Data Structure and Algorithm Design

ME/MSc (Computer) – Pokhara University

Prepared by:

Assoc. Prof. Madan Kadariya (NCIT)

Chapter 2:
Advanced Data Structures and Algorithms (11 hrs)

Outline

1. Advanced Trees
   - B and B+ Trees
2. Advanced Graph
   - Planar Graphs and its applications in Geographic and Circuit Layout Design
3. Network Flow Algorithms
   - Ford-Fulkerson Algorithm
   - Edmonds-Karp Algorithm
4. Advanced Shortest Path Algorithms
   - Bellman-Ford Algorithm
   - Johnson's algorithm
5. Graphical Data Structures
   - Quadtrees, KD Trees, R-Trees
6. Computational Geometry
   - Convex Hull and Voronoi Diagrams
7. Case Studies in Red-Black Tree and Its Applications, Applications of Directed Acyclic Graph, Applications of Minimum Spanning Tree in Network Design

Computational Geometry

1. Convex Hull
2. Voronoi Diagrams

Computational Geometry

- ***Computational geometry*** is a branch of computer science that focuses on the study of algorithms which can be expressed in terms of geometry.
- It deals with the design and analysis of algorithms for solving geometric problems involving points, lines, polygons, and other geometric objects

- Two main branches of computational geometry:
  1. **Combinatorial Computational Geometry**: This branch deals with geometric objects as discrete entities. It includes problems like finding the convex hull of a set of points, determining the intersections of line segments, and solving the closest pair of points problem.
  - **Numerical Computational Geometry**: Also known as geometric modeling or computer-aided geometric design (CAGD), this branch focuses on representing real-world objects in forms suitable for computer computations, often used in CAD/CAM systems.

Computational Geometry

Applications

1. **Computer Graphics**: Used in rendering scenes, modeling shapes, and animations. Algorithms for hidden surface removal, ray tracing, and mesh generation are based on computational geometry.
2. **Robotics**: Essential for motion planning, collision detection, and pathfinding. Robots use geometric algorithms to navigate environments and avoid obstacles.
3. **Geographic Information Systems (GIS)**: Helps in managing and analyzing spatial data. Applications include map overlay, terrain modeling, and spatial querying.
4. **Computer-Aided Design (CAD)**: Used in designing and manufacturing products. Geometric algorithms help in modeling complex shapes, surface fitting, and tool path generation.
5. **Computer Vision**: Involves 3D reconstruction, object recognition, and image segmentation. Geometric algorithms help in interpreting visual data and reconstructing 3D models from 2D images.

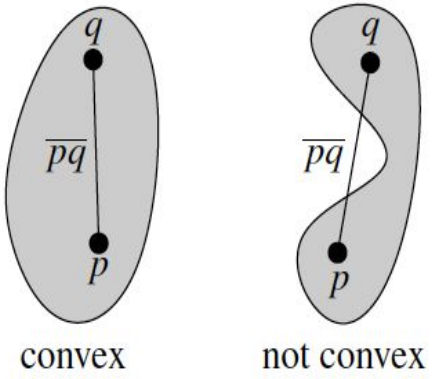
Computational Geometry

Applications

6. **Integrated Circuit Design**: Used in the layout and verification of circuits. Algorithms help in optimizing the placement of components and routing of connections.
7. **Virtual Reality (VR) and Augmented Reality (AR)**: Geometric algorithms are used to create immersive environments and overlay digital information onto the real world.
8. **Medical Imaging**: Helps in reconstructing 3D images from 2D scans, such as CT or MRI scans, and in planning surgical procedures.

Computational Geometry (Convex Hull)

- The **convex hull** of a set of points is the smallest convex polygon that can enclose all the points.
- A subset S of the plane is called convex if and only if for any pair of points $p,q \in S$ the line segment $\overline{pq}$ is completely contained in S. The convex hull $CH(S)$ of a set S is the smallest convex set that contains S. To be more precise, it is the intersection of all convex sets that contain S.
- Lets visualize what the convex hull looks like by a thought experiment.
- Imagine that the points are nails sticking out of the plane, take an elastic rubber band, hold it around the nails, and let it go. It will snap around the nails, minimizing its length.
- The area enclosed by the rubber band is the convex hull of P.
- This leads to an alternative definition of the convex hull of a finite set P of points in the plane: it is the unique convex polygon whose vertices are points from P and that contains all points of P.



Computational Geometry (Convex Hull)

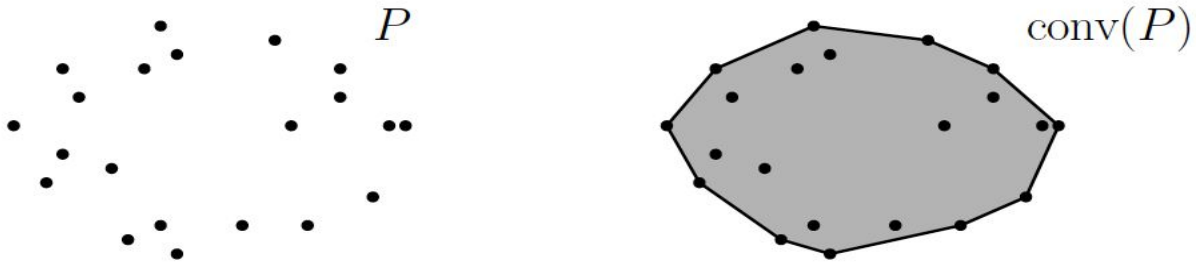


Fig: Point sets and a convex hull

Properties of Convex hull

Convexity: The convex hull is always convex, meaning any line segment between two points inside it lies entirely within the hull.

Minimality: The convex hull is the smallest convex shape that completely contains the given points.

Uniqueness: For a set of points, there is only one convex hull.

Extreme Points: The vertices (corners) of the convex hull are the "outermost" points of the set,

which cannot be formed by combining other points.

Affine Invariance: If the points are transformed (e.g., rotated, scaled, or shifted), the convex hull of

Computational Geometry (Convex Hull)

Finding convex hull using Graham scan Algorithm

- **Input:** A set $P=\{p_1,p_2,\ldots,p_n\}$, where each p_i is a point in the plane (x_i,y_i).
- **Output:** The points forming the convex hull in counterclockwise order.

