CS 2009
Design and Analysis of Algorithms

Waheed Ahmed

Farrukh Salim Shaikh

Geometric Algorithms

Thomas H. Coreman (CLRS), Chapter 33.

Geometric Algorithms

- Applications of **Geometric Algorithms.**
  - Computer vision (Detecting Edge, Corner, Different Shapes)
  - Data mining (Clustering Different Data Point)
  - VLSI Design
  - Mathematical Models
  - Computer graphics (movies, games, virtual reality).

Geometric operations

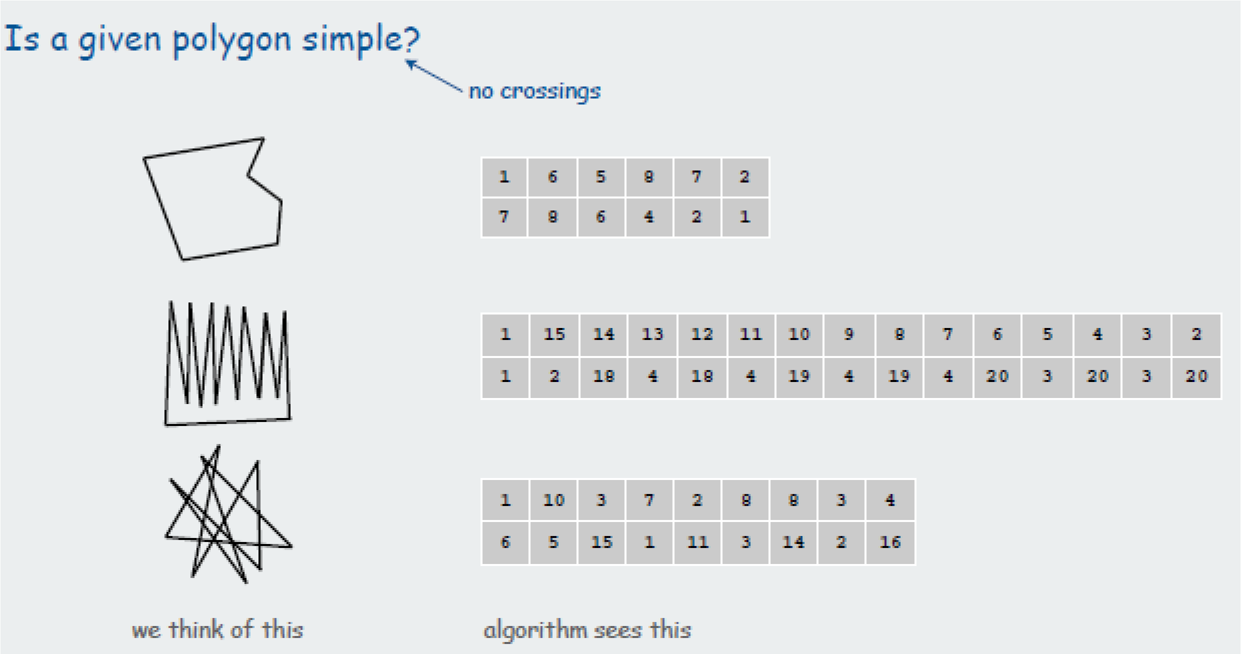
- Point: two numbers (x, y).
- Line segment: two points.
- Polygon: sequence of points.

Primitive operations

- Is a point inside a polygon?
- Compare slopes of two lines
- Do two line segments intersect?
- Given three points p1, p2, p3, is p1-p2-p3 a counterclockwise turn?

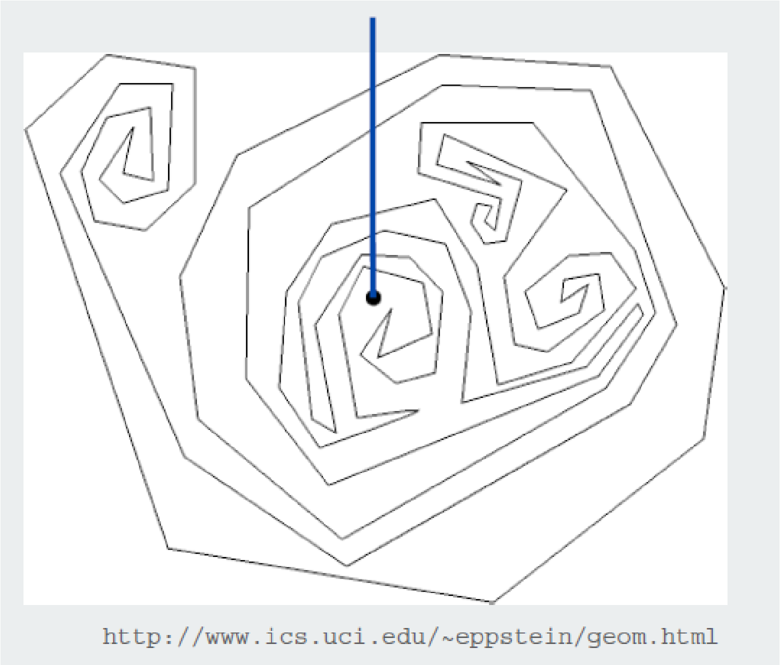
Warning: Intuition may be misleading

- Humans have spatial intuition in 2D and 3D, Computers do not.
- Neither has good intuition in higher dimensions!



Polygon Inside, Outside

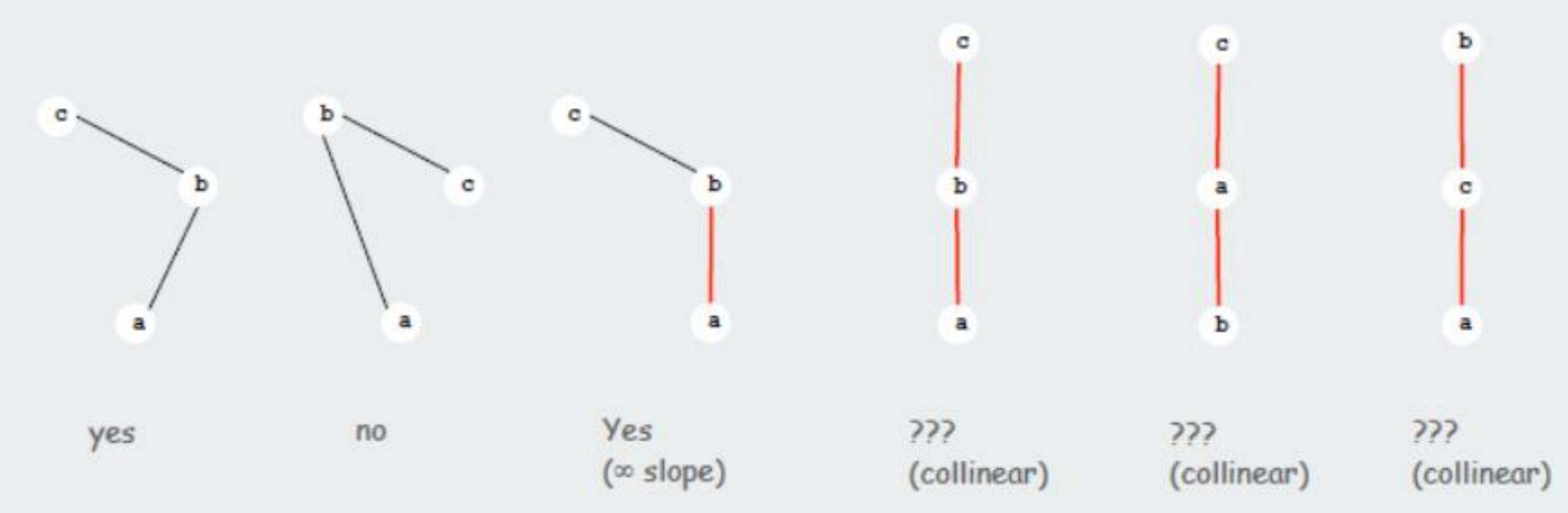
Is a point inside a simple polygon?



Implementing counterclockwise (CCW)

CCW. Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Idea: Compare Slope.



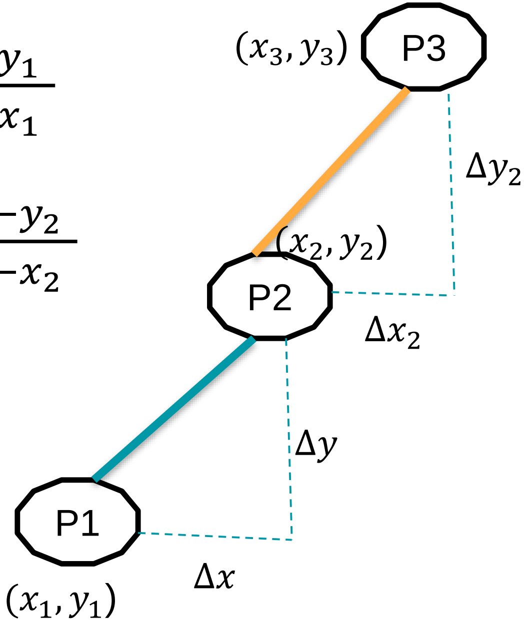
Implementing counterclockwise (CCW)

CCW. Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Idea: Compare Slope.

$$\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\frac{\Delta y_2}{\Delta x_2} = \frac{y_3 - y_2}{x_3 - x_2}$$



$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} > 0$$ Counterclockwise

$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} < 0$$ Clockwise

$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} = 0$$ Collinear

Implementing counterclockwise (CCW)

CCW. Given three point a, b, and c, is a-b-c a counterclockwise turn?

- Idea: Compare Slope.

$$(y_3 - y_2)\,(x_2 - x_1) - (y_2 - y_1)(x_3 - x_2) > 0$$ Counterclockwise

$$(y_3 - y_2)\,(x_2 - x_1) - (y_2 - y_1)(x_3 - x_2) < 0$$ Clockwise

$$(y_3 - y_2)\,(x_2 - x_1) - (y_2 - y_1)(x_3 - x_2) = 0$$ Collinear

$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} > 0$$ Counterclockwise

$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} < 0$$ Clockwise

$$\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} = 0$$ Collinear

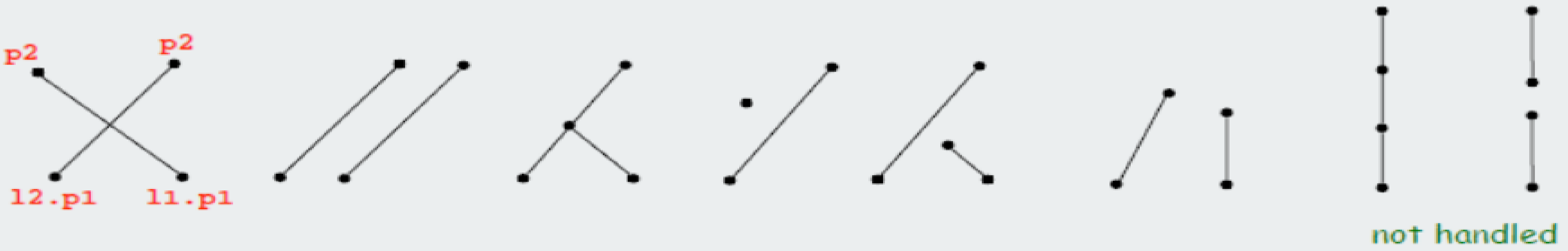
Immutable Point ADT

```
public final class Point
{
   public final int x;
   public final int y;

   public Point(int x, int y)
   {  this.x = x; this.y = y;  }


   public double distanceTo(Point q)
   {  return Math.hypot(this.x - q.x, this.y - q.y);  }

   public static int ccw(Point a, Point b, Point c)
   {
      double area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
      if else (area2 < 0) return -1;
      else if (area2 > 0) return +1;
      else if (area2 > 0  return  0;
   }

   public static boolean collinear(Point a, Point b, Point c)
   {
      return ccw(a, b, c) == 0;
   }
}
```

Sample ccw client: Line intersection

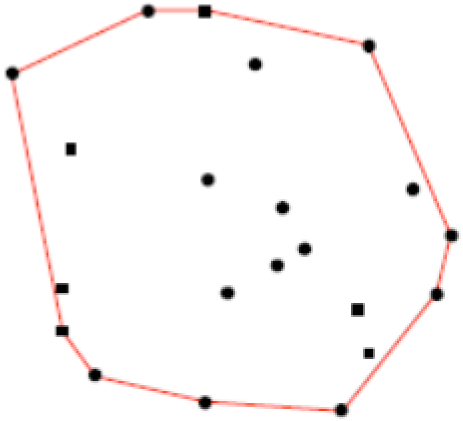
Intersect: Given two line segments, do they intersect?

