Graduate Algorithms

CS673-2016F-19

Computational Geometry

David Galles

Department of Computer Science
University of San Francisco

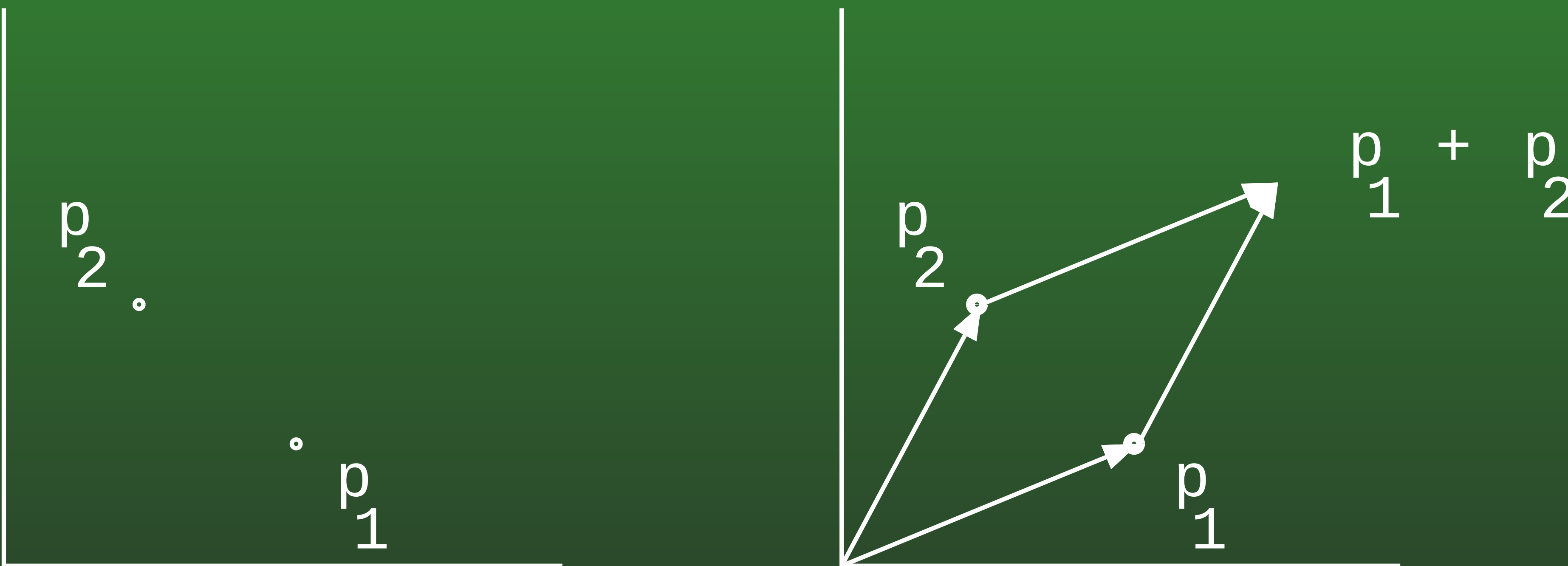
19-0: Cross Products

- Given any two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
  - Cross Product: $p_1 \times p_2 = x_1 y_2 - x_2 y_1$

$$\begin{aligned} p_1 \times p_2 &= x_1 y_2 - x_2 y_1 \\ &= -1 * (x_2 y_1 - x_1 y_2) \\ &= -p_2 \times p_1 \end{aligned}$$

19-1: Cross Products

- Cross Product $p_1 \times p_2$ as Signed Area

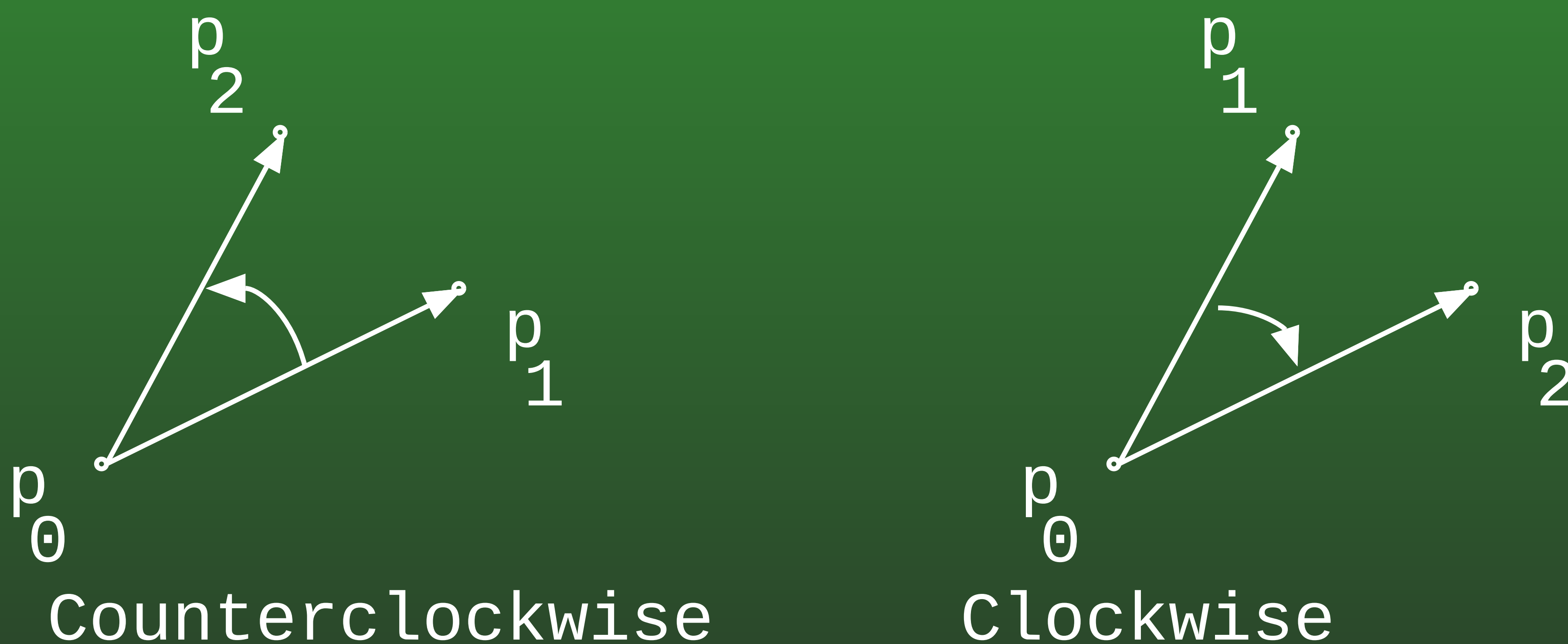


- Area is positive if p_1 is “below” p_2
- Area is negative if p_1 is “above” p_2

19-2: Cross Products

- Given two vectors that share an origin:
  - $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_0p_2}$
- Is $\overrightarrow{p_0p_2}$ clockwise or counterclockwise relative to $\overrightarrow{p_0p_2}$?

19-3: Cross Products

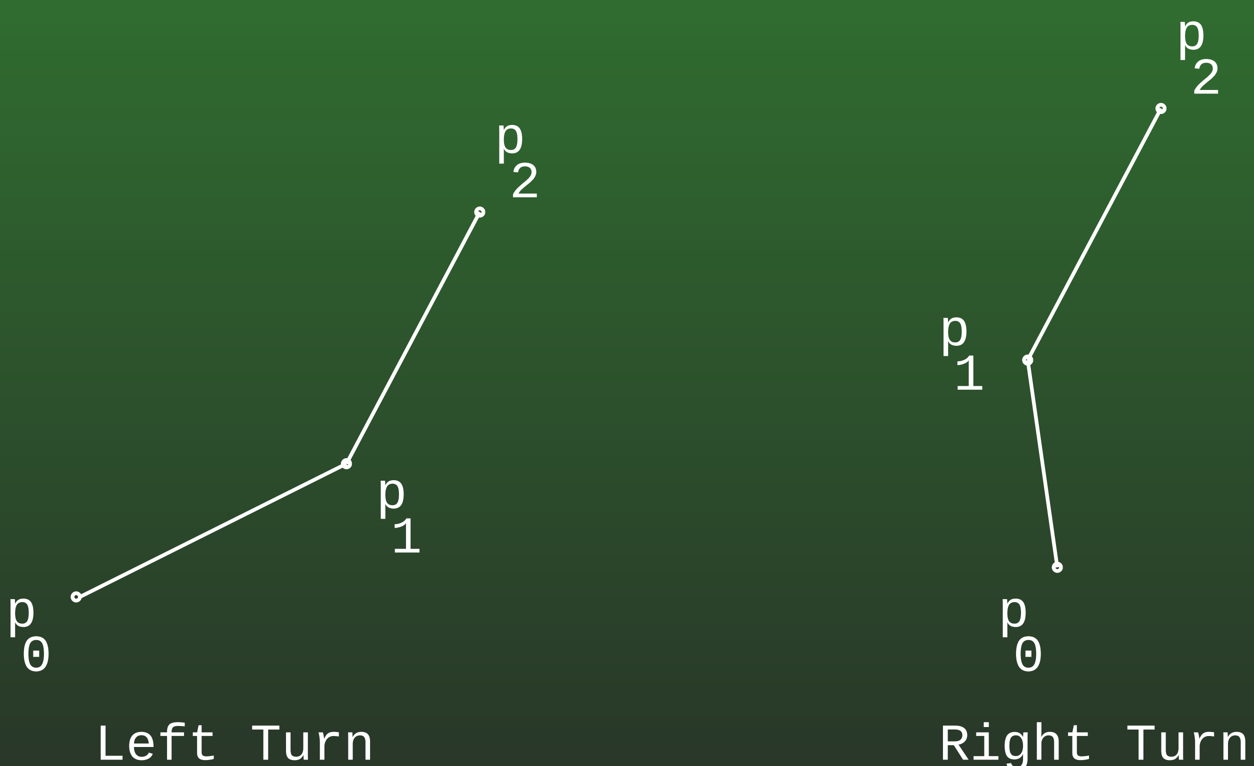


19-4: Cross Products

- Given two vectors that share an origin:
  - $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_0p_2}$
- Is $\overrightarrow{p_0p_2}$ clockwise or counterclockwise relative to $\overrightarrow{p_0p_2}$?
  - $(p_1 - p_0) \times (p_2 - p_0)$ is positive, $\overrightarrow{p_0p_2}$ is counterclockwise from $\overrightarrow{p_0p_1}$

19-5: Cross Products

- Given two line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$, which direction does angle $\angle p_0p_1p_2$ turn?



19-6: Cross Products

- Given two line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$, which direction does angle $\angle p_0p_1p_2$ turn?
  - $(p_2 - p_0) \times (p_1 - p_0)$ is positive, left turn
  - $(p_2 - p_0) \times (p_1 - p_0)$ is negative, right turn
  - $(p_2 - p_0) \times (p_1 - p_0)$ is zero, no turn (colinear)

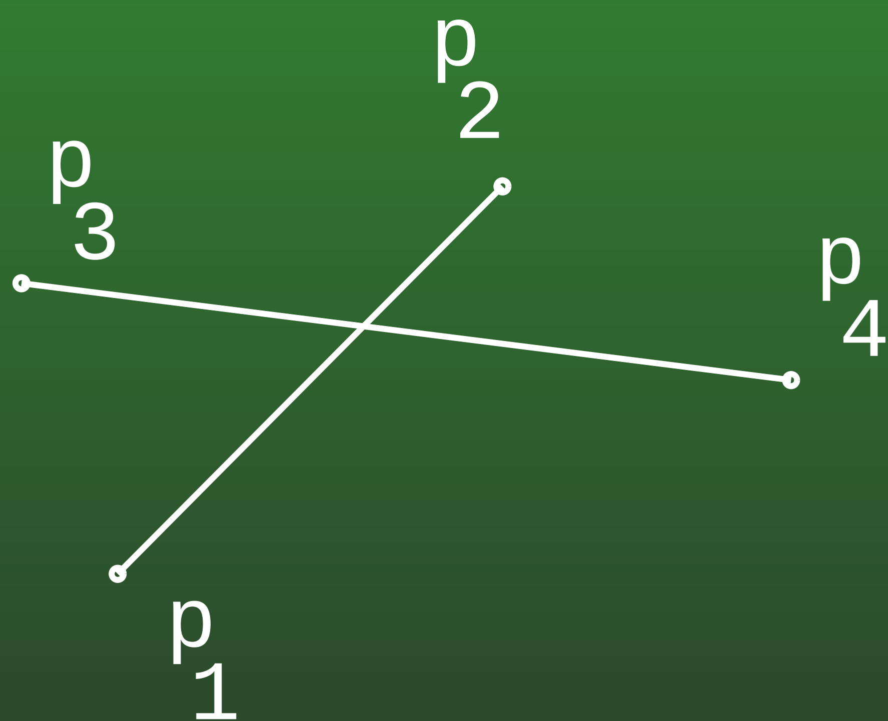
19-7: Line Segment Intersection

- Given two line segments $\overline{p_1p_2}$ and $\overline{p_3p_4}$, do they intersect?
  - How could we determine this?

19-8: Line Segment Intersection

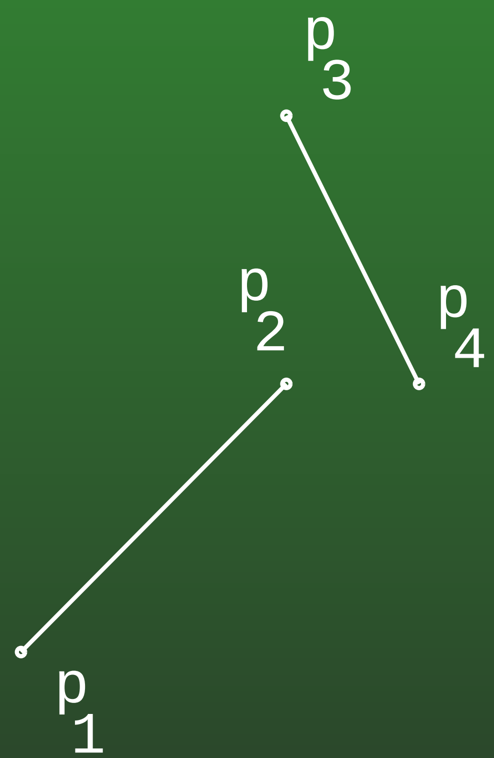
- Given two line segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$, do they intersect?
  - Each segment straddles the line containing the other
  - An endpoint of one segment lies on the other segment

19-9: Line Segment Intersection



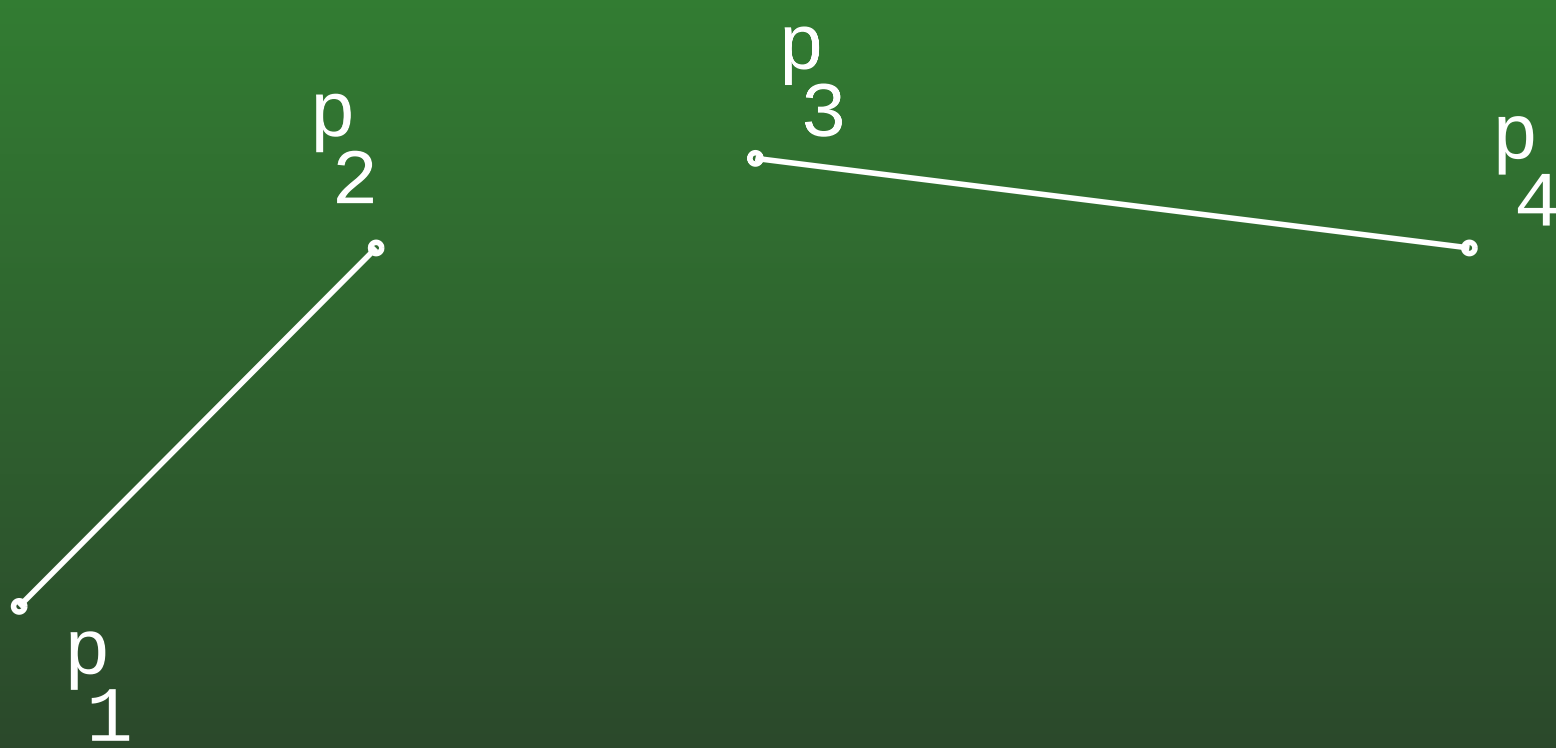
- p_3 and p_4 straddle line defined by p_1 and p_2
- p_1 and p_2 straddle line defined by p_3 and p_4

19-10: Line Segment Intersection



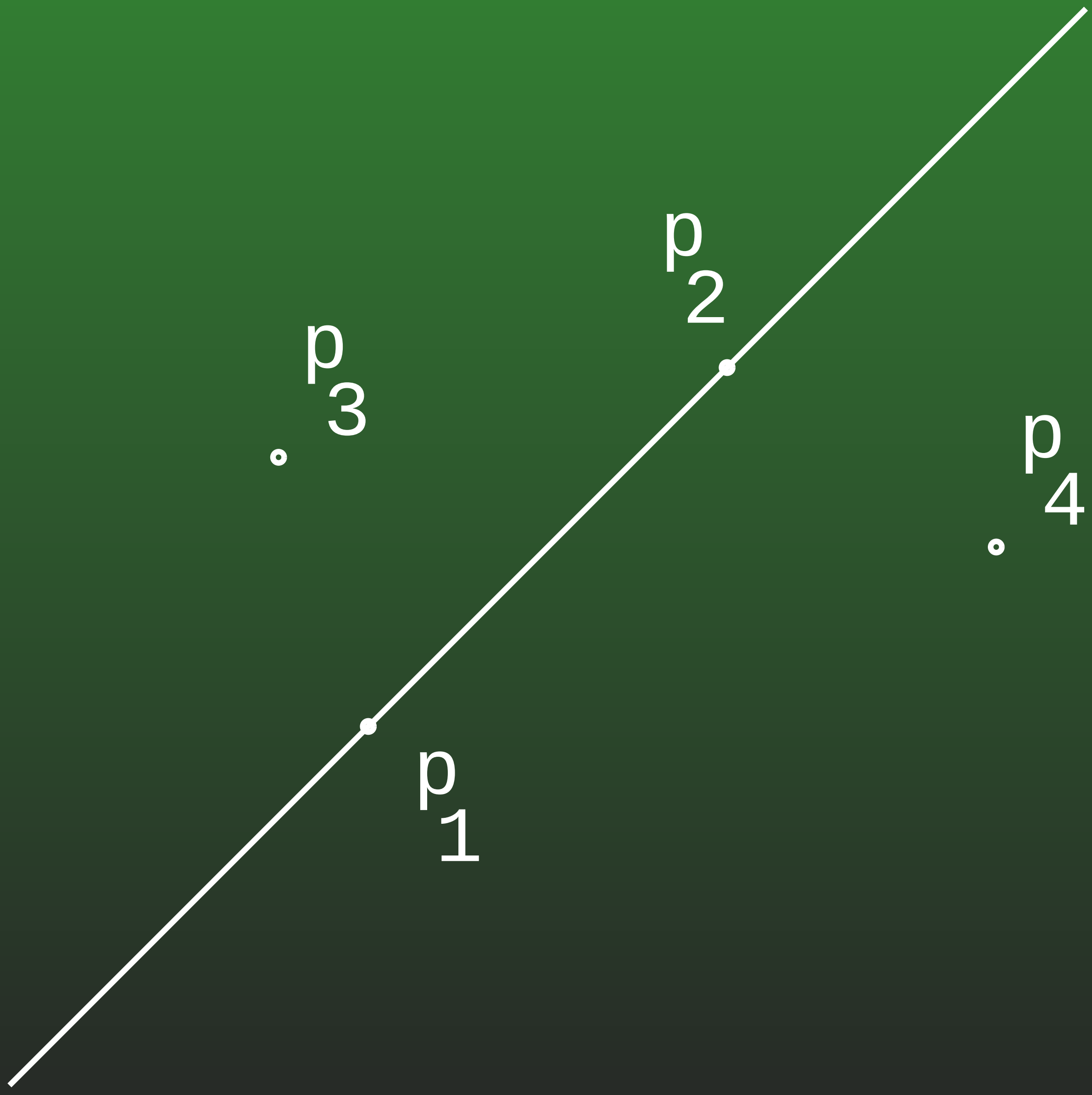
- p_3 and p_4 straddle line defined by p_1 and p_2
- p_1 and p_2 do not straddle line defined by p_3 and p_4

19-11: Line Segment Intersection

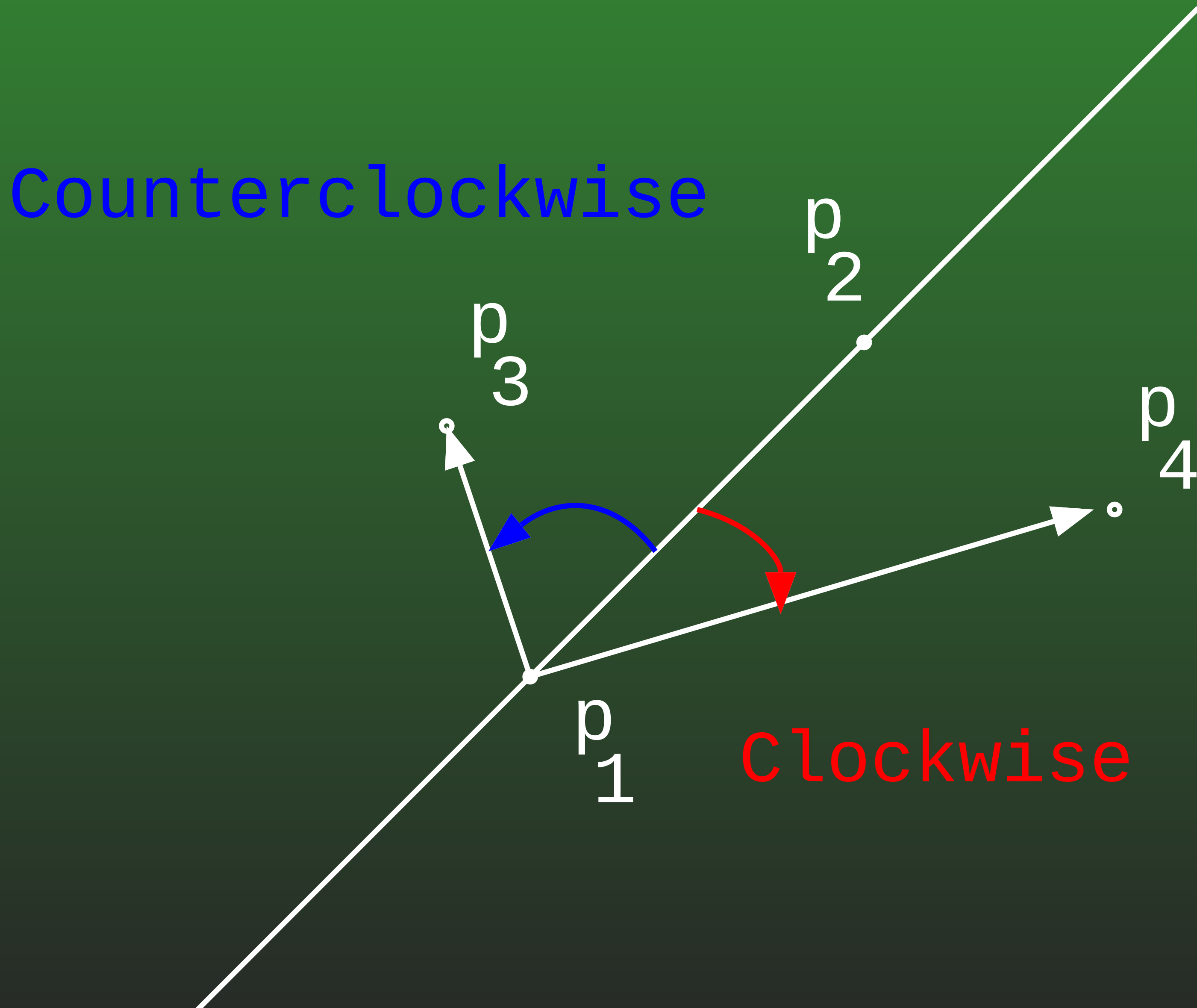


- p_3 and p_4 do not straddle line defined by p_1 and p_2
- p_1 and p_2 do not straddle line defined by p_3 and p_4

