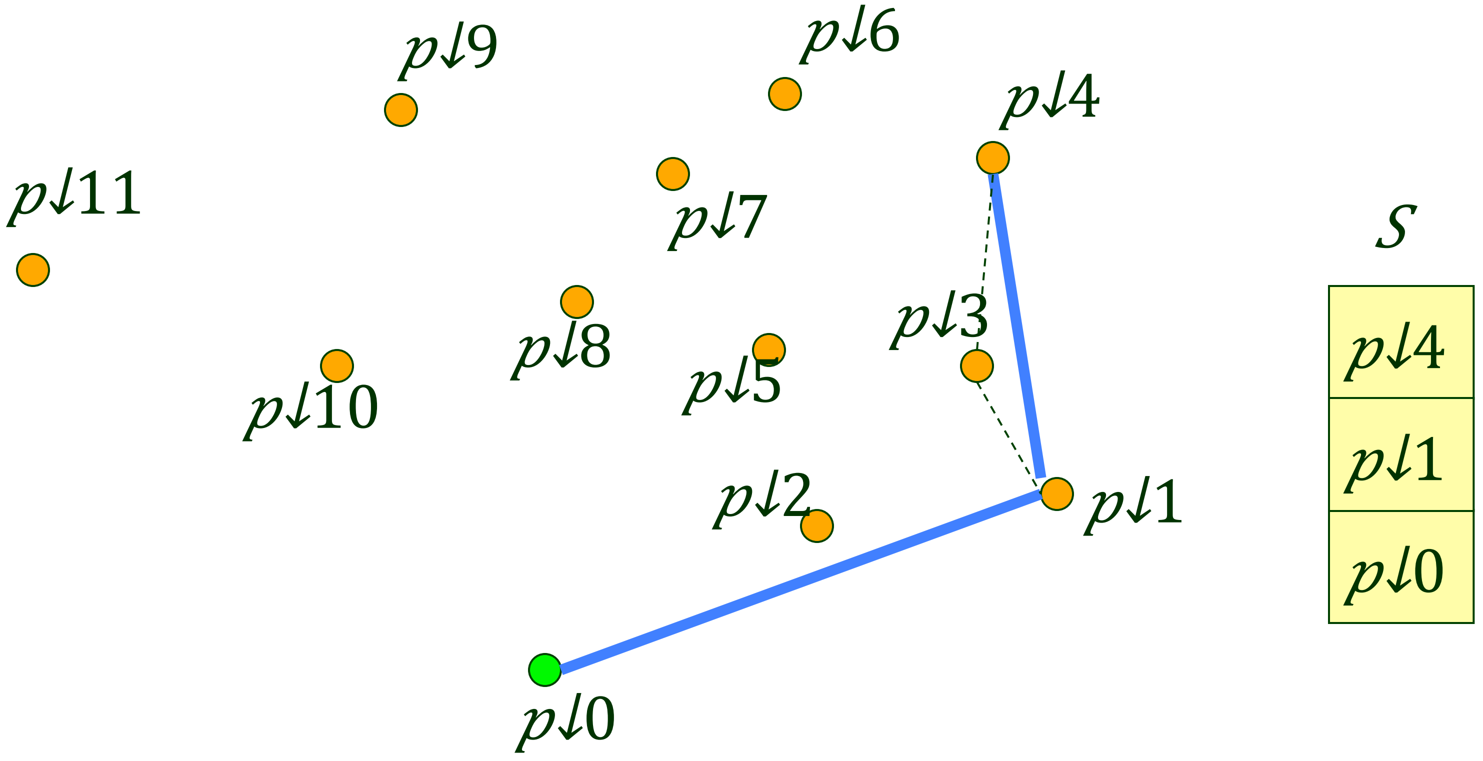


Stack Initialization

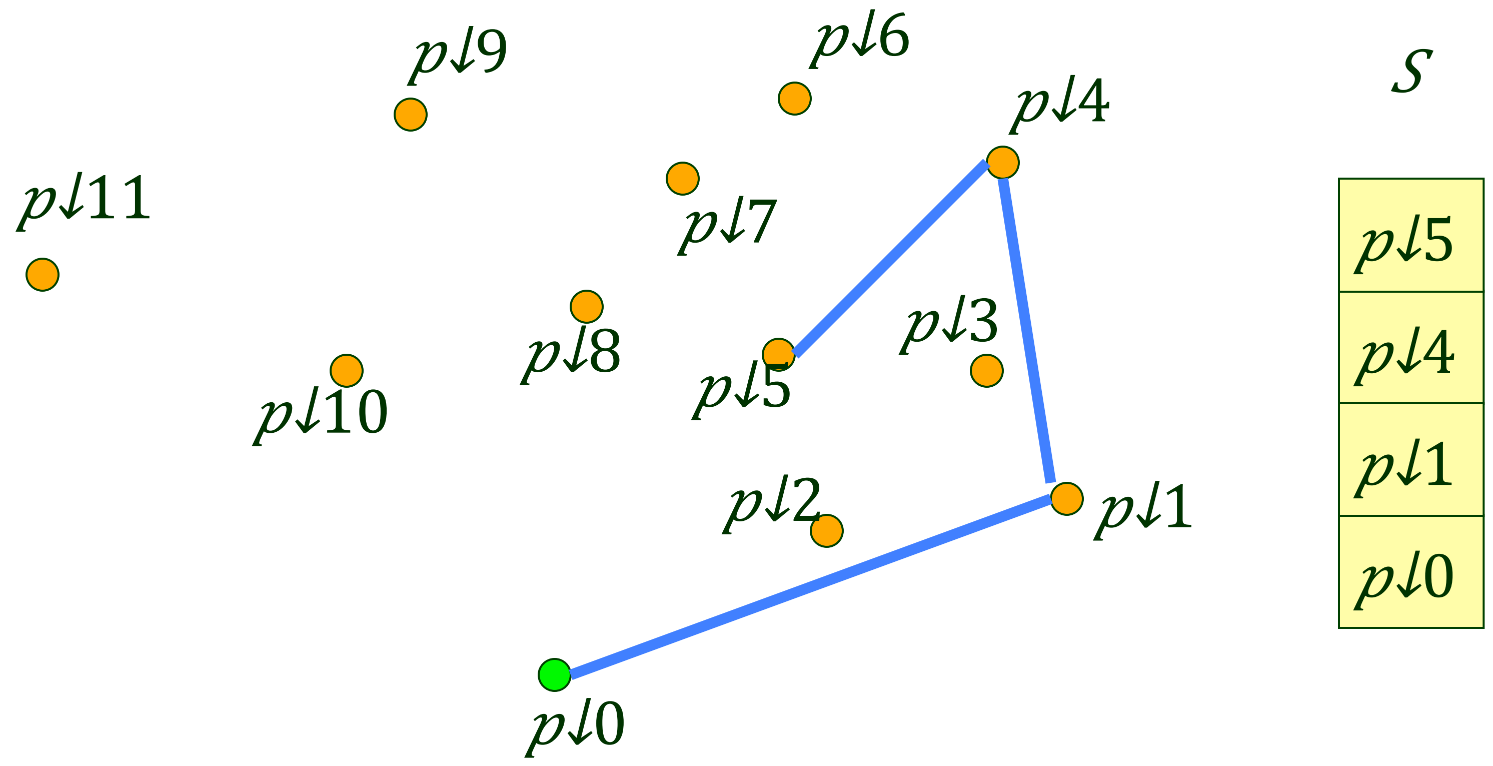
3) Scan points in the increasing order of polar angle, maintaining a stack.