- Idea 1: find intersection point using algebra and check.
- Idea 2: check if the endpoints of one line segment are on different "sides" of the other line segment.
- 4 ccw computations.



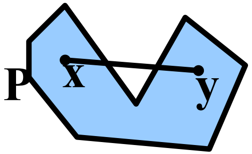

```
public static boolean intersect(Line l1, Line l2)
{
   int test1, test2;
   test1 = Point.ccw(l1.p1, l1.p2, l2.p1)
           * Point.ccw(l1.p1, l1.p2, l2.p2);
   test2 = Point.ccw(l2.p1, l2.p2, l1.p1)
           * Point.ccw(l2.p1, l2.p2, l1.p2);
   return (test1 <= 0) && (test2 <= 0);
}
```

Convex Hull

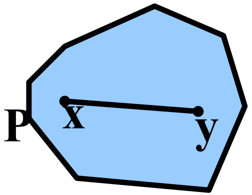


Convex vs. Concave

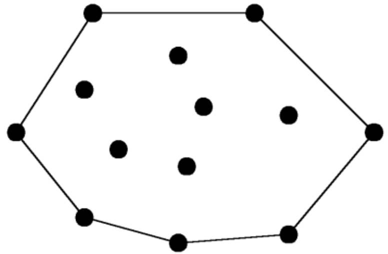
- A polygon P is convex if for every pair of points x and y in P, the line xy is also in P; otherwise, it is called concave.



concave



convex


Convex Hall

- The convex hull of a set of planar points is the smallest convex polygon containing all of the points.

Brute-force algorithm

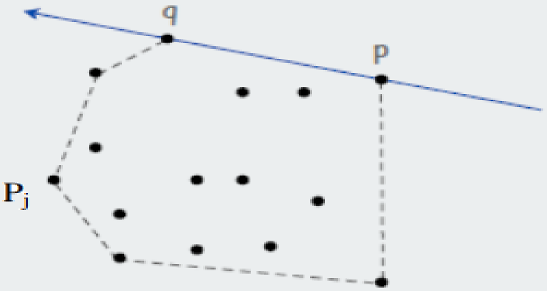
Observation 1.

Edges of convex hull of P connect pairs of points in P.

Observation 2.

p-q is on convex hull if all other points are counterclockwise of $\overrightarrow{pq}$.

for each point P_i
    for each point P_j where $P_j \neq P_i$
        Compute the line segment for P_i and P_j
        for every other point P_k where $P_k \neq P_i$ and $P_k \neq P_j$
            If each P_k is on one side of the line segment, label P_i and P_j in the convex hull



O(N^3) algorithm.

For all pairs of points p and q in P

- compute ccw(p, q, x) for all other x in P
- p-q is on hull if all values positive

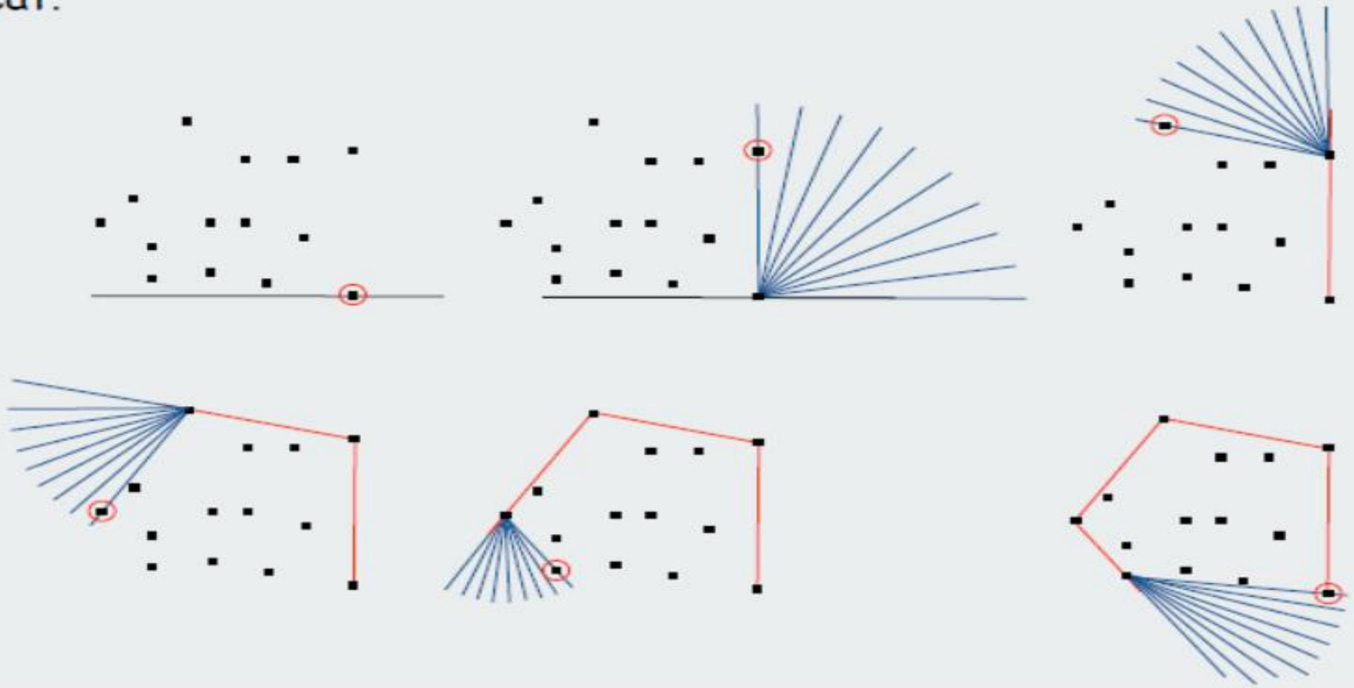
for each of n(n − 1)/2 pairs points, we need to find the sign for each of the other n − 2 points (are all on one side).

Jarvis March

Package Wrap (Jarvis March)

Package wrap.

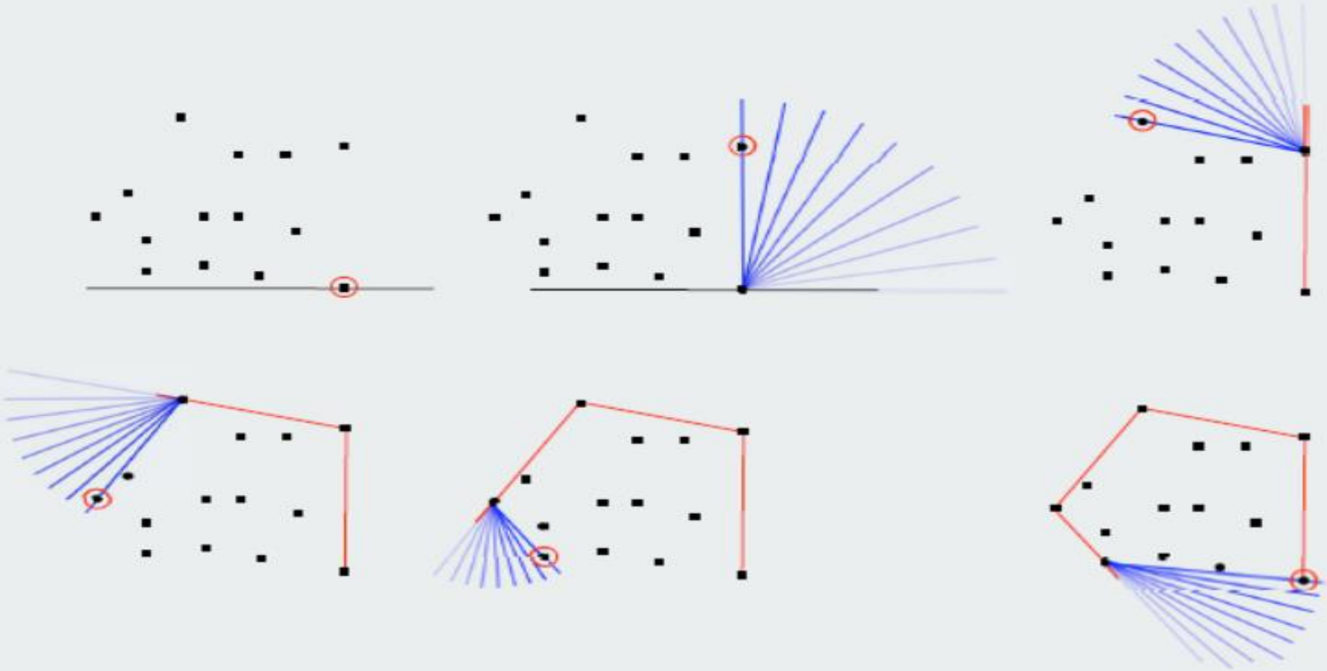
- Start with point with smallest y-coordinate.
- Rotate sweep line around current point in ccw direction.
- First point hit is on the hull.
- Repeat.



Package Wrap (Jarvis March)

Implementation.

- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- $\Theta(N)$ per iteration.



How Many Points on the Hull?

Parameters.

- N = number of points.
- h = number of points on the hull.

Package wrap running time. $\Theta(N\,h)$ per iteration.

How many points on hull?

- Worst case: $h = N$.
- Average case: difficult problems in stochastic geometry.
  - in a disc: $h = N^{1/3}$.
  - in a convex polygon with $O(1)$ edges: $h = \log N$.