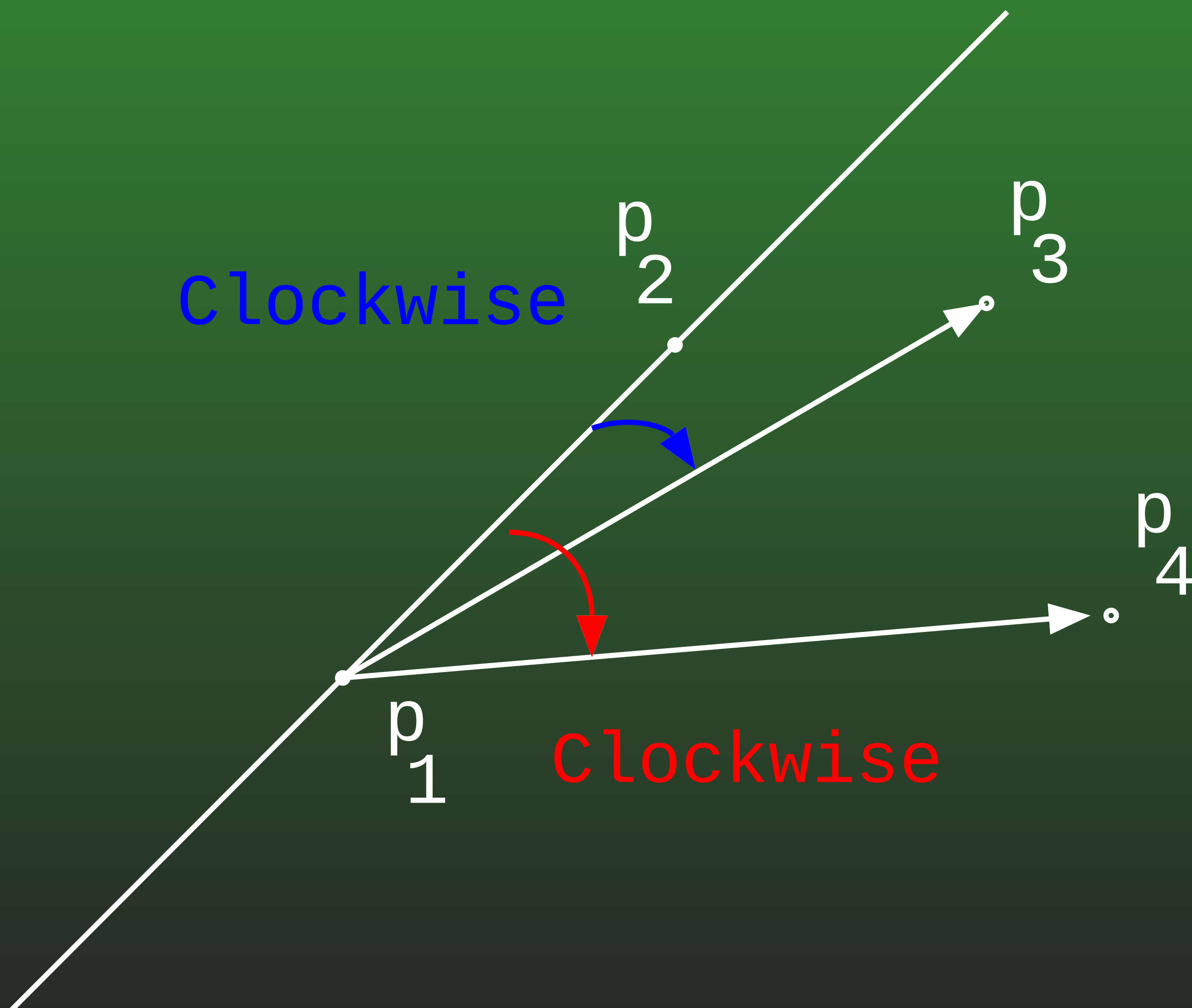
19-12: Line Segment Intersection



19-13: Line Segment Intersection



19-14: Line Segment Intersection



19-15: Line Segment Intersection

- p_3 and p_4 straddle line define by p_1 and p_2 if:
  - $\overrightarrow{p_1 p_3}$ is counterclockwise of $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_1 p_4}$ is clockwise of $\overrightarrow{p_1 p_2}$
    - $(p_2 - p_1) \times (p_3 - p_1) > 0$ and $(p_2 - p_1) \times (p_4 - p_1) < 0$
  - $\overrightarrow{p_1 p_3}$ is clockwise of $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_1 p_4}$ is counterclockwise of $\overrightarrow{p_1 p_2}$
    - $(p_2 - p_1) \times (p_3 - p_1) < 0$ and $(p_2 - p_1) \times (p_4 - p_1) > 0$

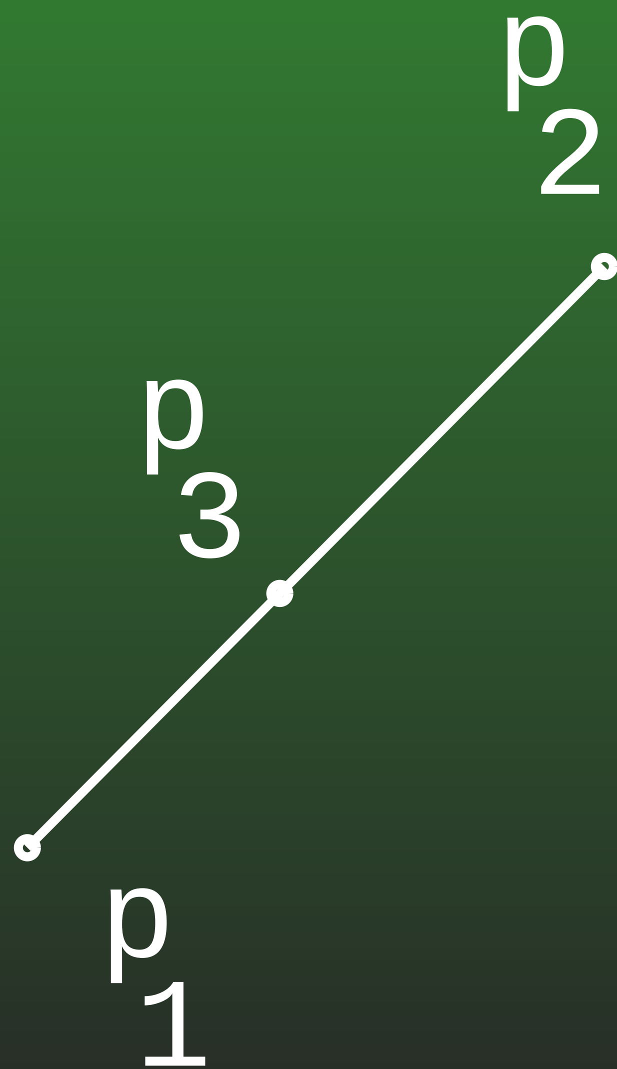
19-16: Line Segment Intersection

- How can we determine if p_3 is on the segment $\overline{p_1 p_2}$?

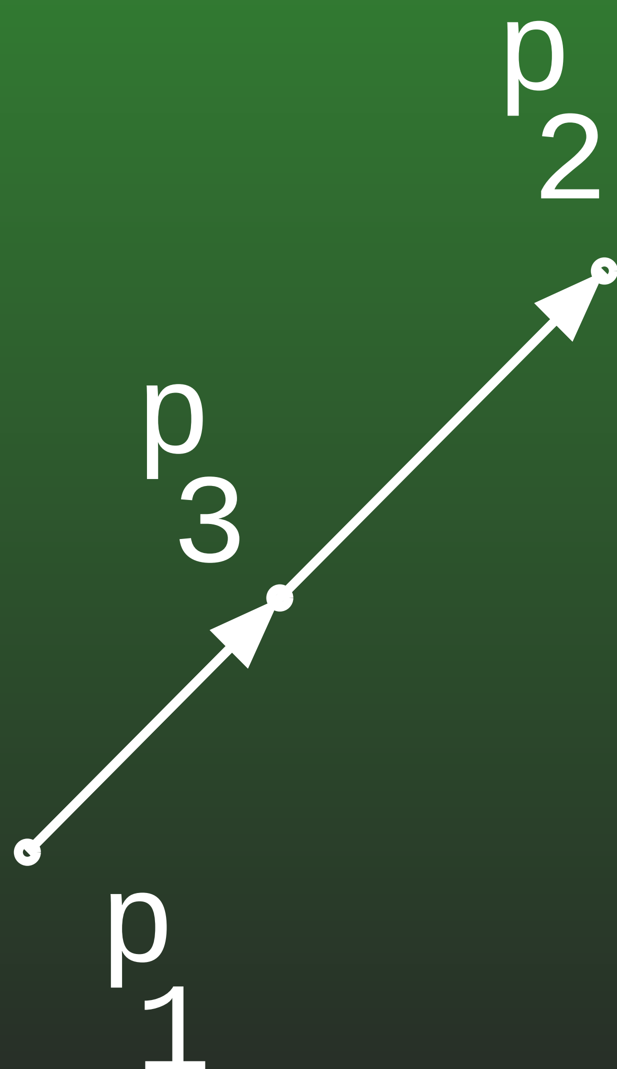
19-17: Line Segment Intersection

- How can we determine if p_3 is on the segment $\overline{p_1 p_2}$?
  - p_3 is on the line defined by p_1 and p_2
  - p_3 is in the proper range along that line

Line Segment Intersection



Line Segment Intersection



Line Segment Intersection

- How can we determine if p_3 is on the segment $\overline{p_1 p_2}$?
  - p_3 is on the line defined by p_1 and p_2
    - $(p_2 - p_1) \times (p_3 - p_1) = 0$
  - p_3 is in the proper range along that line
    - $p_{3_x} \geq p_{1_x} \&\& p_{3_x} \leq p_{2_x}$ or $p_{3_x} \leq p_{1_x} \&\& p_{3_x} \geq p_{2_x}$
    - $p_{3_y} \geq p_{1_y} \&\& p_{3_y} \leq p_{2_y}$ or $p_{3_y} \leq p_{1_y} \&\& p_{3_y} \geq p_{2_y}$

Line Segment Intersection

- Given a set of n line segments, do any of them intersect?
  - What is a brute force method for solving this problem?
  - How long does it take (if there are n total line segments)

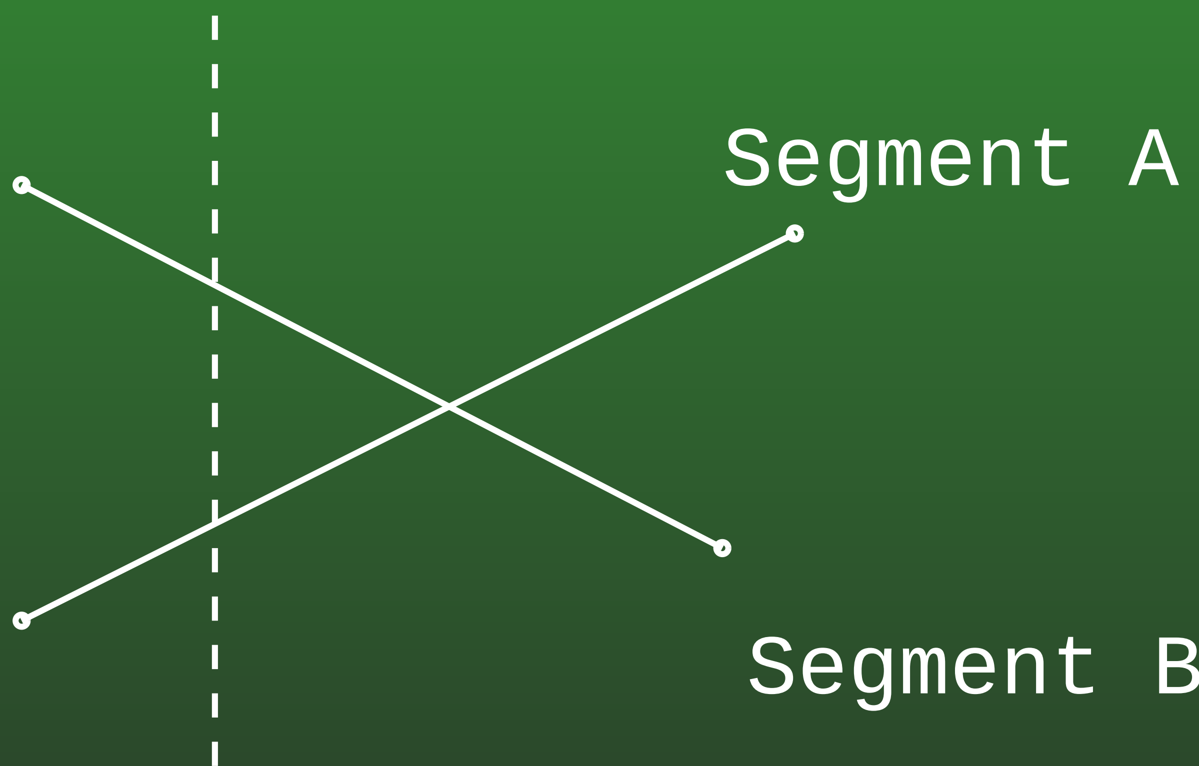
19-22: Line Segment Intersection

- Given a set of n line segments, do any of them intersect?
  - What is a brute force method for solving this problem?
    - Check each pair of line segments, see if they intersect using the previous technique
  - How long does it take (if there are n total line segments)
    - Each of the n segments needs to be comparted to $n-1$ other segments, for a total time of $O(n^2)$
- We can do better!

19-23: Line Segment Intersection

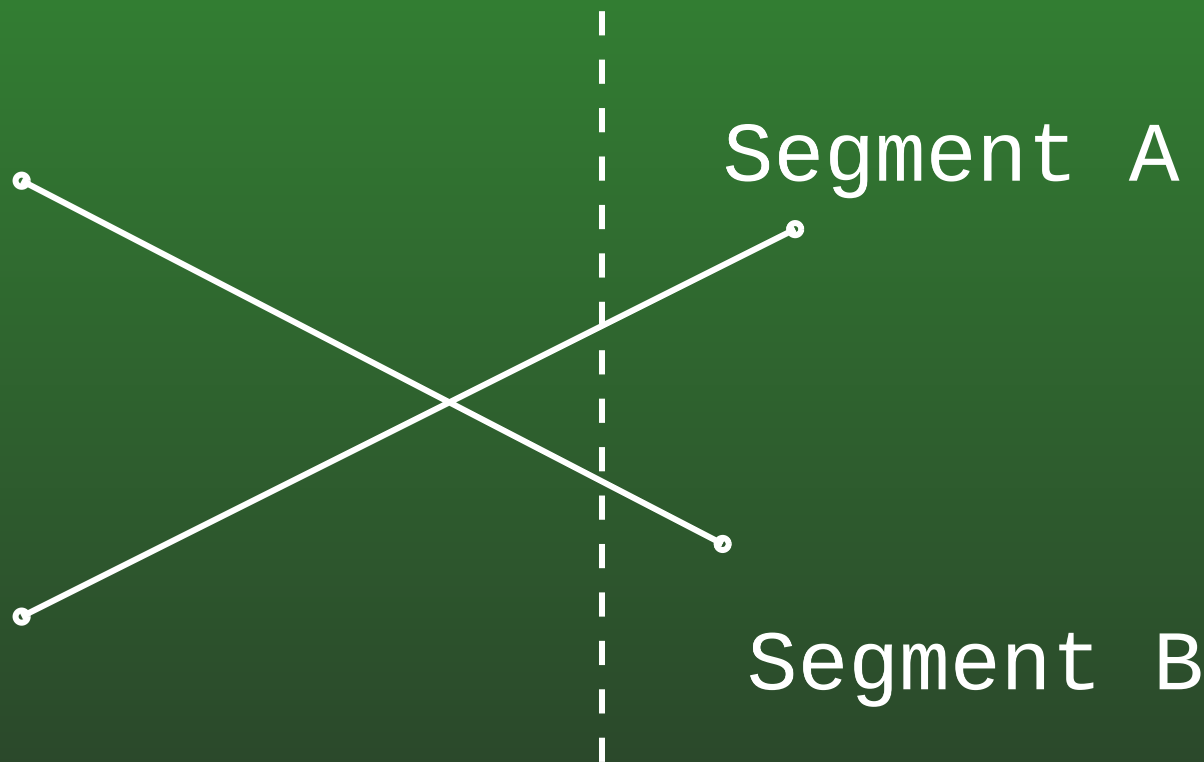
- Basic idea:
  - Assume that there are no vertical line segments
  - Sweep a vertical line across the segments
  - Segment A is above segment B at the line, and as we move the line to the right, Segment B becomes above Segment A, then the segments have crossed

19-24: Line Segment Intersection



- Segment B is above Segment A

19-25: Line Segment Intersection



- Segment A is above Segment B
- The two segments must have crossed

19-26: Line Segment Intersection

- Maintain an ordered list of the segments that intersect with the current sweep line
- Whenever two segments become adjacent on this list, check to see if they intersect
- Only need to check endpoints of segments

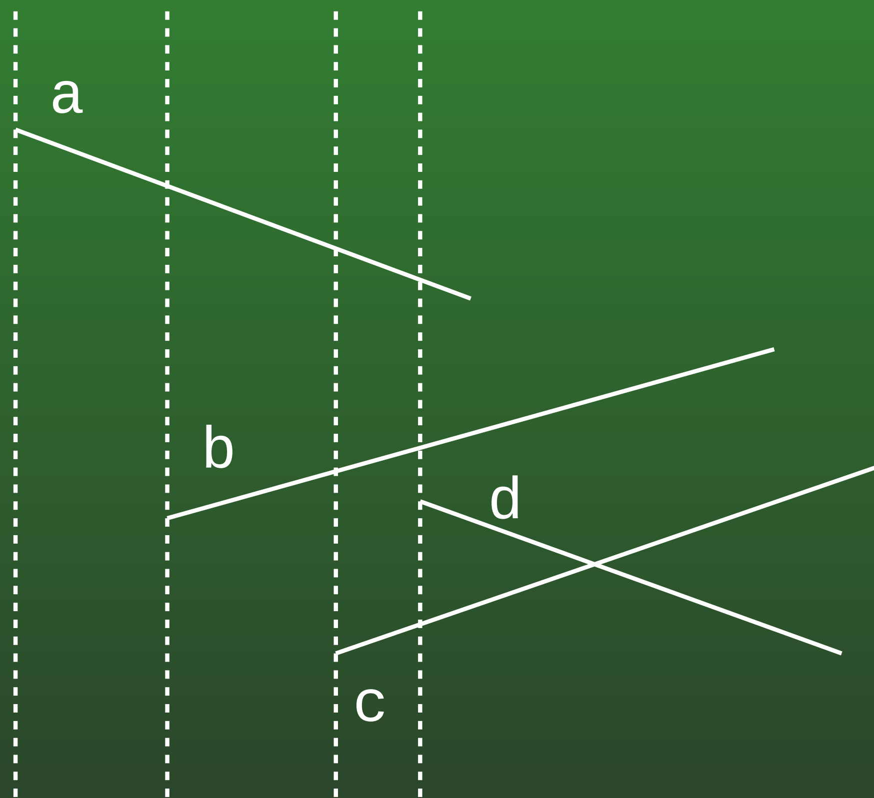
19-27: Line Segment Intersection

- Maintian a data structure that lets us:
  - Insert a segment s into T
  - Delete a segment s from T
  - Find the segment above s in T
  - Find the segment below s in T
- Use a red-black tree, using cross products to see if segment 1 is above segment 2 at a certain point

19-28: Line Segment Intersection

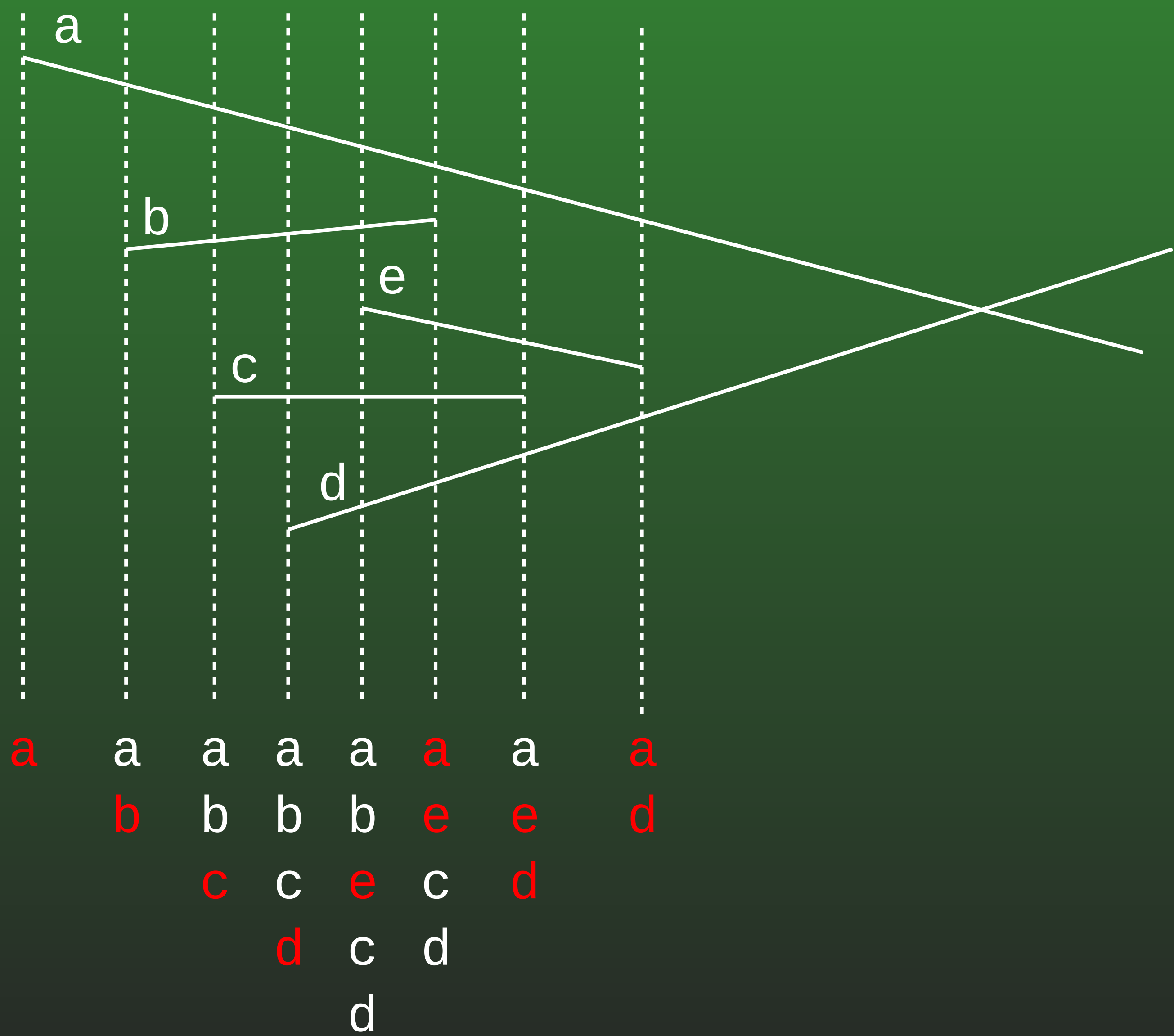
```
Sort endpoints of segments from left to right
    Break ties:
        Left endpoints before right endpoints
        lowest y-coordinate first
for each point p in endpoint list
    if p is the left endpoint of a segment s
        Insert s into T
        if there is a segment above s in T that intersects s
            or a segment below s in T that intersects s
                return true
    if p is the right endpoint of a segment s
        if there is a segment above s and below s in T
            and these segments intersect
                return true
        Delete s from T
return false
```

19-29: Convex Hull



a	a	a	a
	b	b	b
		c	d
			c

Convex Hull

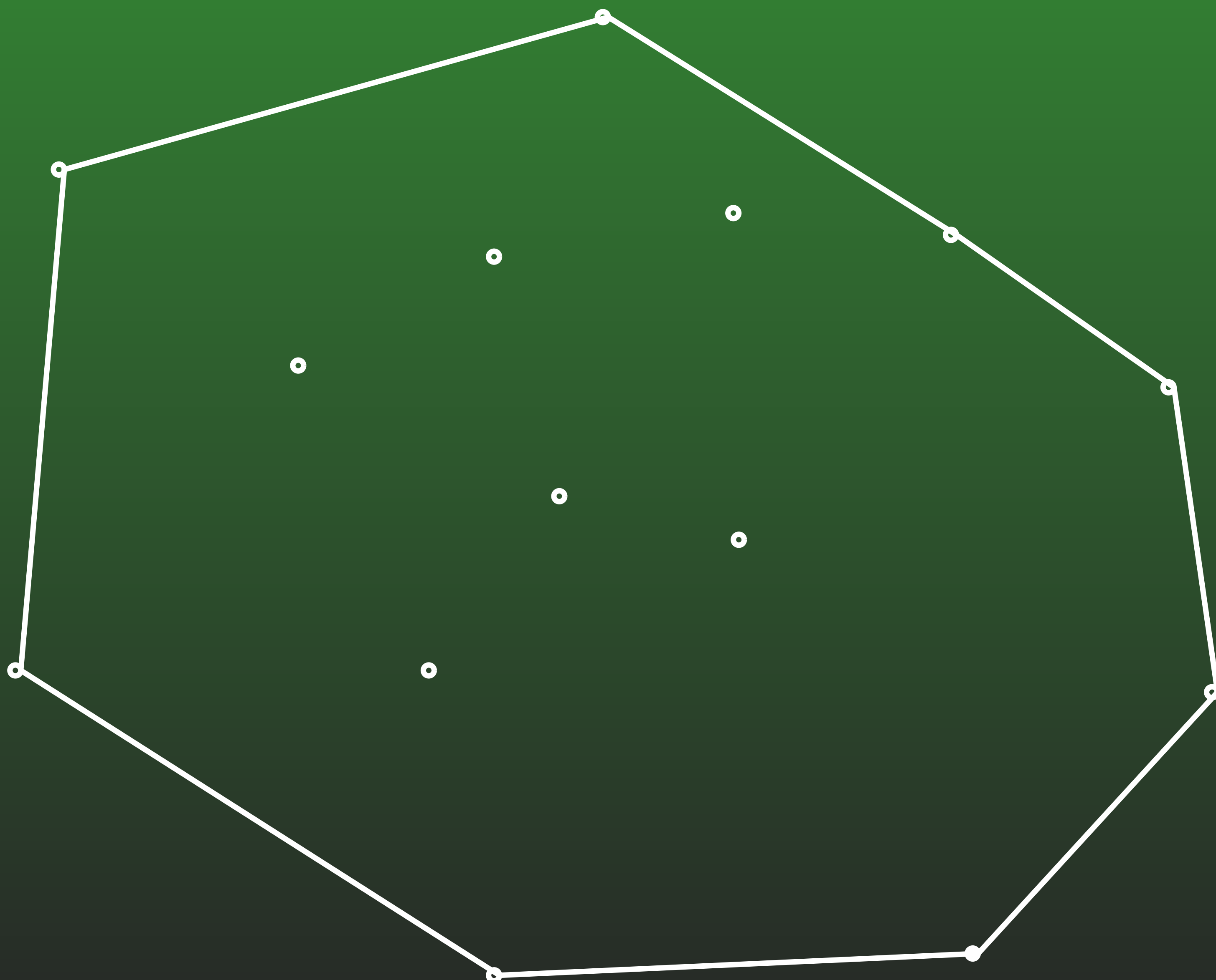


19-31: Convex Hull

- Given a set of points, what is the smallest convex polygon that contains all points
- Alternately, if all of the points were nails in a board, and we placed a rubber band around all of them, what shape would it form?