Steps:

1. **Find the Point with the Lowest Y-coordinate**:
   1. Identify the point p_0 with the lowest *y-coordinate* (and the lowest *x-coordinate* in case of ties). This point is guaranteed to be part of the convex hull.
2. **Sort Points by Polar Angle**:
   1. Sort the remaining points $\{p_1,p_2,\ldots,p_n\}$(excluding p_0) based on the **polar angle** they make with p_0.
      1. **Formula for Polar Angle**: $\theta=arctan(y-y_0/x-x_0)$ where (x,y) is the current point and (x_0,y_0) is p_0.
   2. If two points have the same angle, keep only the farthest point from p_0 using the Euclidean $d = \sqrt{(x - x_0)^2 + (y - y_0)^2}$

Computational Geometry (Convex Hull)

3. **Construct the Hull**:
   - Start with p_0 and the first two points in the sorted list as the initial hull (Stack).
4. **Check Orientation Using Cross Product**:
   - For each point p_i in the sorted list:
     - Compute the **orientation** of the triplet of points $(p_{prev}, p_{curr}, p_i)$:Cross Product: $(x_{curr} - x_{prev}) \cdot (y_i - y_{curr}) - (y_{curr} - y_{prev}) \cdot (x_i - x_{curr})$ OR

$$\text{Cross Product} = (x_2 - x_1) \cdot (y_3 - y_2) - (y_2 - y_1) \cdot (x_3 - x_2)$$

       - If the result > 0: Left turn (valid for convex hull).
       - If the result < 0: Right turn (remove p_{curr} from the hull).
       - If the result = 0: Collinear points (retain the farthest point).
     - Keep removing the last point from the hull until the orientation forms a left turn.
5. **Close the Hull**:
   - After processing all points, the final list of points in the stack forms the convex hull.

Computational Geometry (Convex Hull)

GRAHAM_SCAN(P):

1. Find the point p_0 with the lowest y-coordinate (and smallest x-coordinate if there are ties).
2. Sort the points in P by polar angle with respect to p_0:
   a. For each point pi in P:
      - Compute the polar angle $\theta = \arctan(y_i - y_0, x_i - x_0)$.
      - If angles are the same, keep the point farthest from p_0.
3. Initialize a stack (HULL) to store the vertices of the convex hull.
   - Push p_0, the first sorted point, and the second sorted point onto HULL.
4. For each point p_i in the sorted list (starting from the third point):
   a. While there are at least two points in HULL and the last three points in HULL do not form a left turn:
      - Pop the top point from HULL.
   b. Push p_i onto HULL.
5. The points in HULL now form the convex hull in counterclockwise order.
6. Return HULL.

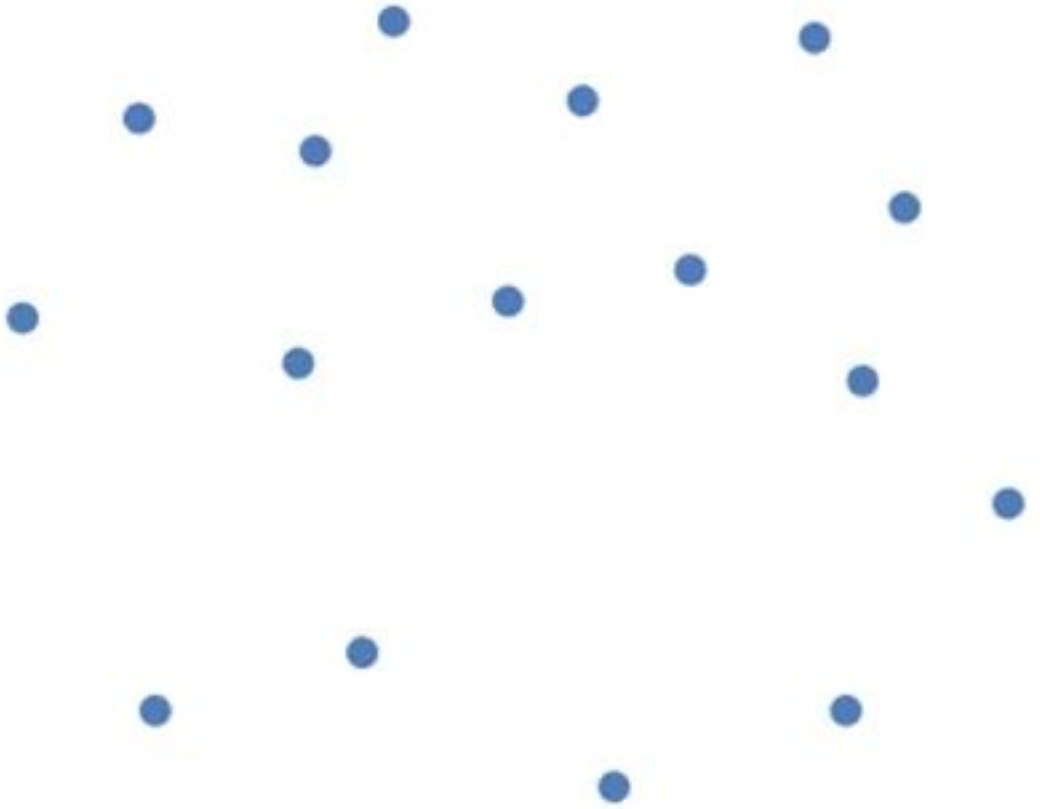
Graham Scan Algorithm Pseudocode

Graham's Scan Algorithm

Input: S, a set of n points in $\Re^2$.
Output: Σ, a stack containing conv(S) with the points in counter-clockwise order.

$p_0 \leftarrow$ the point in S with the minimum y-coordinate.
Sort S - $\{p_0\}$ with the orientations of $\overline{p_0 p_i}$ so that they are ordered $p_1, p_2, \ldots, p_{n-1}$.
PUSH(p_0, Σ)
PUSH(p_1, Σ).
for (i == 2) to $n - 1$ **do**
    while (*orient*(second(Σ), top(Σ), p_i) ≤ 0) **do**
        POP(Σ)
    end-while
    PUSH(p_i, Σ)
end-for
▷▷ Comment: Σ now contains the sequence of points forming conv(S) in counter clockwise order.
return Σ

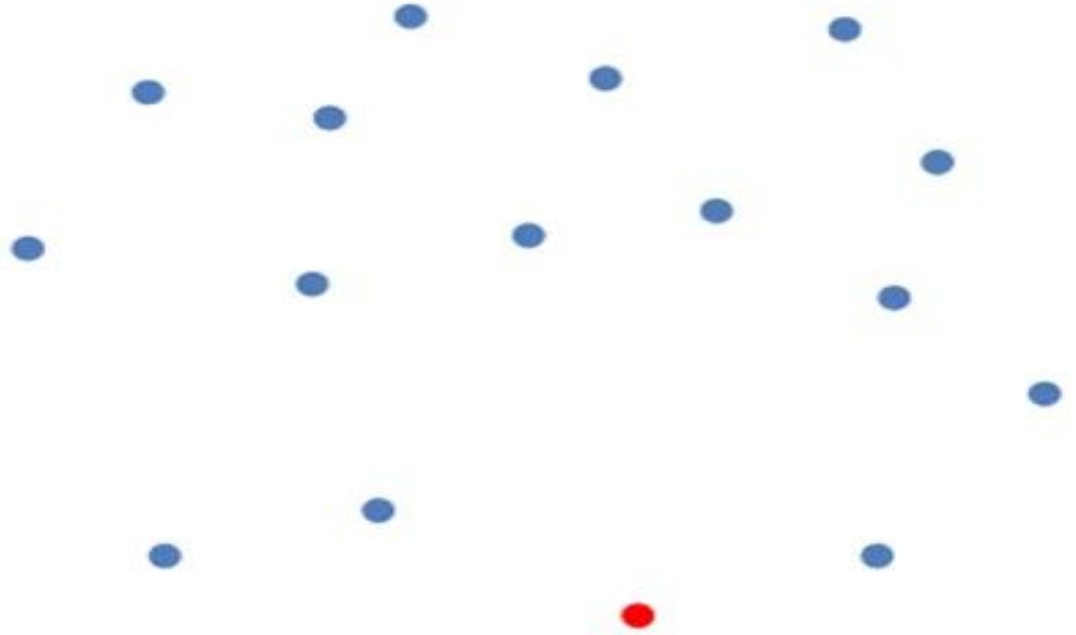
Graham scan Algorithm(Example)



