Computing CONVEX HULLS

Presentation Outline

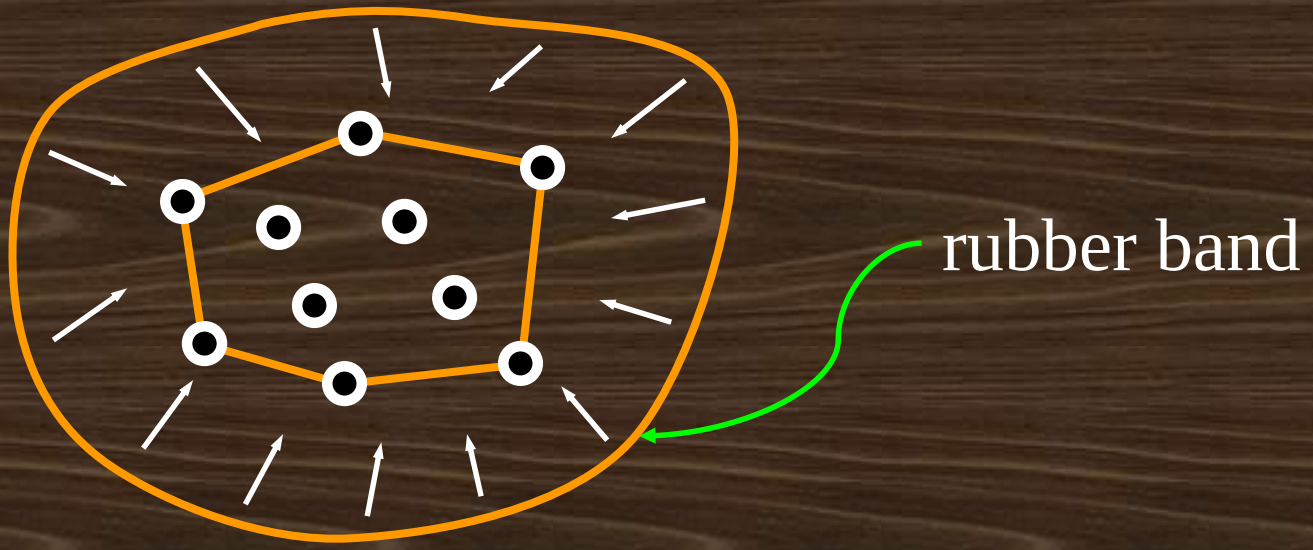
- **Convex Hulls**
  - **Definitions and Properties**
  - **Approaches:**
    - ***Brute Force***
    - ***Gift Wrapping***
    - ***QuickHull***
    - ***Graham Scan***
    - ***Incremental***
    - ***Divide and Conquer***
    - ***By Delaunay Triangulation & Voronoi Diagram***

Some Applications

- Collision Avoidance
  - robot motion planning
- Finding Smallest Box
  - collision detection
- Shape Analysis

CONVEX HULLS

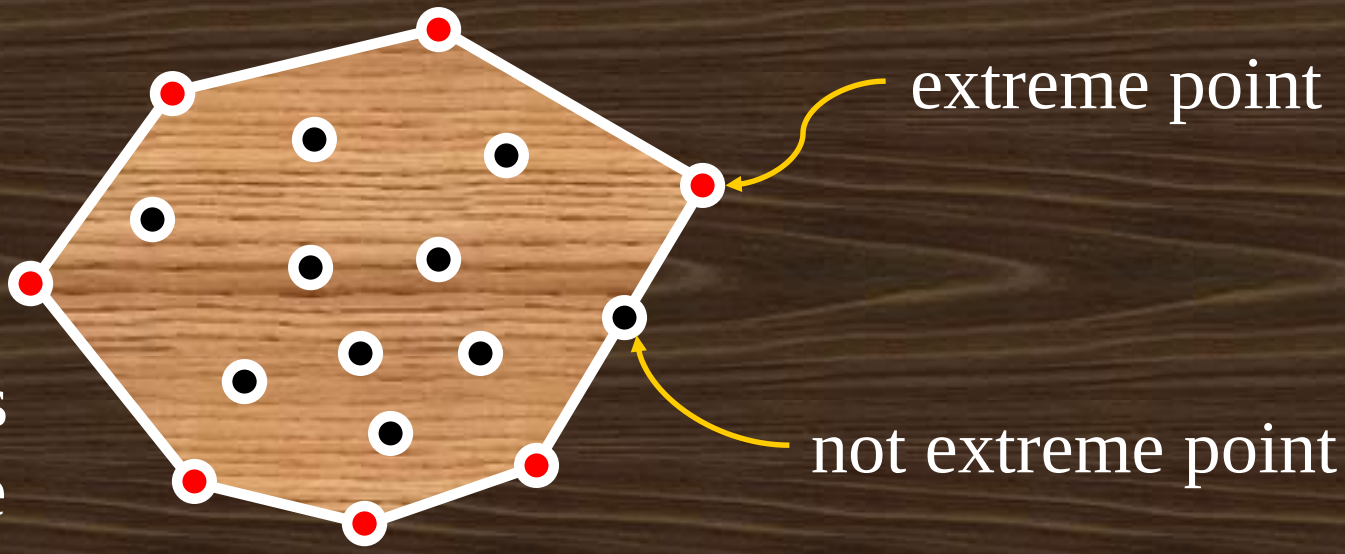
Definitions and Properties



- Intersection of all convex sets containing *P*
- Smallest convex set containing *P*
- Intersection of all half-planes containing *P*
- Union of all triangles formed by points of *P*