 # Convex Hull

19-33: Convex Hull



Convex Hull

- Several computational geometry problems have finding the convex hull as a subproblem
  - Like many graph algorithms have finding a topological sort as a subproblem
- For instance: Finding the two furthest points
  - Must lie on the convex hull

19-35: Convex Hull

- Graham’s Scan Algorithm
  - Go through all the points in order
  - Push points onto a stack
  - Pop off points that don’t form part of the convex hull
  - When we’re done, stack contains the points in the convex hull

Convex Hull

Gram-Scan

Let p_0 be the point with the minimum y-coordinate

Sort the points by increasing polar angle around p_0

Push p_0, p_1, and p_2 on the stack S

for $i \leftarrow$ 3 to n do

  while angle formed by top two points on S

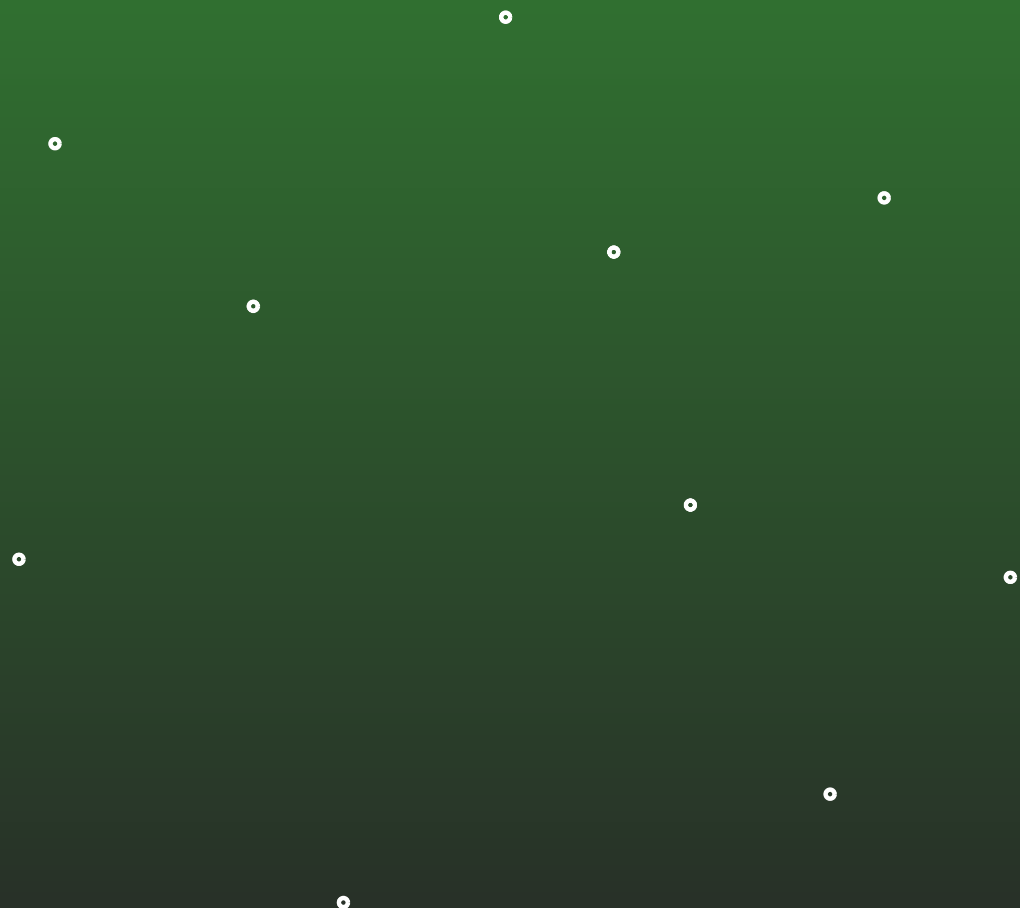
  doesn't turn left do

    Pop

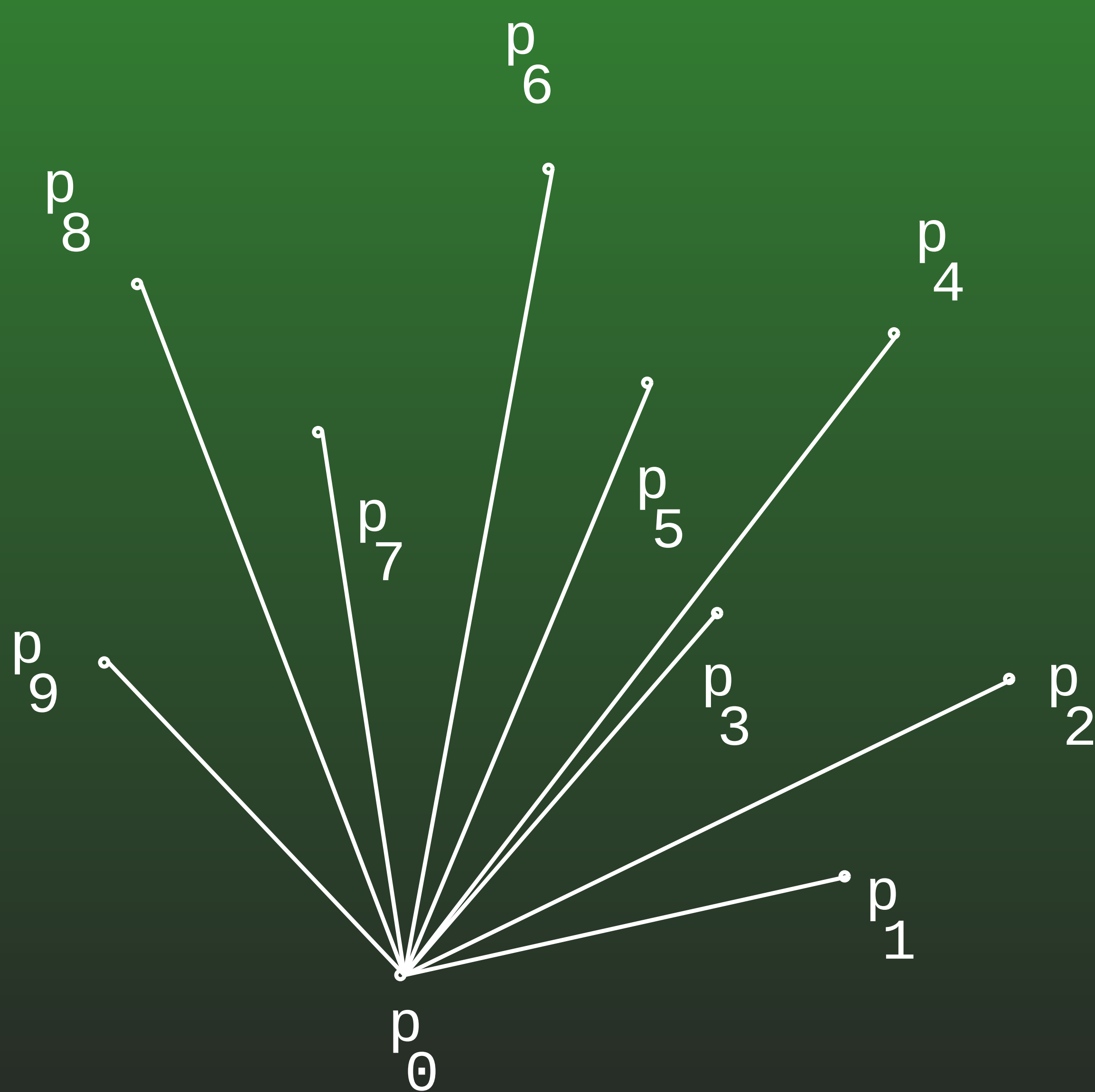
  Push(p_i)