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

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

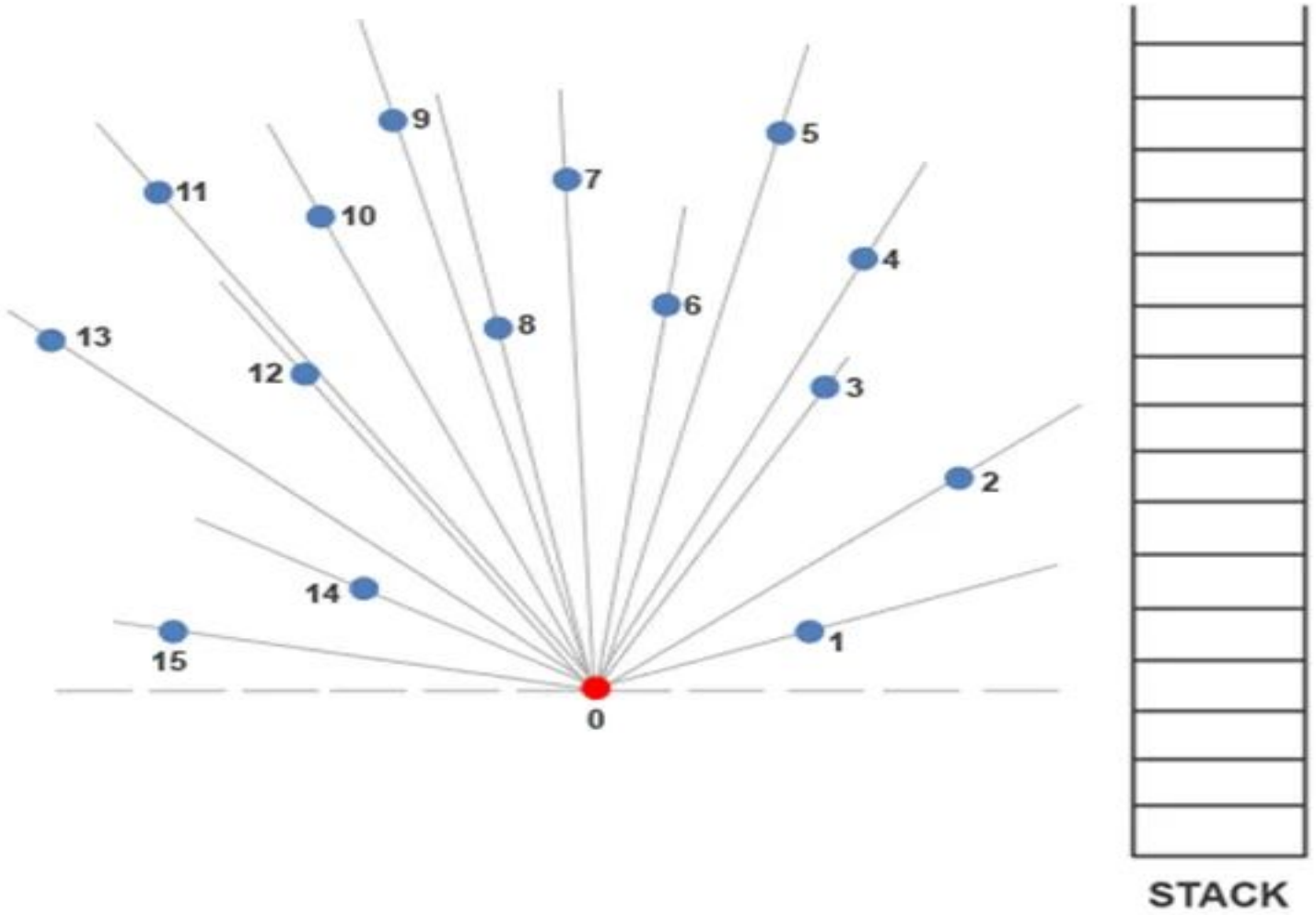


Computational Geometry (Convex Hull)

Graham scan Algorithm

(Example)