Definitions and Properties

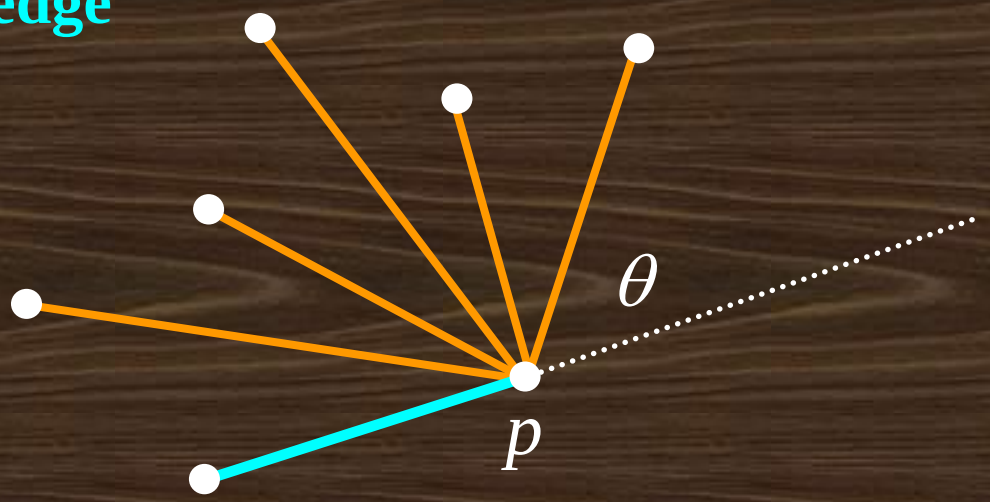
- Smallest *convex polygon* containing *P*
- *All vertices of hull are some points of P*



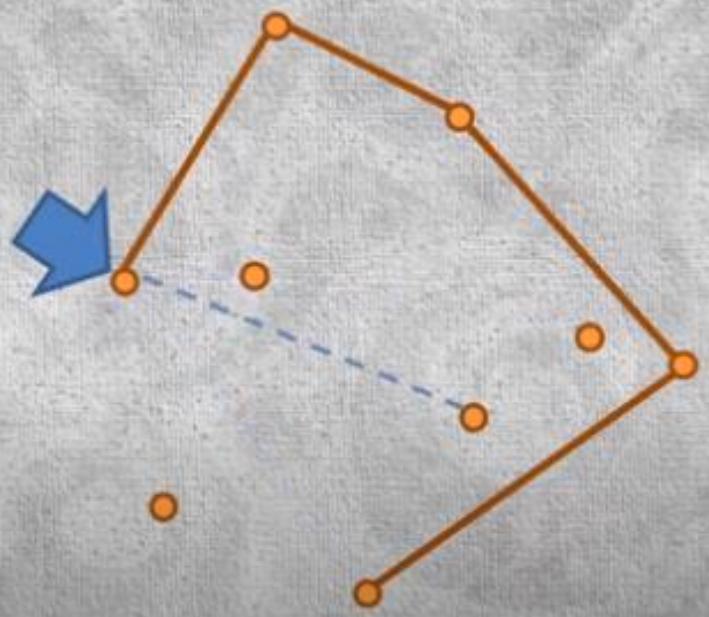
- **NOTE: convex hull is the closed solid region, not just the boundary**

Jarvis March/Gift Wrapping/Package Wrapping

$p \leftarrow$ the lowest point p_0
repeat
  for each $q \in P$ and $q \neq p$ do
    compute counterclockwise angle θ from previous hull edge
  let r be the point with smallest θ
  output (p, r) as a hull edge
  $p \leftarrow r$
until $p = p_0$