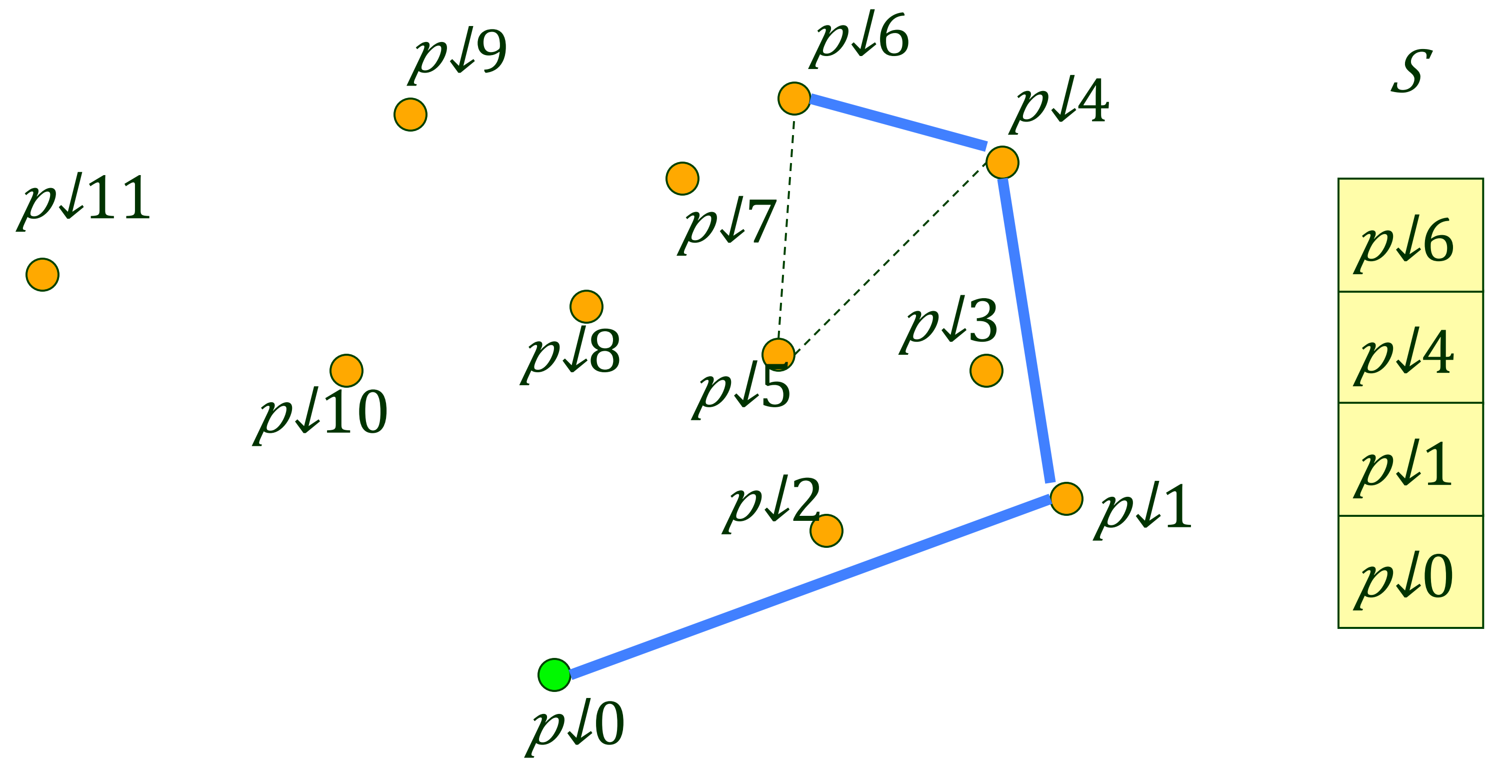



p_9
p_6
p_4
p_{11}
p_7
S
p_8
p_3
p_5
p_{10}
p_2
p_1
p_0
p_3
p_1
p_0

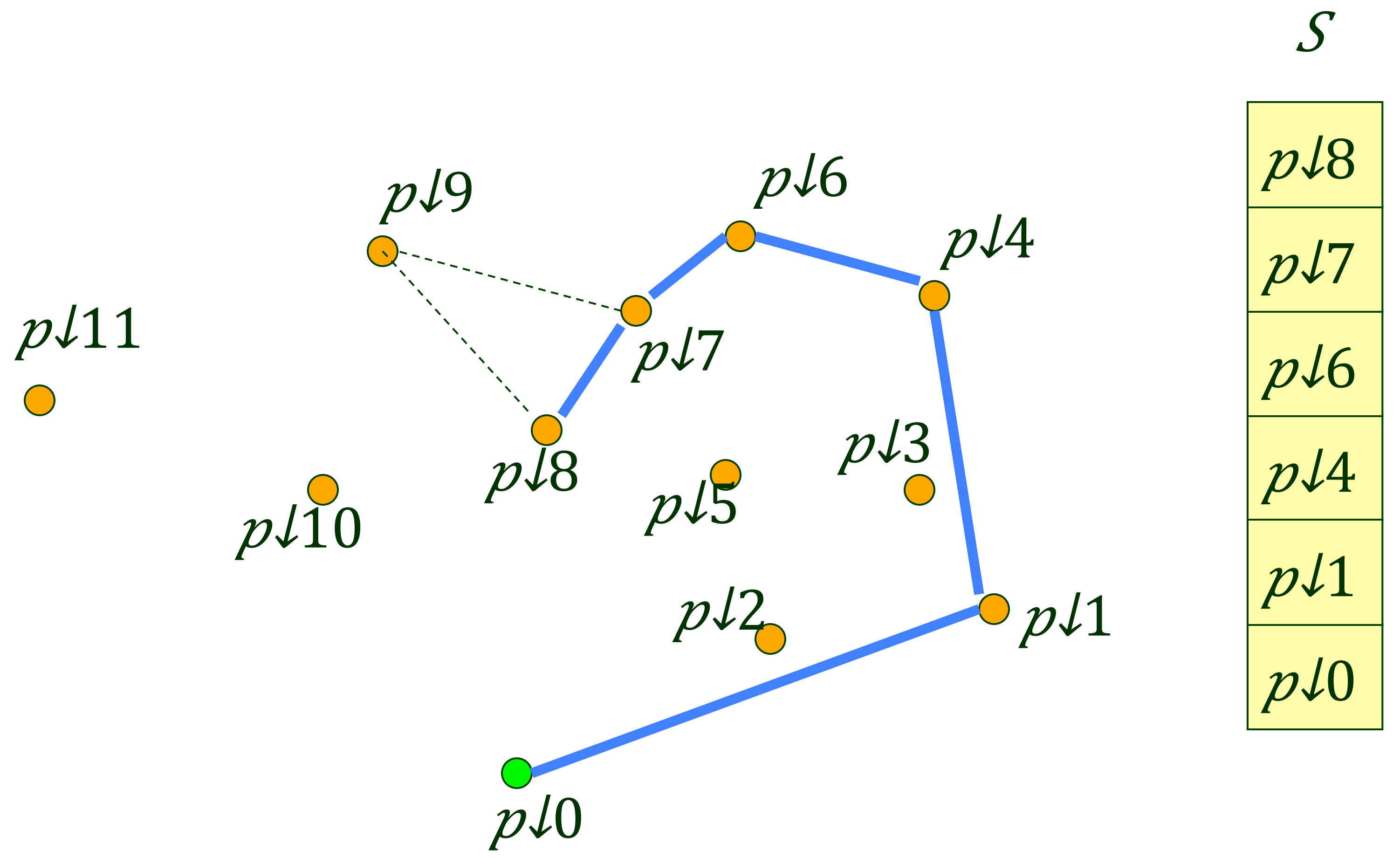