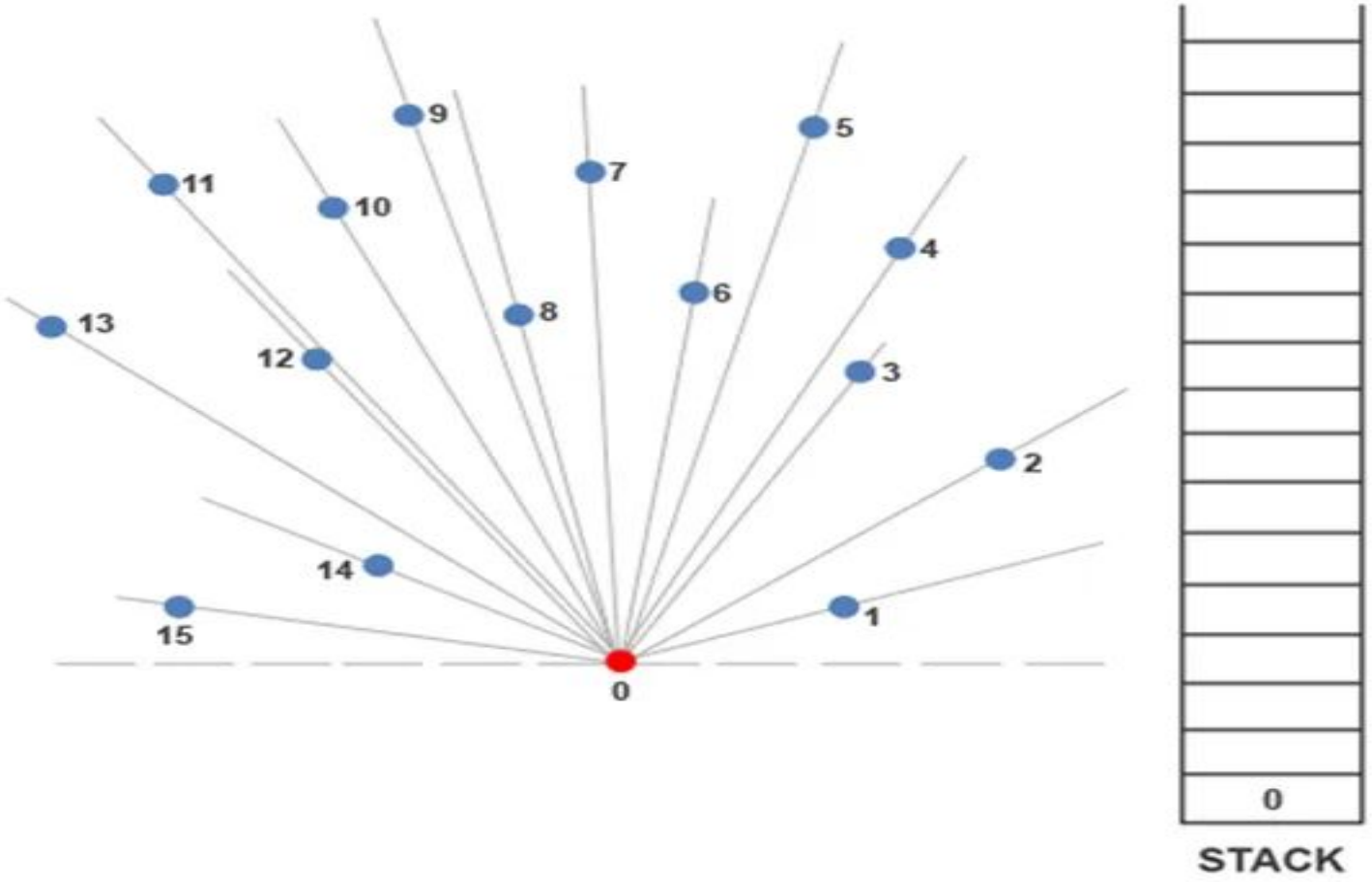
From the, points are ordered in increasing angle.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

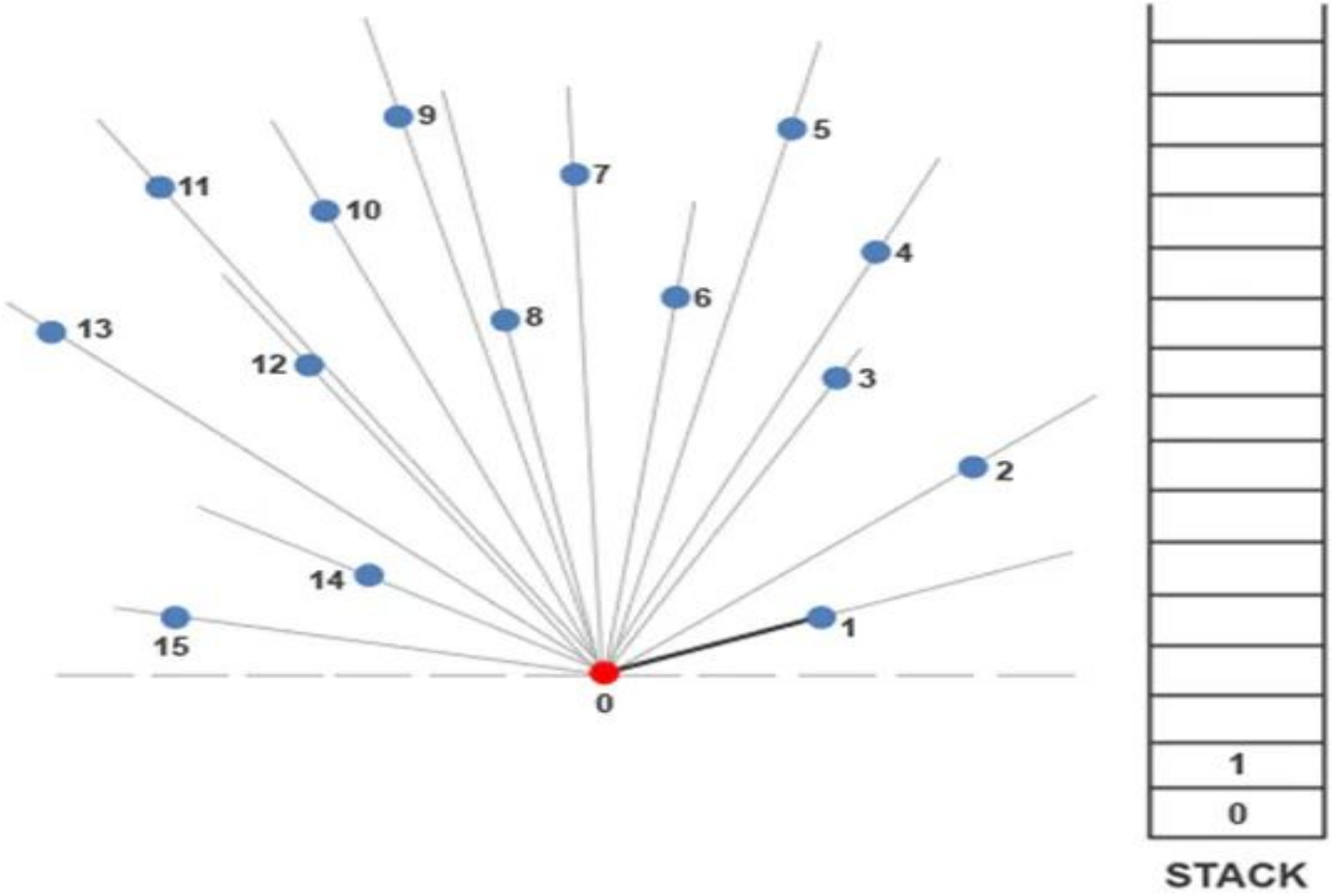
Now we can follow Graham's scan to find out which points create the convex hull. Point 0 is pushed onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