Graham's Scan

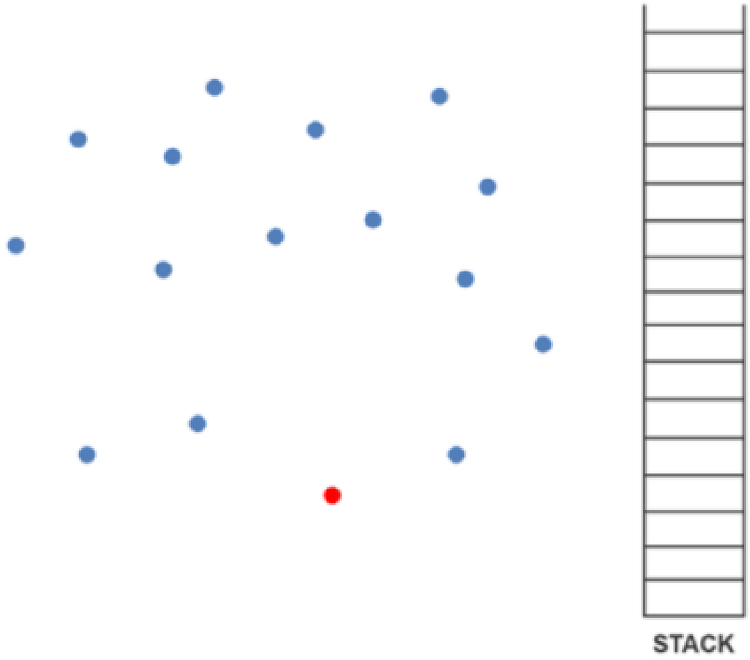
Start at point guaranteed to be on the hull. (the point with the minimum y value)

Sort remaining points by polar angles of vertices relative to the first point.

Go through sorted points, keeping vertices of points that have left turns and dropping points that have right turns.

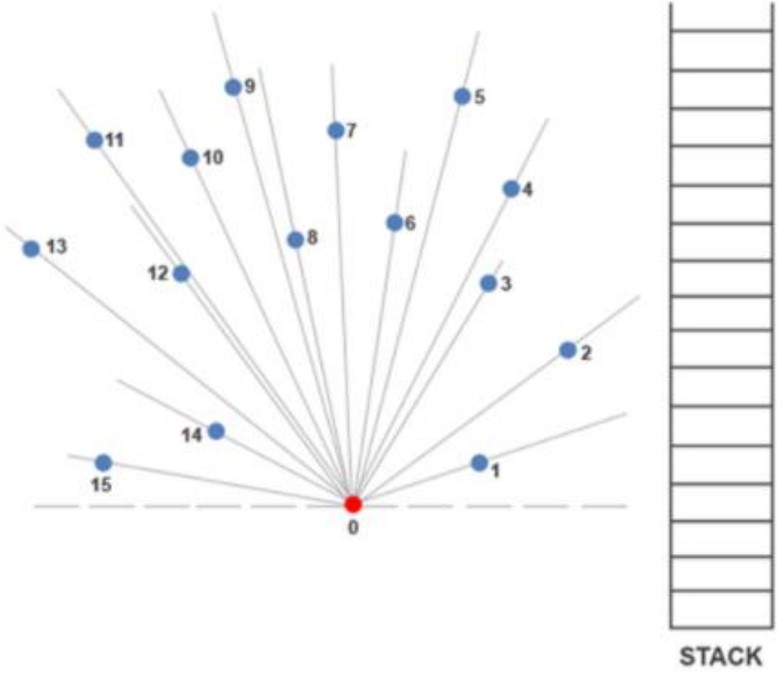
Graham’s Scan

We have a set of points. It’s clear which point has the lowest y-coordinate value.



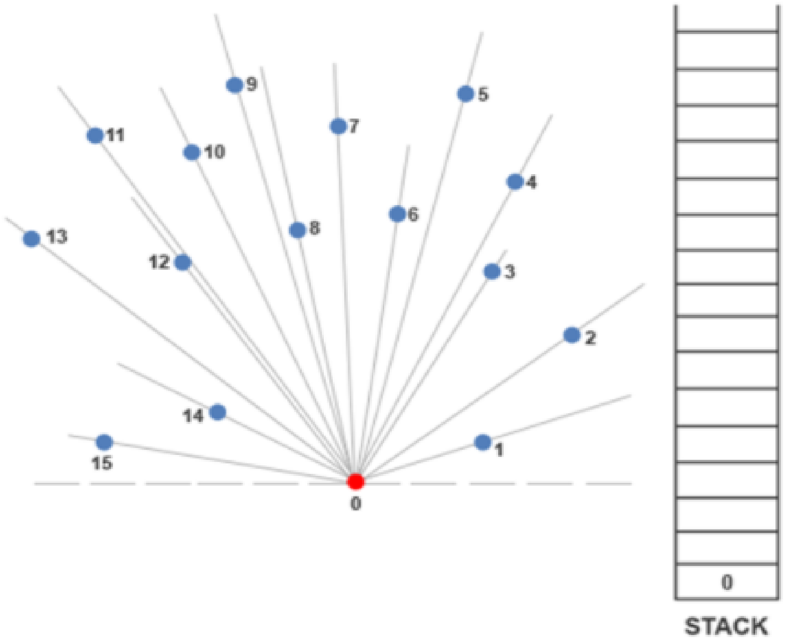
Graham’s Scan

From there, points are ordered in increasing angle.



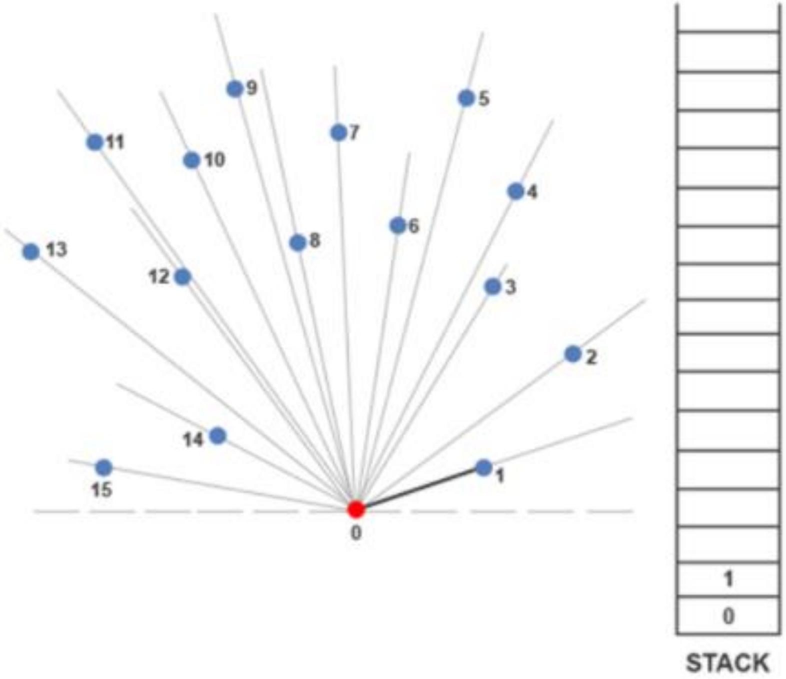
Graham's Scan

Now we can follow Graham's scan to find out which points create the convex hull. Point 0 is pushed onto the stack.



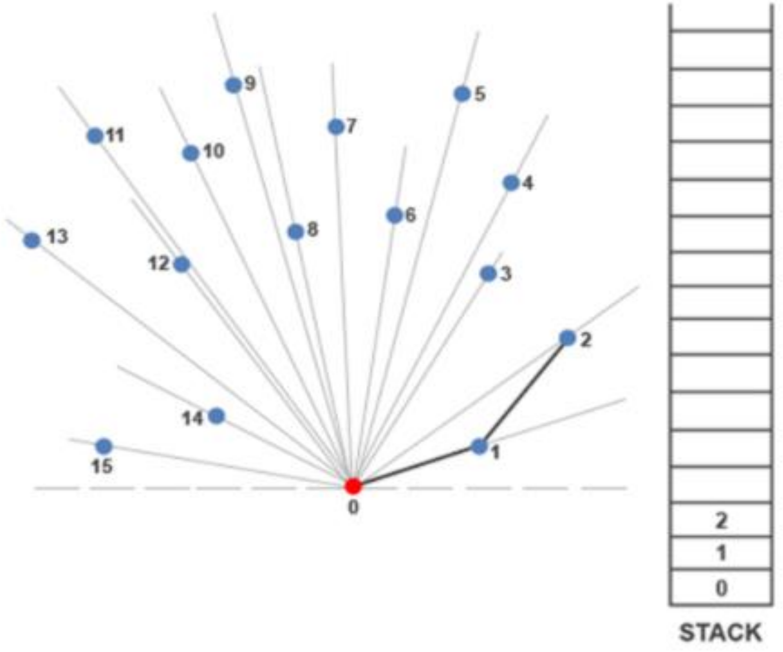
Graham’s Scan

Point 1 is pushed onto the stack immediately after.



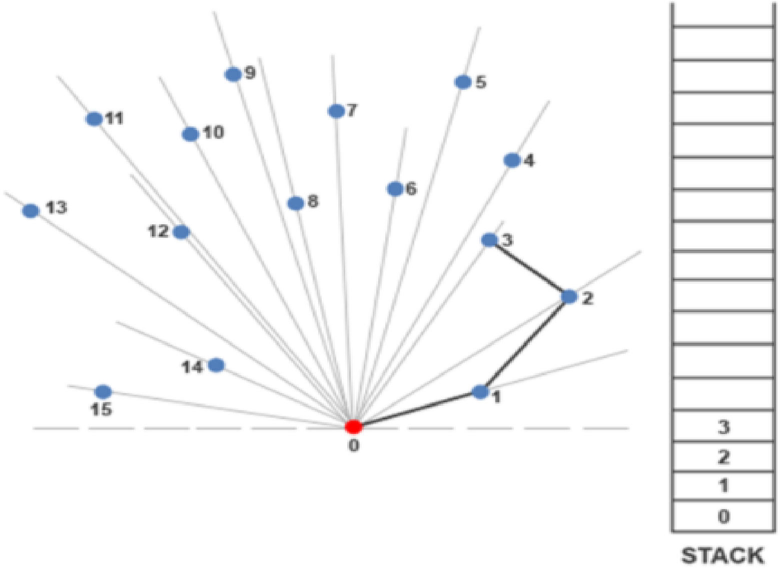
Graham’s Scan

The next point to be added to the stack is 2. A line is made from point 1 to point 2.



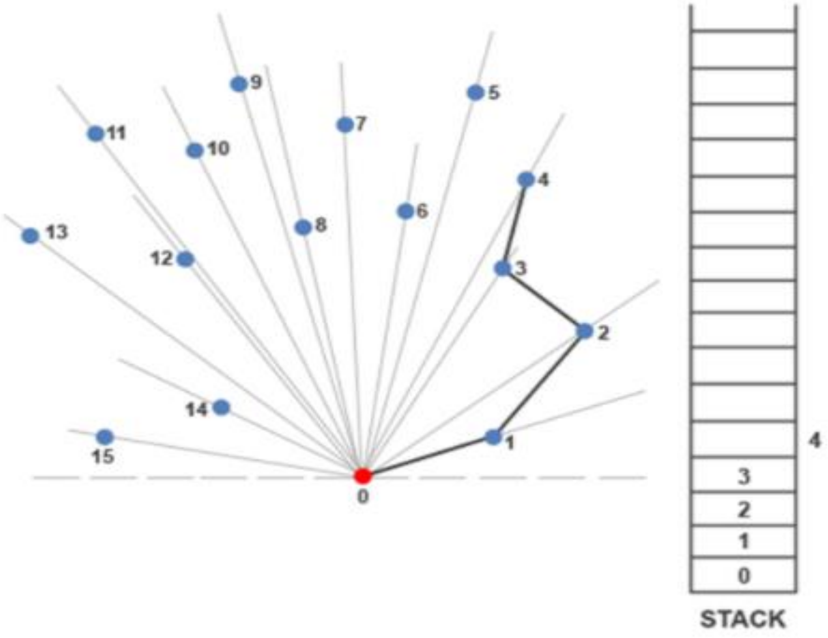
Graham’s Scan

Whenever a left turn is made, the point is presumed to be part of the convex hull. We can clearly see a left turn being made to reach 2 from point 1. To get to point 3, another left turn is made. Currently, point 3 is part of the convex hull. A line segment is drawn from point 2 to 3 and 3 is pushed onto the stack.