Jarvis march algorithm

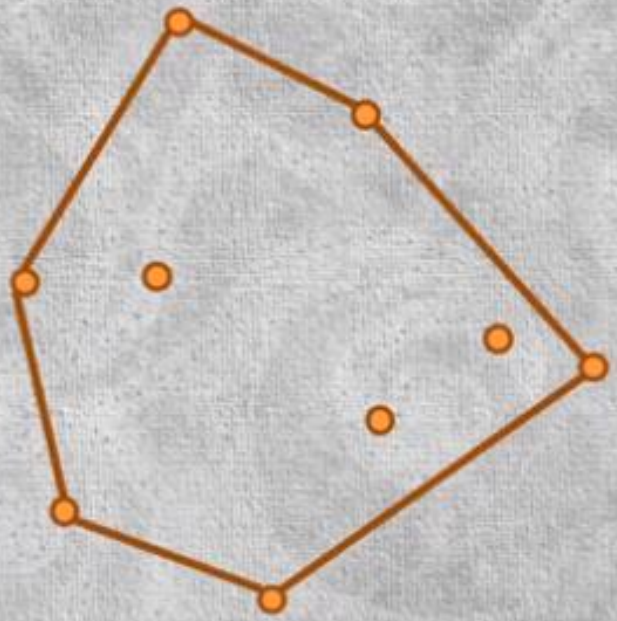


select the point with the lowest Y coordinate - the 1st vertex

select the point with smallest counterclockwise in reference to previous vertex

repeat until reaching the starting vertex

Jarvis march algorithm



select the point with the lowest Y coordinate - the 1st vertex

select the point with smallest counterclockwise in reference to previous vertex

repeat until reaching the starting vertex

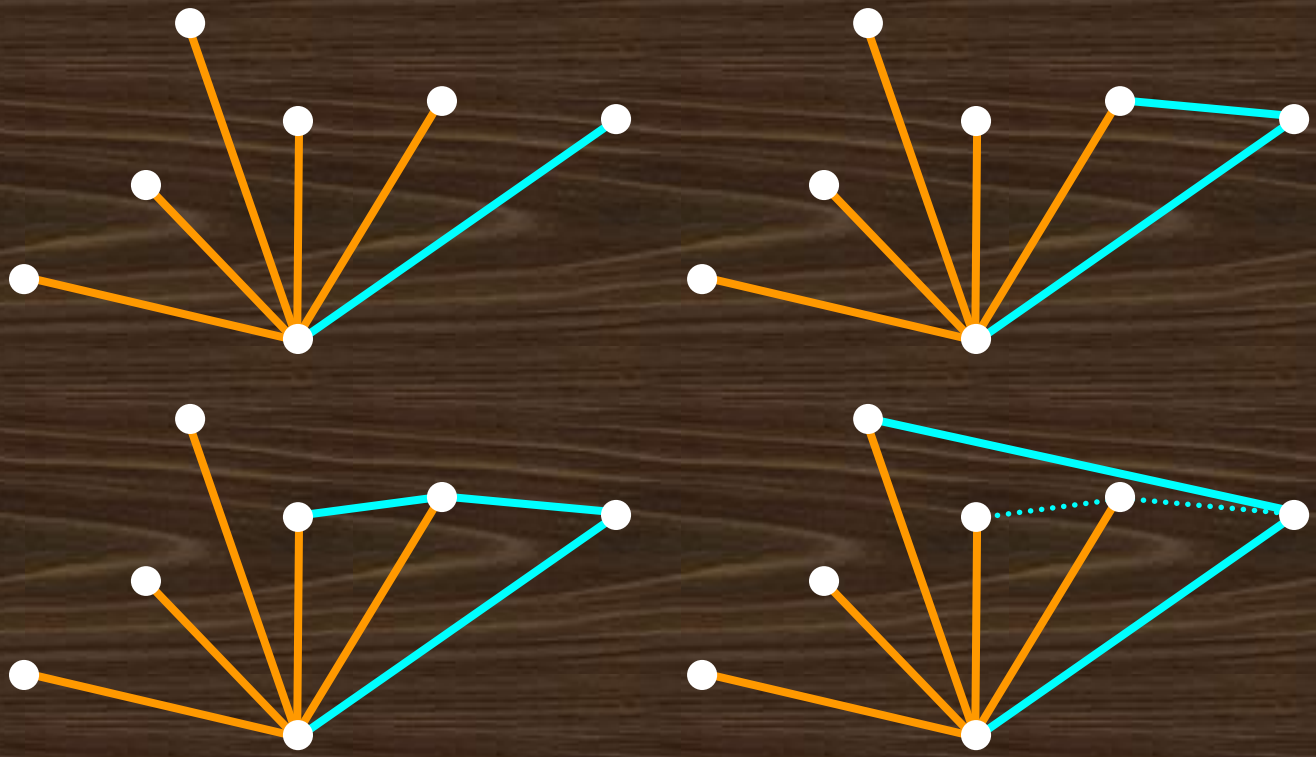
Gift Wrapping

- First suggested by Chand and Kapur (1970)
- Worst-case time: $O(n^2)$
- Output-sensitive time: $O(nk)$
  - where k is the # of vertices of hull
- Can be extended to higher dimension
  - was the primary algorithm for higher dimensions for quite some time

Graham Scan

- By Graham (1972)
- First algorithm to achieve optimal running time
- Uses angular sweep

Graham Scan



Graham Scan

Find lowest point p_0

Sort all other points angularly about p_0,

    break ties in favor of closeness to p_0;

    label them $p_1, p_2, \ldots, p_{n-1}$

Stack $S = (p_{n-1}, p_0) = (p_{t-1}, p_t)$; t indexes top

$i \leftarrow 1$

while $i < n$ do

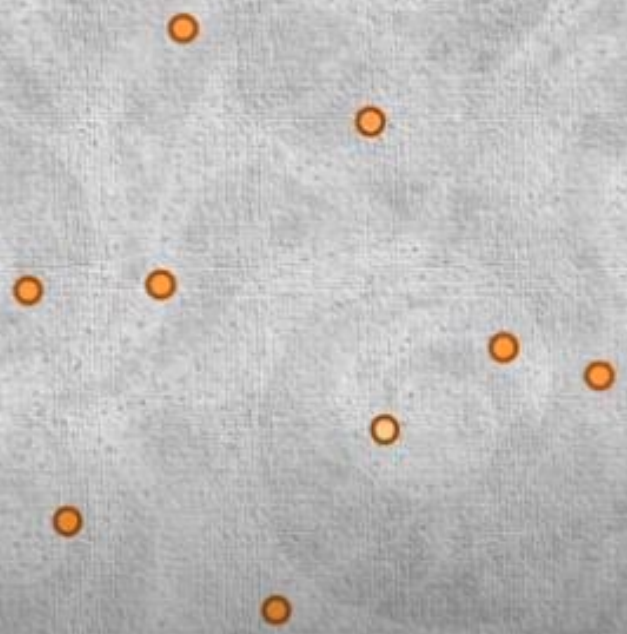
    if p_i is strictly left of (p_{t-1}, p_t) then

        Push(S, i); i++

    else Pop(S)