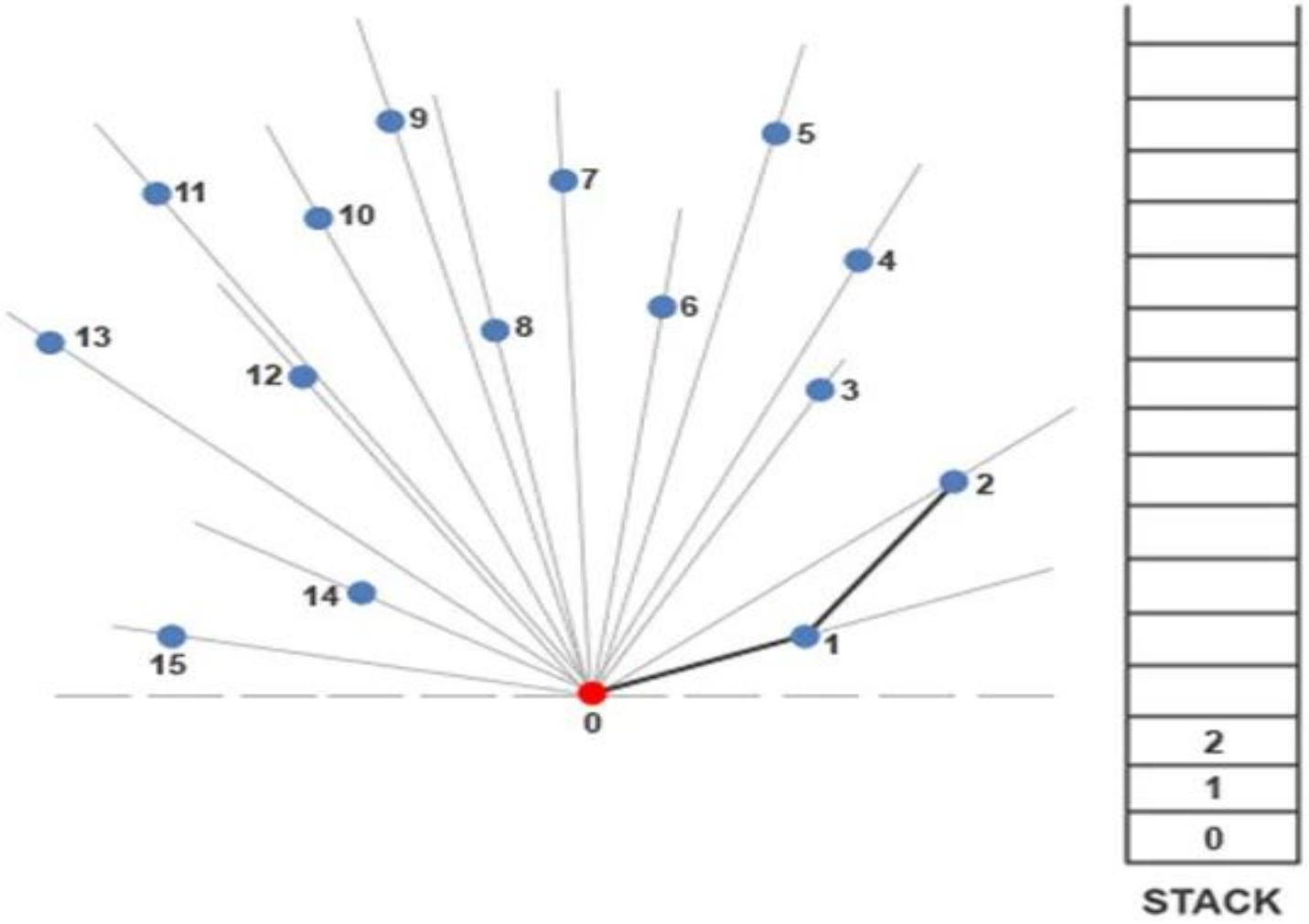
Point 1 is pushed onto the stack immediately after.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

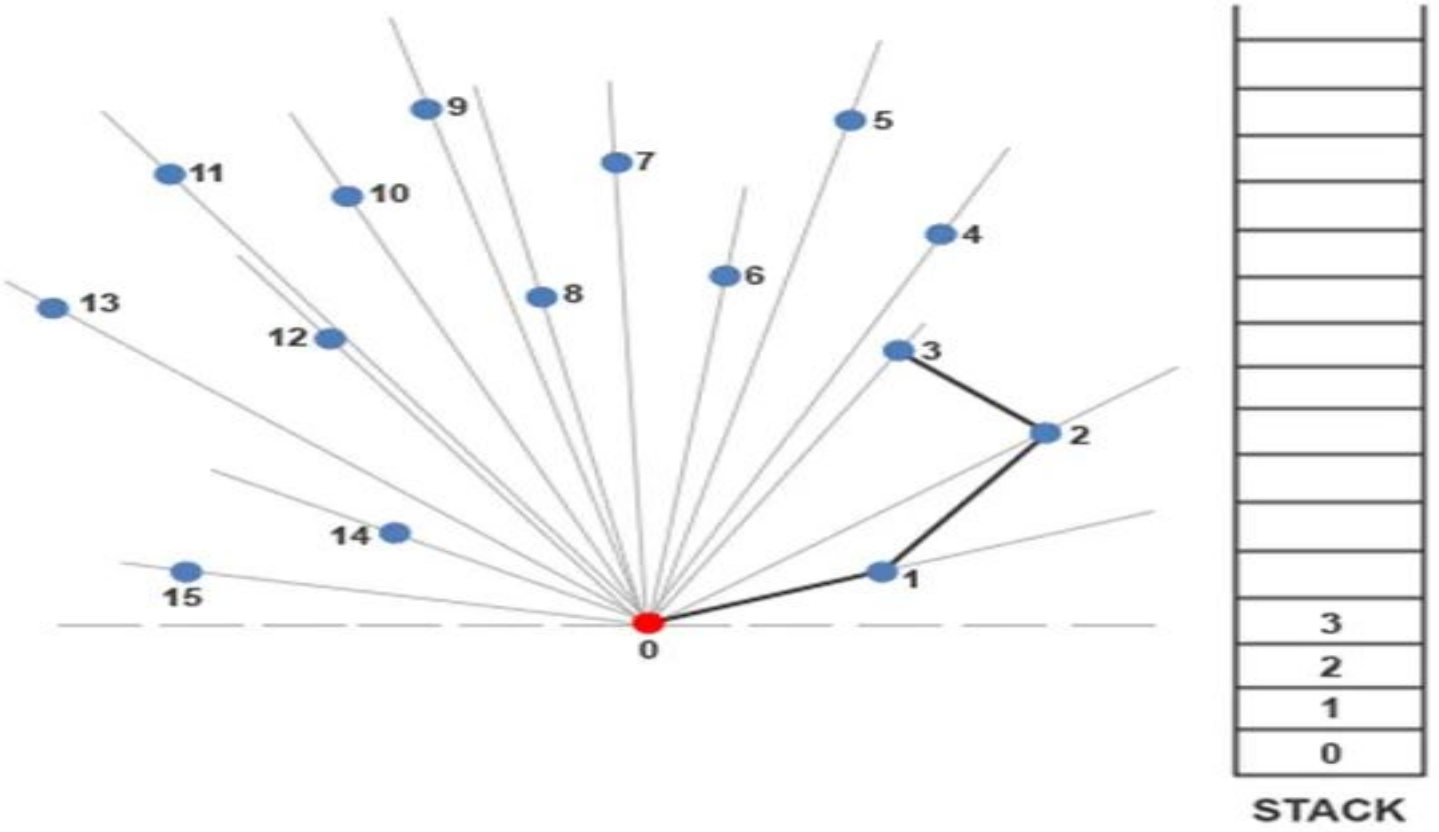
The next point to be added to the stack is 2. A line is made from point 1 to point 2.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