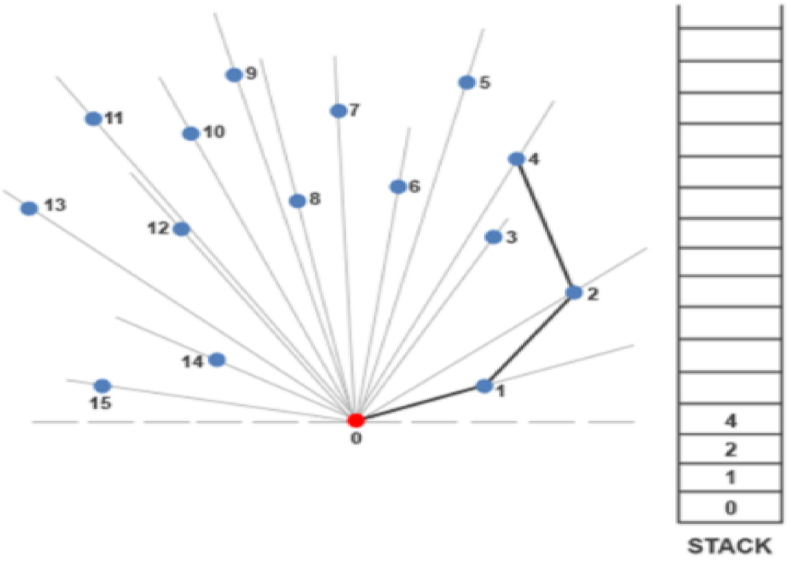
Graham's Scan

We make a right turn going to point 4. We'll draw the line to point 4 but will not push it onto the stack.



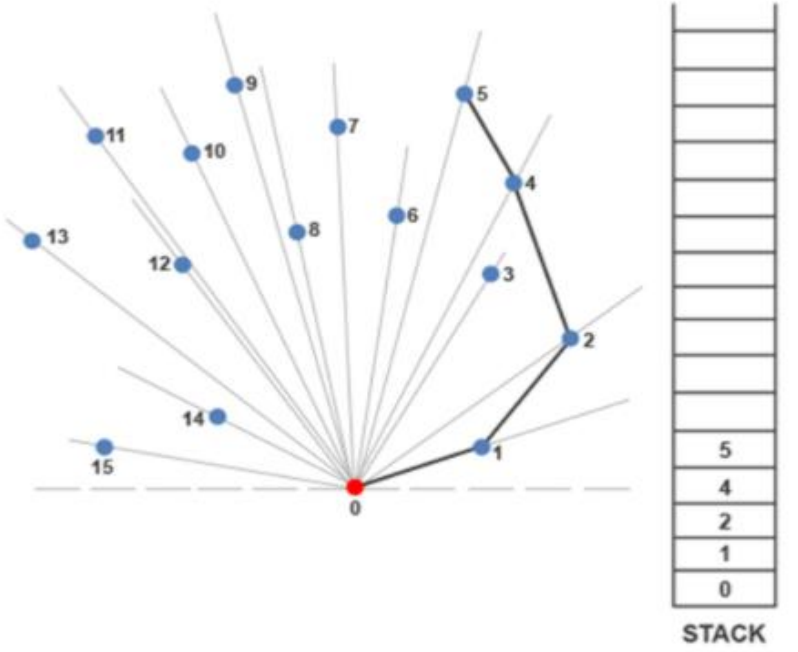
Graham’s Scan

Whenever a right turn is made, Graham’s scan algorithm pops the previous value from the stack and compares the new value with the top of the stack again. In this case, we’ll pop 3 from the top of the stack and we’ll see if going from point 2 to point 4 creates a left bend. In this case it does, so we’ll draw a line segment from 2 to 4 and push 4 onto the stack.



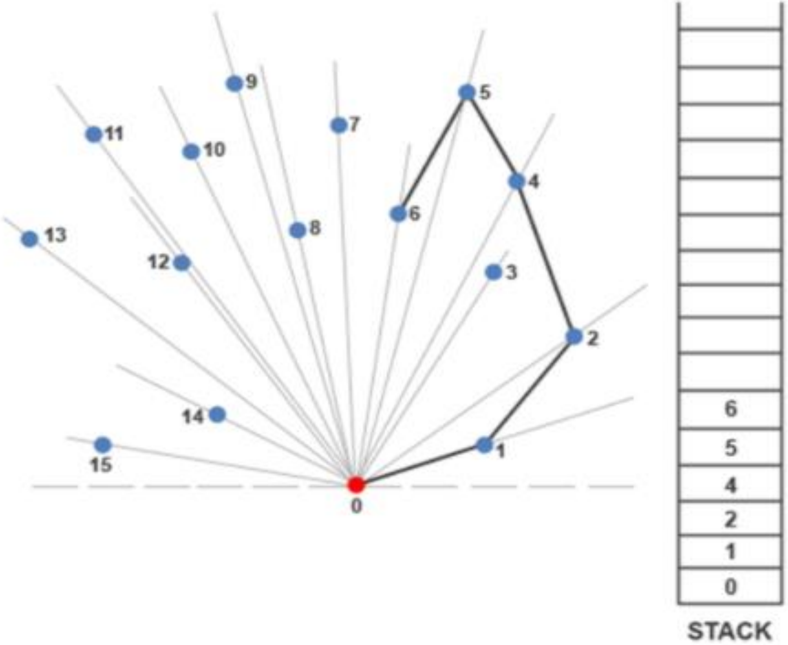
Graham’s Scan

Since going from 4 to 5 creates a left turn, we’ll push 5 onto the stack. Point 5 is currently part of the convex hull.



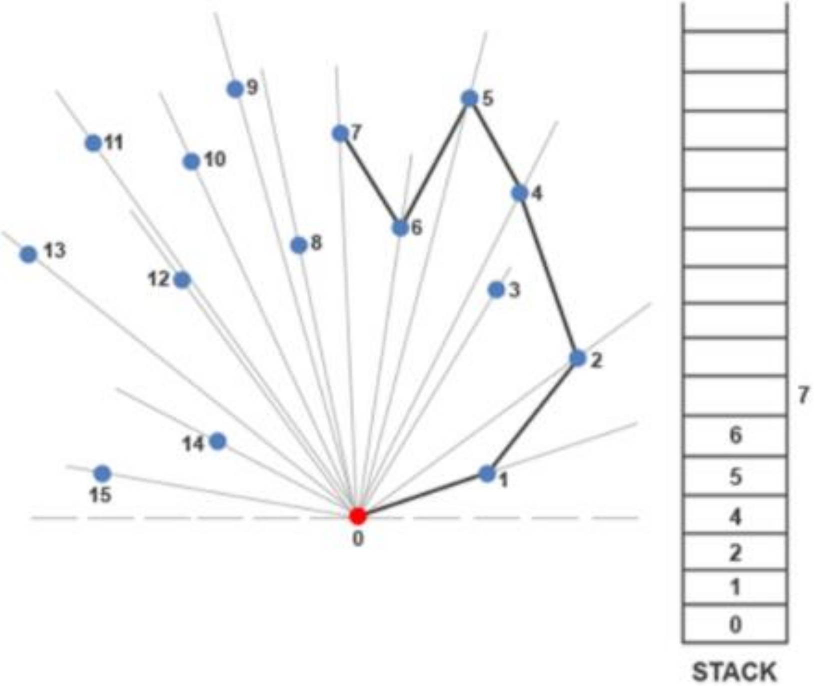
Graham’s Scan

Moving from point 5 to 6 creates a left-hand turn, so we’ll push 6 onto the stack. Point 6 is currently part of the convex hull.



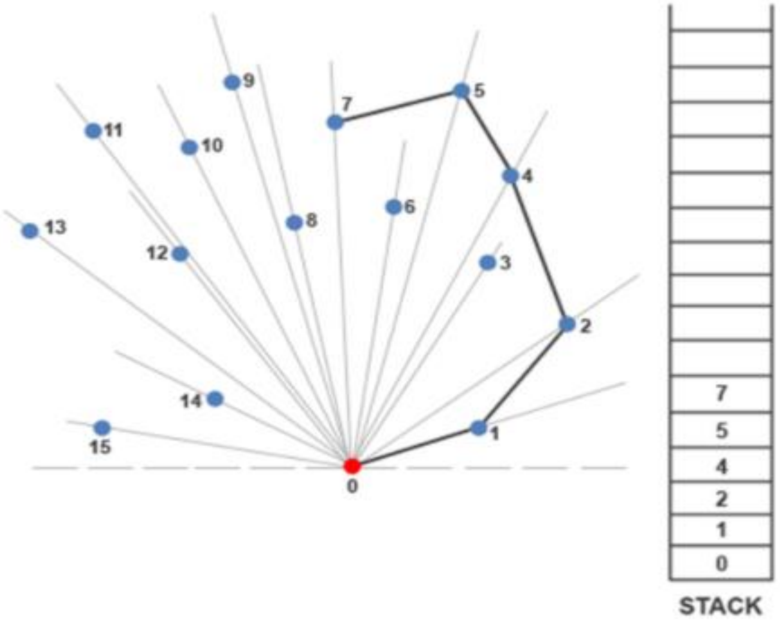
Graham’s Scan

To get to point 7, we must make a right-hand turn at 6.



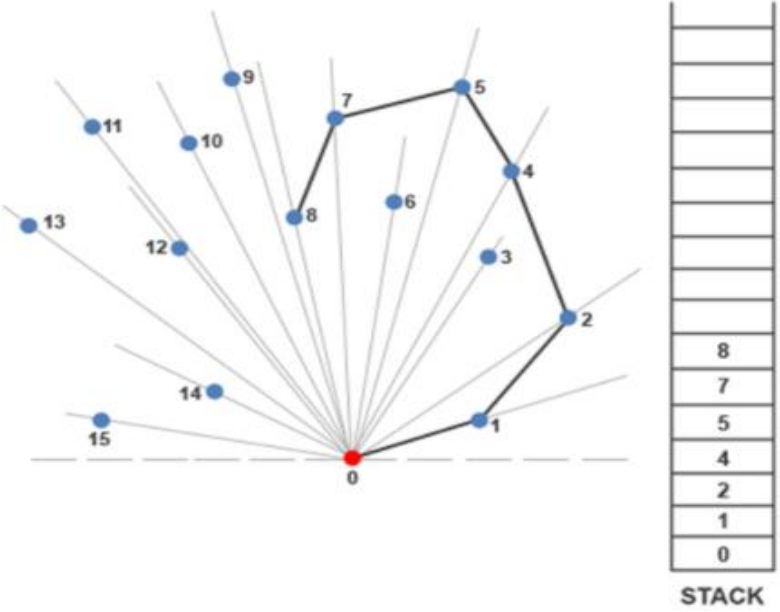
Graham's Scan

Point 6 is popped from the stack and the turn is examined from point 5 to point 7. Since we make a left hand turn from point 5 to point 7, we push point 7 onto the stack.