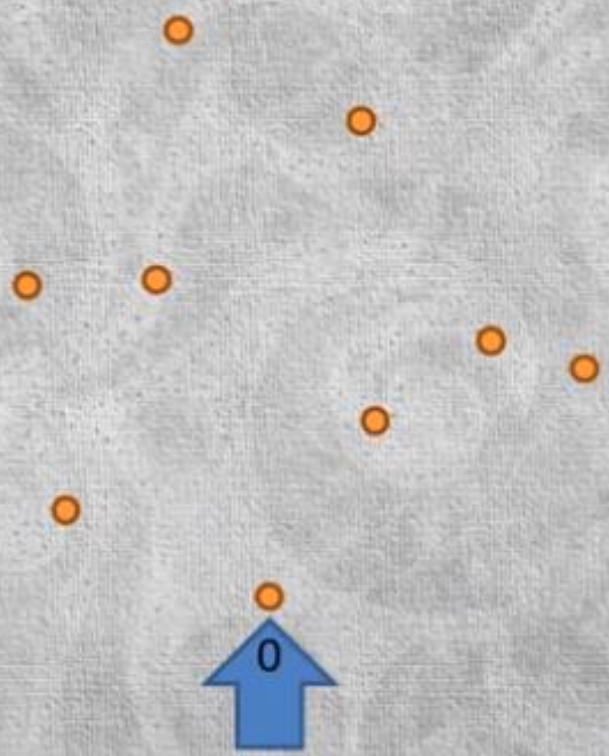
Graham scan algorithm

select the point with the lowest Y coordinate

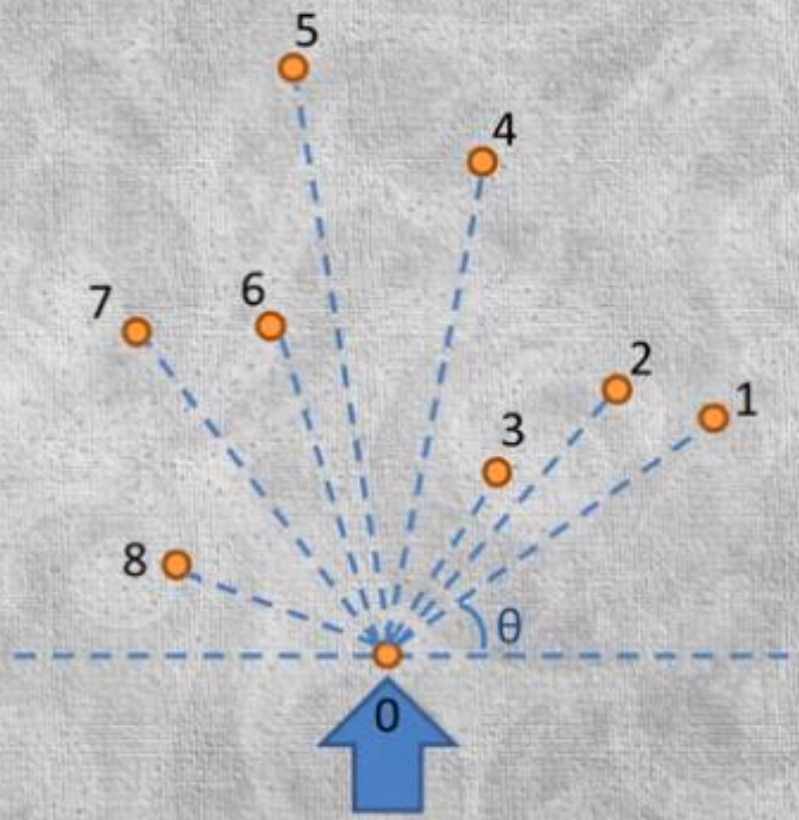


Graham scan algorithm

select the point with the lowest Y coordinate



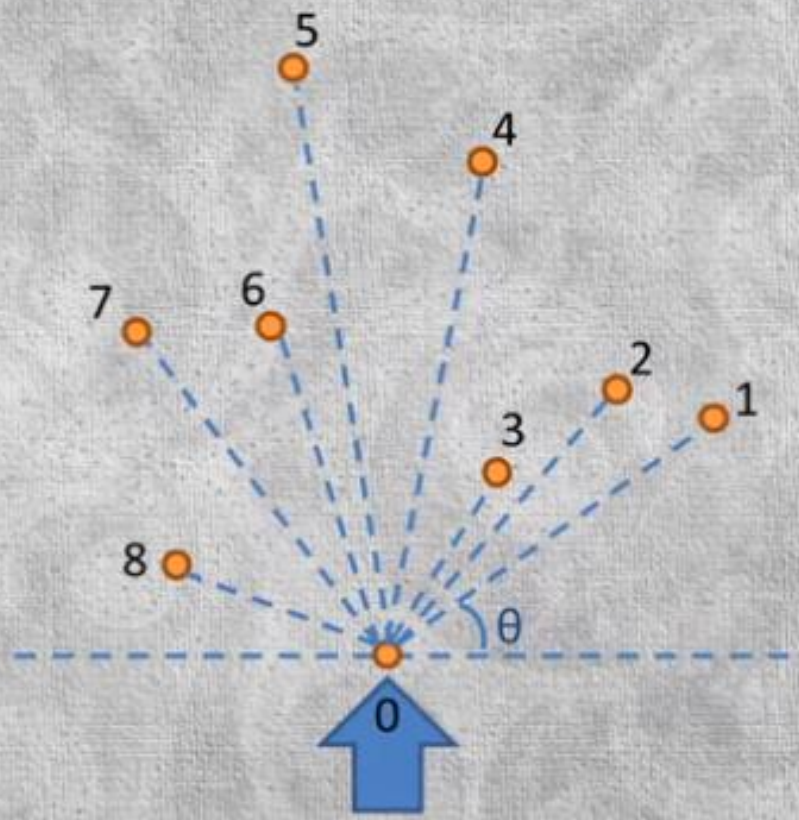
Graham scan algorithm



select the point with the lowest Y coordinate

sort the points by the angle relative to the bottom most point and the horizontal

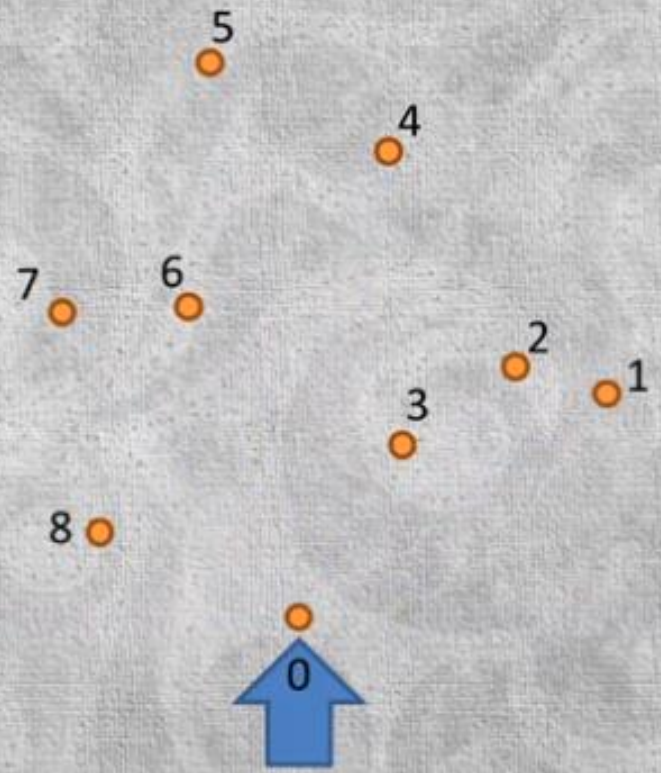
Graham scan algorithm



select the point with the lowest Y coordinate

sort the points by the angle relative to the bottom most point and the horizontal