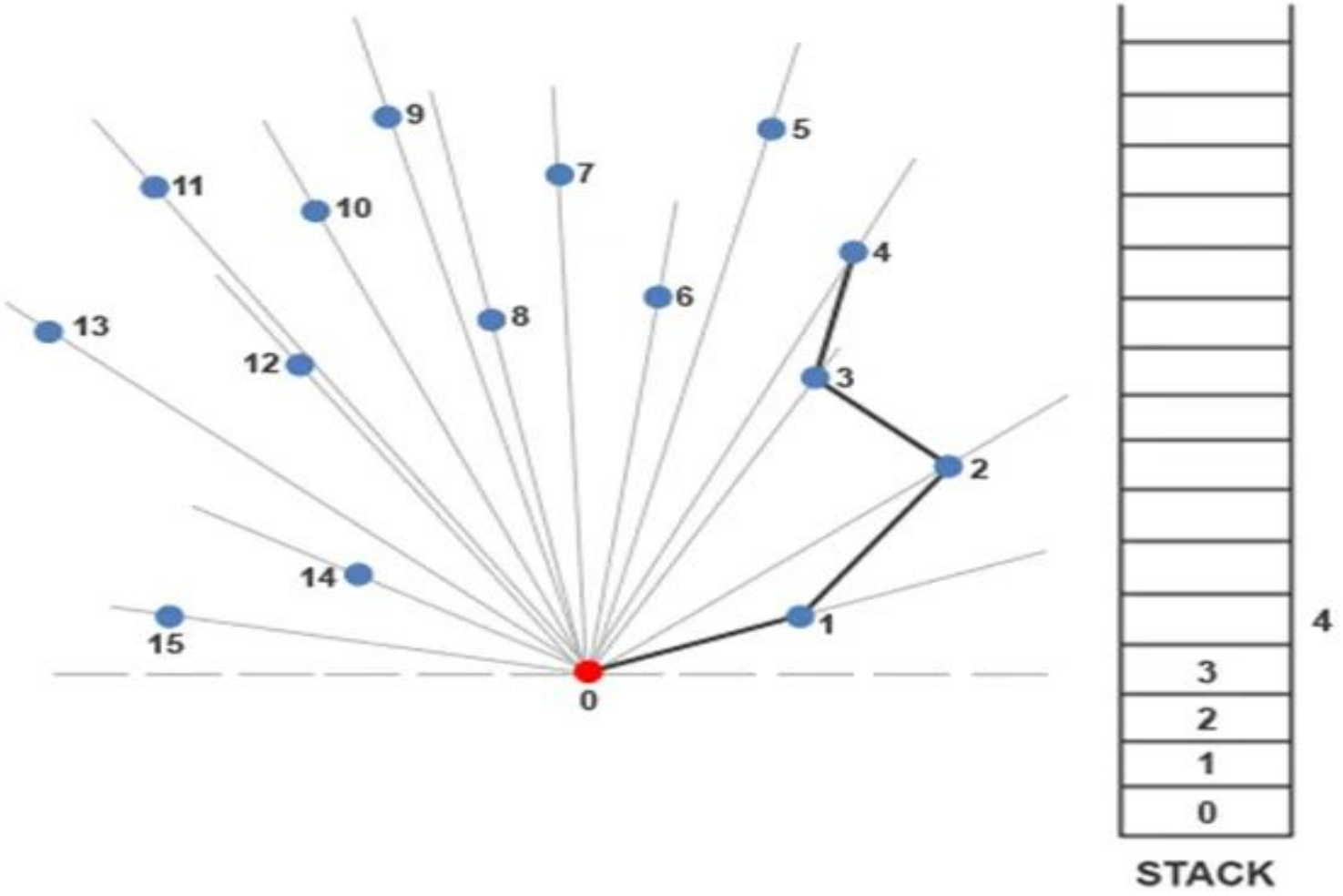
Whenever a left turn is made, the point is presumed to be part of the convex hull. We can clearly see a left turn being made to reach 2 from point 1. To get to point 3, another left turn is made. Currently, point 3 is part of the convex hull. A line segment is drawn from point 2 to 3 and 3 is pushed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



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

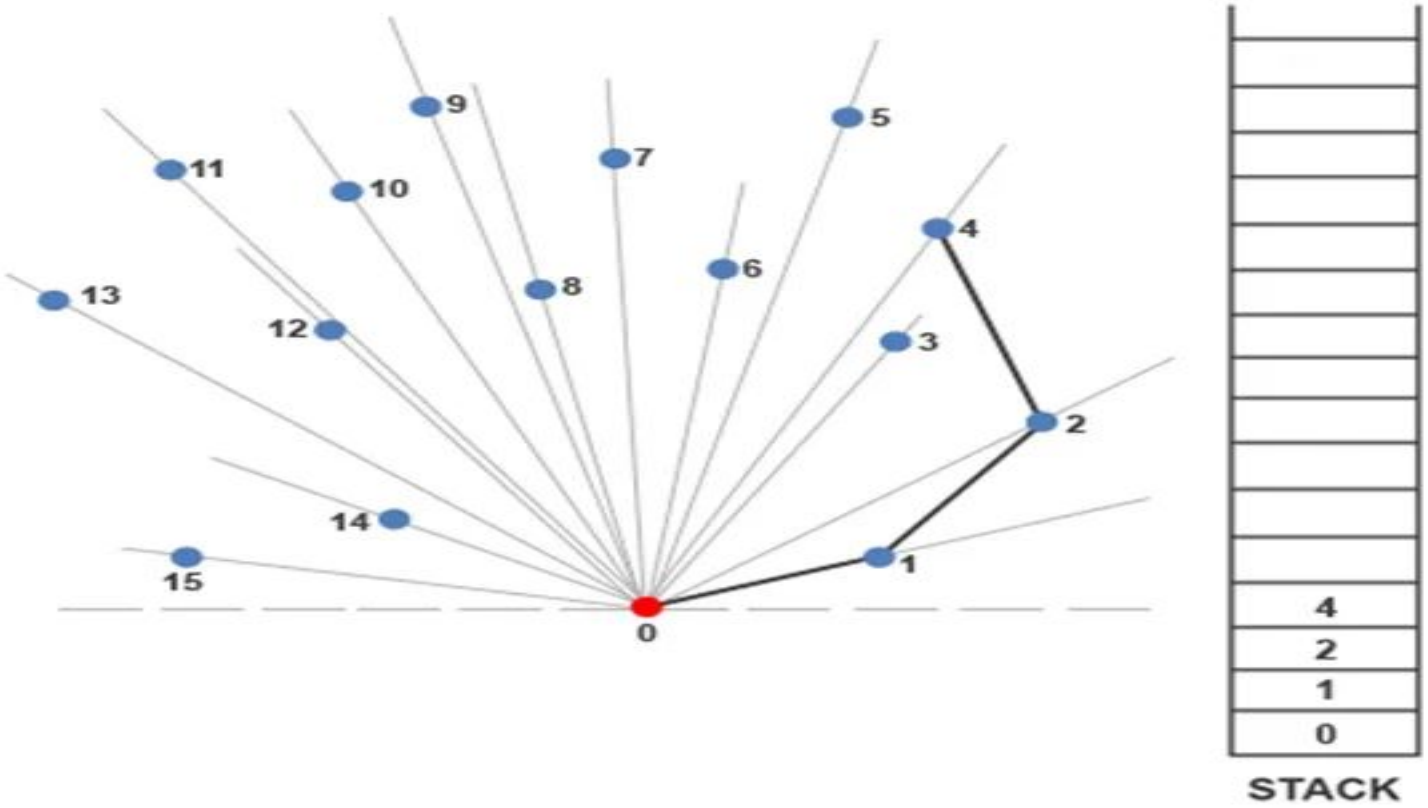
We make a right turn going to point 4. We'll draw the line to point 4 but will not push it onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