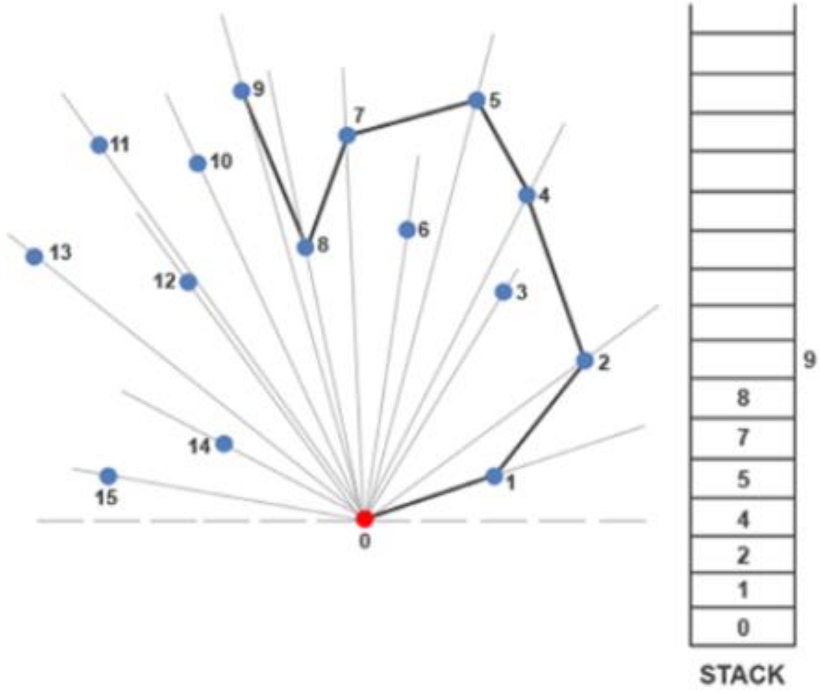
Graham’s Scan

We attempt to push point 8 onto the stack. To get to point 8, we make a left at point 7, therefore point 8 is added to the stack. Point 8 is currently part of the convex hull.



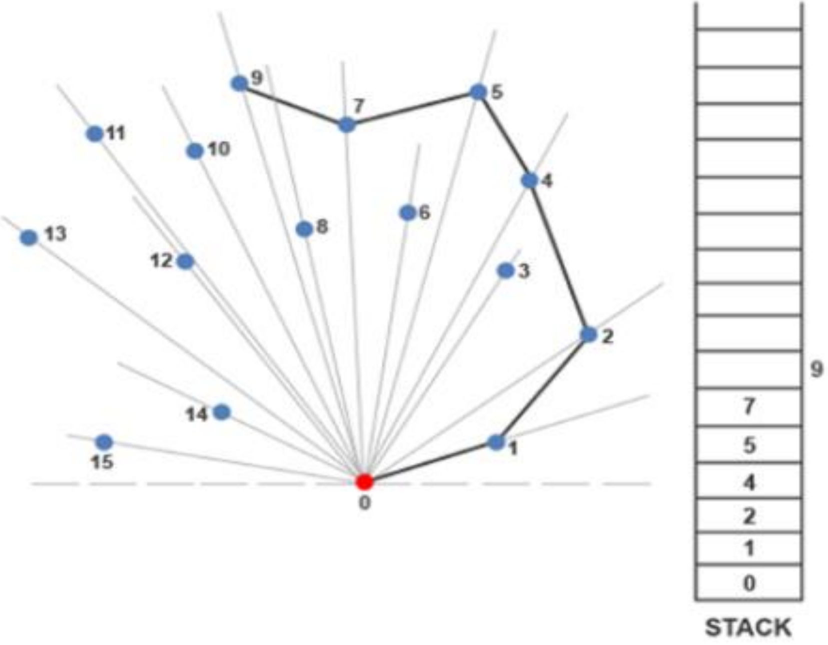
Graham’s Scan

Going to point 9 requires a right-hand turn at point 8.



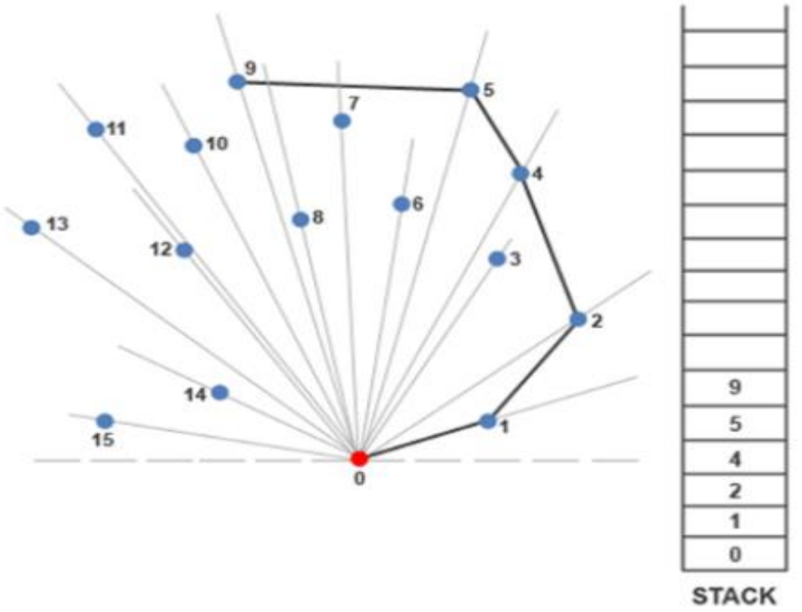
Graham’s Scan

Since there’s a right-hand turn, point 8 is popped from the stack and point 9 is compared with point 7.



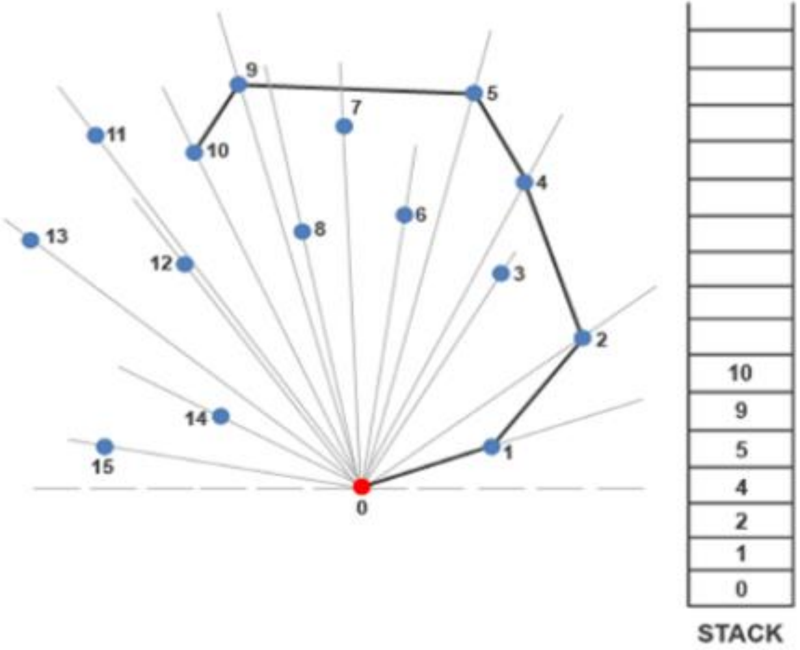
Graham’s Scan

To get to point 9 from point 7 requires another right-turn, so we pop point 7 from the stack too and compare point 9 to point 5. We make a left-hand turn at point 5 to get to point 9, so 9 is pushed onto the stack.



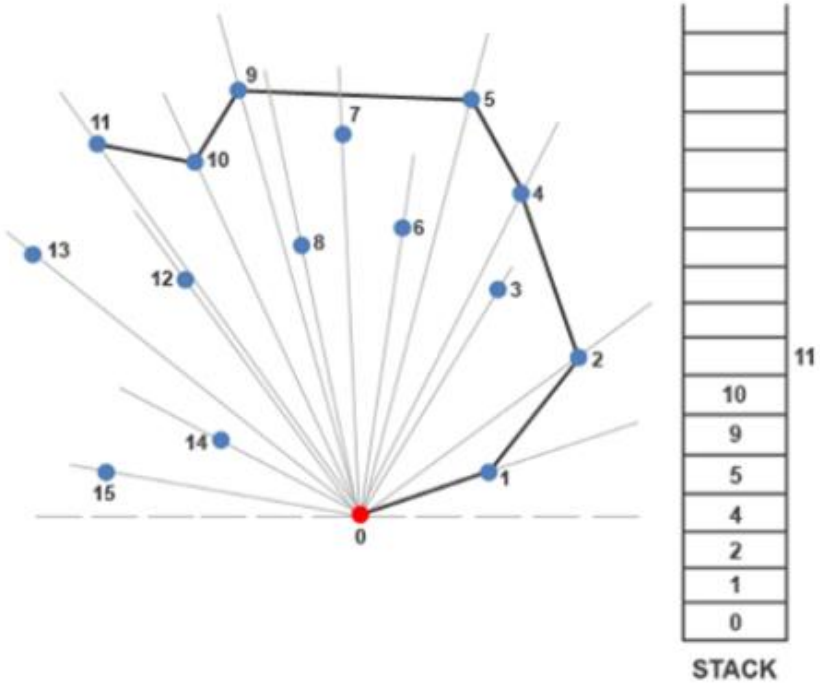
Graham’s Scan

Next, we make a left turn to get to point 10. Point 10 is currently part of the convex hull.



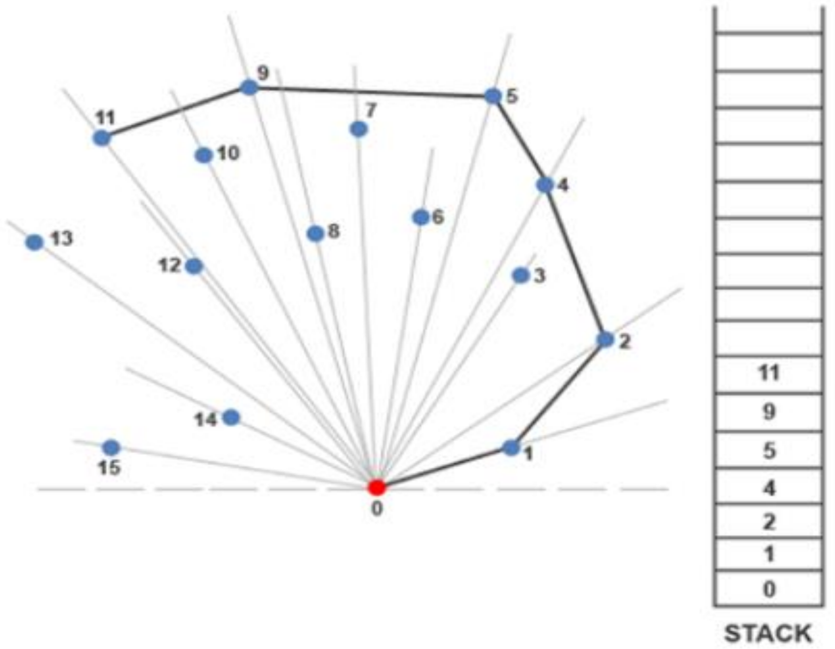
Graham's Scan

A right turn is required to get to point 11 from point 10.