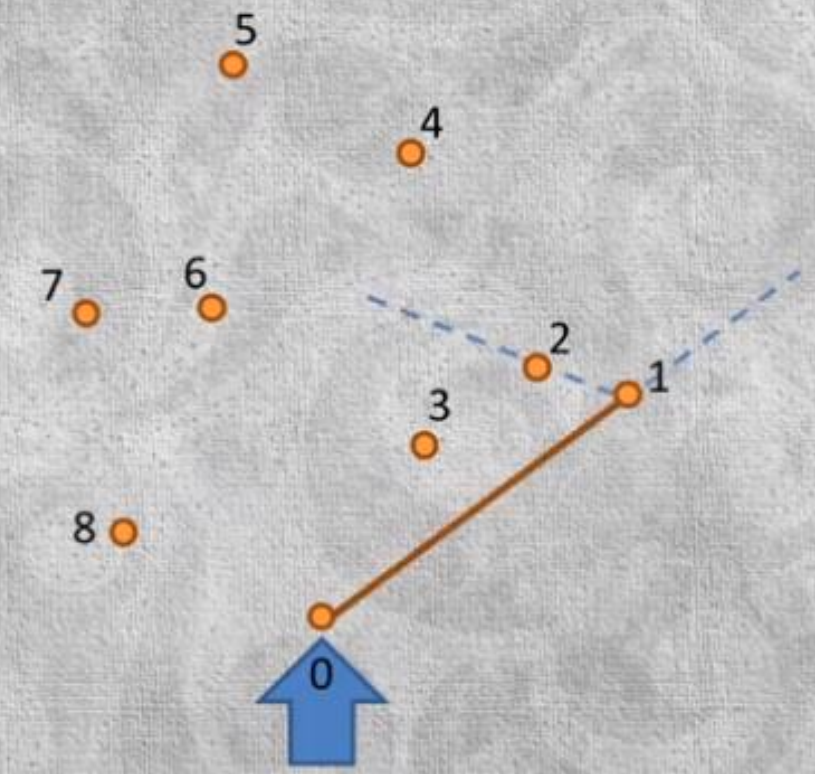
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

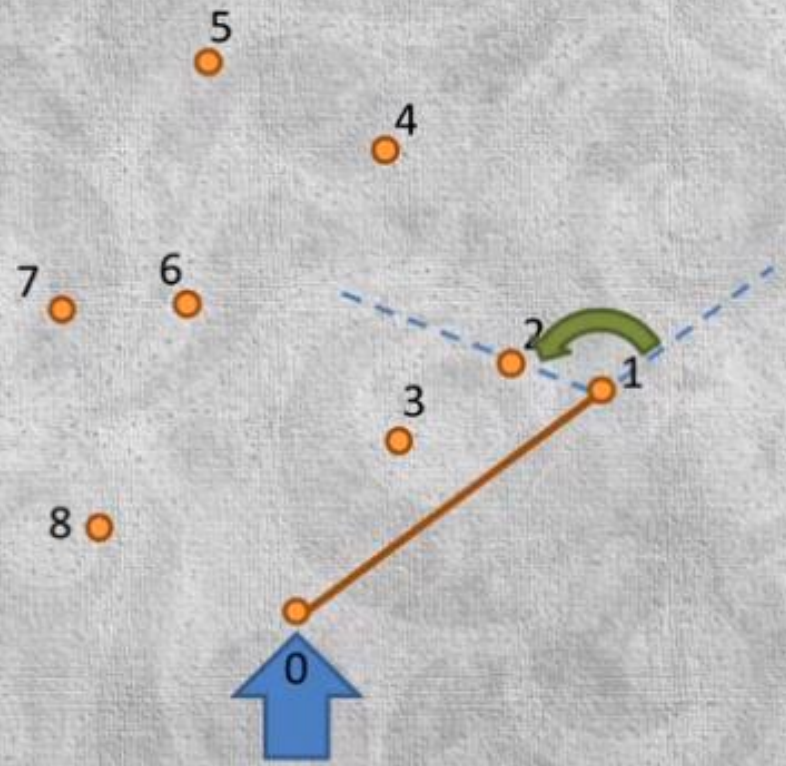
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