p_9
p_6
p_4
p_{11}
p_7
S
p_3
p_8
p_5
p_{10}
p_4
p_1
p_2
p_1
p_0
p_0

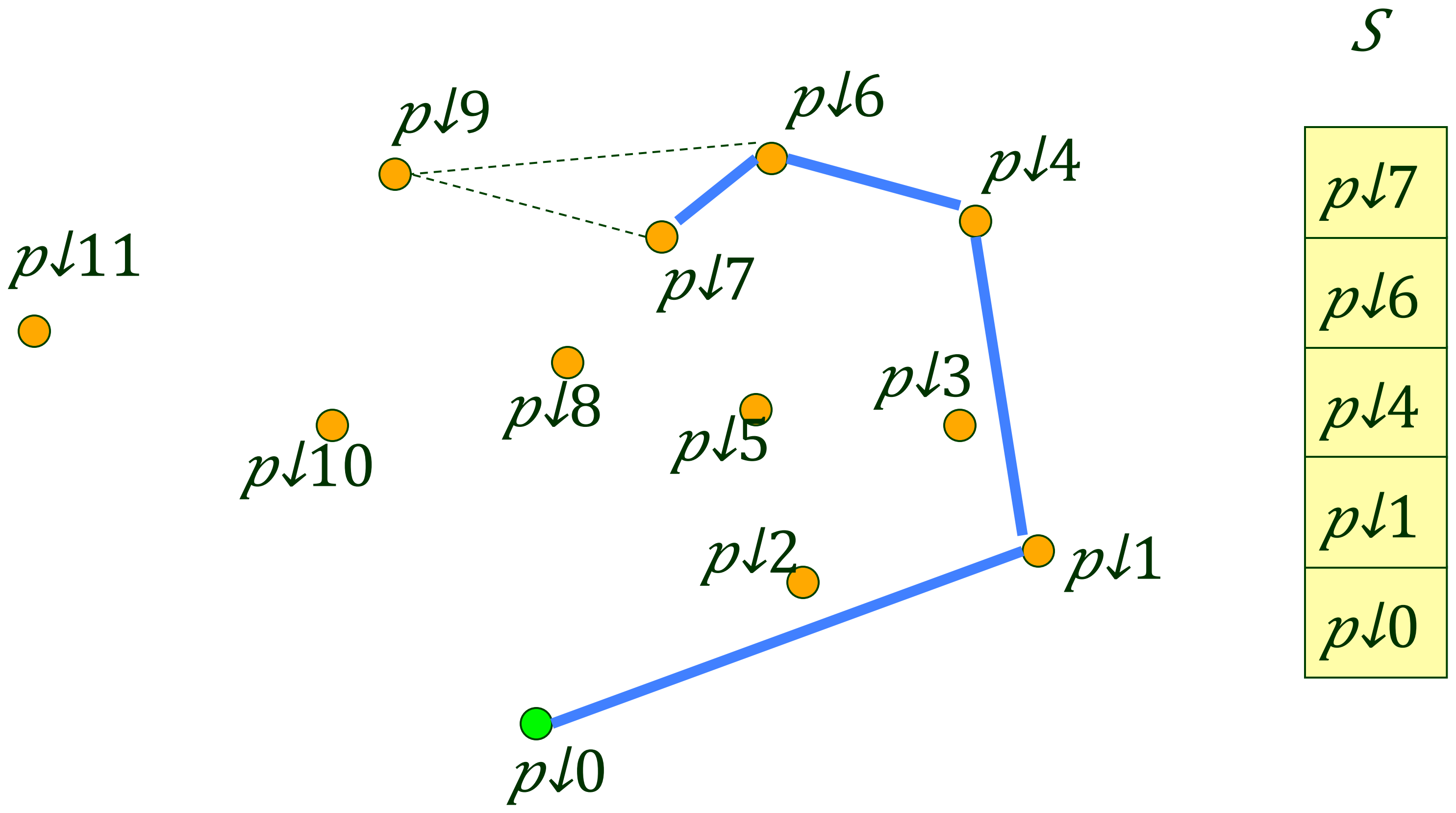

p↓9
p↓6
p↓4
S
p↓11
p↓7