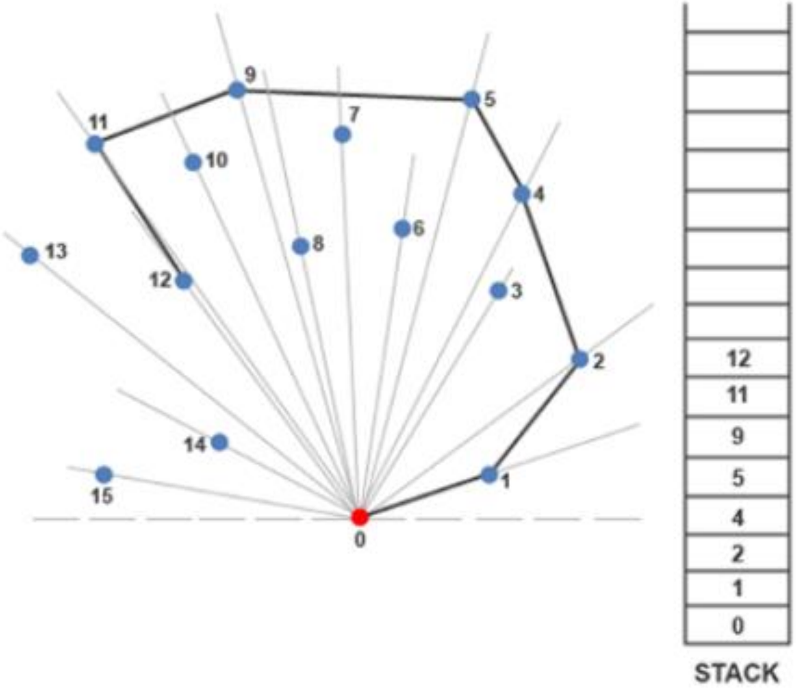
Graham’s Scan

Since a right turn is taken at point 10, point 10 is popped from the stack and the path to point 10 from point 9 is examined. Since a left turn is made at point 9 to get to point 11, point 11 is pushed onto the stack.



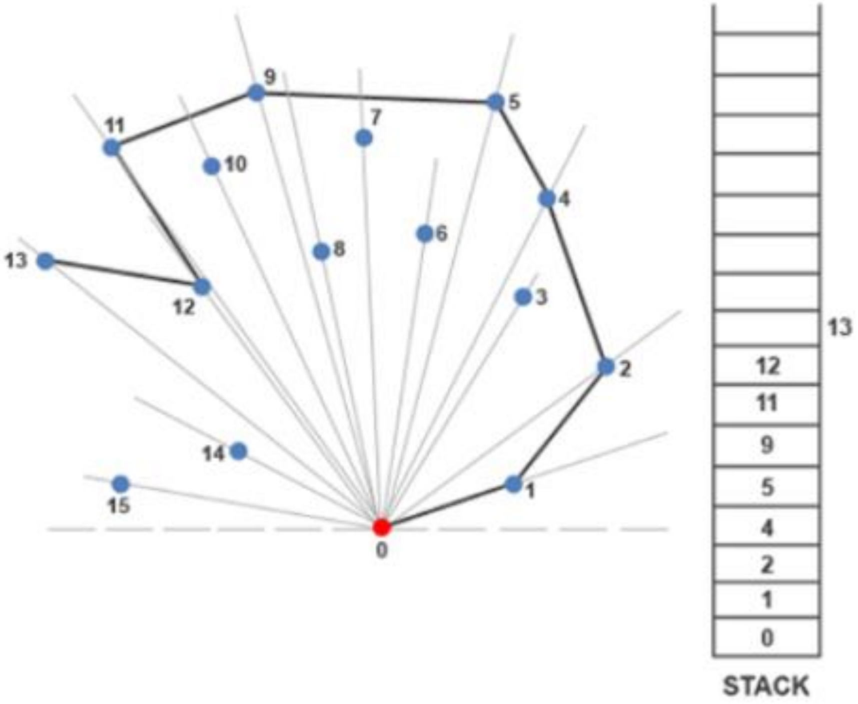
Graham’s Scan

A left turn is made at point 11 to get to point 12. Point 12 is therefore pushed onto the stack and is currently considered part of the convex hull.



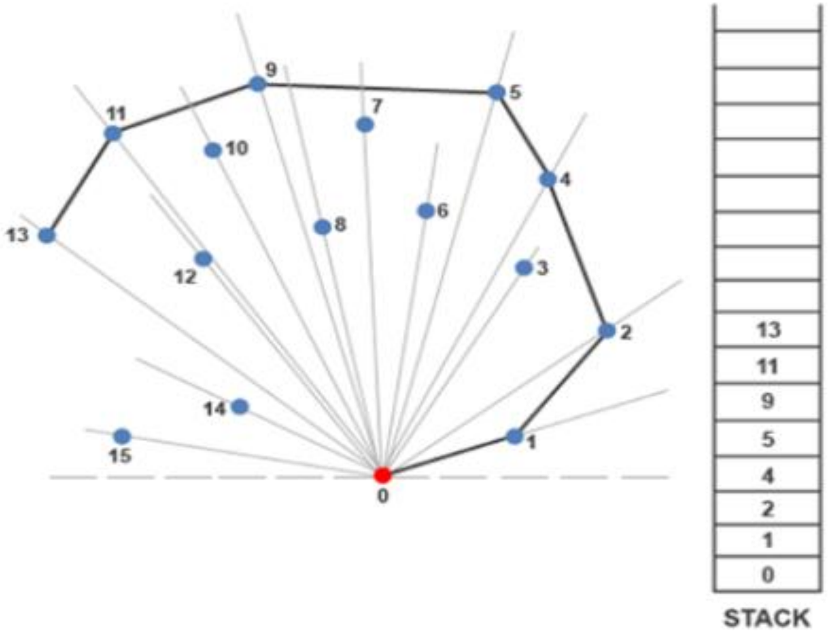
Graham’s Scan

A right turn is required to go to point 13 from point 12.



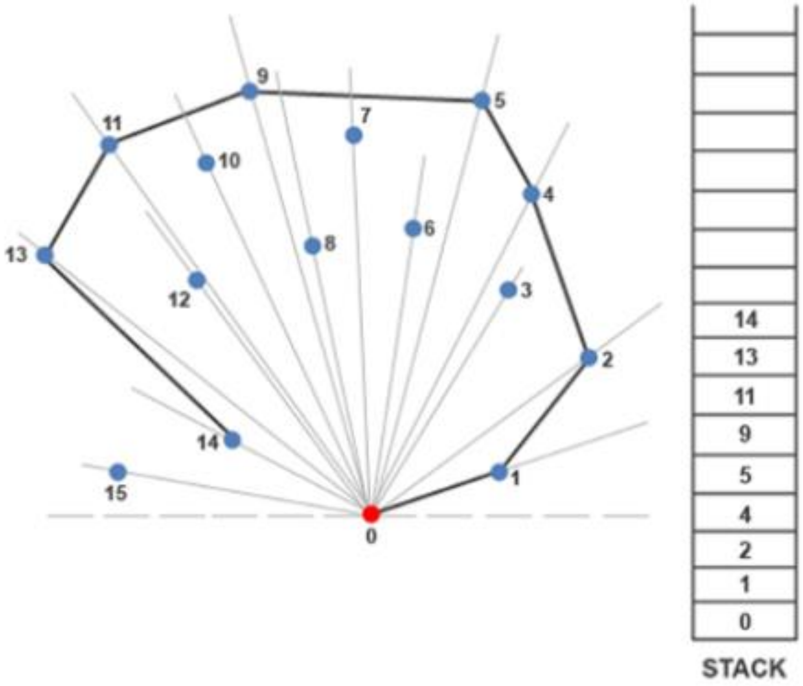
Graham's Scan

Point 12 is popped from the stack and the path to point 13 from point 11 is examined. Since a left turn is made at point 11, point 13 is pushed onto the stack.



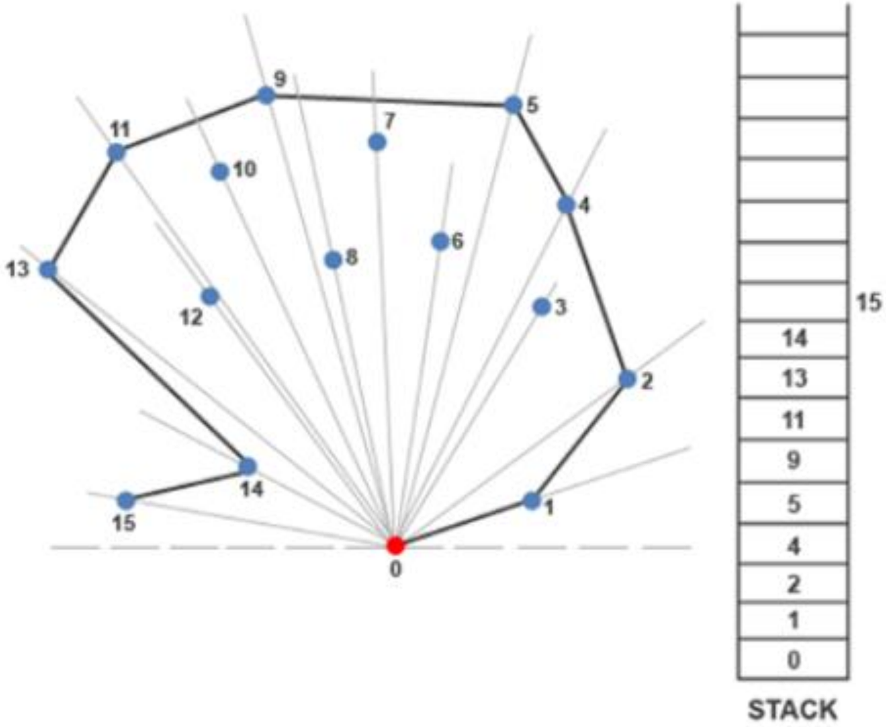
Graham’s Scan

A left turn is made at point 13 to get to point 14, so point 14 is pushed onto the stack.



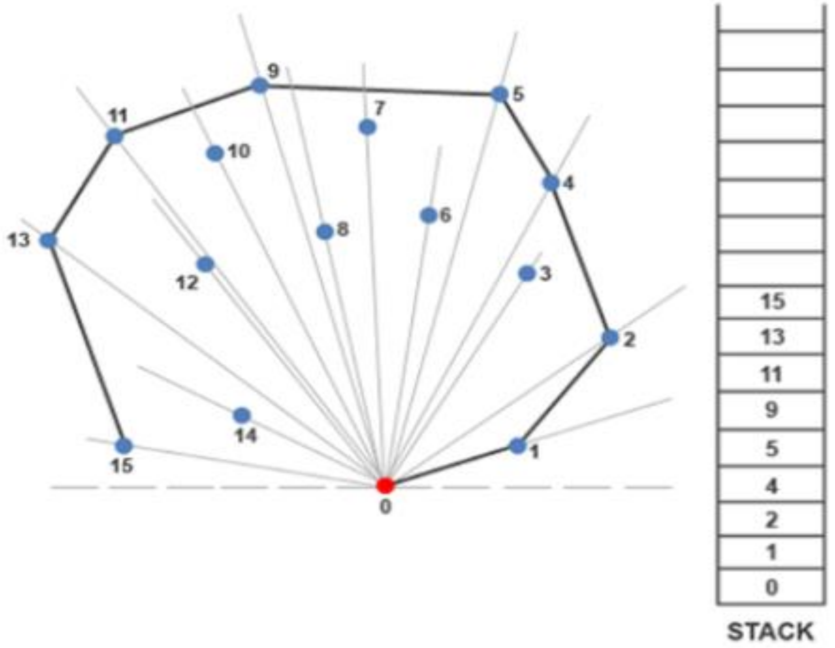
Graham’s Scan

A right turn is required to go from point 14 to point 15.