Graham scan algorithm

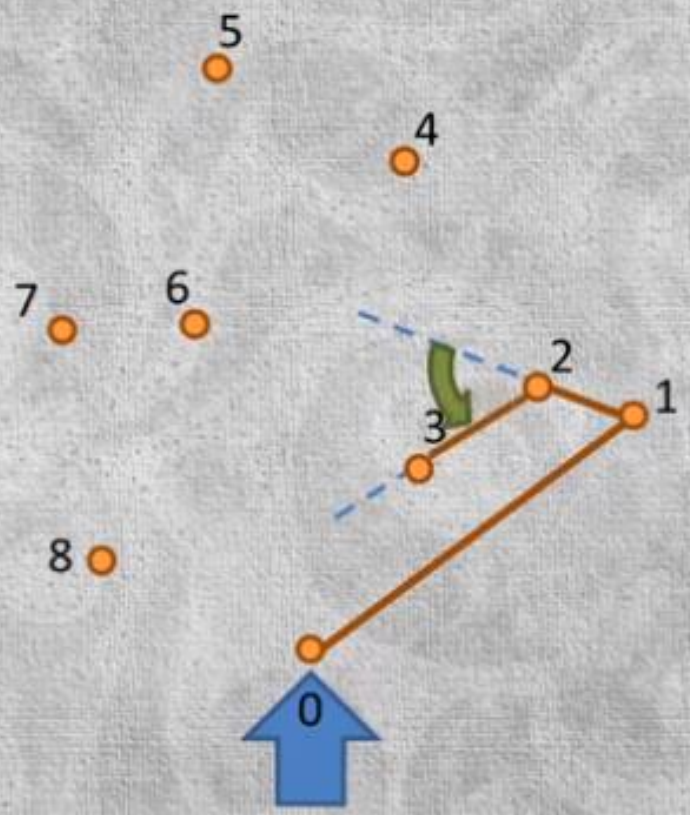


iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

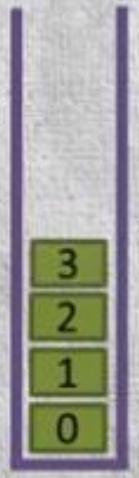
Stable Sor

Graham scan algorithm

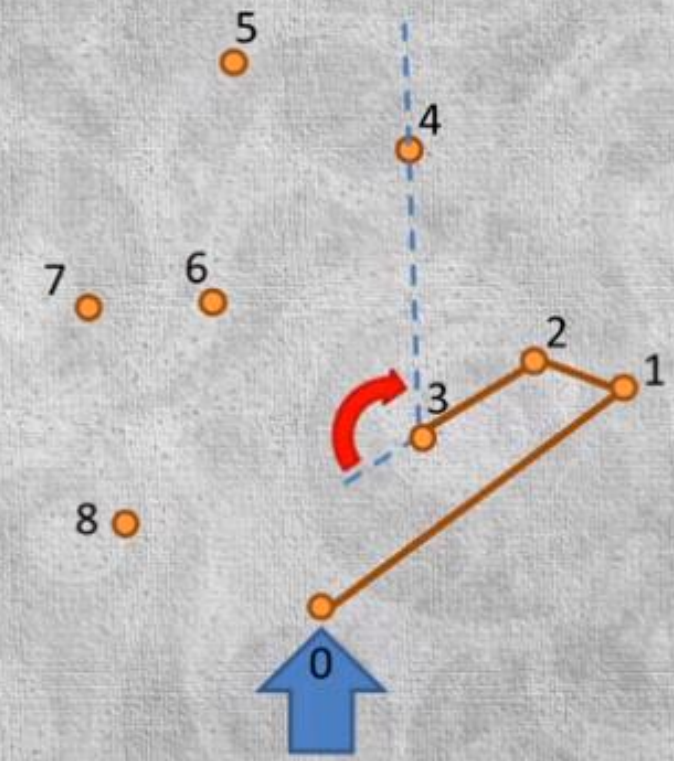


iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn



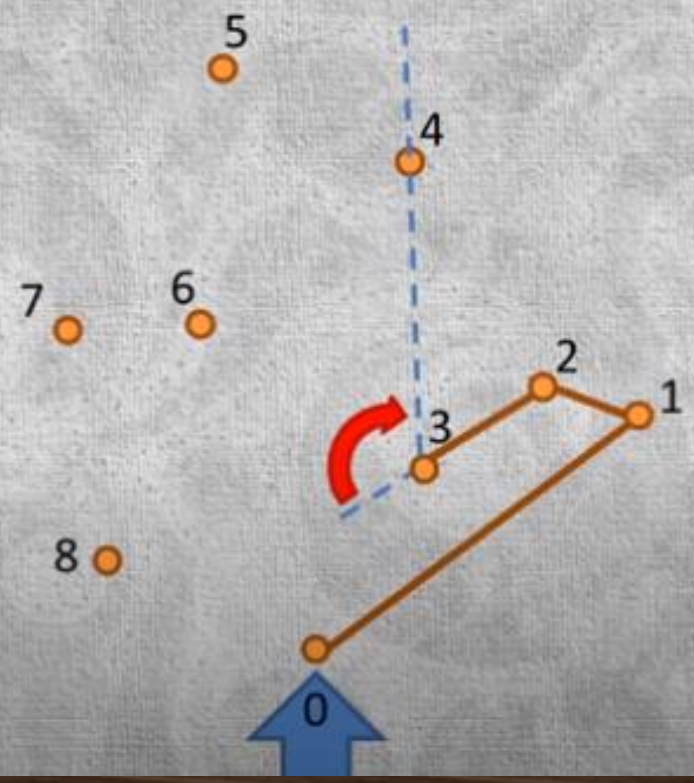
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

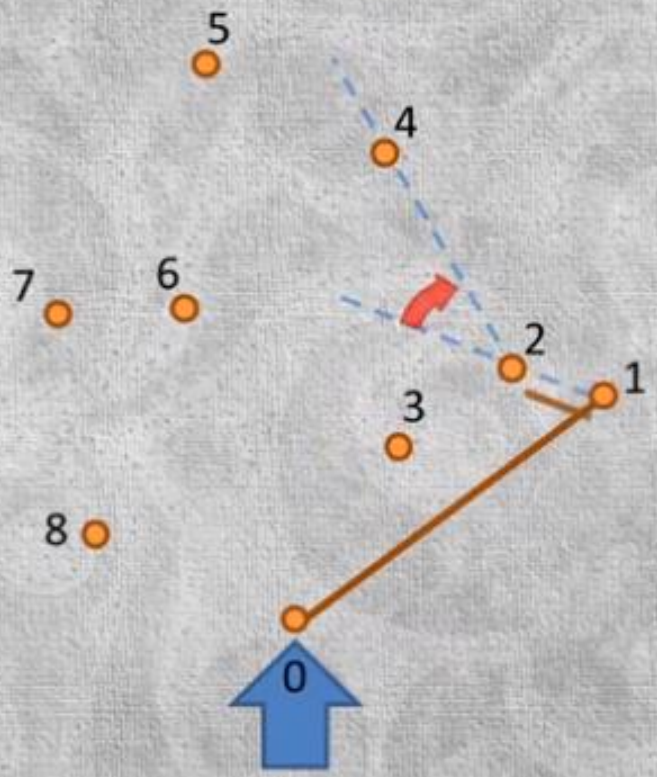
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

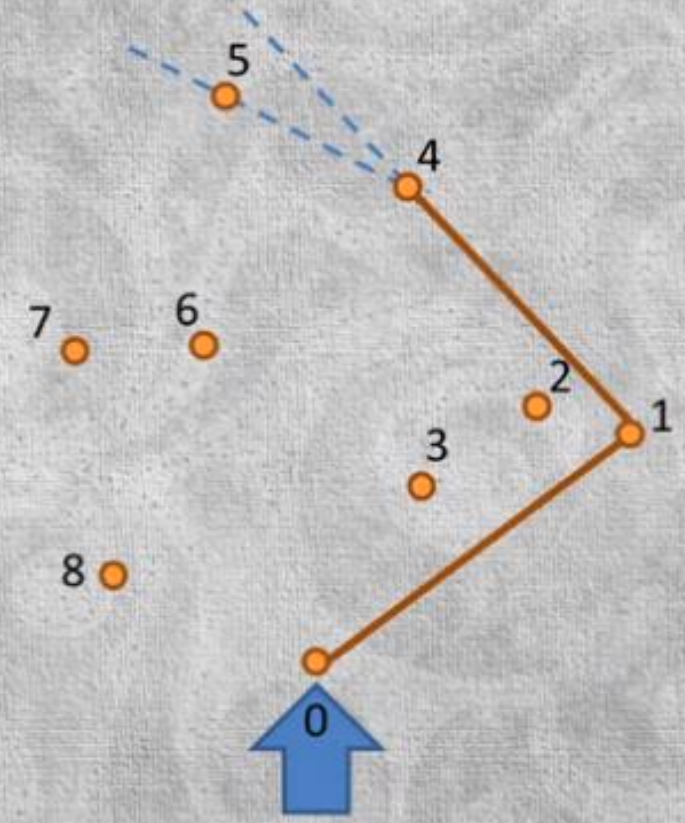
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