p↓5
p↓8
p↓3
p↓4
p↓10
p↓5
p↓1
p↓2
p↓1
p↓0
p↓0

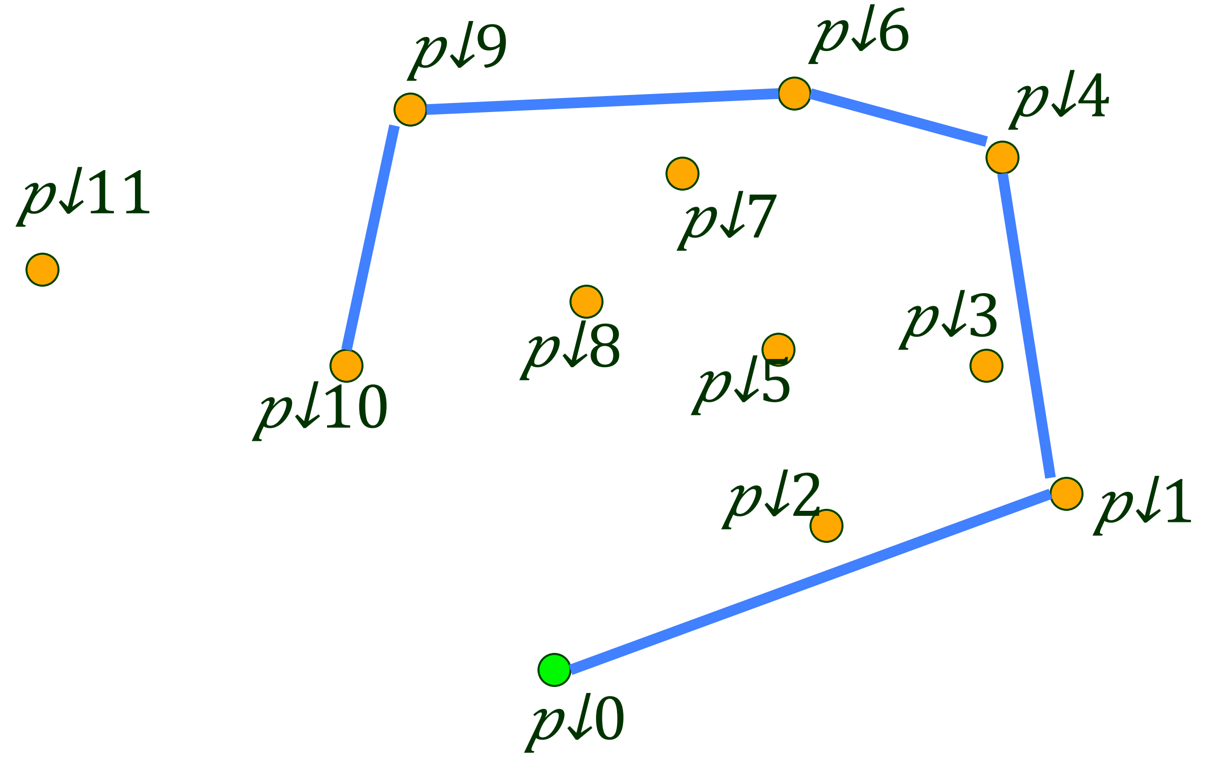


p_6
p_9
p_4
S
p_{11}
p_7
p_8
p_3
p_5
p_{10}
p_2
p_1