return S

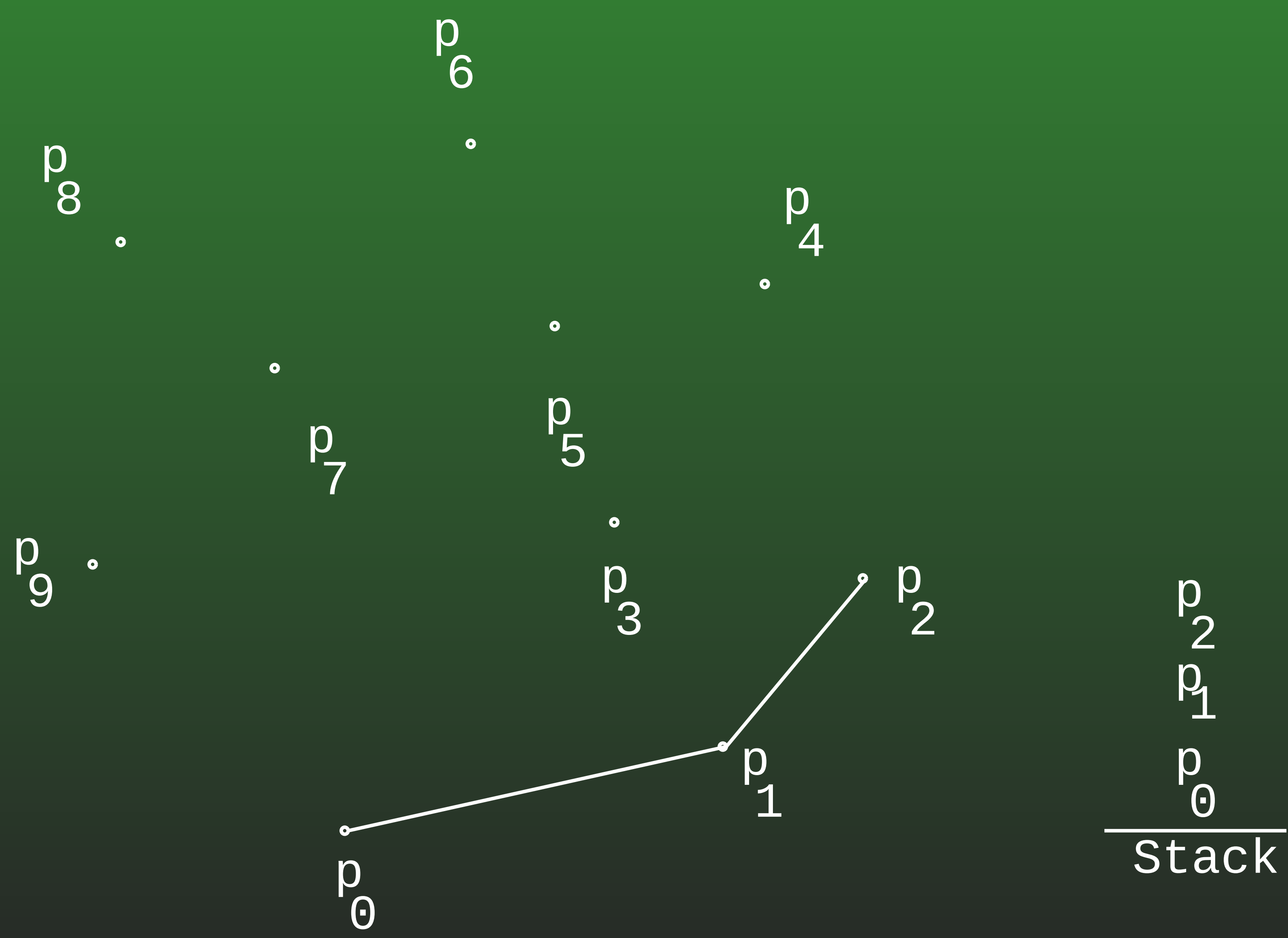
Graham's Scan



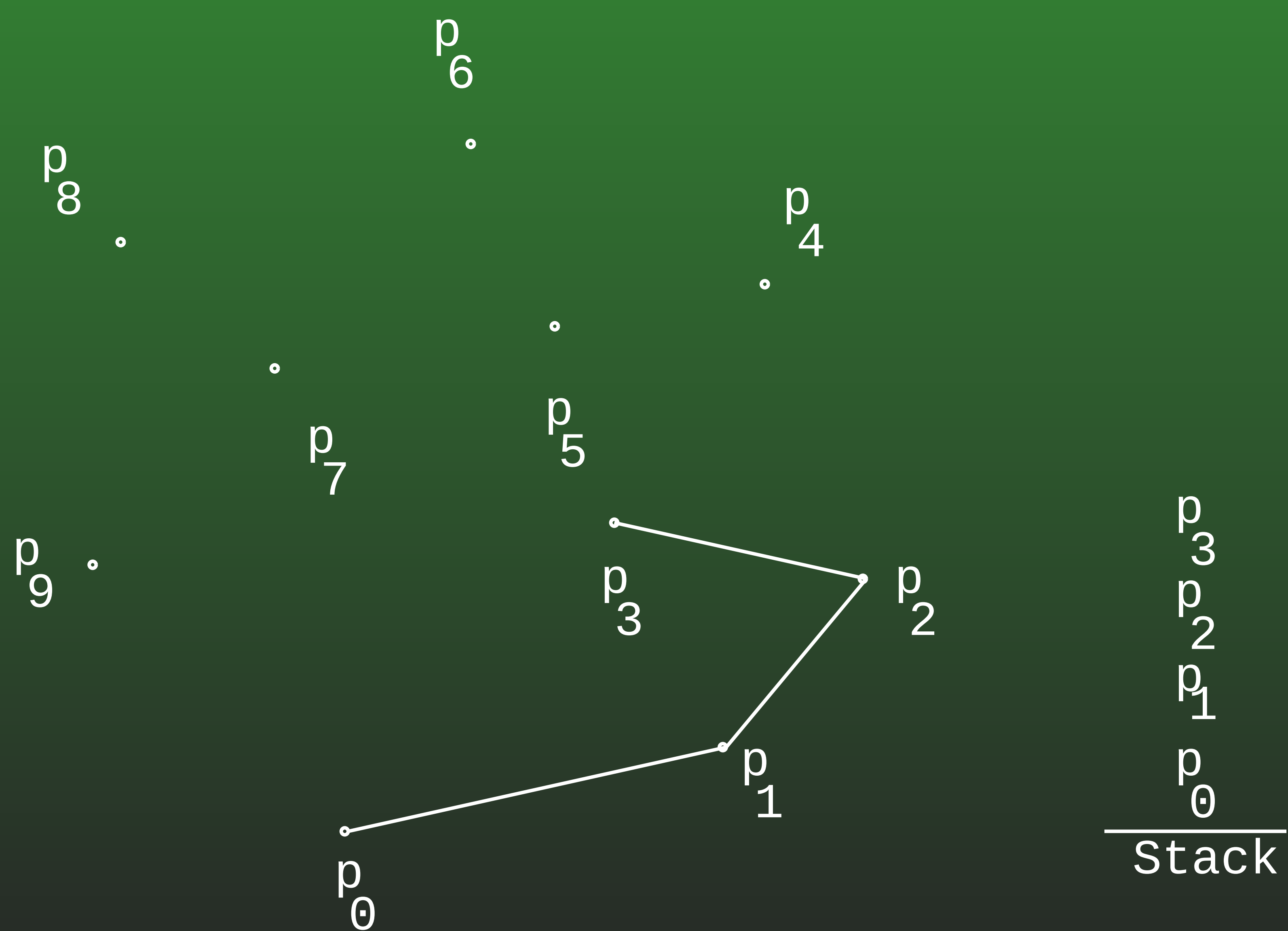
19-38: Graham's Scan



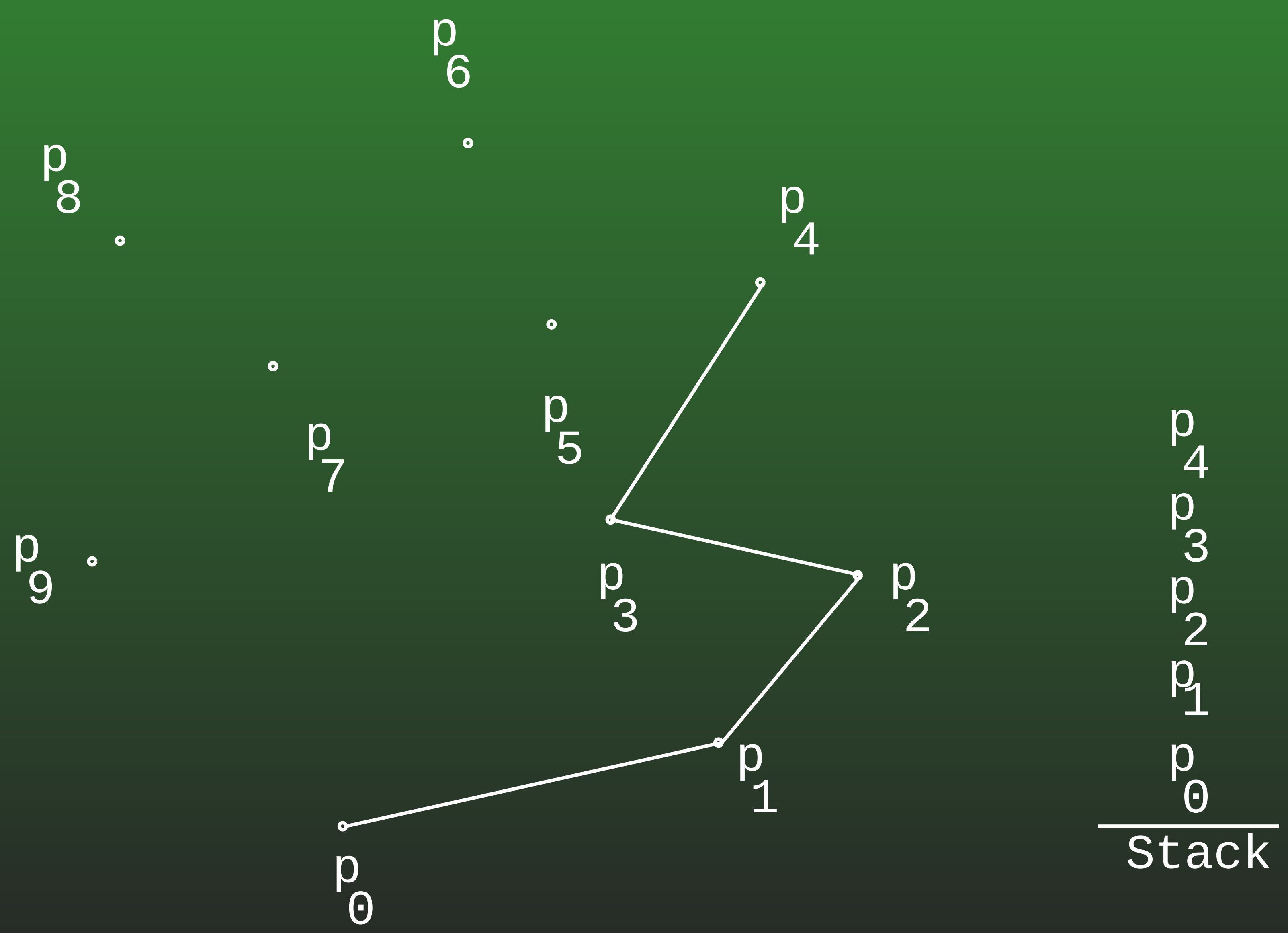
Graham's Scan



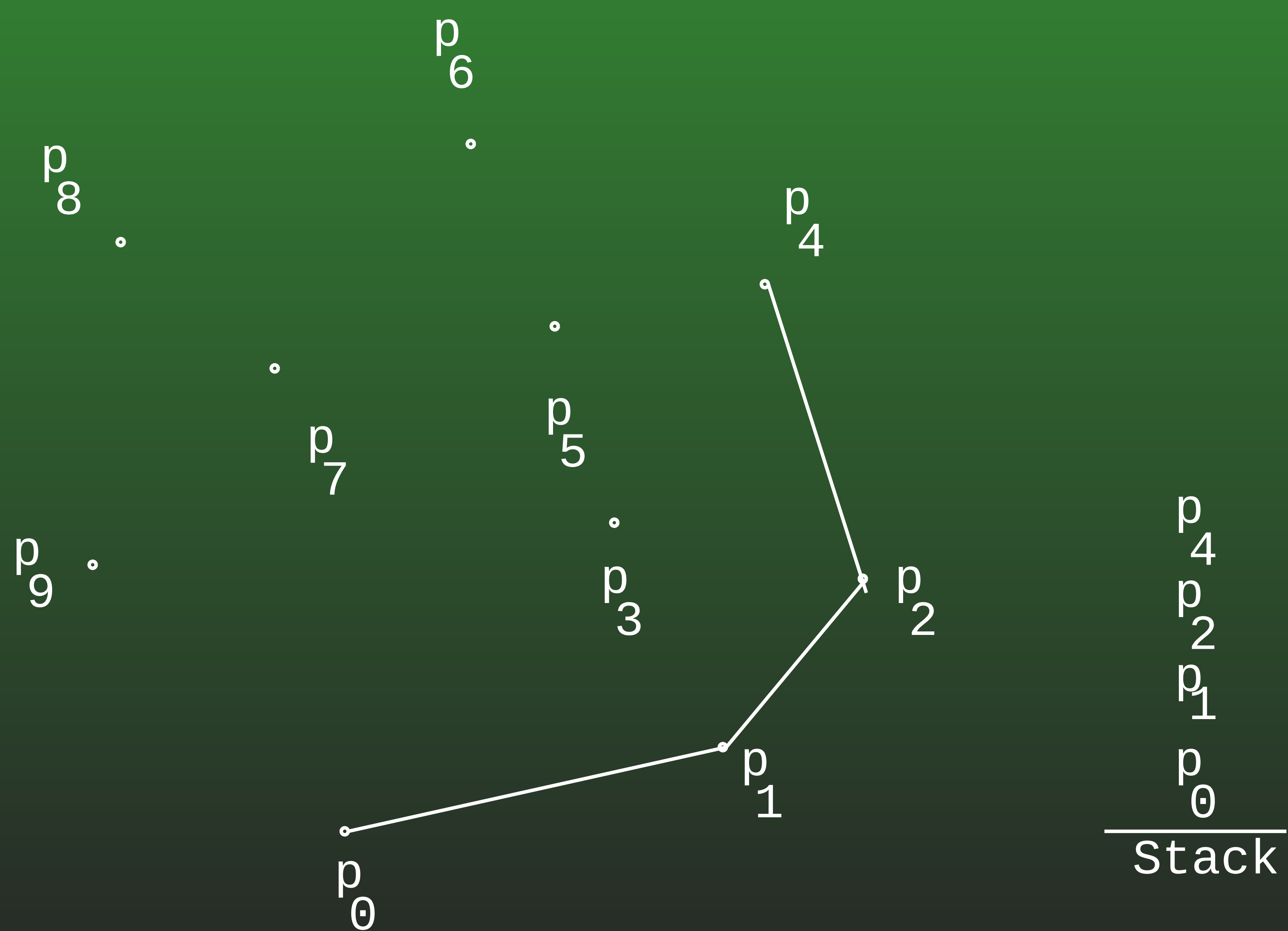
Graham's Scan



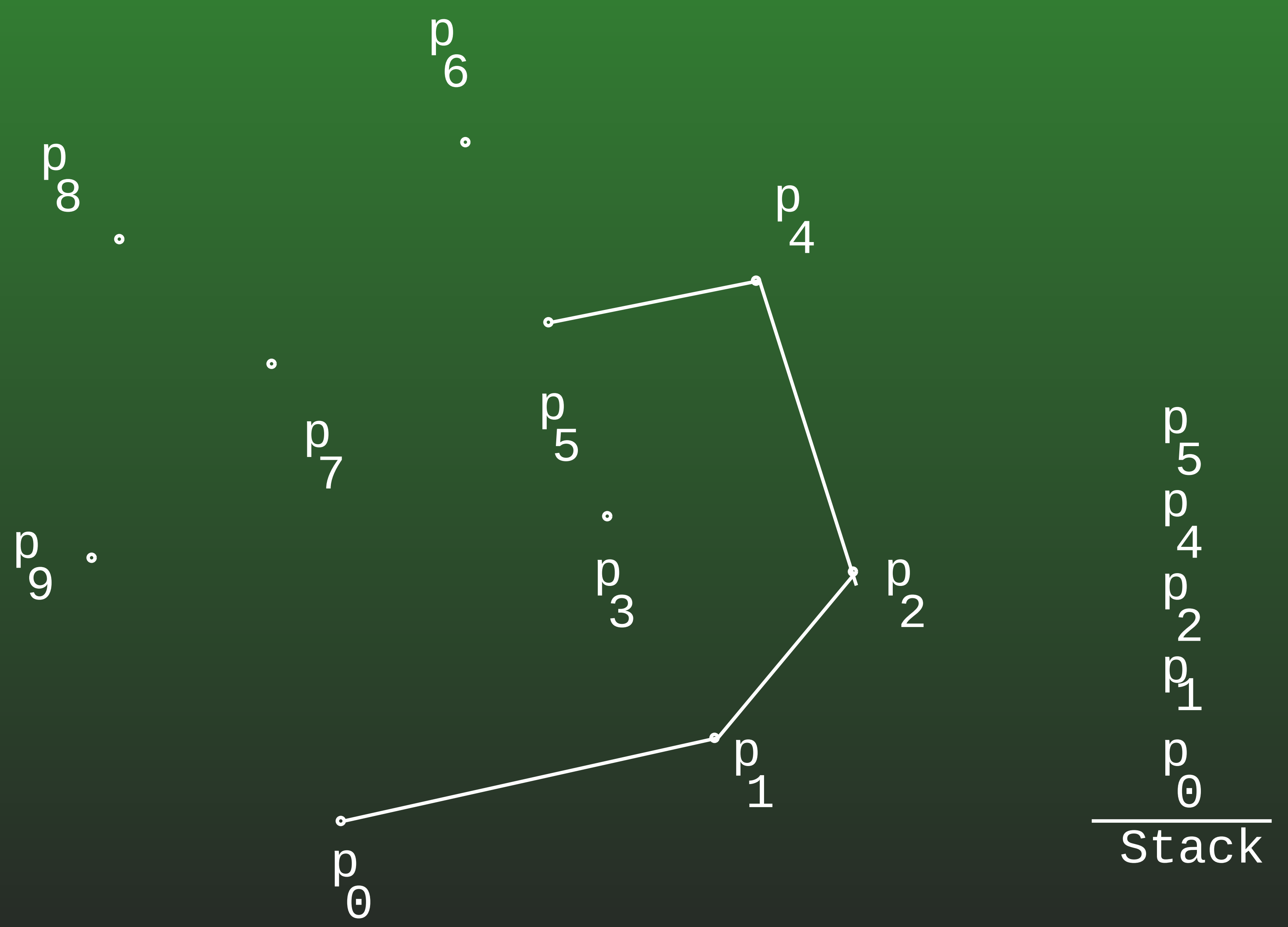
19-41: Graham's Scan



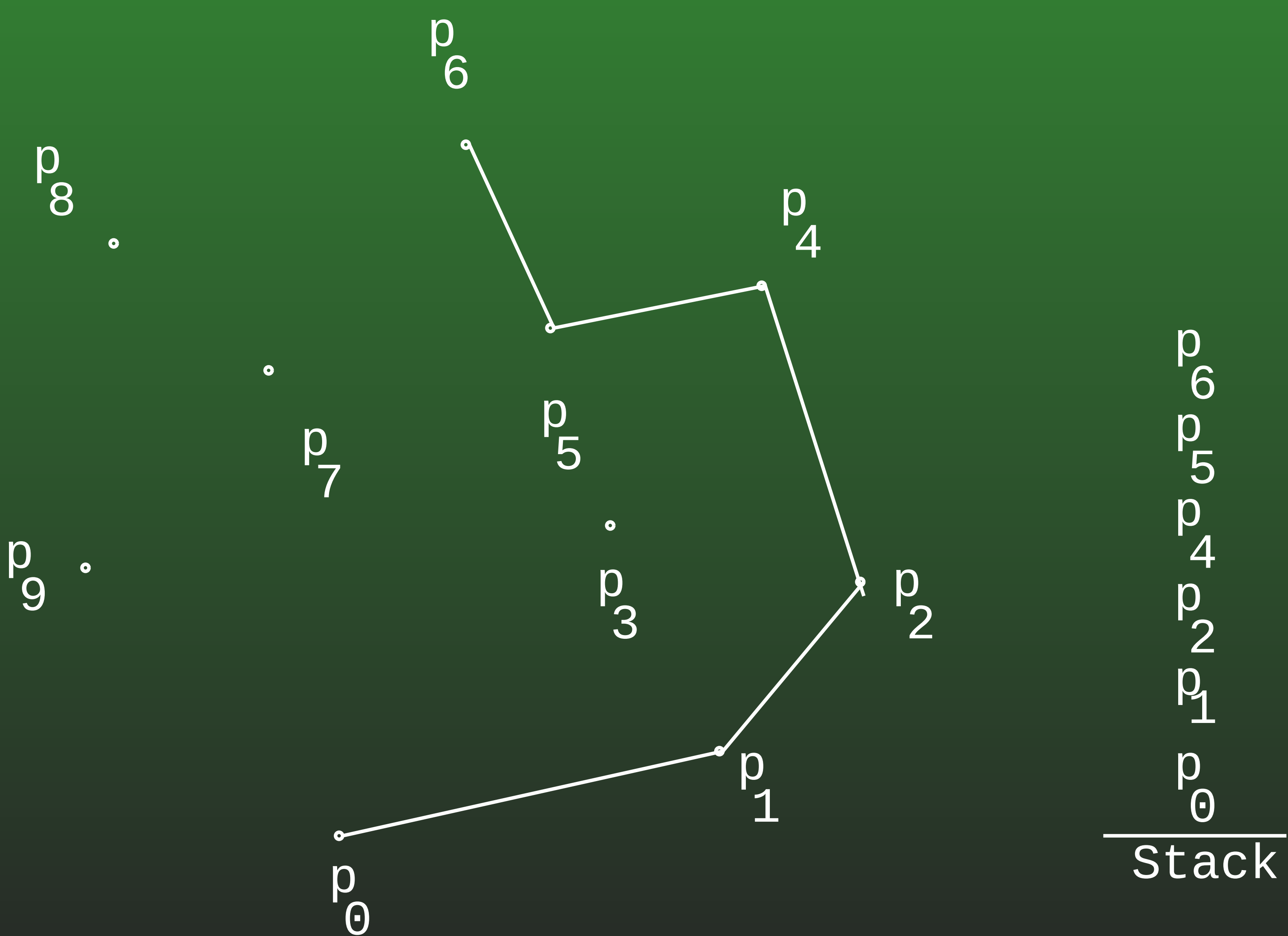
Graham's Scan



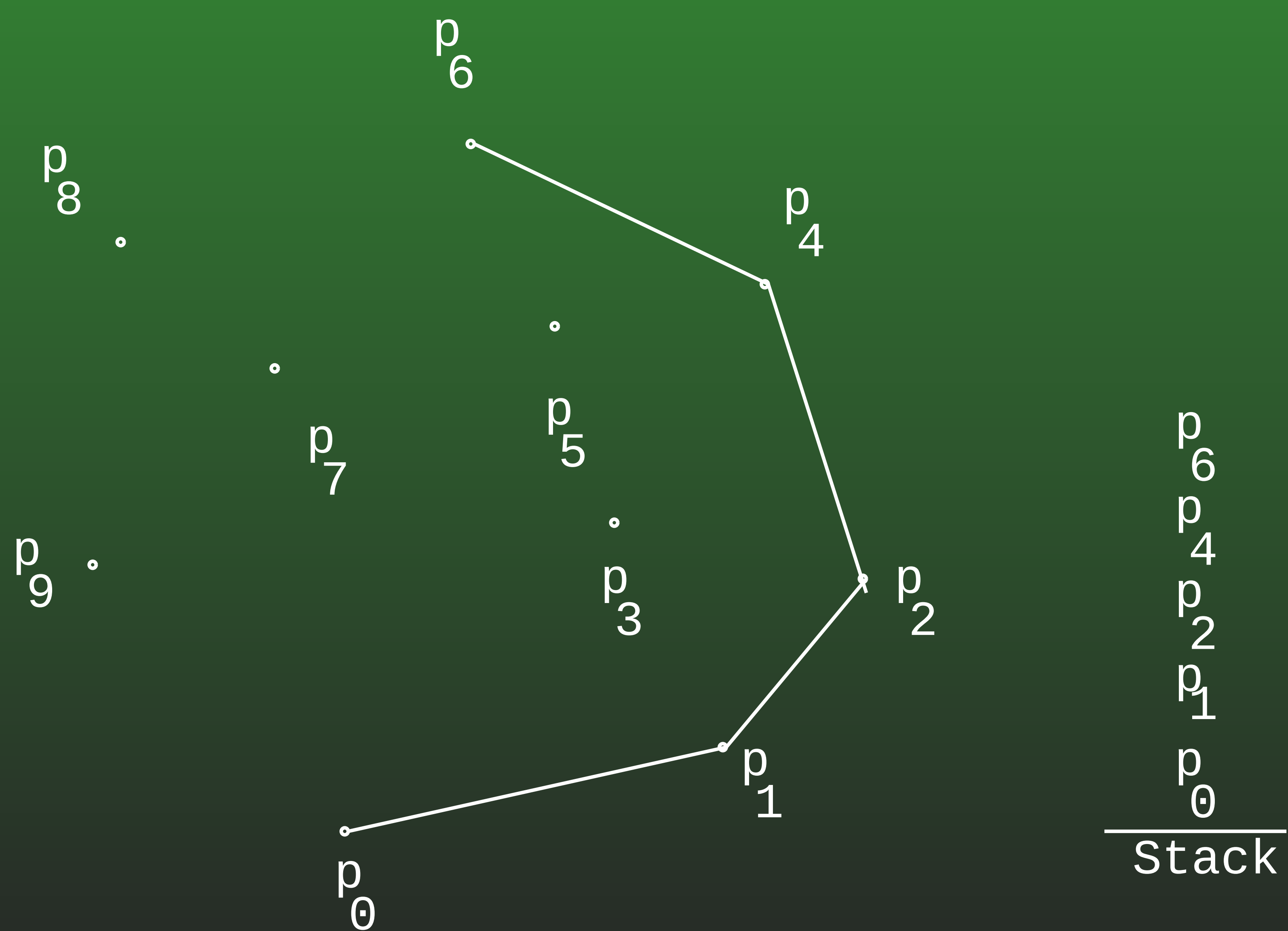
 # Graham's Scan



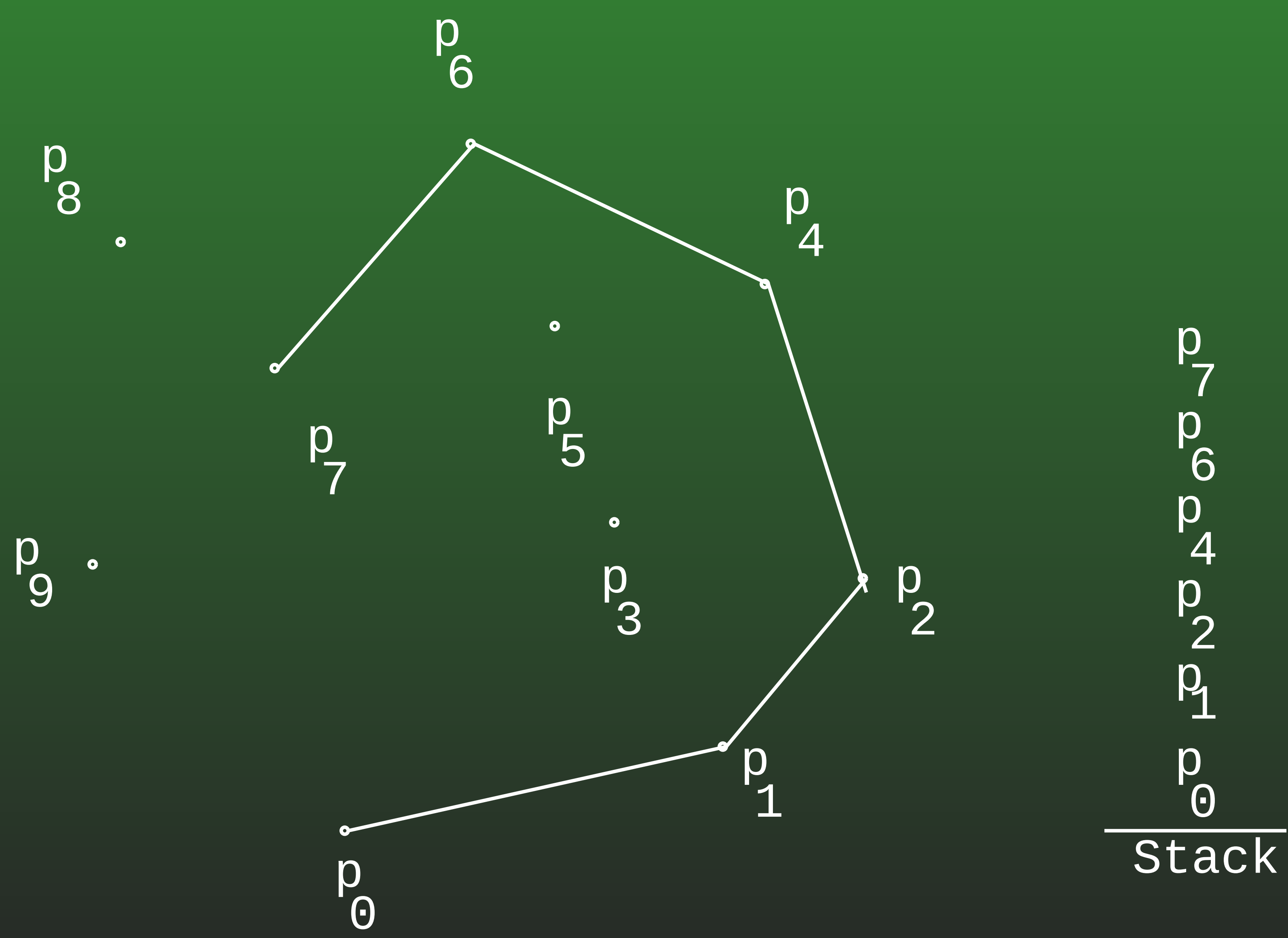
Graham's Scan



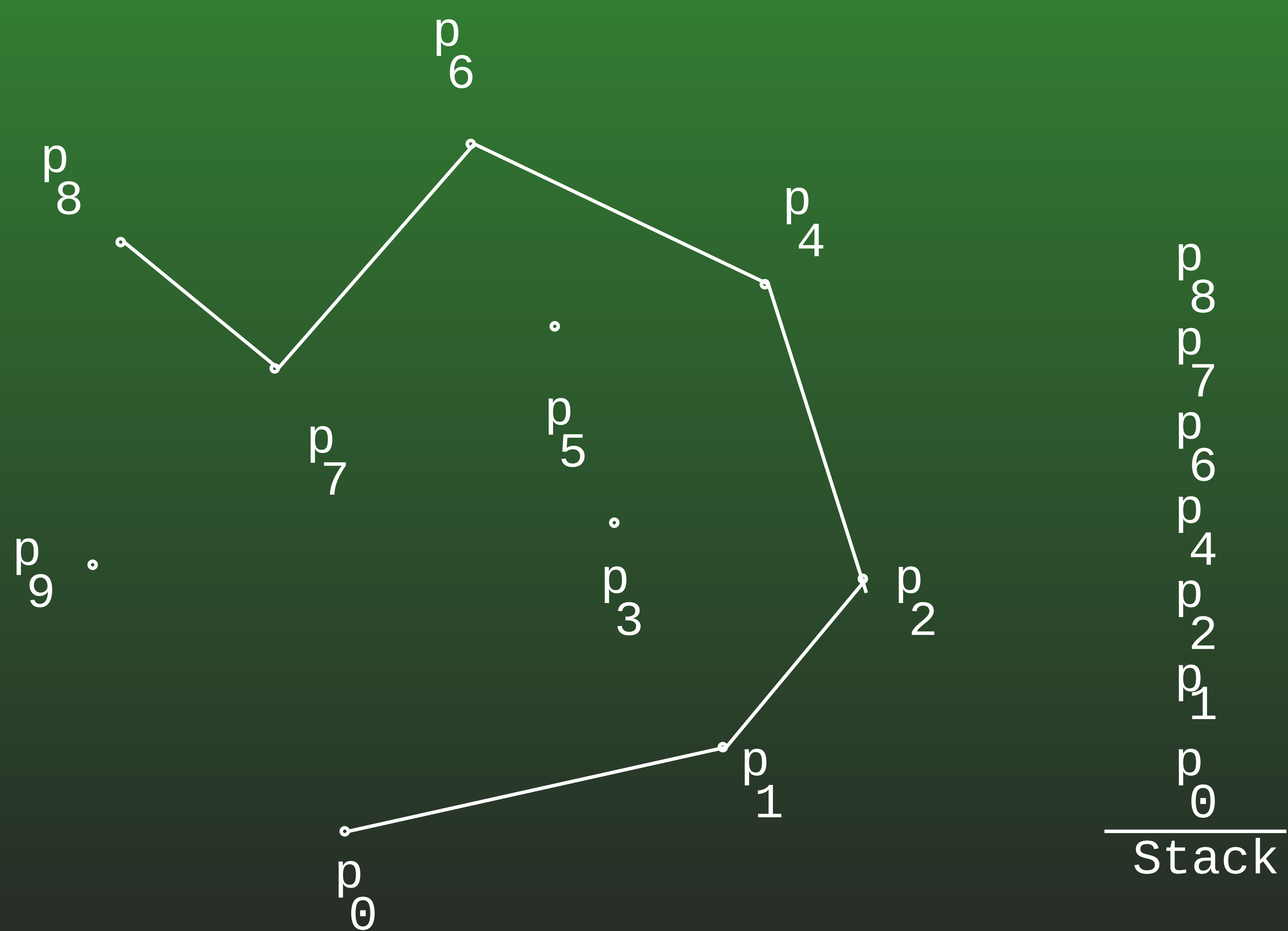
Graham’s Scan



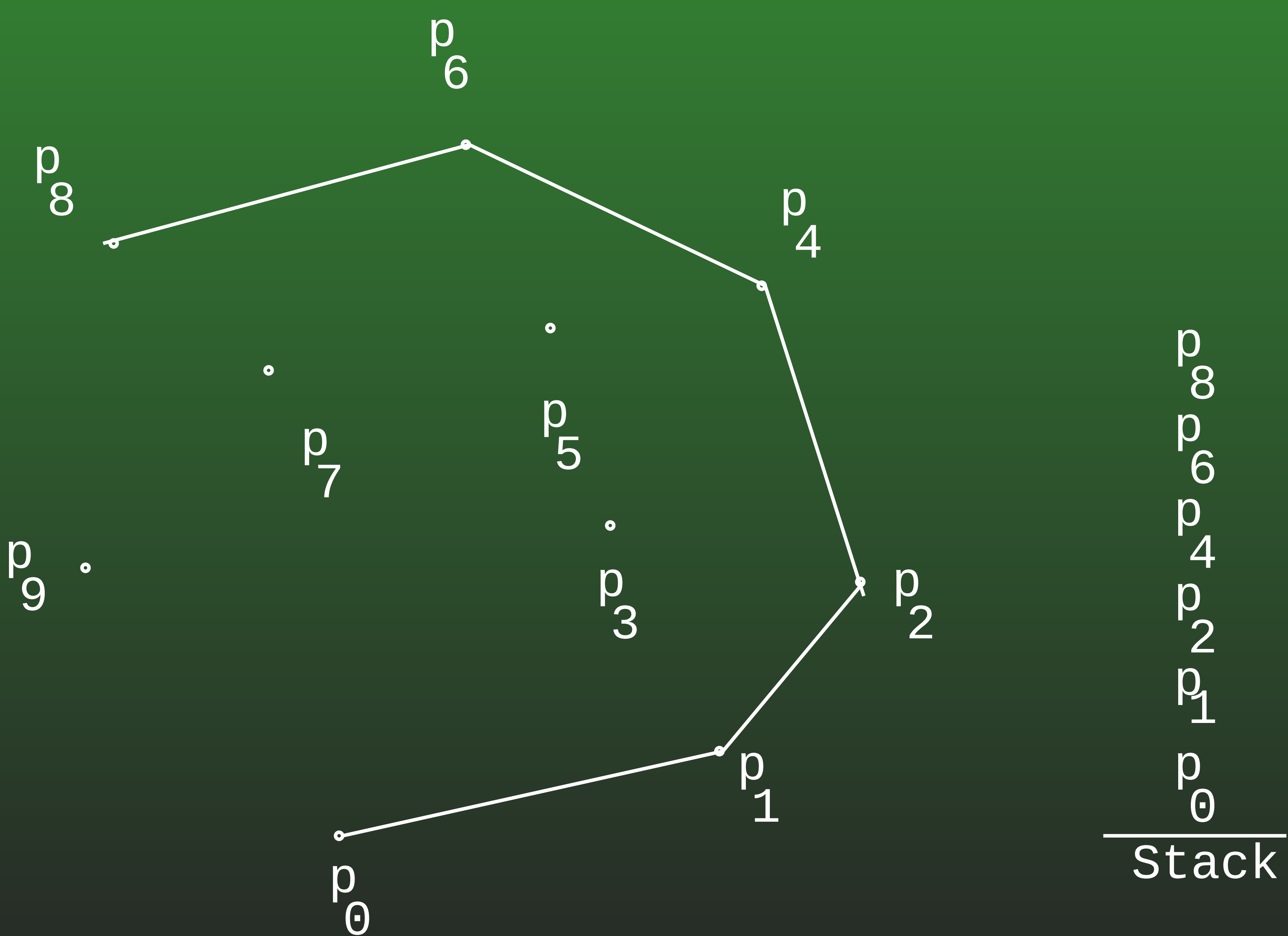
19-46: Graham's Scan



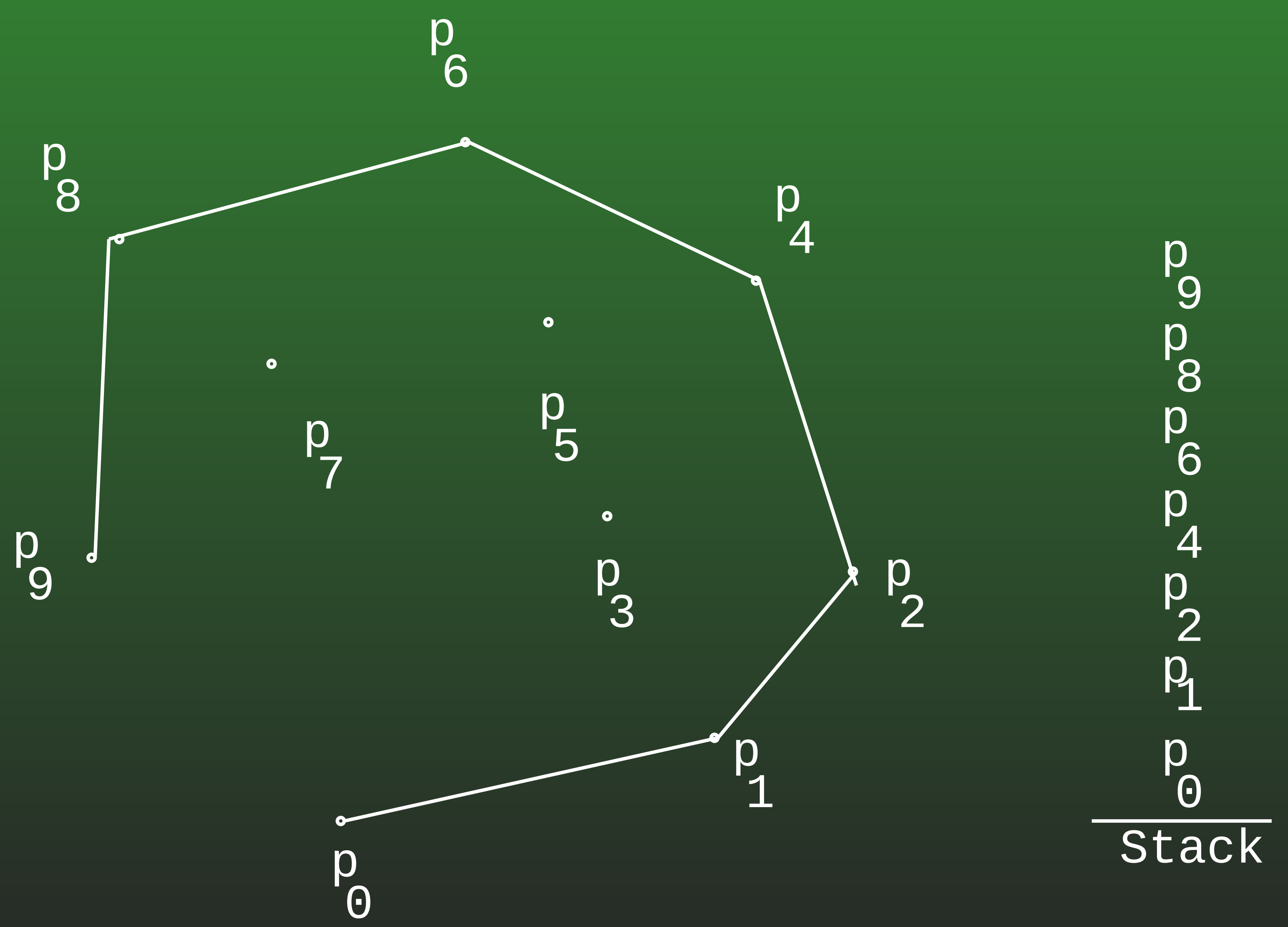
Graham's Scan