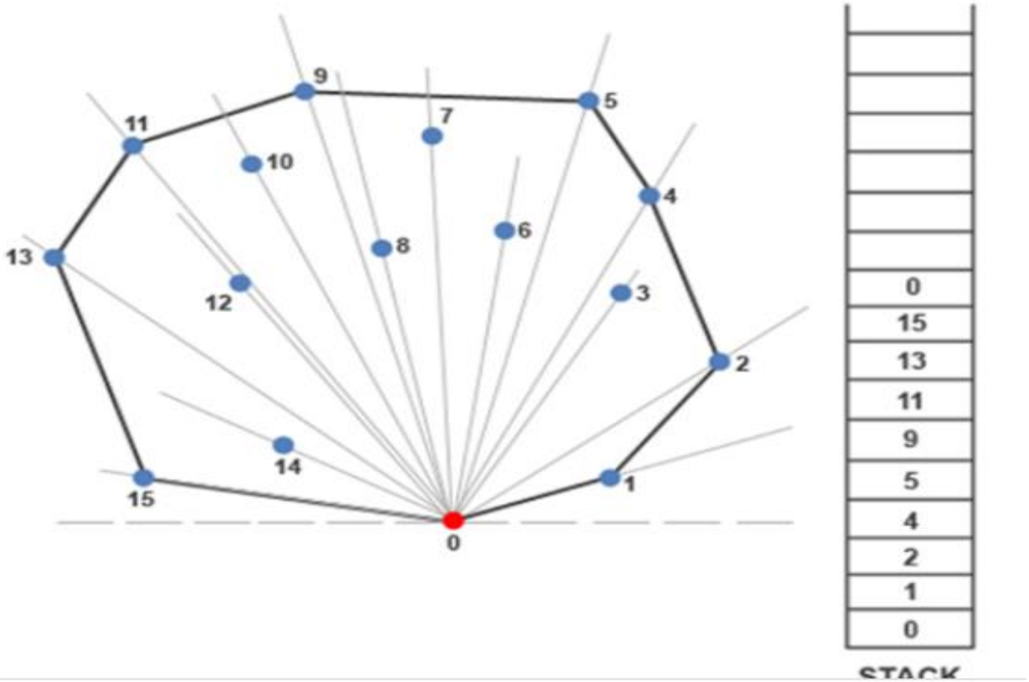
Graham's Scan

Since a right turn was made at point 14, point 14 is popped from the stack. The path to point 15 from point 13 is examined next. A left turn is made at point 13 to get to point 15, so point 15 is pushed onto the stack.



Graham's Scan

Going from point 15 to the starting point 0 requires a left turn. Since the initial point was the point that we needed to reach to complete the convex hull, the algorithm ends.



Graham’s Scan

The points that are needed to create the convex hull are

0–1–2–4–5–9–11–13–15.

Graham's Scan

- Time complexity of Graham's scan algorithm is O(n log n) due to initial sort of angles.

■A more detailed algorithm

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate,
    or the leftmost such point in case of a tie
let $\langle p_1, p_2, \ldots, p_m \rangle$ be the remaining points in Q,
    sorted by polar angle in counterclockwise order around p_0
    (if more than one point has the same angle, remove all but
    the one that is farthest from p_0)
PUSH(p_0, S)
PUSH(p_1, S)
PUSH(p_2, S)
for $i \leftarrow 3$ **to** m
    do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
            and p_i makes a nonleft turn
        do POP(S)
    PUSH(p_i, S)
return S

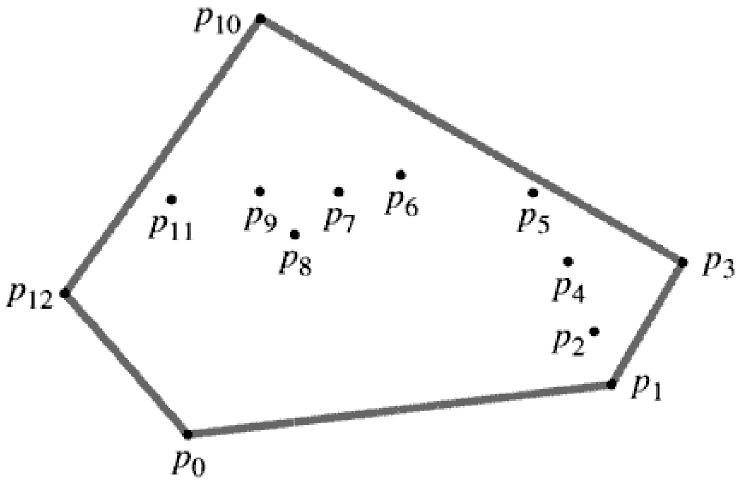


Figure 33.6 A set of points $Q = \{p_0, p_1, \ldots, p_{12}\}$ with its convex hull $\mathrm{CH}(Q)$ in gray.

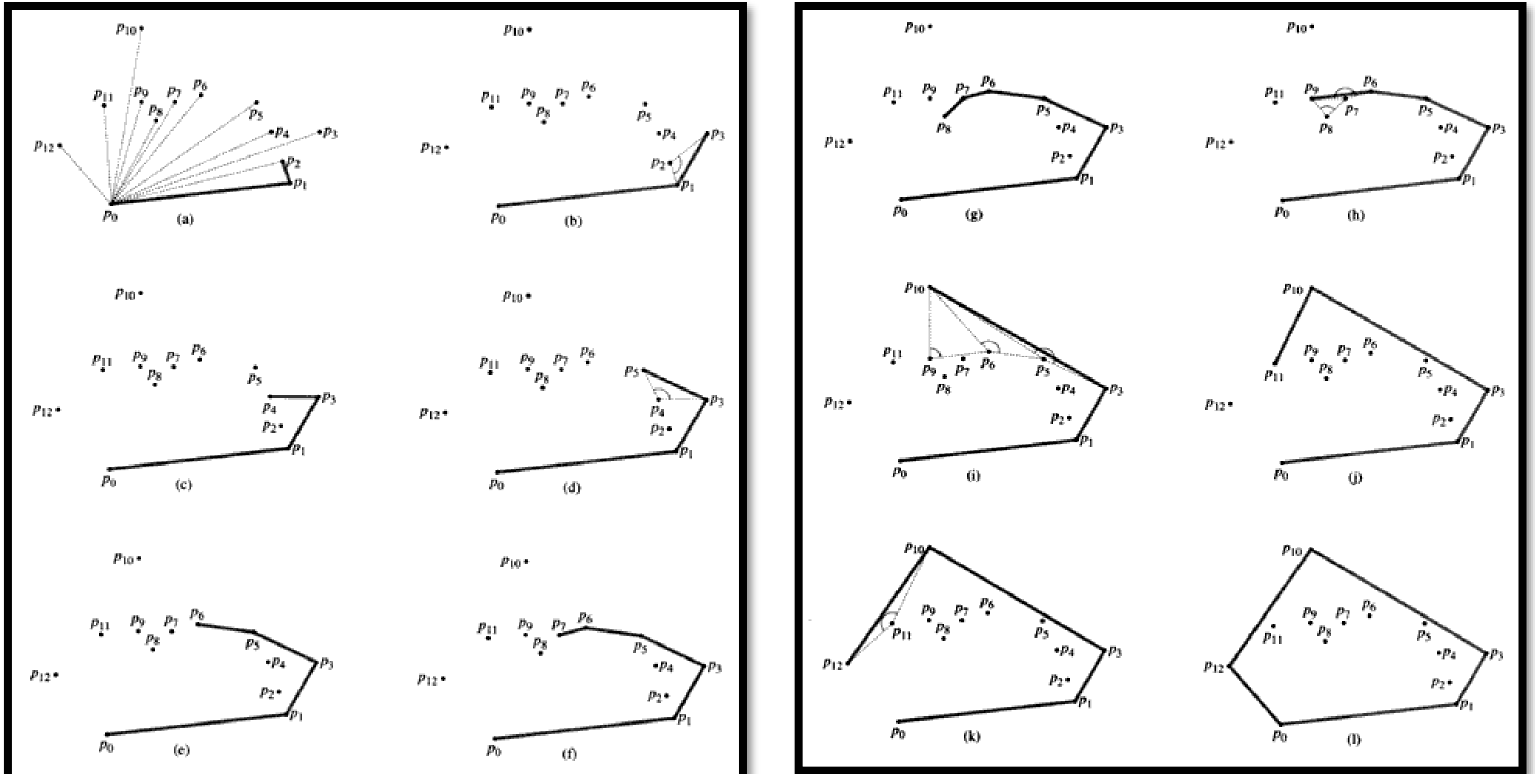


Figure 33.7 The execution of GRAHAM-SCAN on the set Q of Figure 33.6. The current convex hull contained in stack S is shown in gray at each step. **(a)** The sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points numbered in order of increasing polar angle relative to p_0, and the initial stack S containing p_0, p_1, and p_2. **(b)–(k)** Stack S after each iteration of the **for** loop of lines 6–9. Dashed lines show nonleft turns, which cause points to be popped from the stack. In part (h), for example, the right turn at angle $\angle p_7 p_8 p_9$ causes p_8 to be popped, and then the right turn at angle $\angle p_6 p_7 p_9$ causes p_7 to be popped. **(l)** The convex hull returned by the procedure, which matches that of Figure 33.6.

Closest Pair of Points

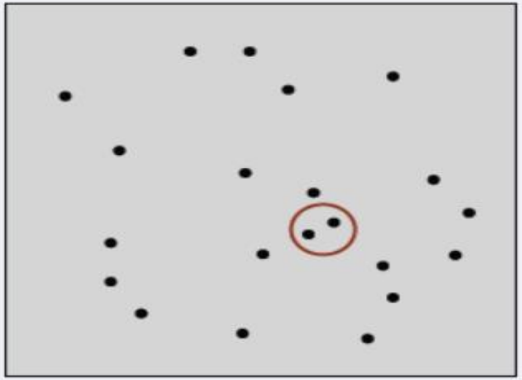
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Fundamental geometric primitive.