p_0
p_6
p_4
p_1
p_0



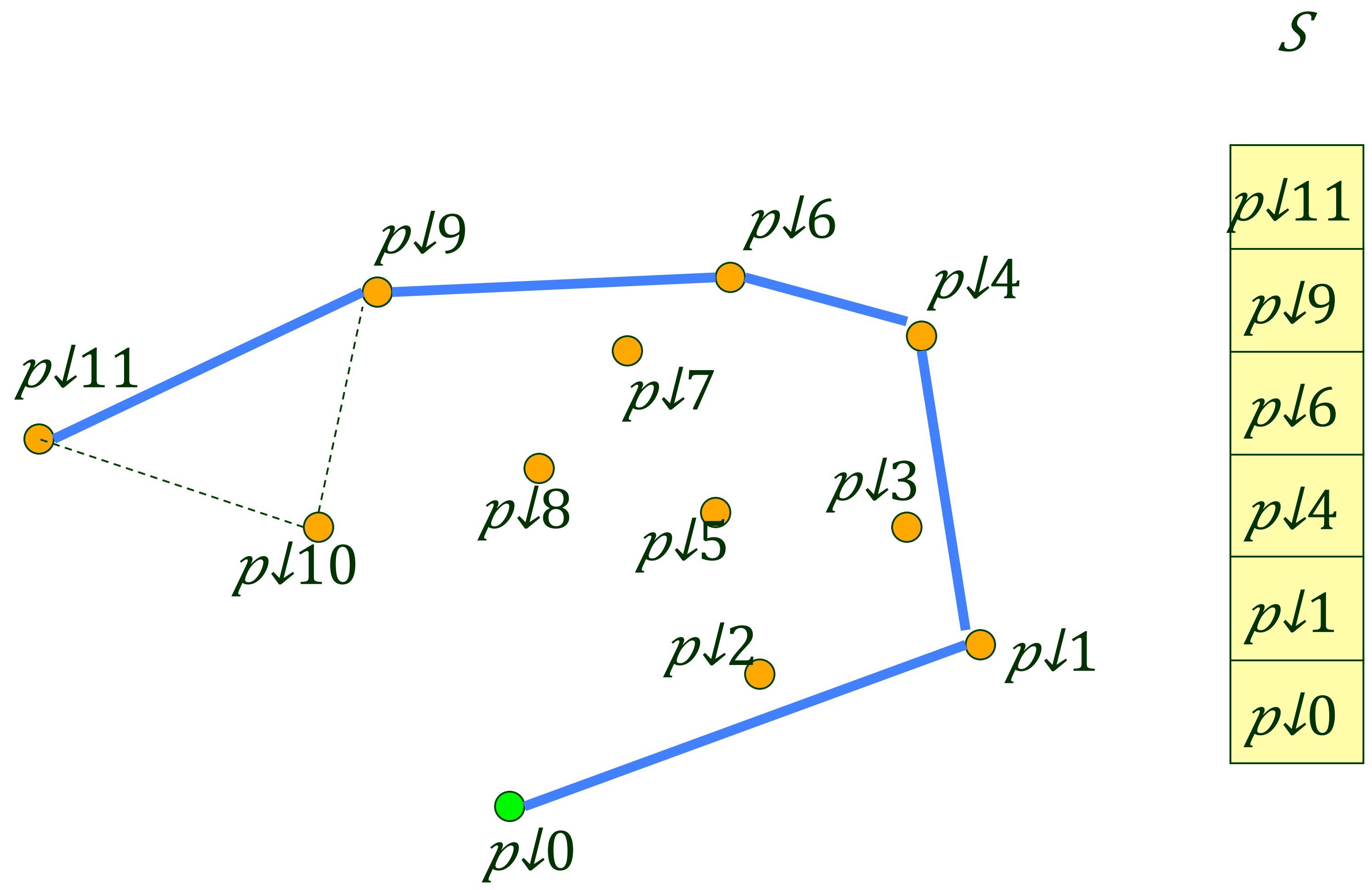
S
p_8
p_7
p_6
p_4
p_1
p_0
p_9
p_6
p_4
p_{11}
p_7
p_3
p_8
p_5
p_{10}
p_2
p_1
p_0