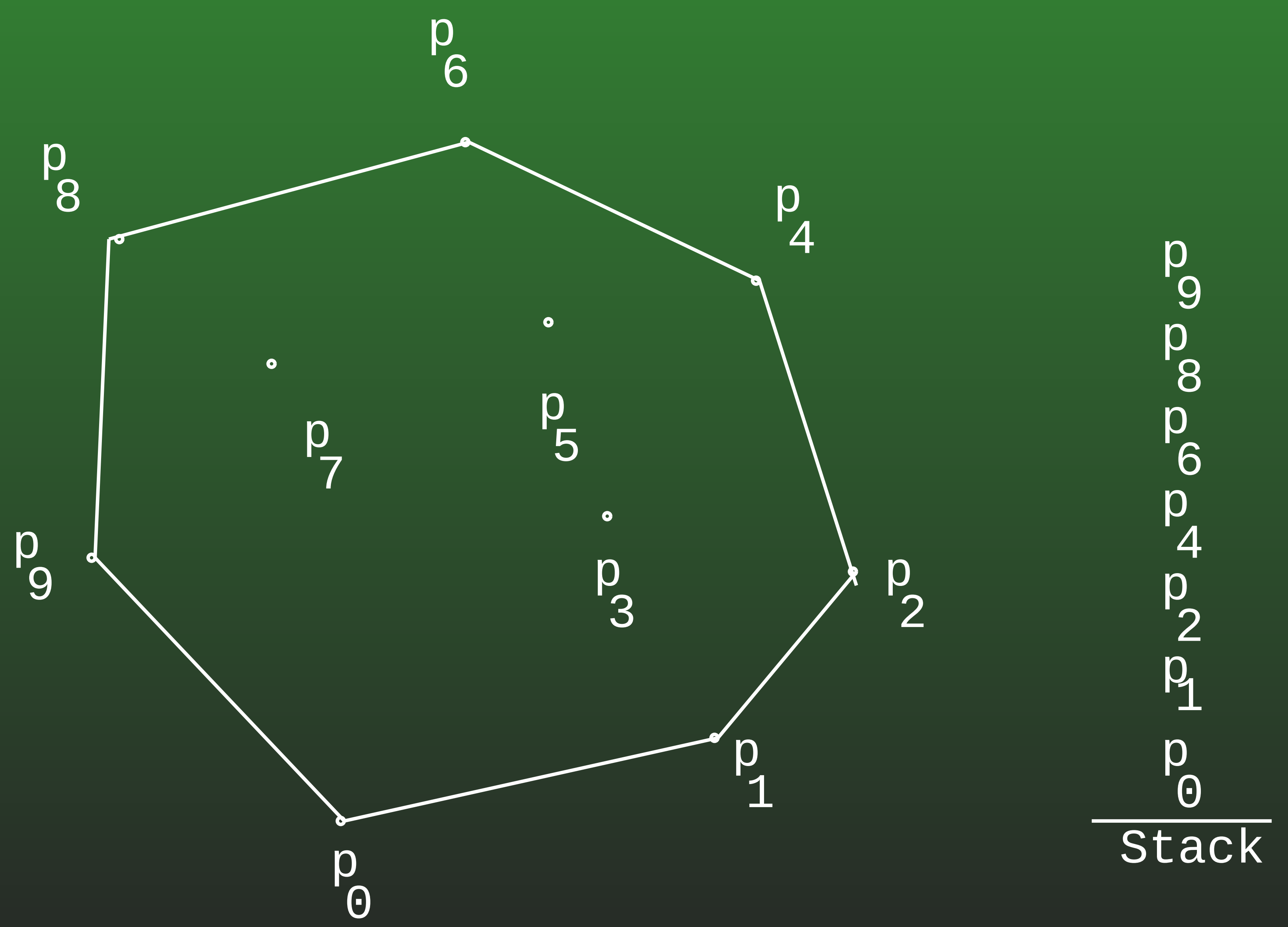
Graham's Scan



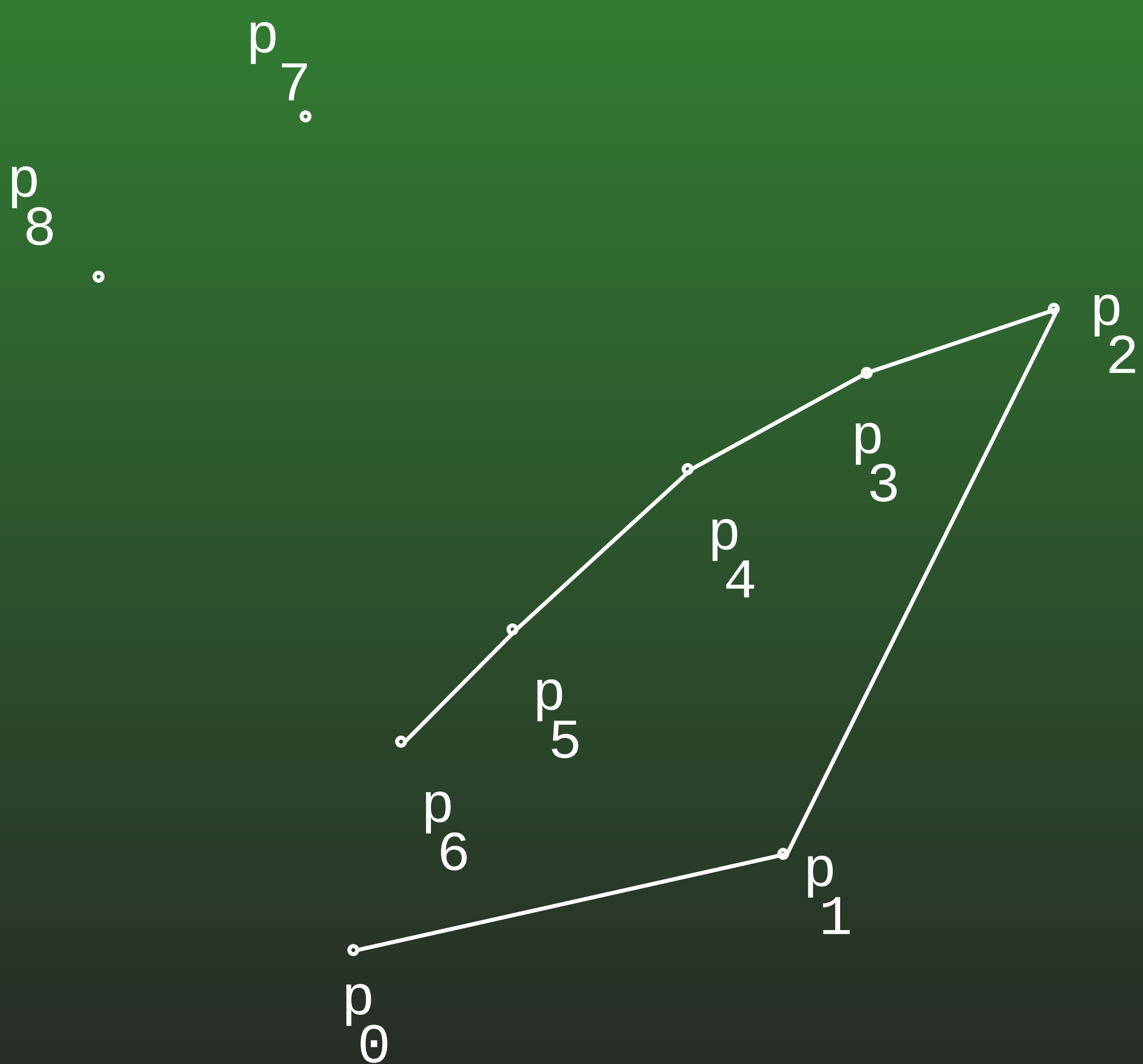
Graham's Scan



Graham's Scan

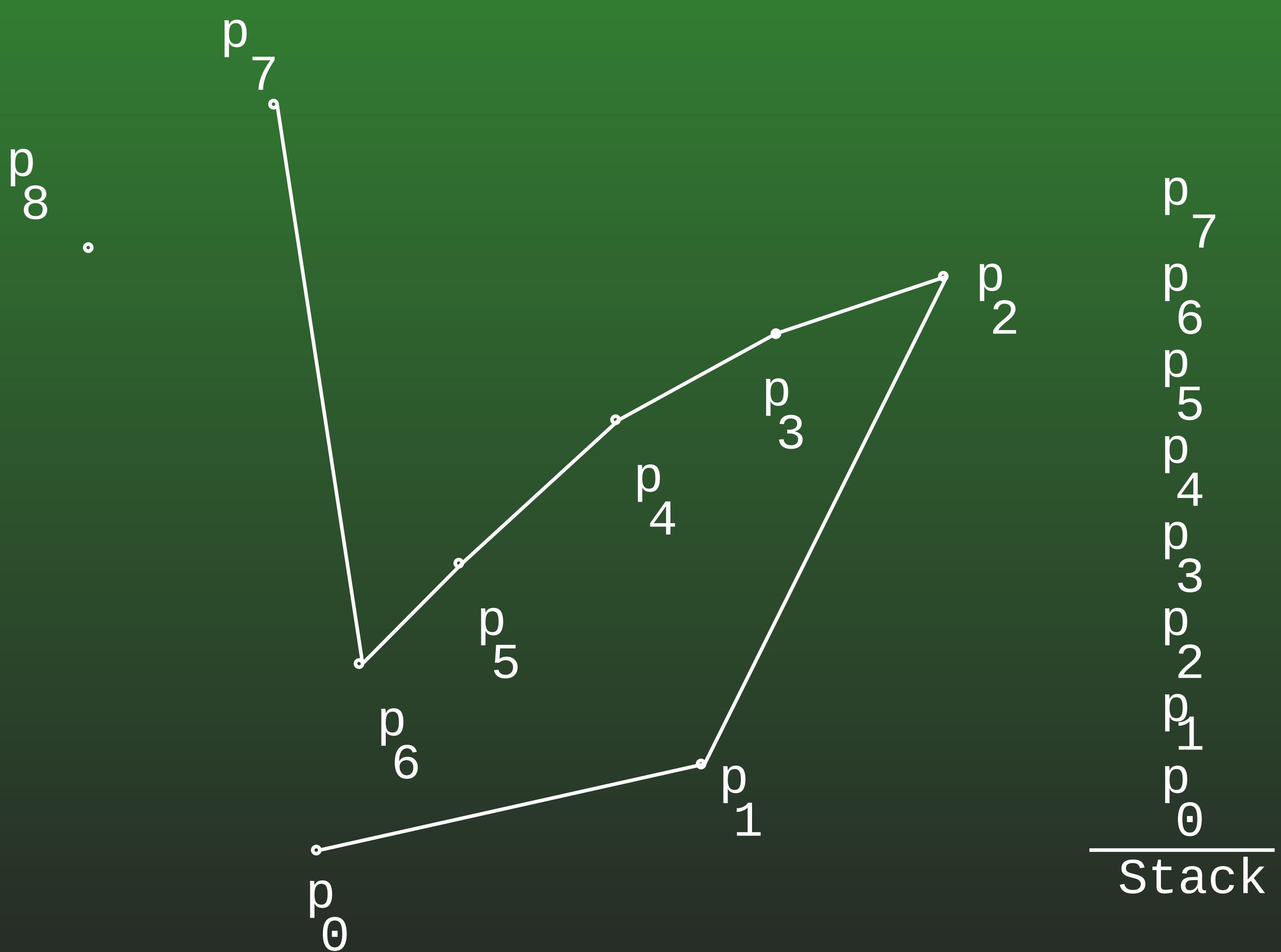


19-51: Graham's Scan

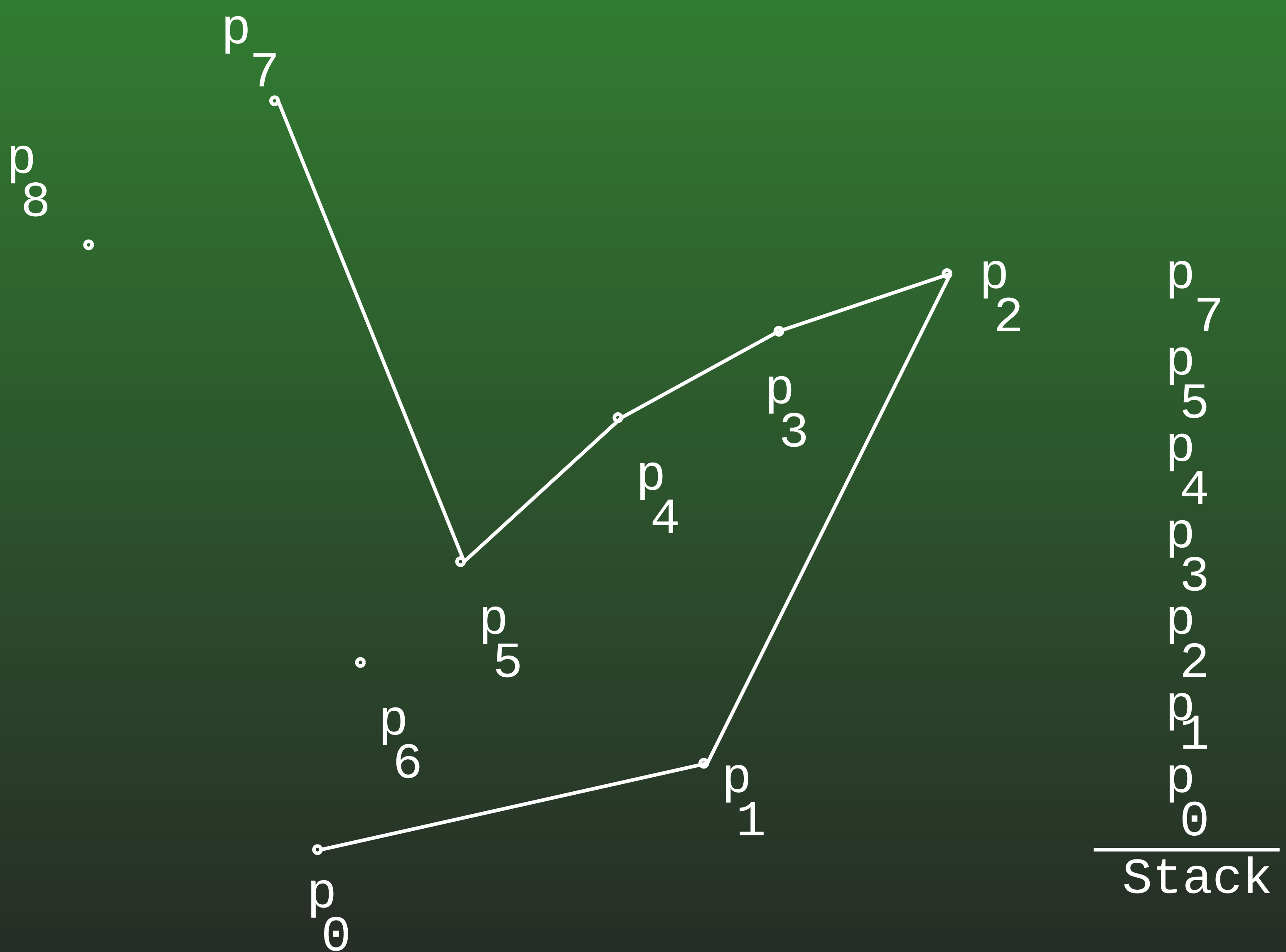


p6
p5
p4
p3
p2
p1
p0
Stack

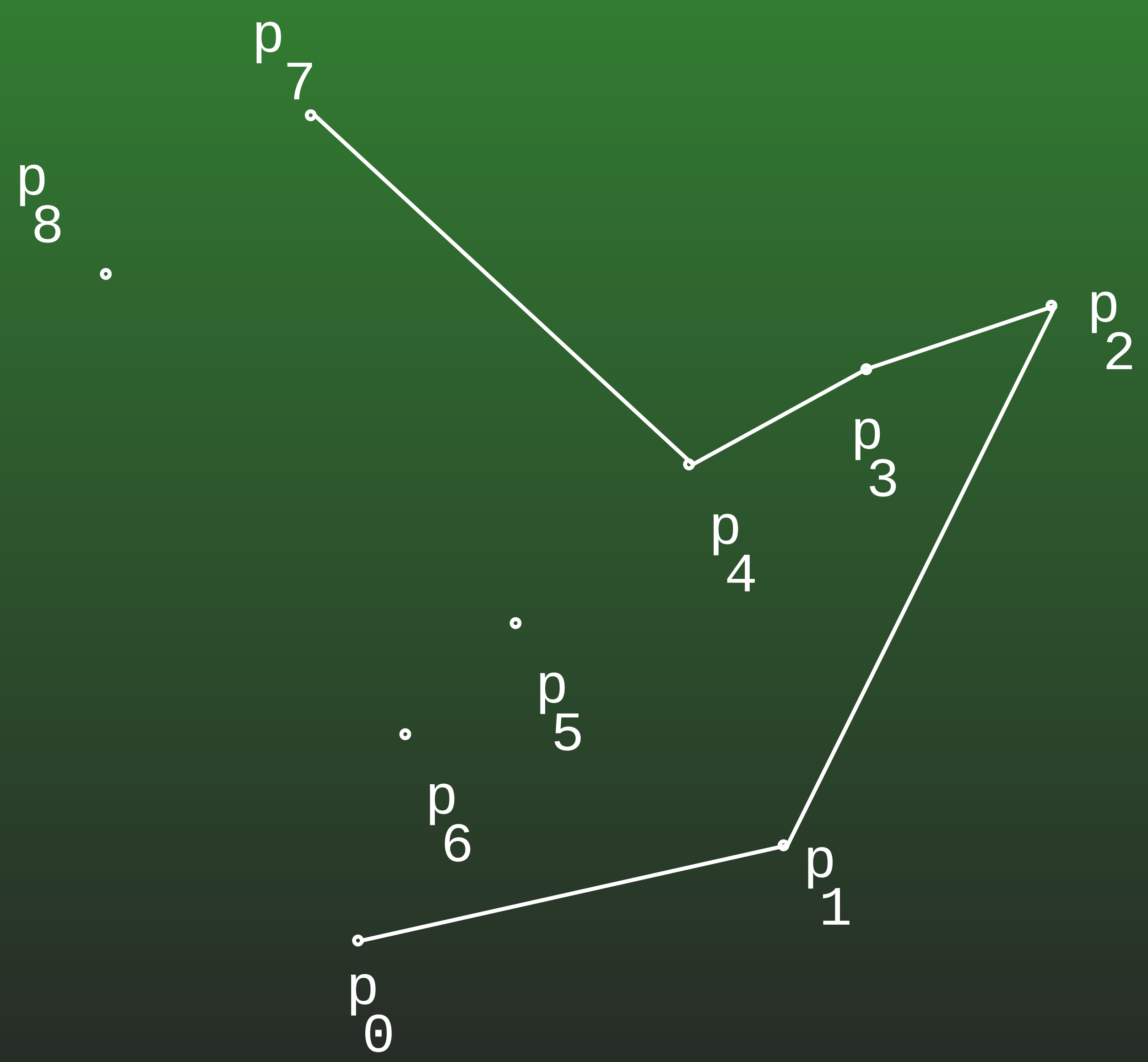
Graham's Scan



Graham's Scan



19-54: Graham's Scan



p_7

p_4

p_3

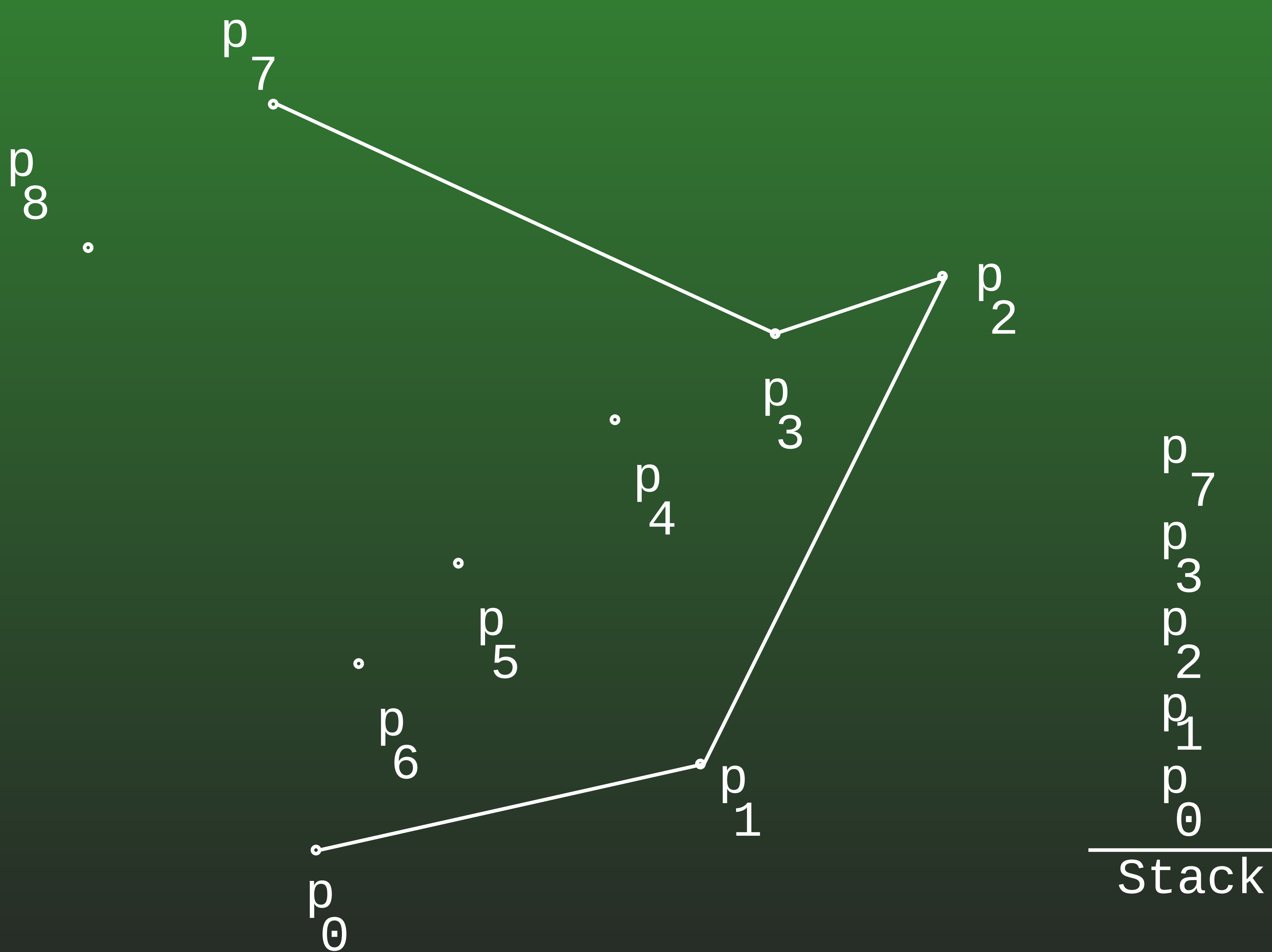
p_2

p_1

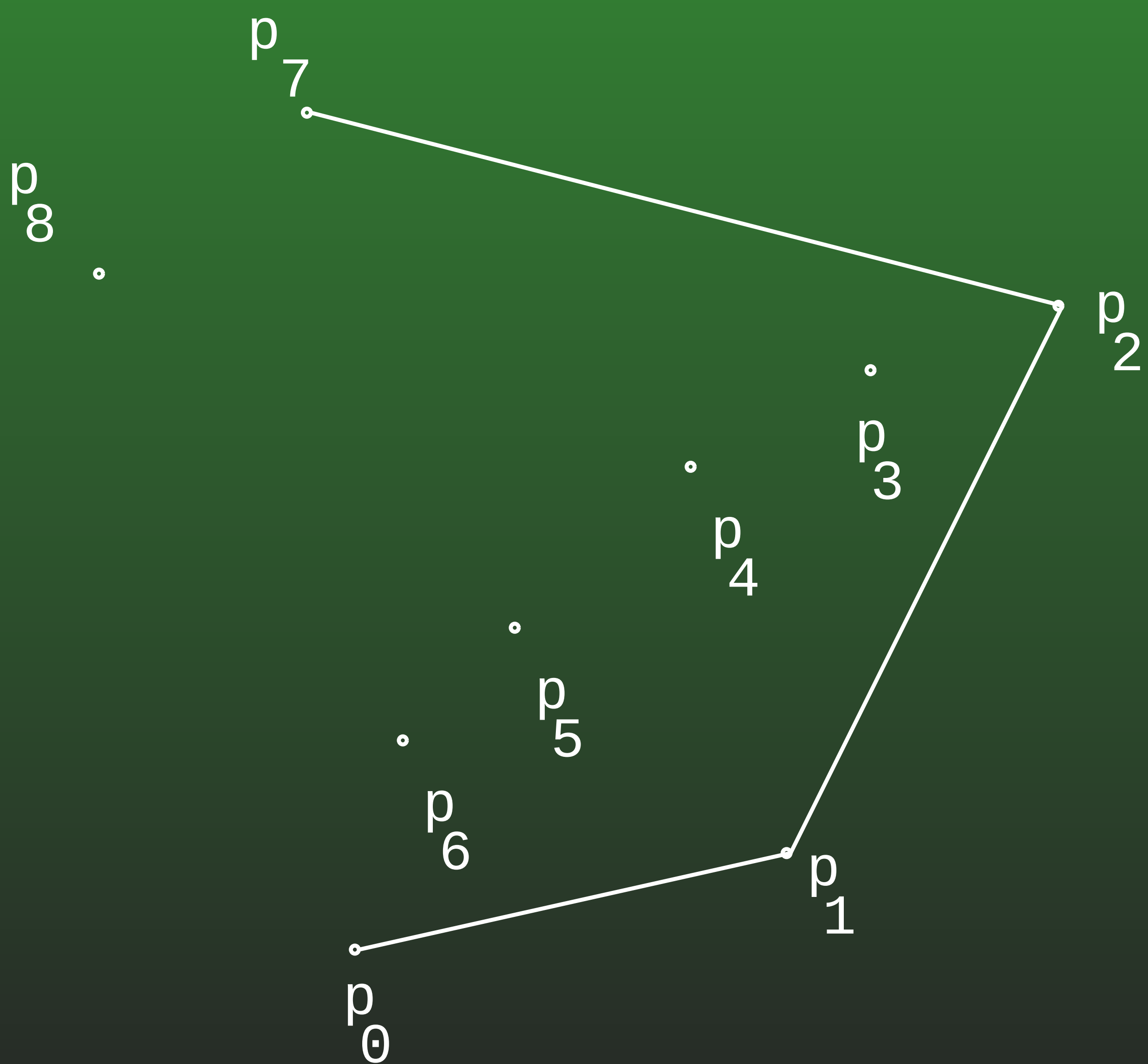
p_0

Stack

Graham’s Scan



Graham's Scan



p 7
p 2
p 1
p 0

Stack

Graham’s Scan

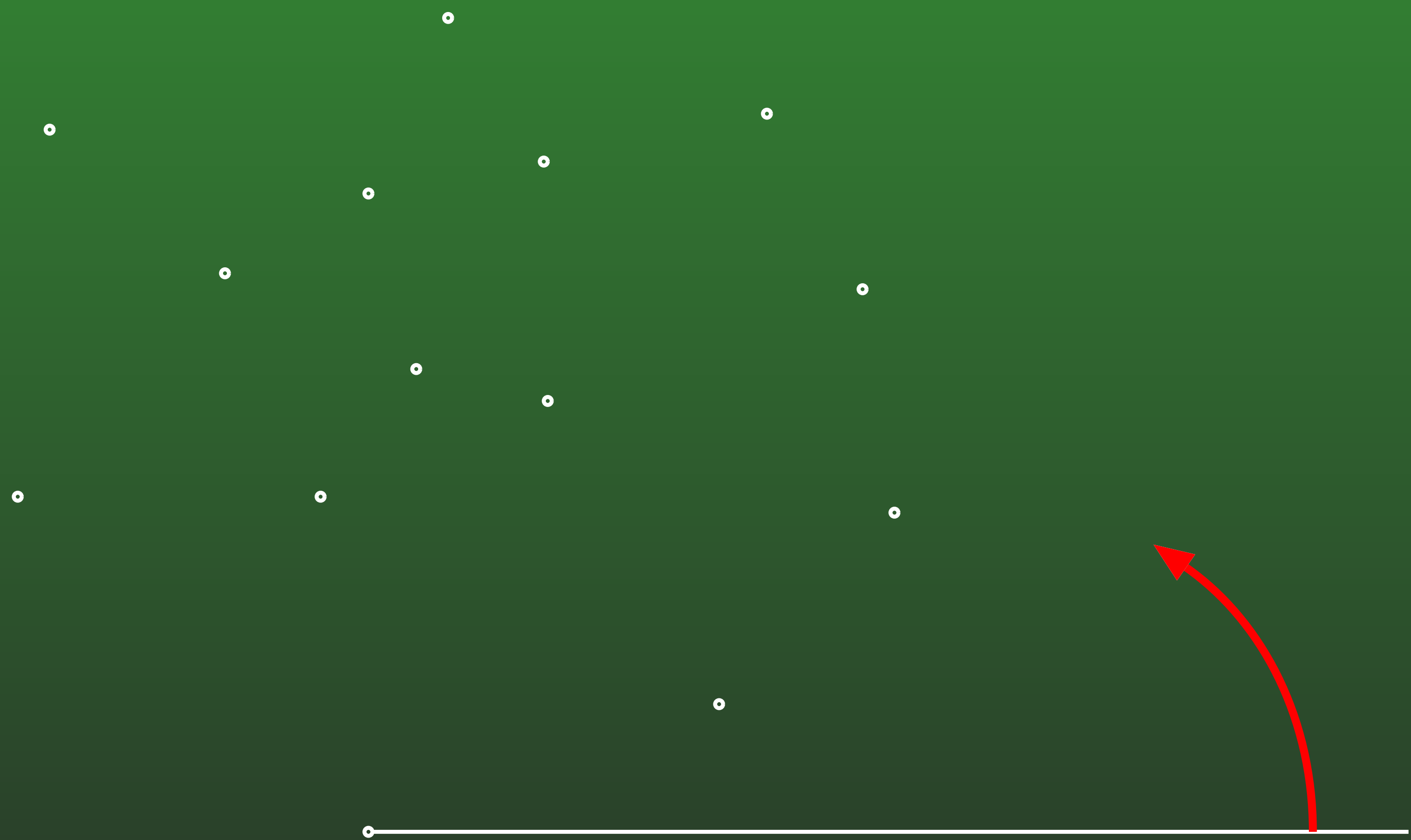
- Time required:
  - $O(n \lg n)$ to sort points by polar degree