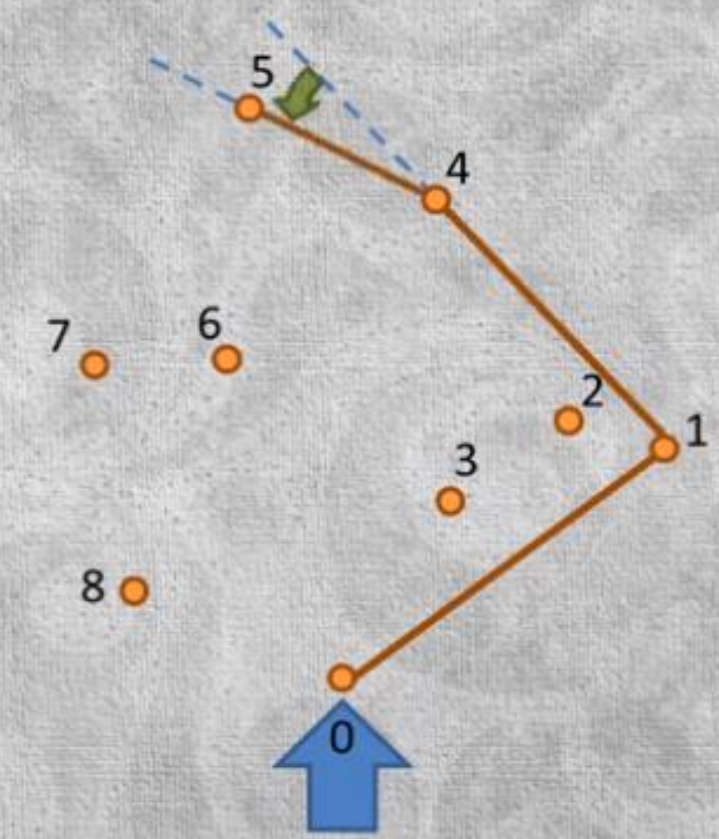
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

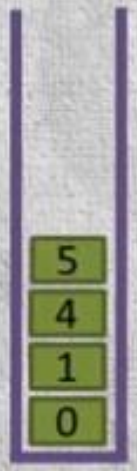
pop previous point off of the stack if making a **clockwise** turn

Graham scan algorithm

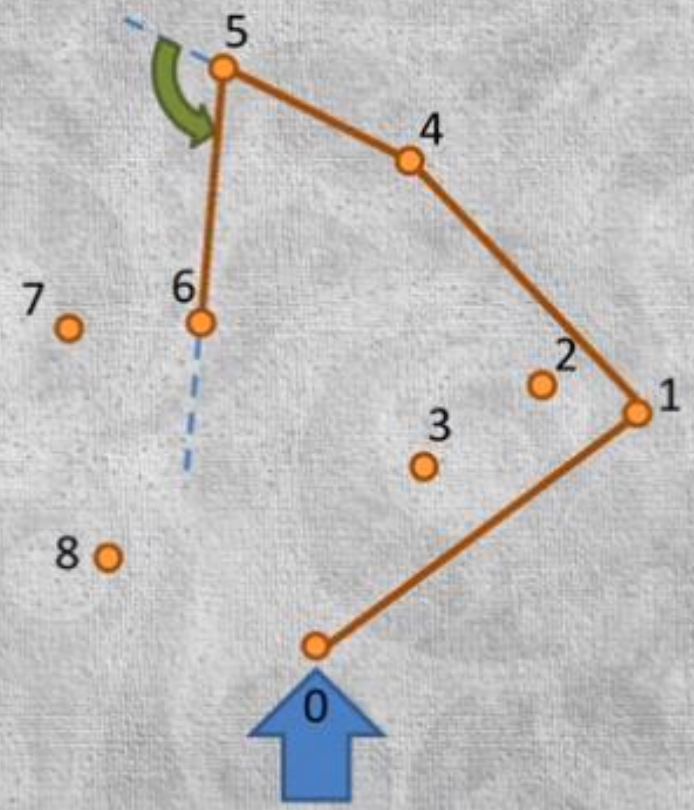


iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn



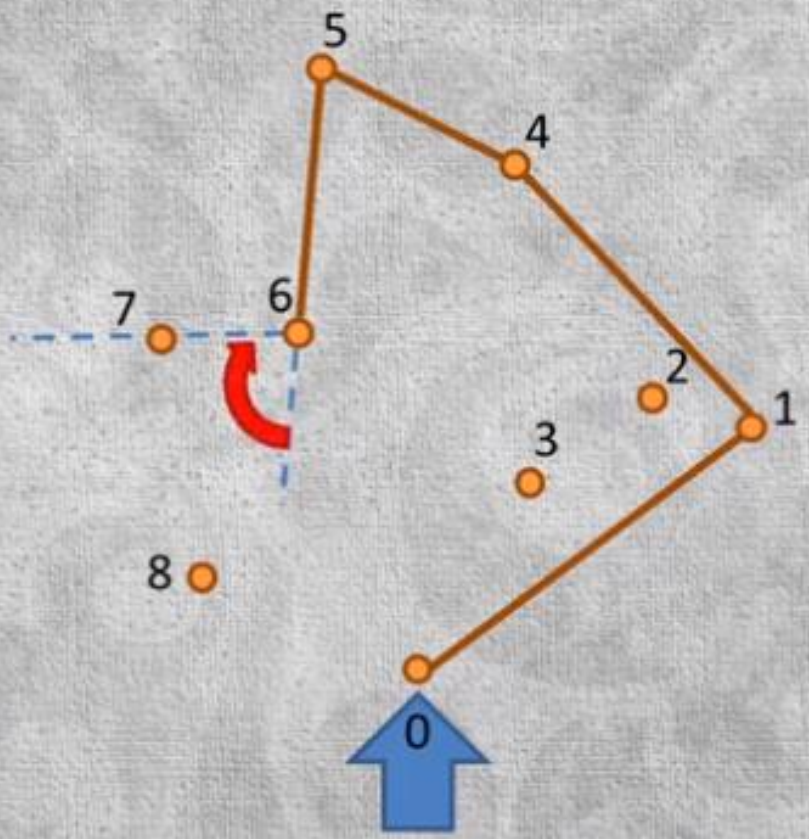
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