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

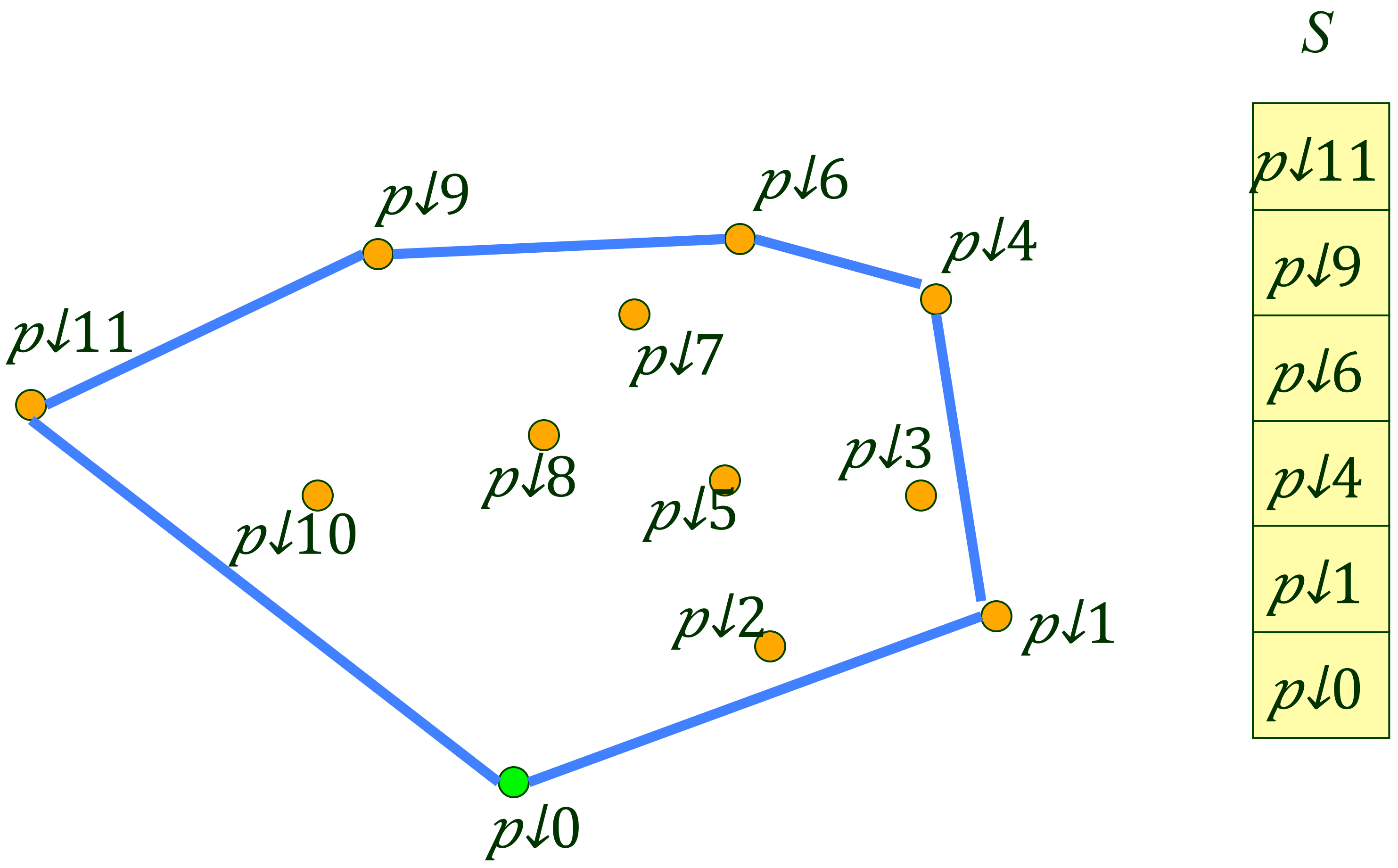


S

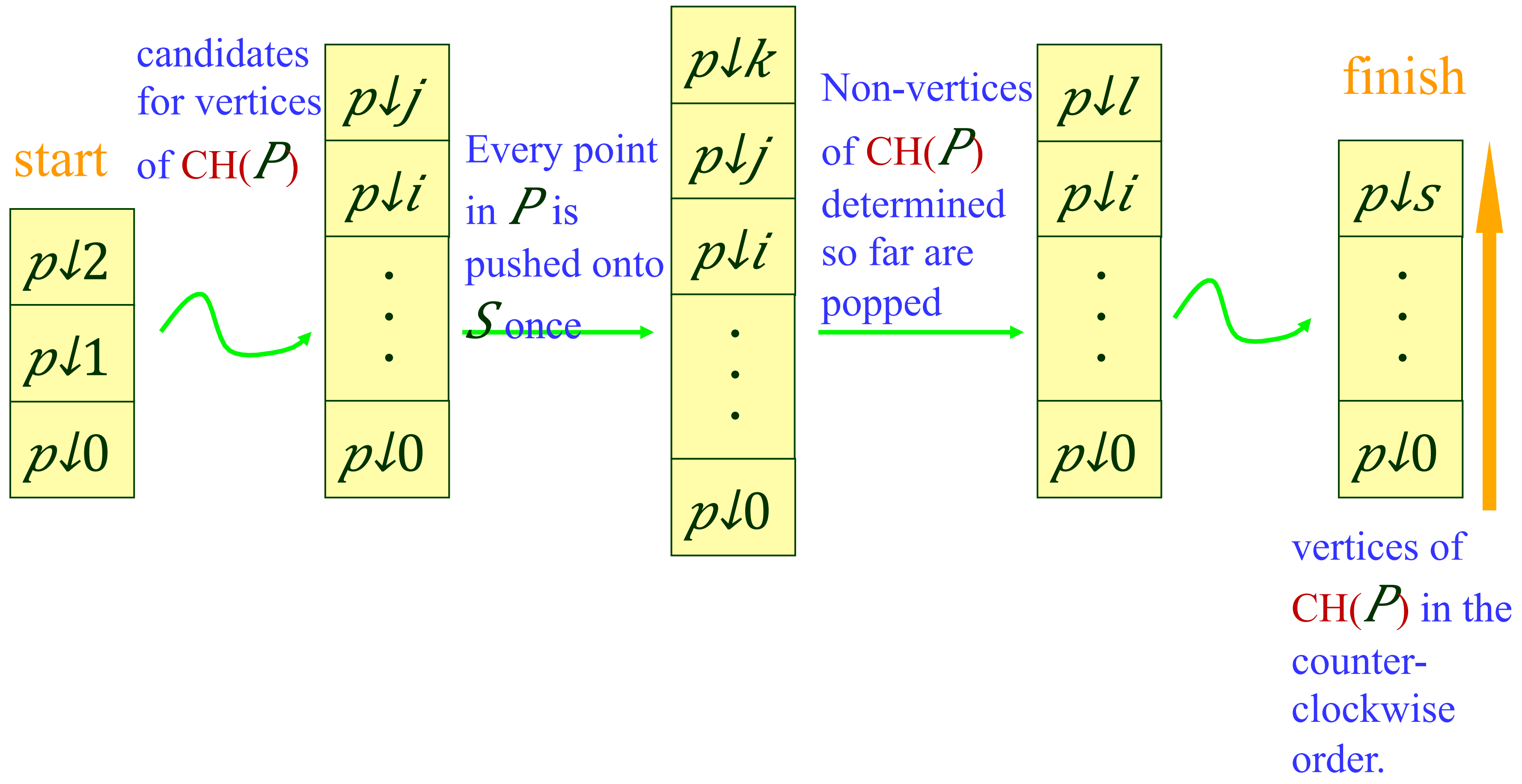
p_{10}
p_9
p_6
p_4
p_1
p_0


S
p↓11
p↓9
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

Finish



Graham's Scan


start
p_2
p_1
p_0
candidates for vertices of CH(P)
p_j
p_i
⋮
p_0
Every point in P is pushed onto S once
p_k
p_j
p_i
⋮
p_0
Non-vertices of CH(P) determined so far are popped
p_l
p_i
⋮
p_0
finish
p_s
⋮
p_0
vertices of CH(P) in the counter-clockwise order.

The Graham Scan Algorithm

```
Graham-Scan($P$)
    let $p_0$ be the point in $P$ with *minimum* $y$-coordinate
    let $\langle p_1, p_2, \ldots, p_{n-1} \rangle$ be the remaining points in $P$
        sorted in counterclockwise order by polar angle around $p_0$.
    Top[$S$] $\leftarrow$ 0
    Push($p_0$, $S$)
    Push($p_1$, $S$)
    Push($p_2$, $S$)
    for $i \leftarrow 3$ 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 $S$
```

Correctness of Graham's Scan

Invariant The points on the stack S always form the vertices of the convex hull of the points scanned so far in counterclockwise order.

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).

Running time

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

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