    - Note that you don’t need to calculate the polar degree, just determine if one vector is clockwise or counterclockwise of another – can be done with a single cross product
  - Each element is added to the stack once, and removed at most once (each taking constant time) for a total time of $O(n)$
  - Total: $O(n \lg n)$

19-58: Convex Hull

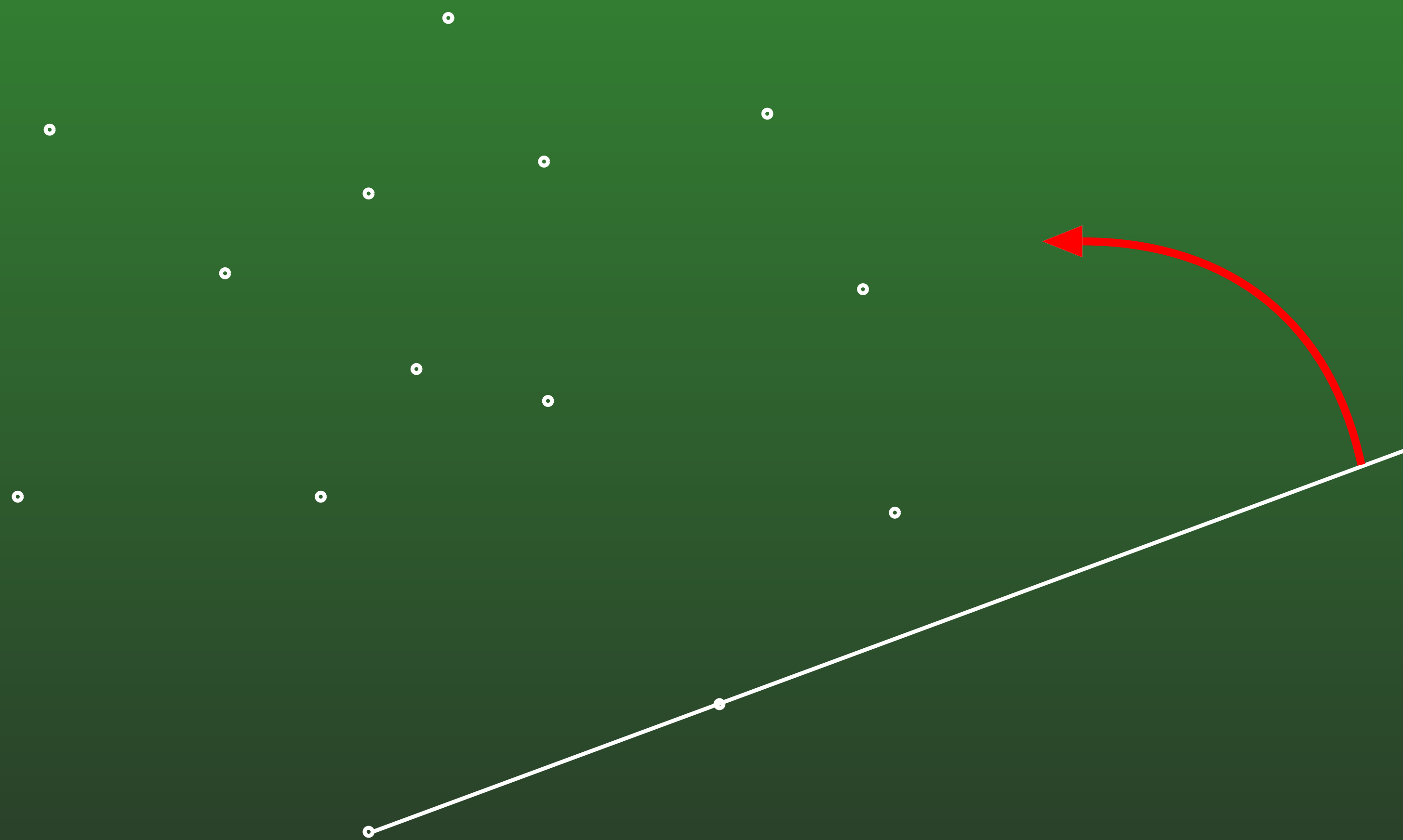
- Different Convex Hull algorithm
- Idea:
  - Attach a string to the lowest point
  - Rotate string counterclockwise, unti it hits a point – this point is in the Convex Hull
  - Keep going until the highest point is reached
  - Continue around back to initial point