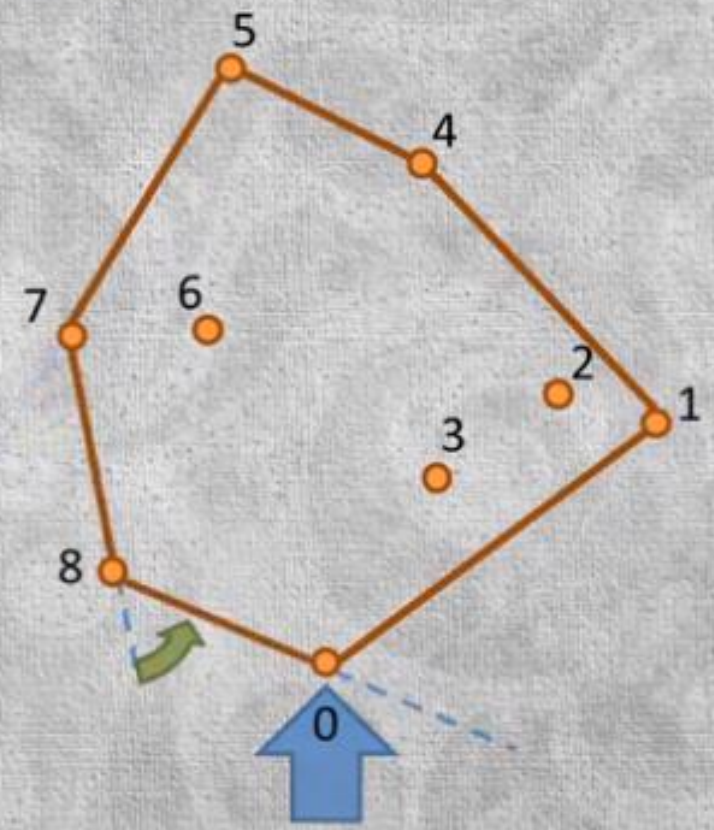
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

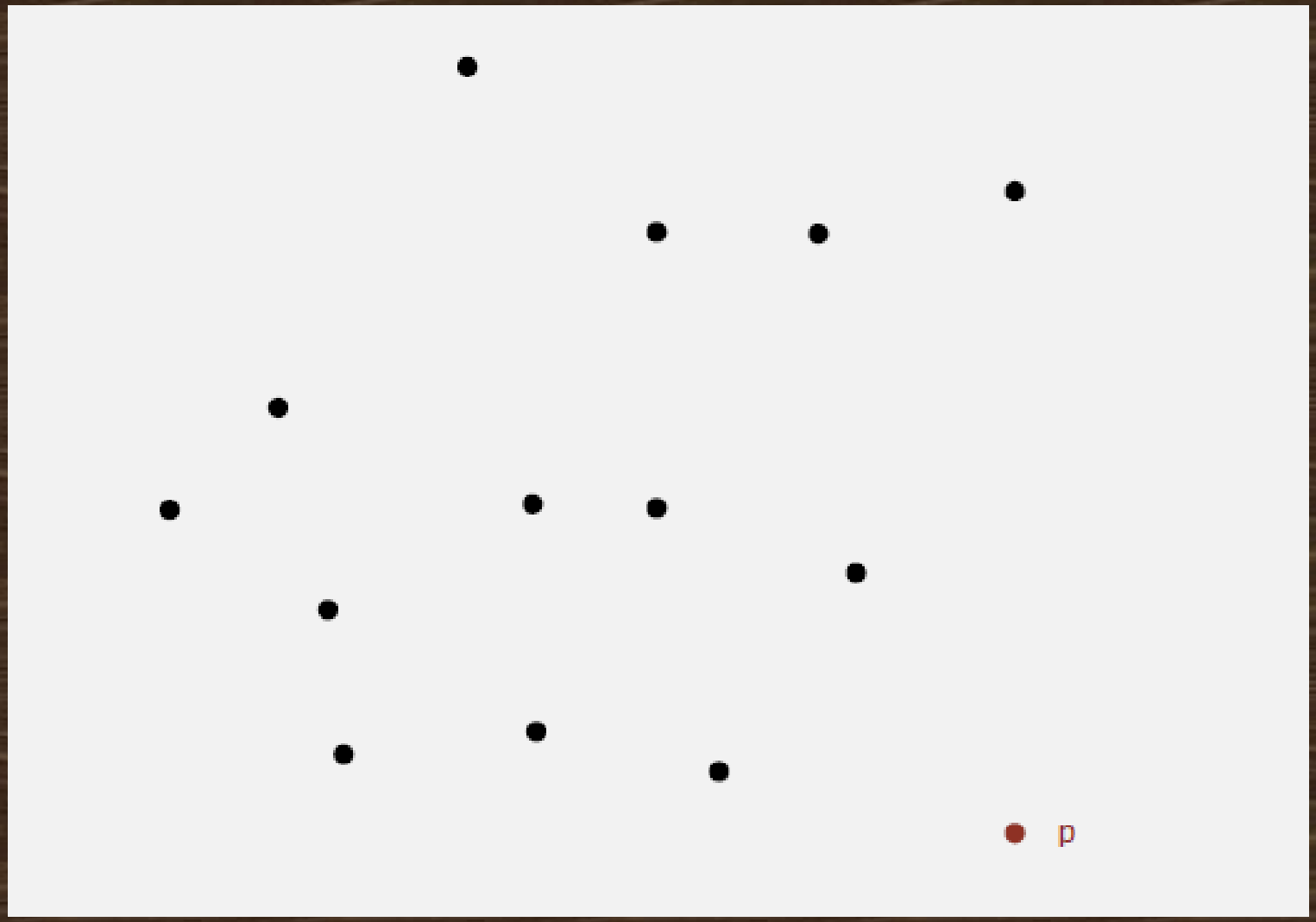
Graham scan algorithm



iterate in sorted order, placing each point on a stack, but only if it makes a **counterclockwise** turn relative to the previous 2 points on the stack

pop previous point off of the stack if making a **clockwise** turn

Practice Problem



Graham Scan

- Running time: $O(n \lg n)$
  - the whole sweep takes $O(n)$ only because each point can be pushed or popped at most once
  - $O(n \lg n)$ due to sorting of angles

- No obvious extension to higher dimensions