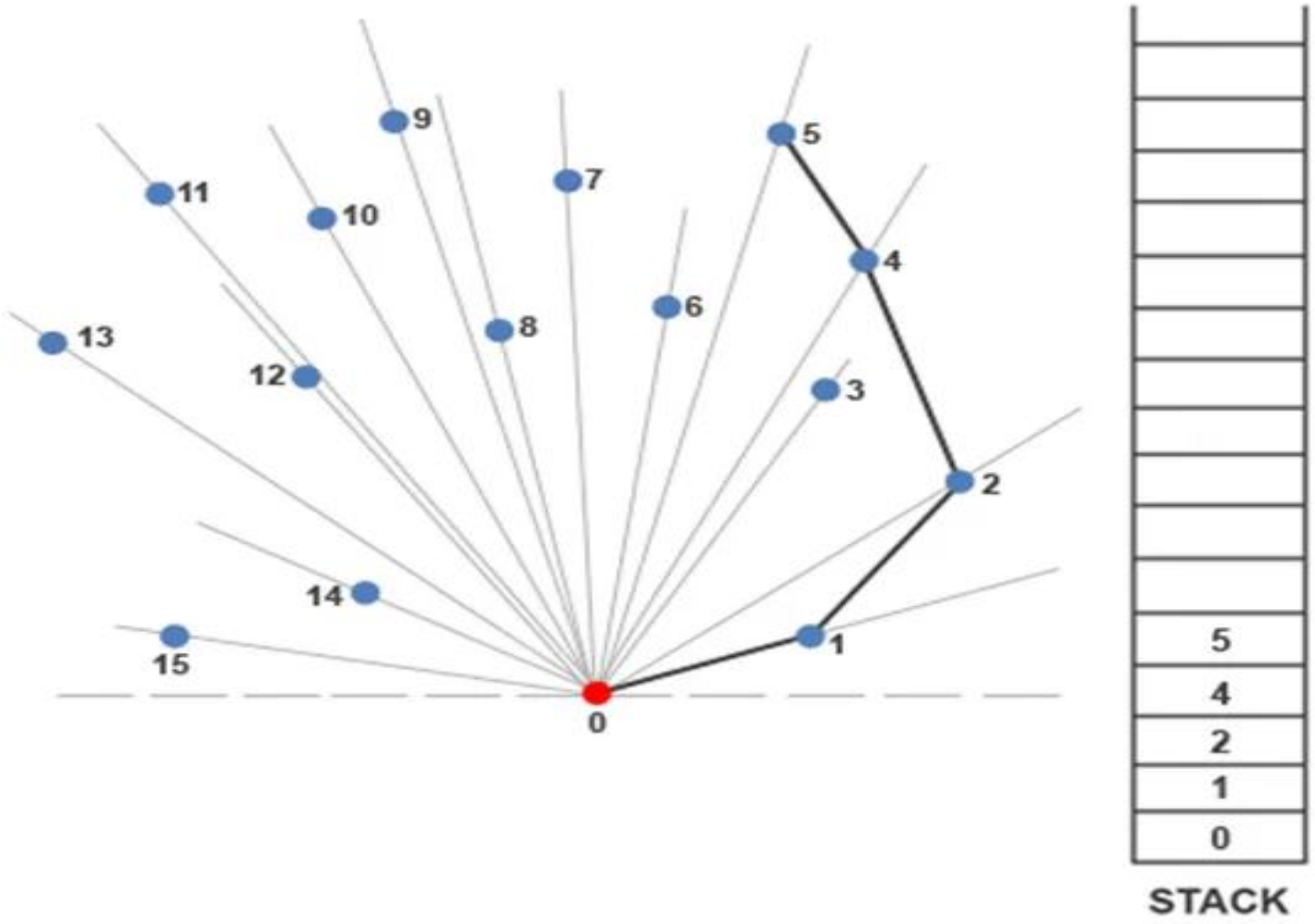
Whenever a right turn is made, Graham's scan algorithm pops the previous value from the stack and compares the new value with the top of the stack again. In this case, we'll pop 3 from the top of the stack and we'll see if going from point 2 to point 4 creates a left bend. In this case it does, so we'll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