19-59: Jarvis's March

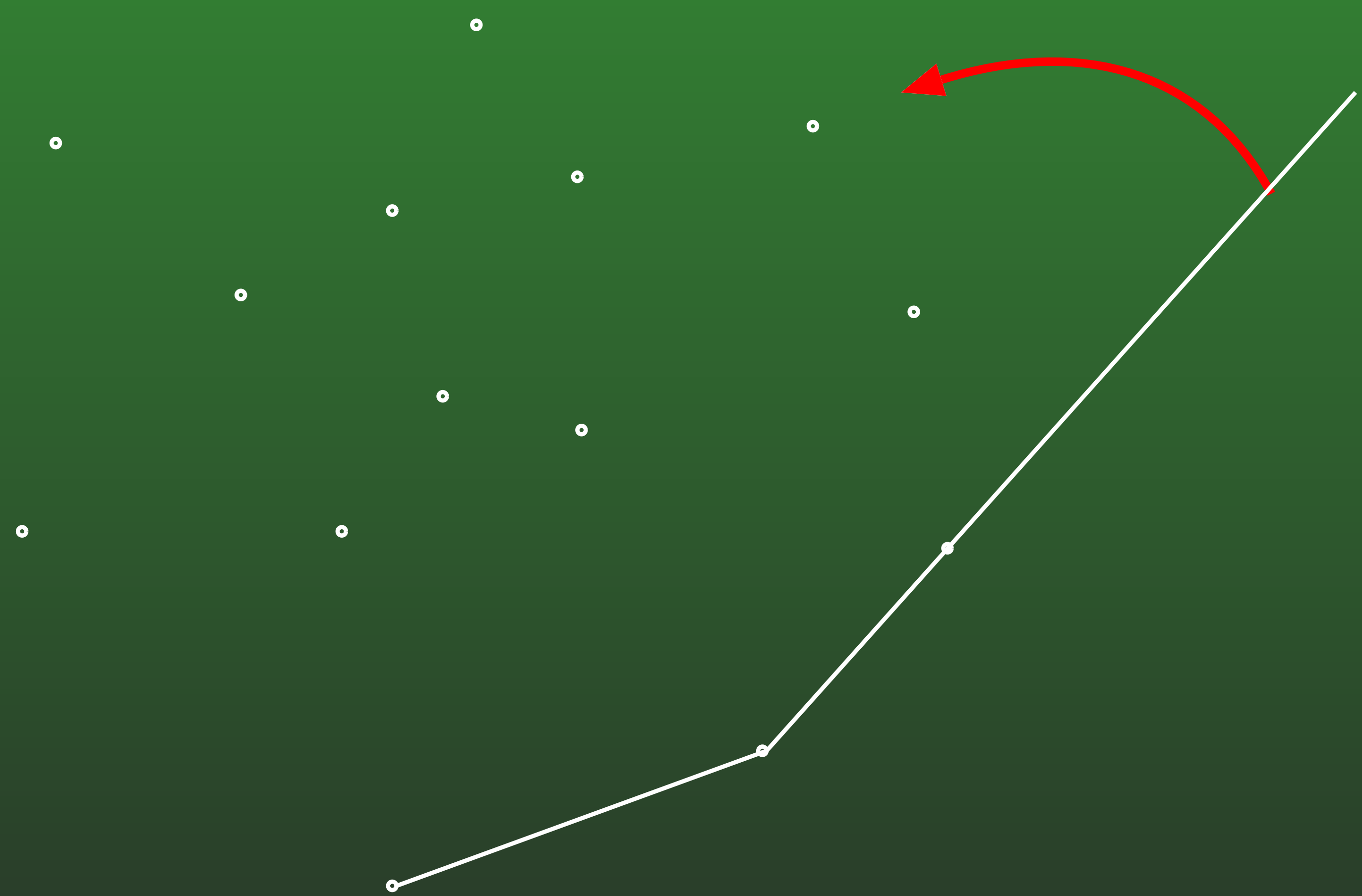
 # Jarvis's March



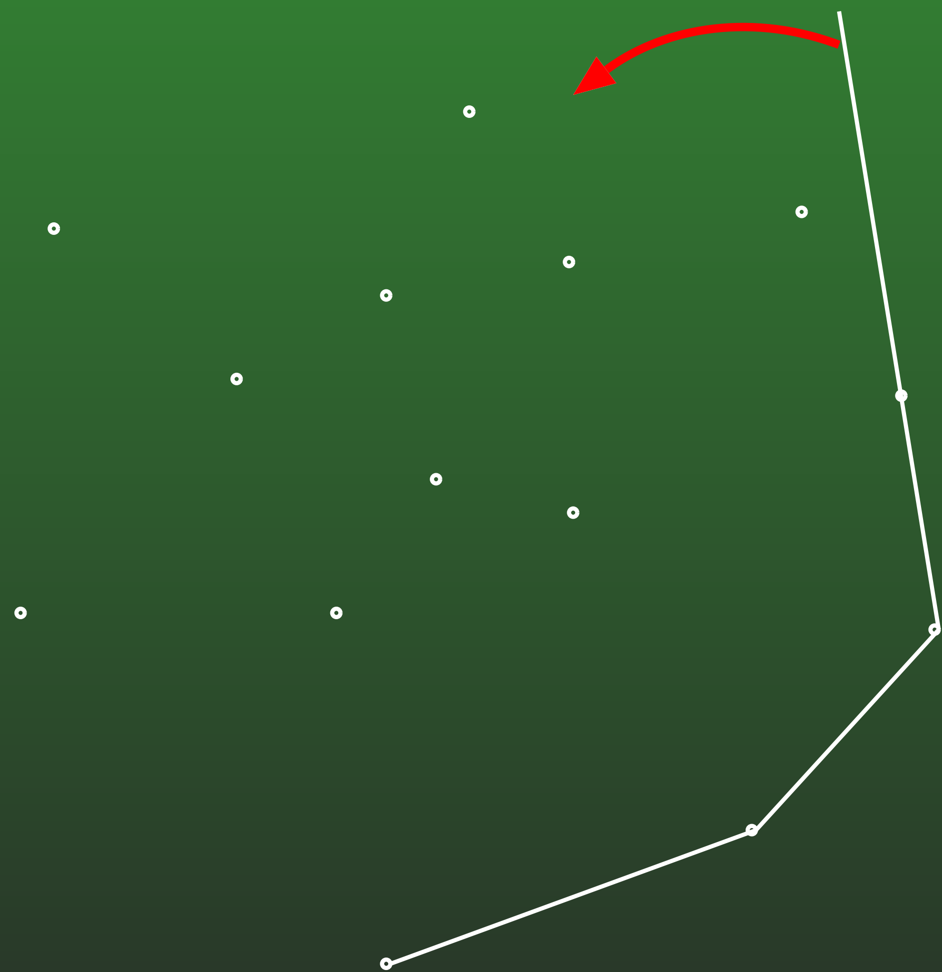
19-61: Jarvis's March



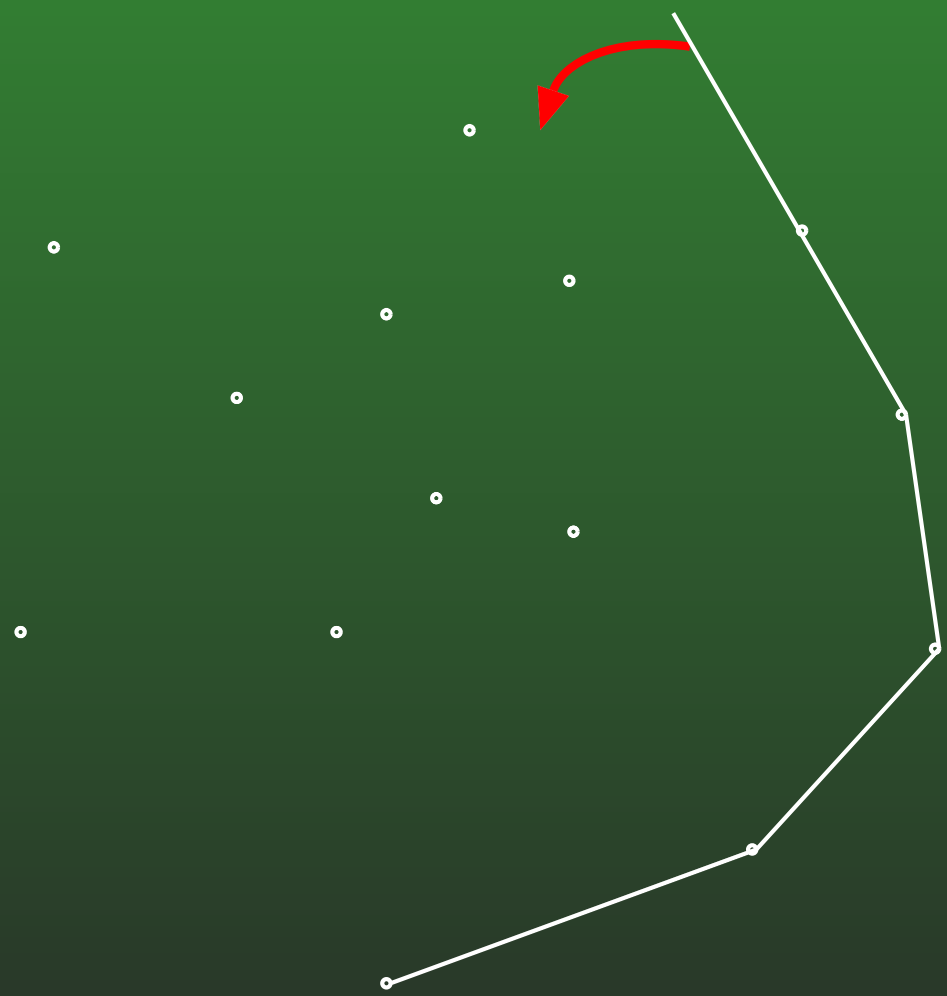
Jarvis's March



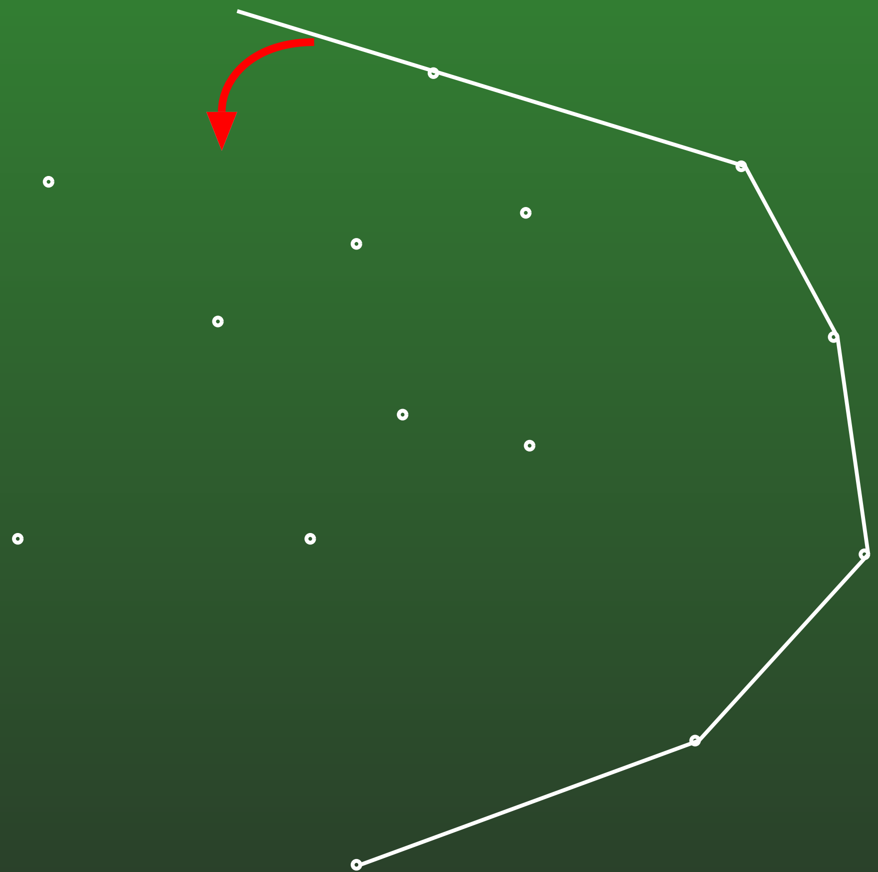
19-63: Jarvis's March



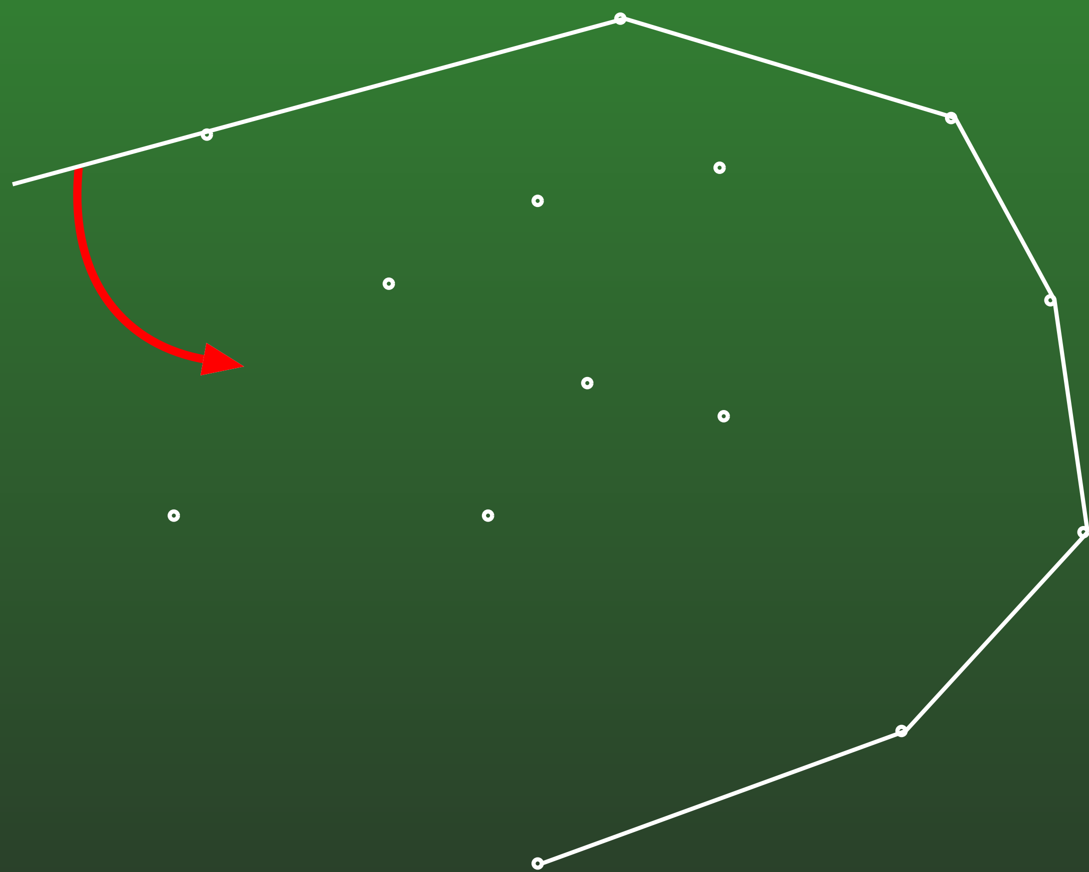
19-64: Jarvis's March



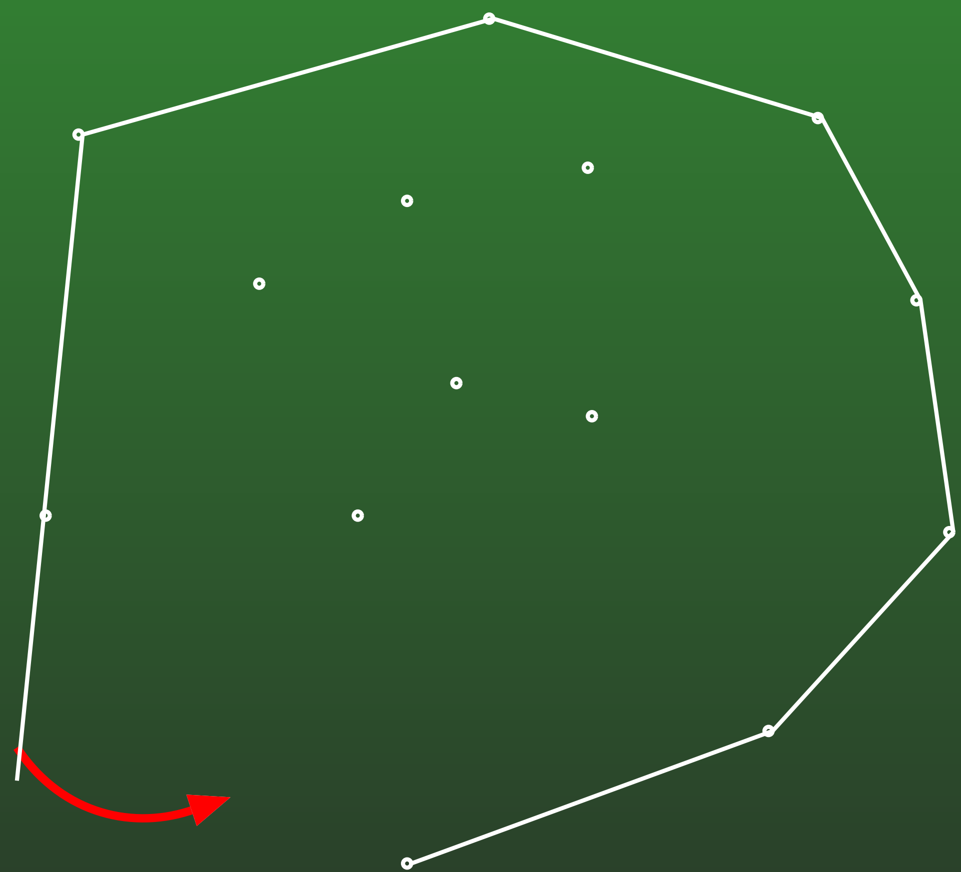
Jarvis's March



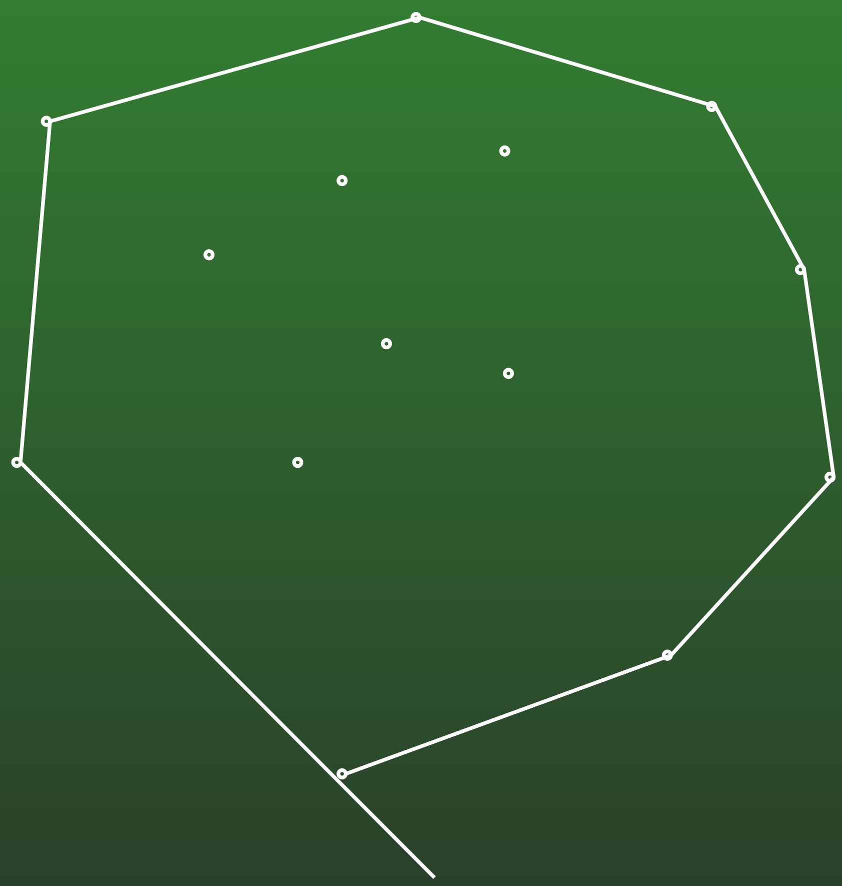
Jarvis's March



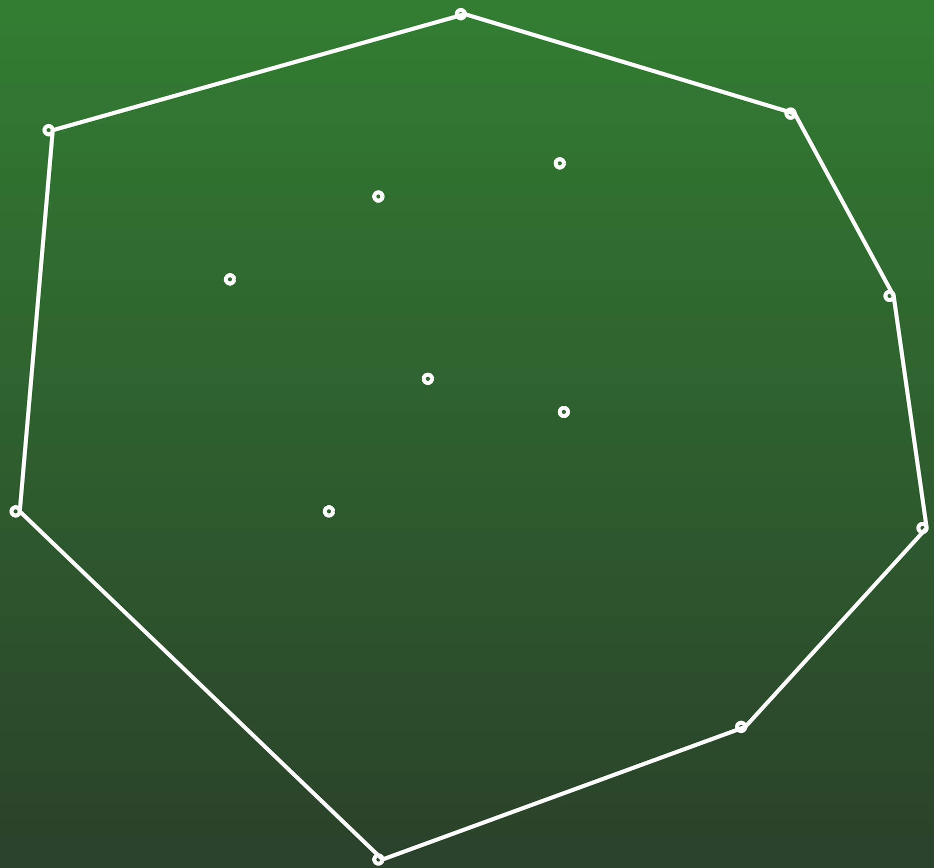
Jarvis's March



19-68: Jarvis's March



19-69: Jarvis's March



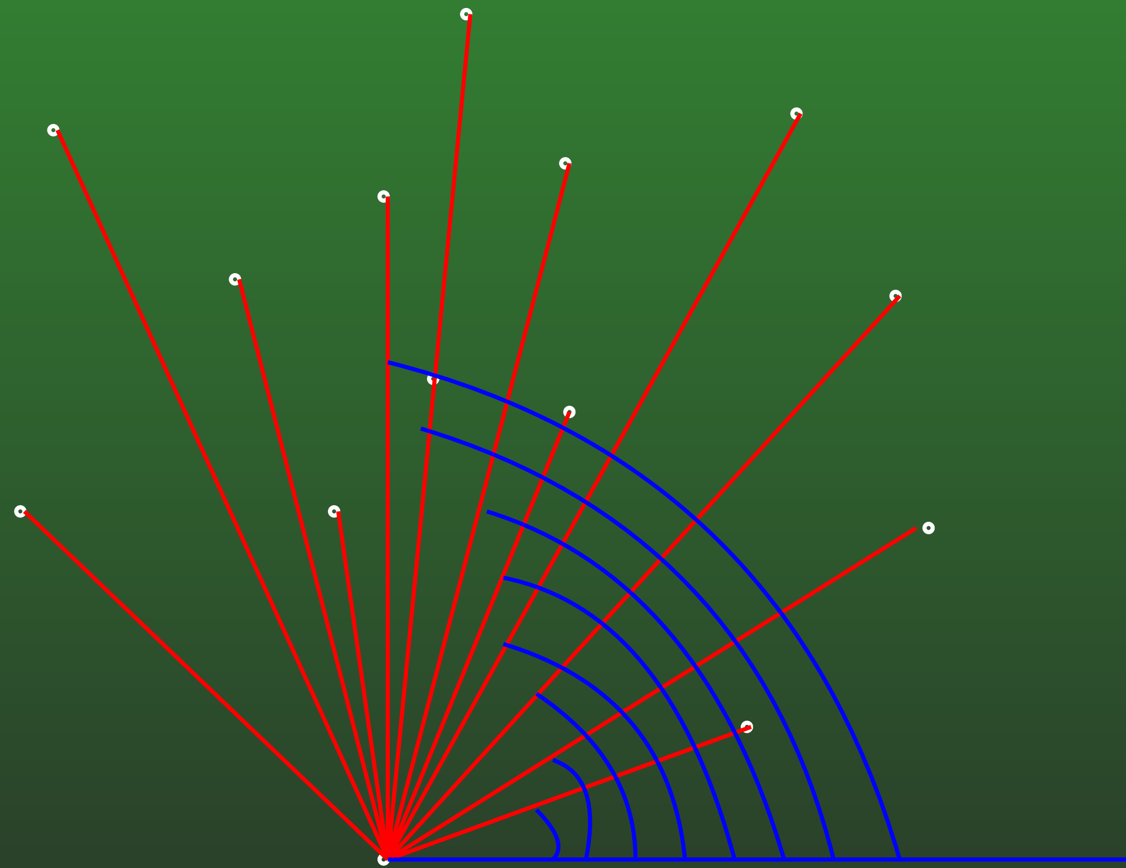
19-70: Jarvis's March

- How do we determine which point wraps next?
  - When we're going from lowest to highest point, the smallest polar angle between previous point and the next point
  - When going from highest point back to lowest point, smallest polar angle (from negative)