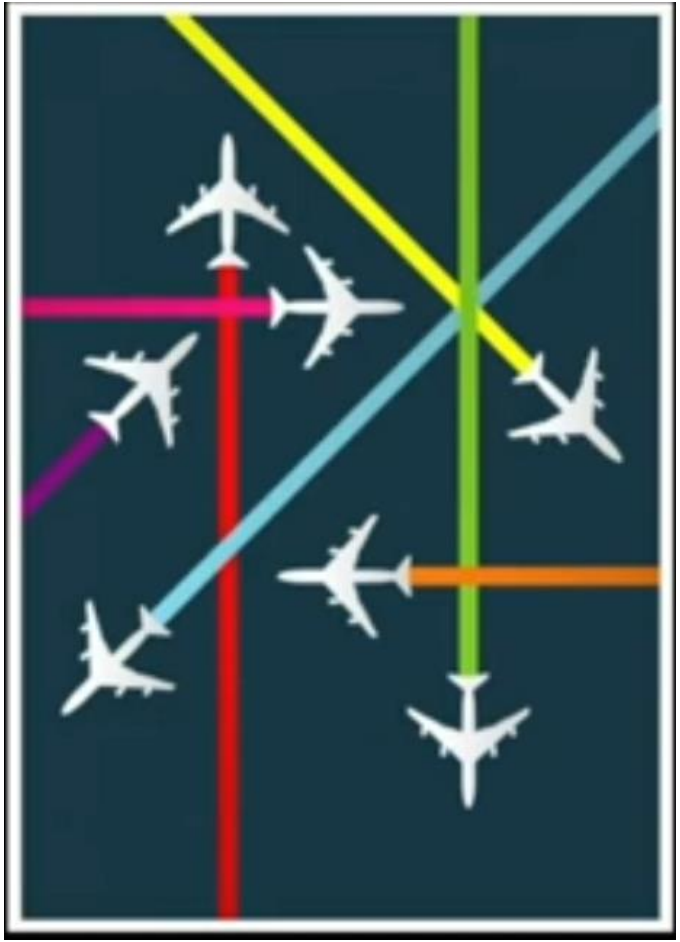
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



Closest Pair of Points

- Applications of Closest Pair.
  - Track the closest pairs in air traffic control to detect and prevent collision.
  - Points can represent database records, statistical samples, DNA sequences, and so on.

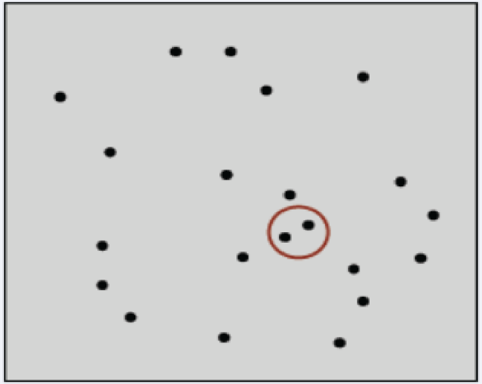


Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

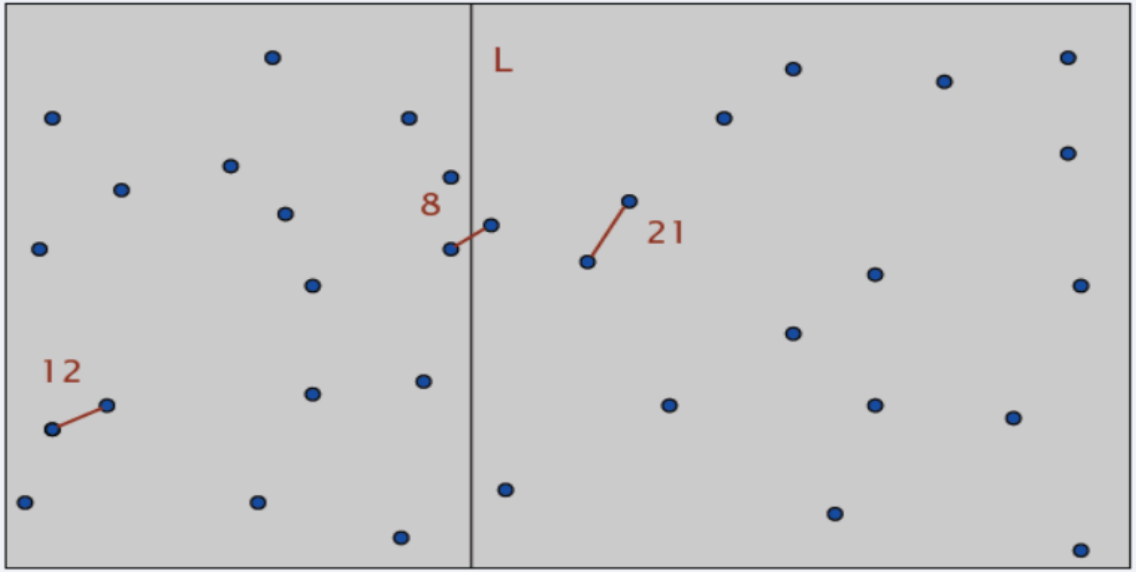
1d version. Easy $O(n \log n)$ algorithm if points are on a line.



Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

seems like $\Theta(N^2)$

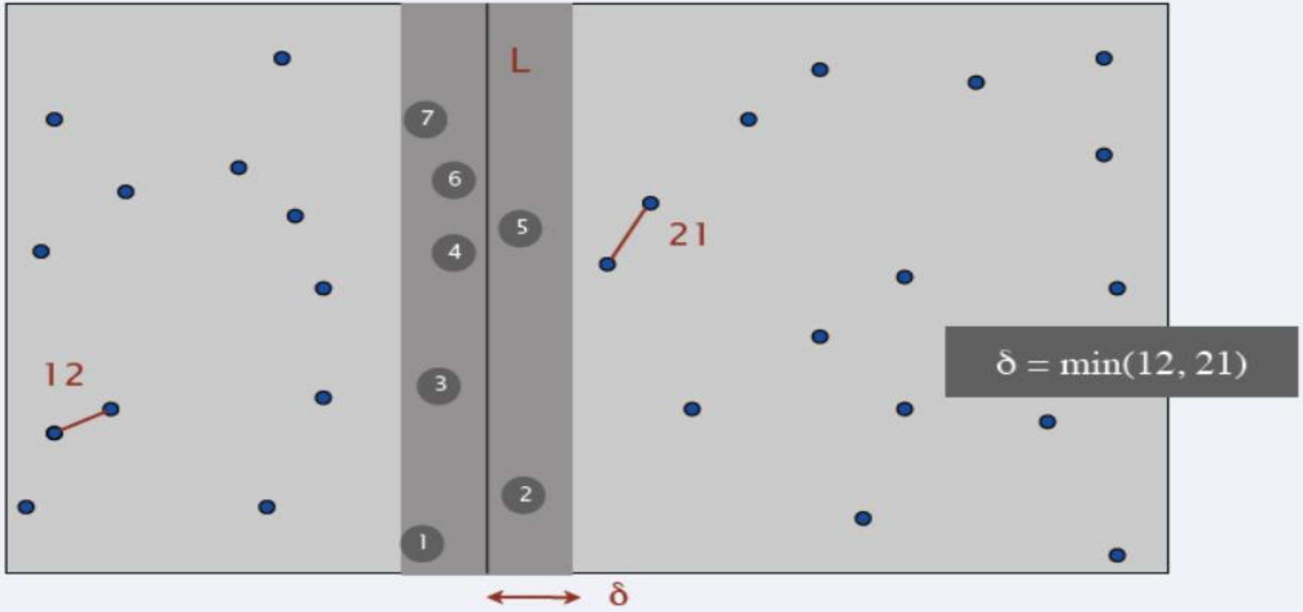


How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y-coordinate.
- Only check distances of those within 11 positions in sorted list!

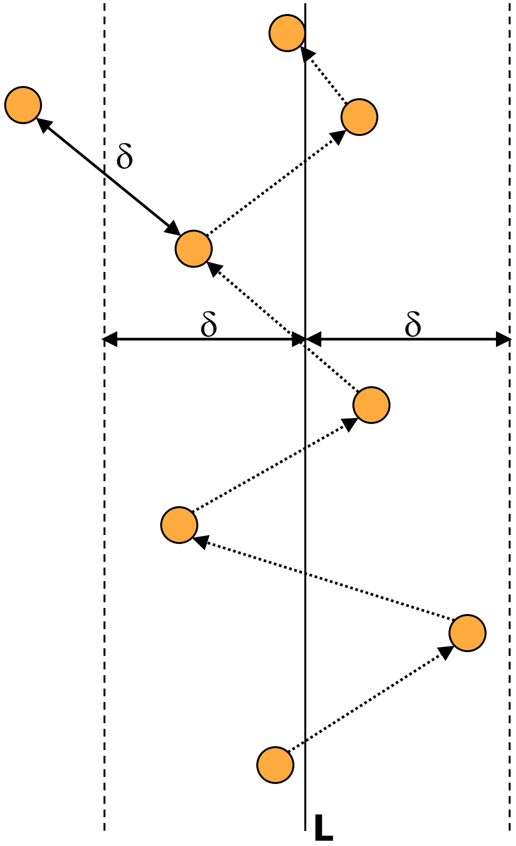
why 11?



Closest pair in the strip

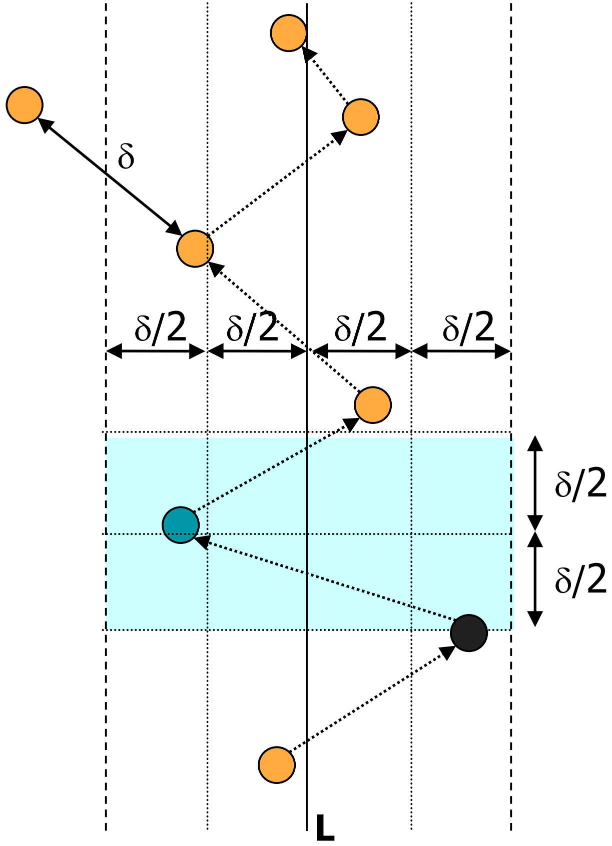
1. Find a sublist **P** of **V** containing all points with a distance at most δ from the line **L** - in linear time.

2. For each element in **P** check its distance to the next 8 elements and remember the closest pair ever found



Why to check only 8 points ahead?

1. Partition the strip into squares of length $\delta/2$ as shown in the picture.
2. Each square contains at most 1 point - by definition of δ.
3. If there are at least 2 squares between points then they can not be the closest points.
4. There are at most 8 squares to check.



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR $(p_1, p_2, \ldots, p_n)$

Compute separation line L such that half the points are on each side of the line. ← $O(n \log n)$

$\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).

$\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half). ← $2\,T(n / 2)$

$\delta \leftarrow \min \{ \delta_1 , \delta_2 \}$.

Delete all points further than δ from line L. ← $O(n)$

Sort remaining points by y-coordinate. ← $O(n \log n)$

Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than δ, update δ. ← $O(n)$

RETURN δ.

Closest Pair of Points: Analysis

Algorithm gives upper bound on running time

Recurrence

$$T(N) \leq 2T(N/2) + O(N \log N)$$

Solution

$$T(N) = O(N (\log N)^2)$$

avoid sorting by y-coordinate from scratch

Upper bound. Can be improved to $O(N \log N)$.

Lower bound. In quadratic decision tree model, any algorithm for closest pair requires $\Omega(N \log N)$ steps.