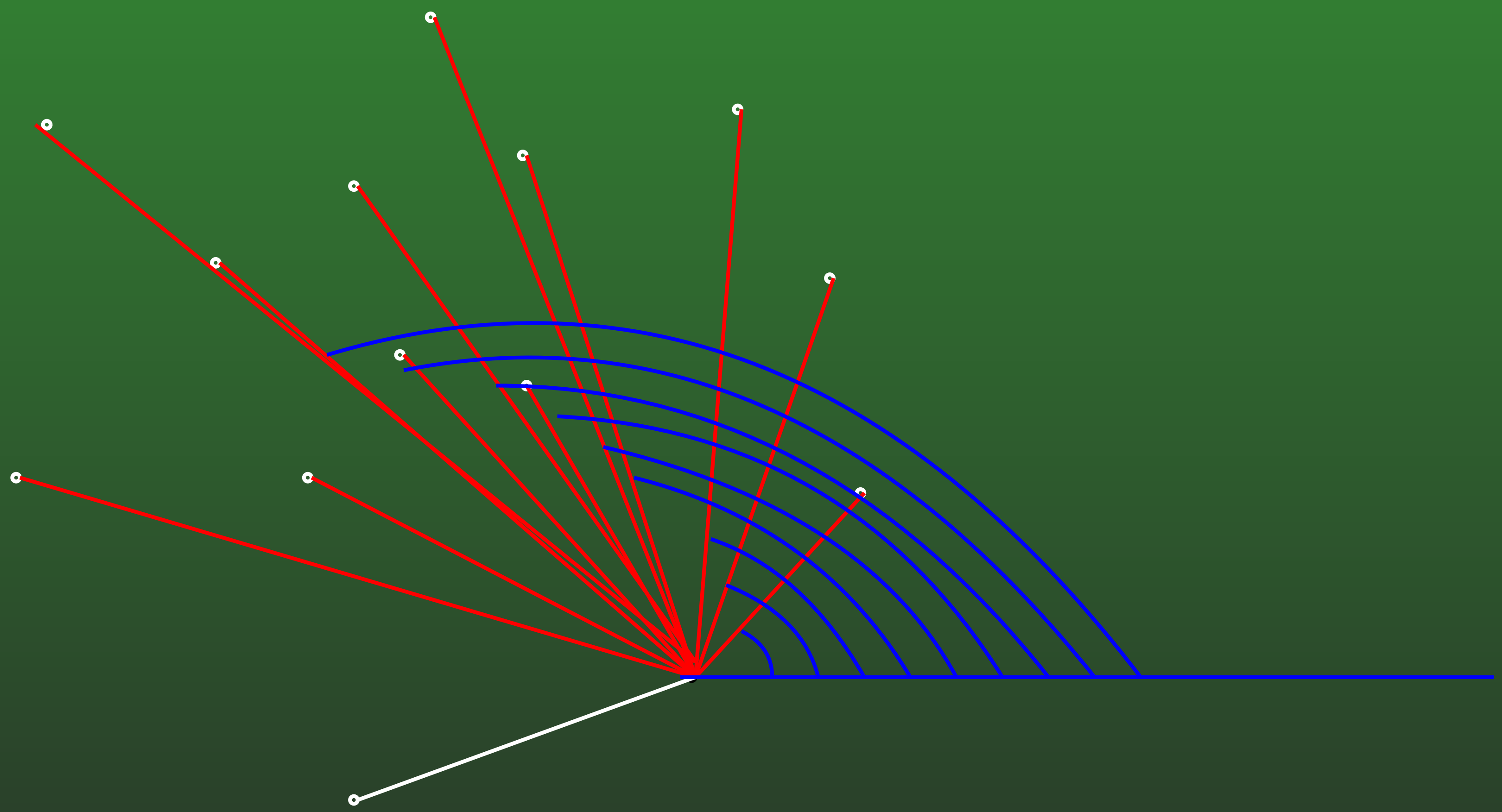
19-71: Jarvis's March

 Jarvis’s March

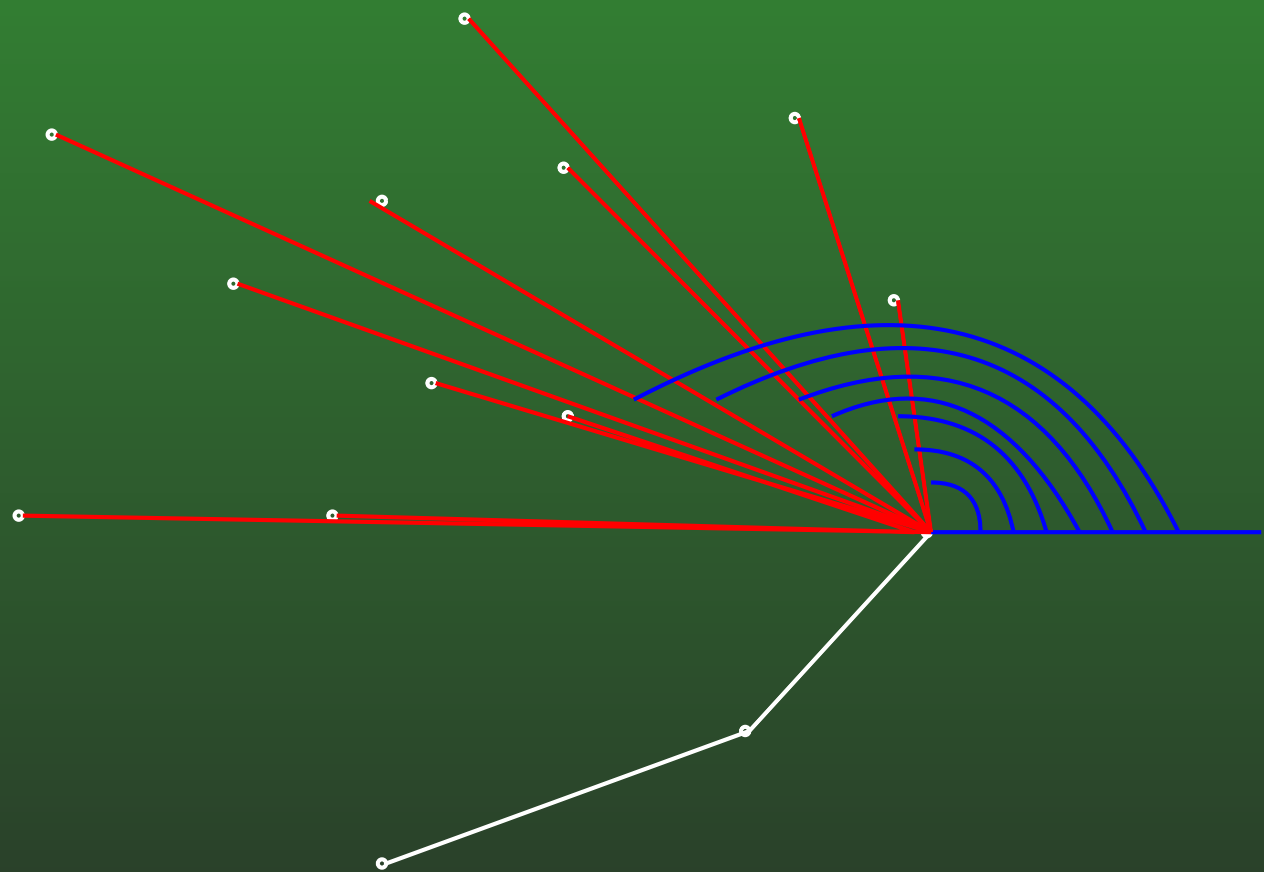
19-73: Jarvis's March



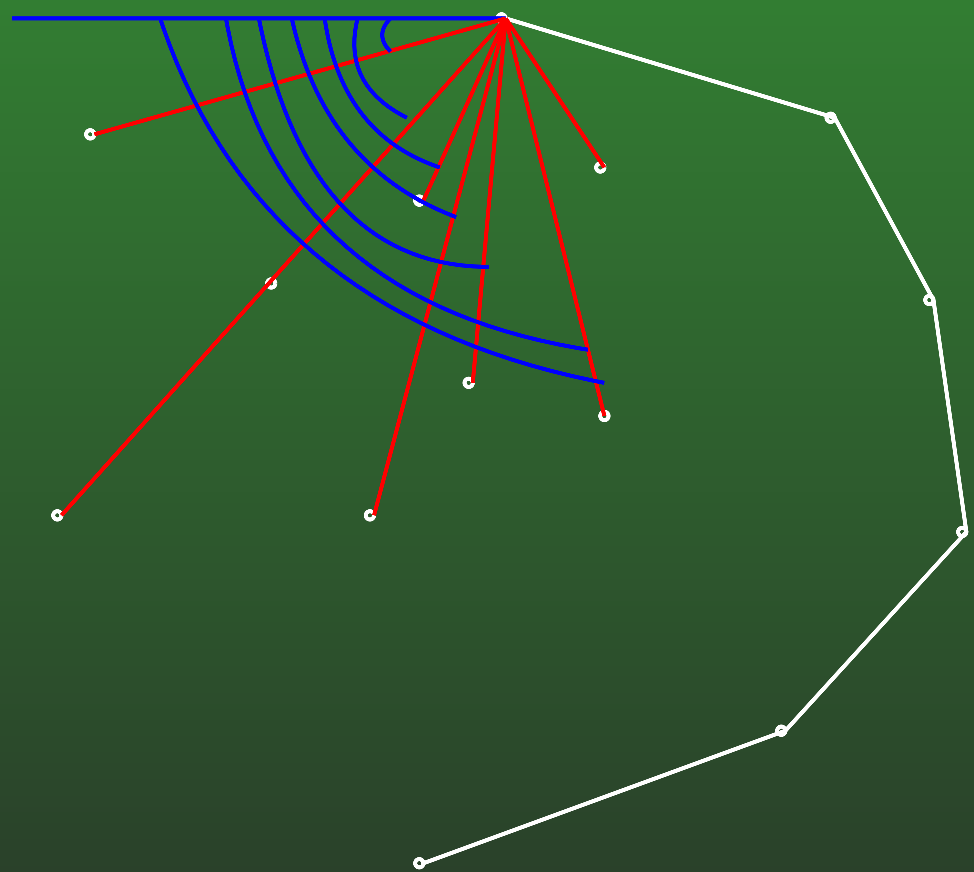
19-74: Jarvis's March



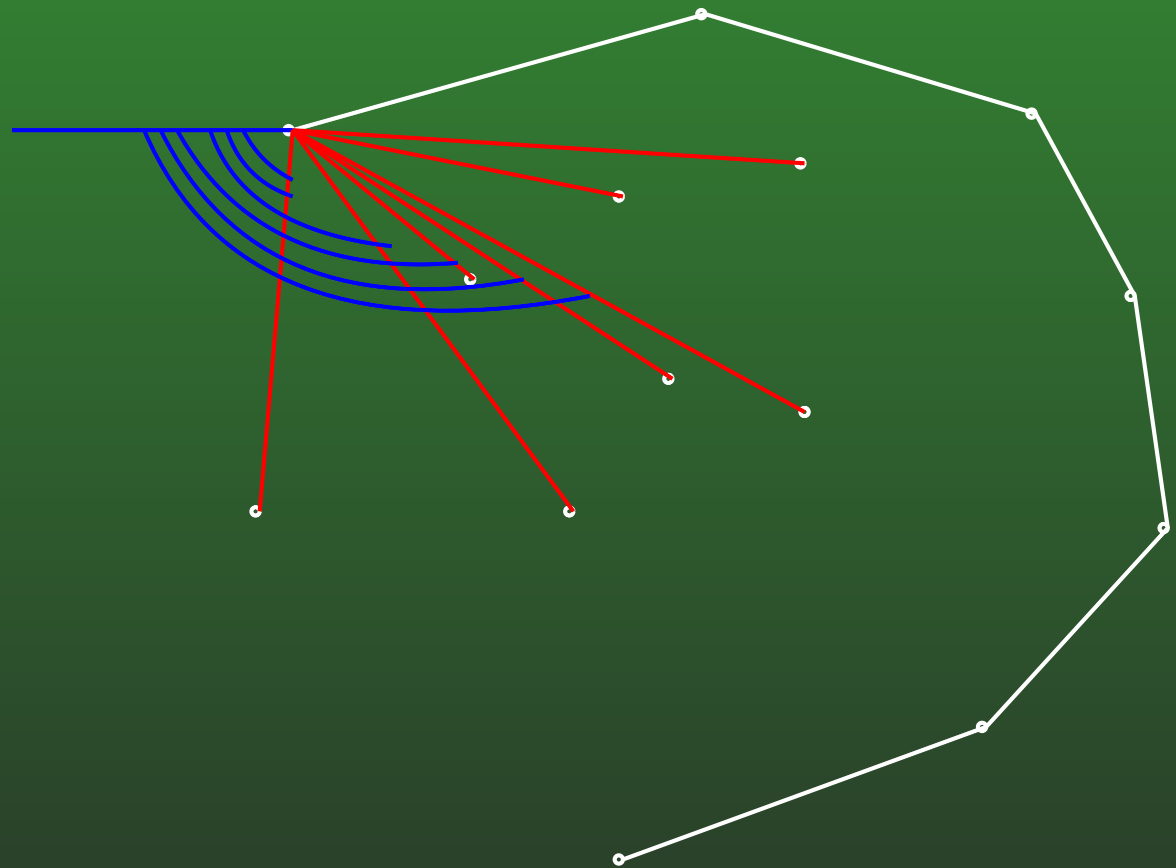
19-75: Jarvis's March



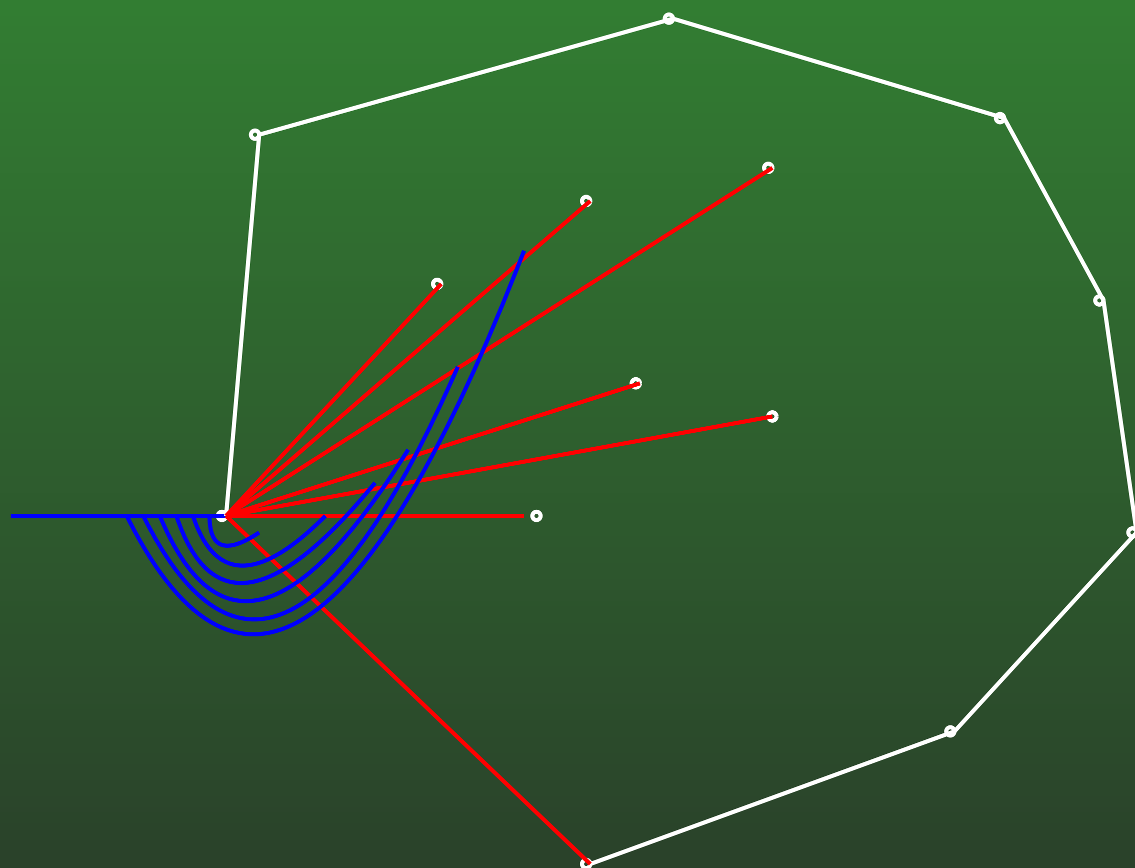
Jarvis's March



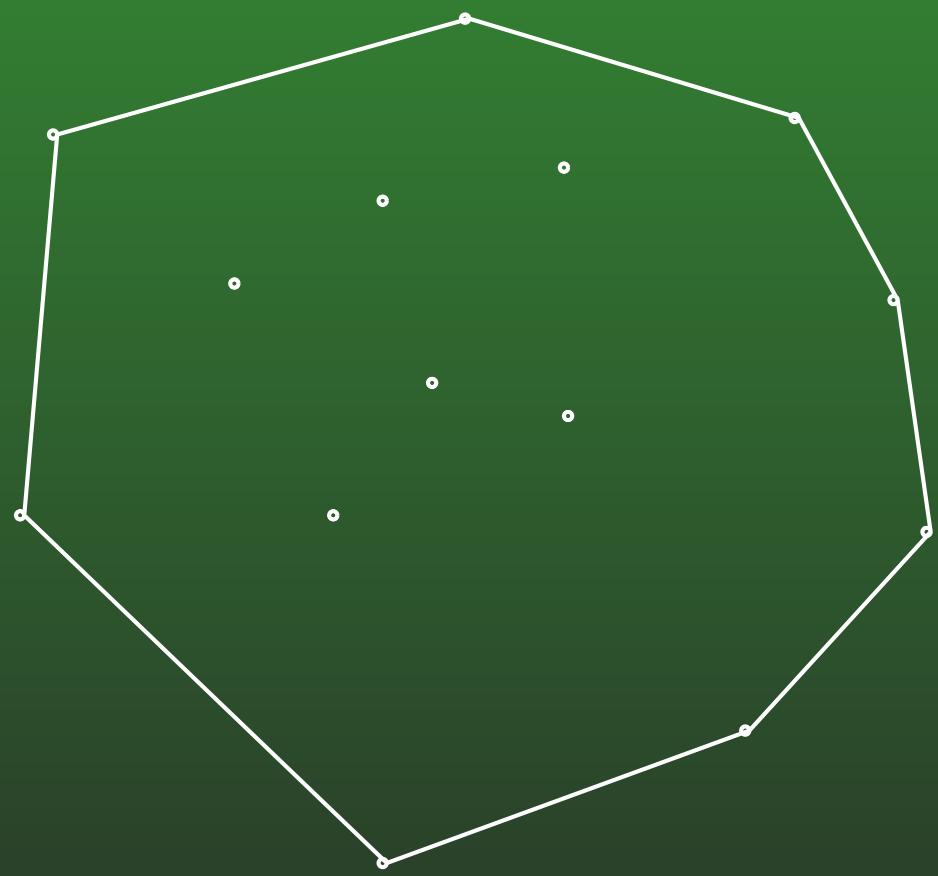
19-77: Jarvis's March



19-78: Jarvis's March



 # Jarvis's March



Jarvis's March

- We don't need to actually compute polar angles
  - We just need to compare them, which can be done with a cross product
- From point p_k, comparing angles from p_i and p_j (going up)
  - Is $\overline{p_k p_i}$ clockwise of $\overline{p_k p_j}$?
  - (is $(p_i - p_k) \times (p_j - p_k)$ positive)?

Jarvis's March

- Time for Jarvis's march:
  - For each vertex in the convex hull, we need to look at up to n other vertices to find the next vertex in the convex hull.
  - Total time: $O(nh)$, where h is the number of vertices in the convex hull
  - Is this better or worst than Graham's Scan

Closest Pair of Points

- We have a large number of points $p_1, \ldots, p_n$
- Want to determine which pair of points p_i, p_j is closest together
- How long would a brute force solution take?
- Can you think of another way?

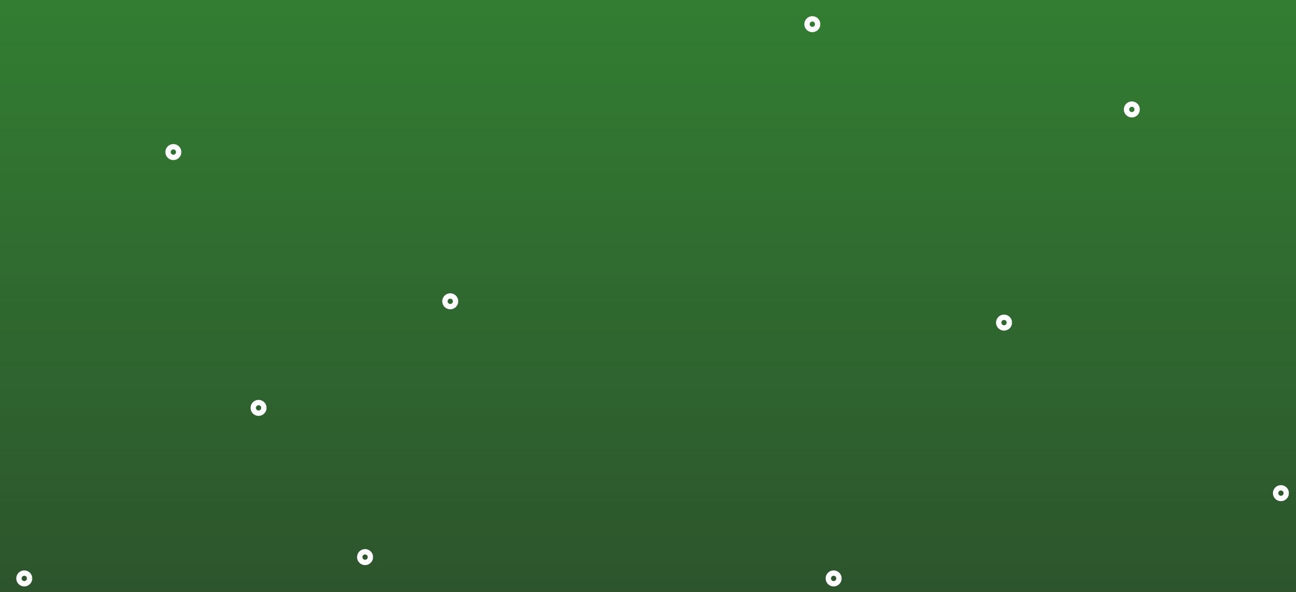
19-83: Closest Pair of Points

- Divide & Conquer
  - Divide the list points in half (by a vertical line)
  - Recursively determine the closest pair in each half
  - . . . and then what?

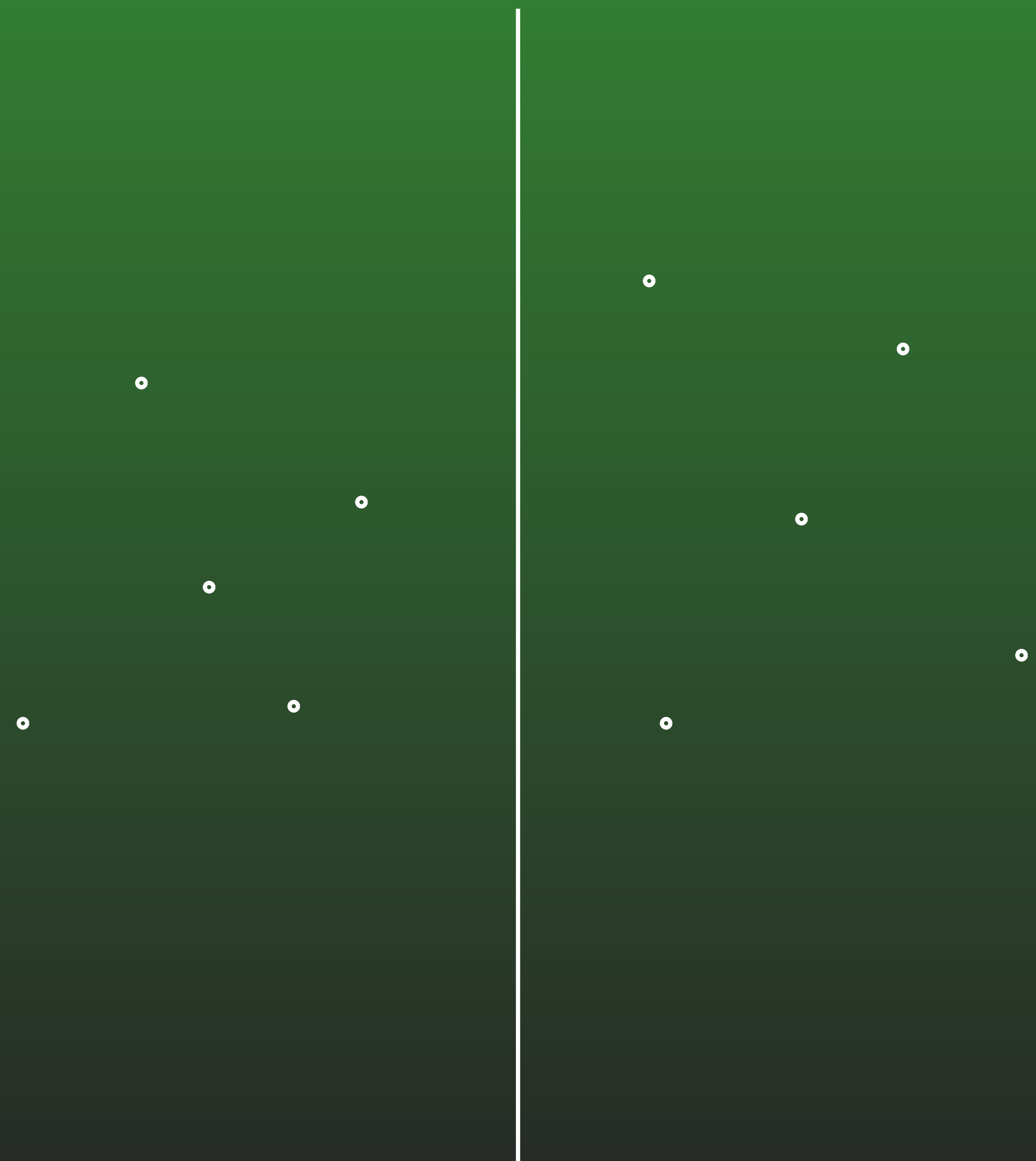
Closest Pair of Points

- Divide & Conquer
  - Divide the list points in half (by a vertical line)
  - Recursively determine the closest pair in each half
  - Smallest distance between points is the minimum of:
    - Smallest distance in left half of points
    - Smallest distance in right half of points
    - Smallest distance that crosses from left to right

Closest Pair of Points



 Closest Pair of Points



Closest Pair of Points

- To find smallest distance that crosses from left to right:
  - If we compare all $\frac{n}{2}$ elements in the left sublist with all $\frac{n}{2}$ elements in the right sublist, how much time would that take?

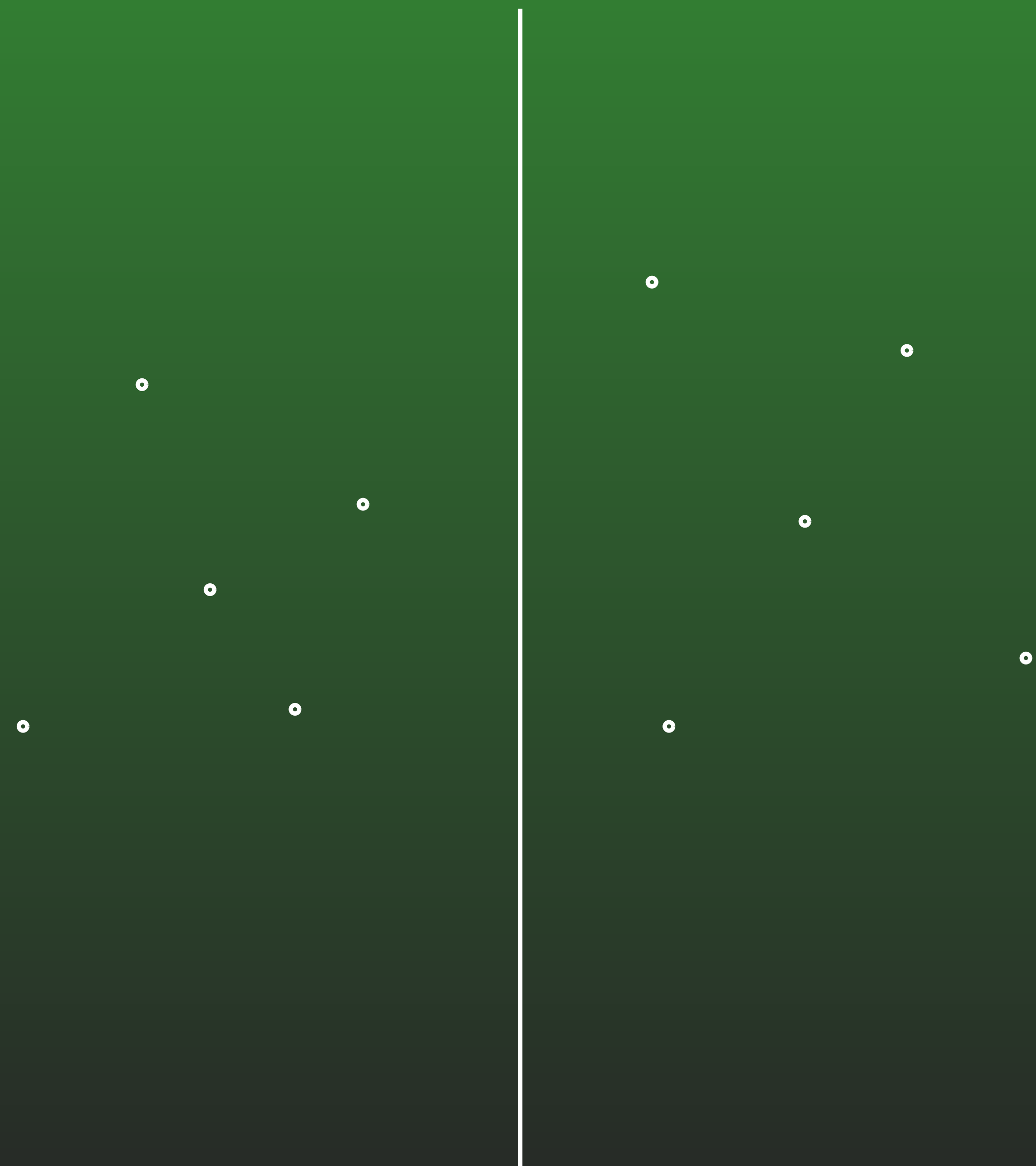
Closest Pair of Points

- To find smallest distance that crosses from left to right:
  - If we compare all $\frac{n}{2}$ elements in the left sublist with all $\frac{n}{2}$ elements in the right sublist, how much time would that take?
  - $\Theta(n^2)$, no better than brute force solution!

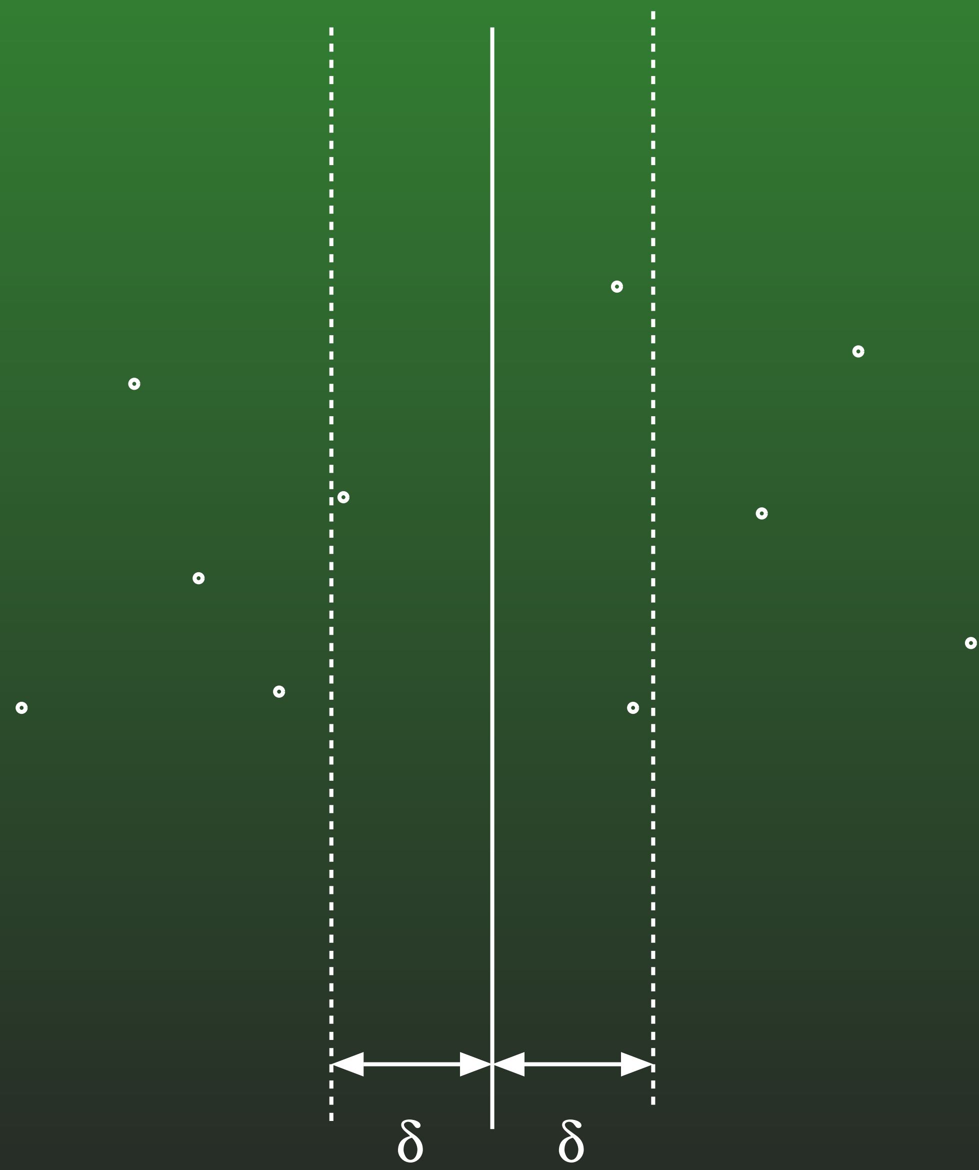
Closest Pair of Points

- To find smallest distance that crosses from left to right:
  - Let δ be the smallest distance in the two sublists
  - Examine only the points that are within δ of the centerline

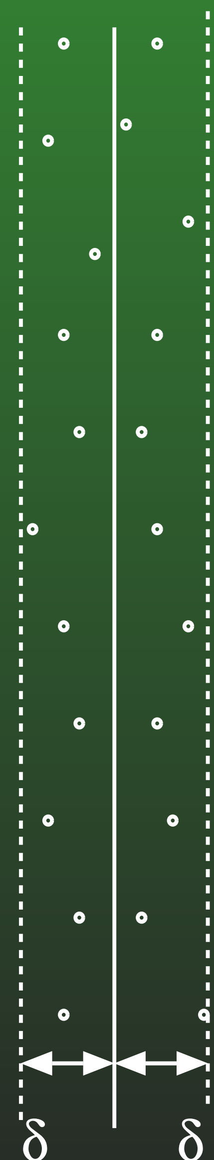
19-90: Closest Pair of Points



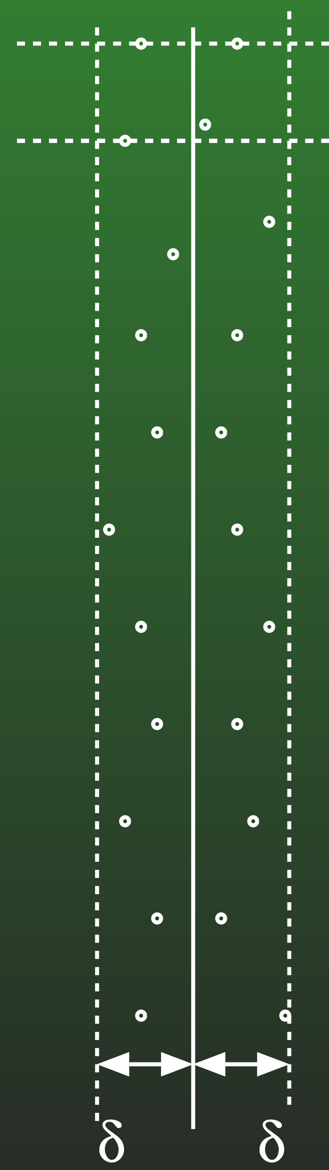
19-91: Closest Pair of Points



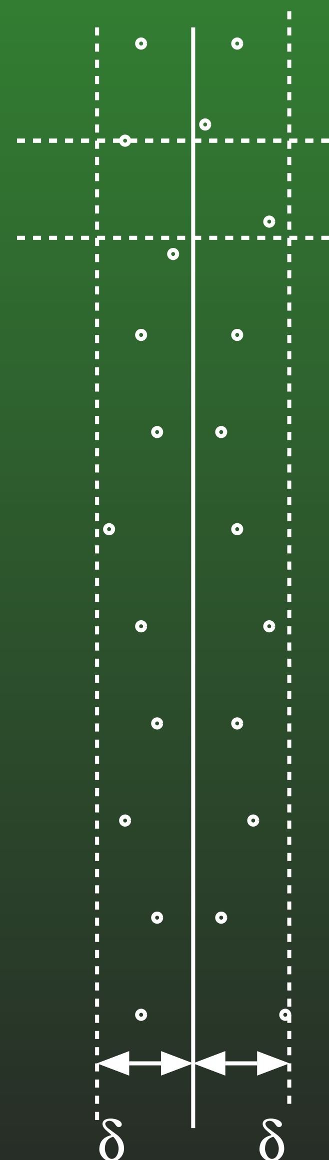
19-92: Closest Pair of Points



Closest Pair of Points

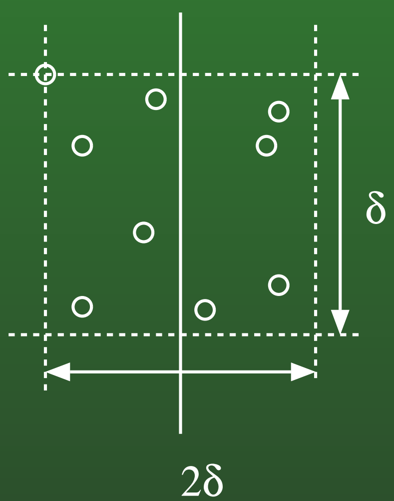


19-94: Closest Pair of Points



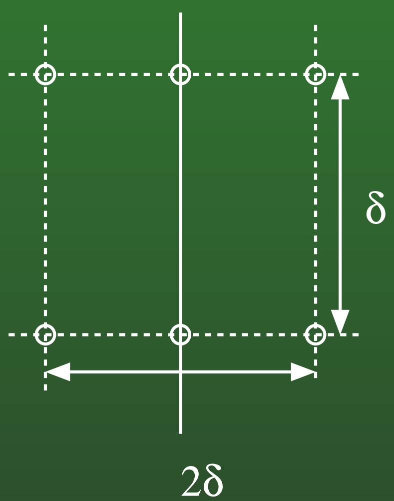
Closest Pair of Points

- How many points can be in the $\delta \times 2\delta$ rectangle?



19-96: Closest Pair of Points

- How many points can be in the $\delta \times 2\delta$ rectangle?



Closest Pair of Points

- Create two lists of the points:
  - One sorted by x-coordinate
  - One sorted by y-coordinate
- Call Find-Smallest using these two lists
  - Find-Smallest(XList,YList)

Closest Pair of Points

FindSmallest(L_x, L_y)
    if $|L_x| \leq 3$
        do brute force search on 3 points
    Split list L_x in half
        Put first 1/2 in L_{XL}
        Put second 1/2 in L_{XR}
    Split list L_Y in half
    For each point p in L_y:
        If $p \in L_{XL}$, put p in L_{YL}
        If $p \in L_{XR}$, put p in L_{YR}
    $\delta \leftarrow$ FindSmallest(L_{XL}, L_{YL})
    $\delta \leftarrow$ Min(δ, FindSmallest(L_{XR}, L_{YR}))
    $\delta \leftarrow$ FindSmallestAcross(L_{YR}, L_{YR}, δ)
    return δ

19-99: Closest Pair of Points

- Time:
  - Sorting: $O(n \lg n)$ using mergesort
  - Recursive call:

$$
\begin{aligned}
T(1) &= T(2) = T(3) = c_1 \\
T(n) &= 2T(n/2) + c_2 * n
\end{aligned}
$$

    - $\Theta(n \lg n)$ by the Master Method
  - Total time: $O(n \lg n)$