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

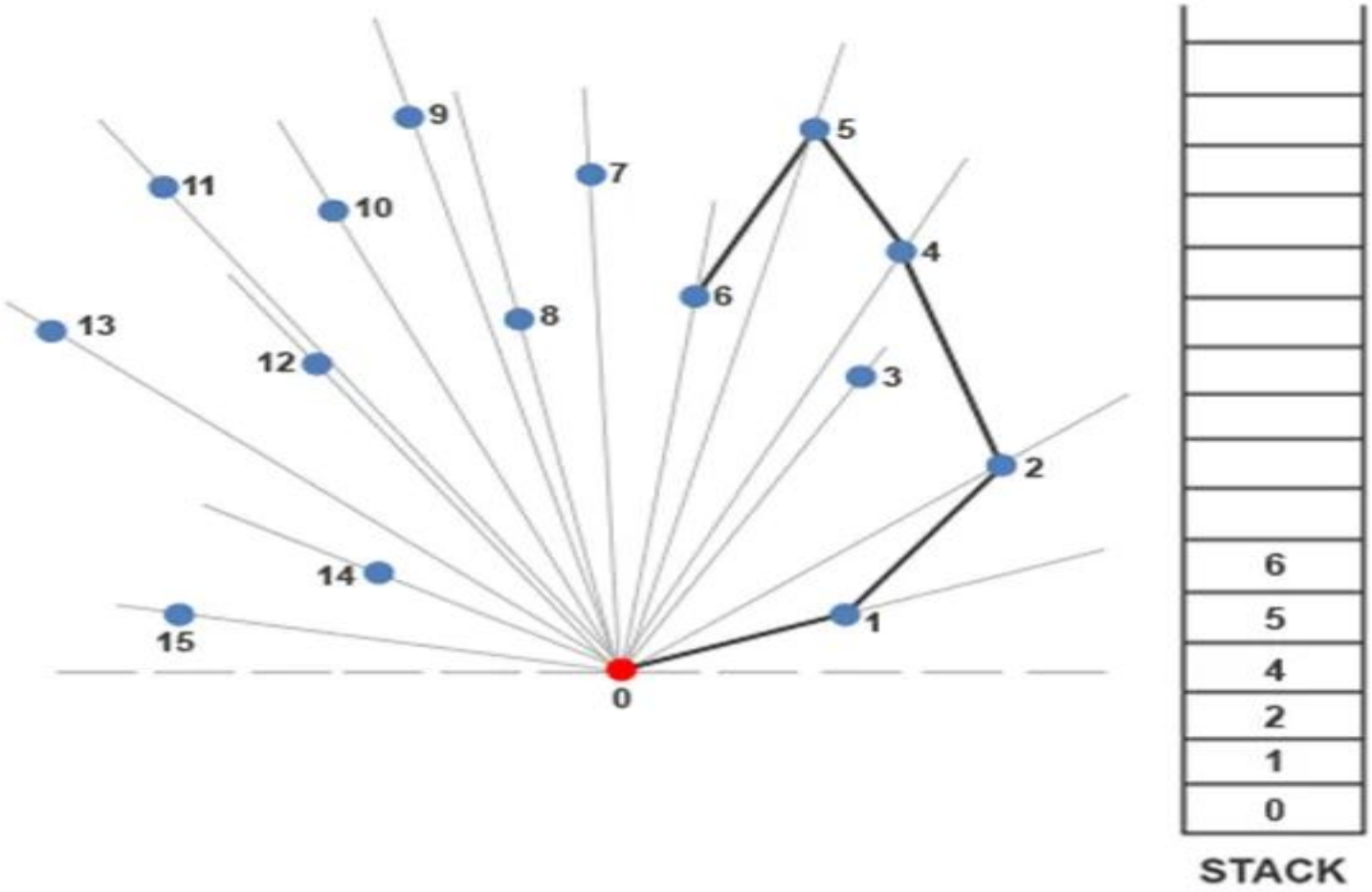
Since going from 4 to 5 creates a left turn, we'll push 5 onto the stack. Point 5 is currently part of the convex hull.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

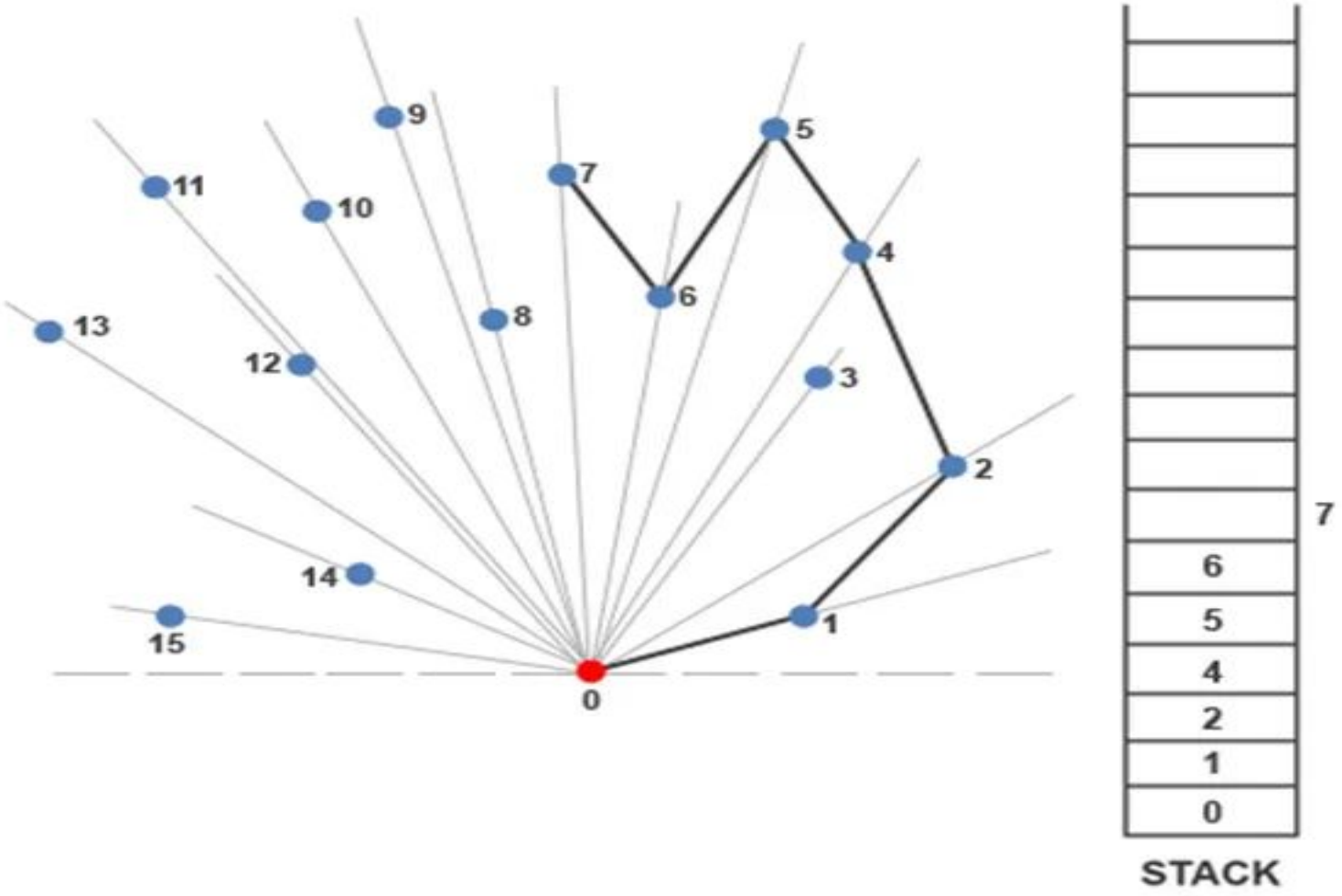
Moving from point 5 to 6 creates a left-hand turn, so we'll push 6 onto the stack. Point 6 is currently part of the convex hull.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

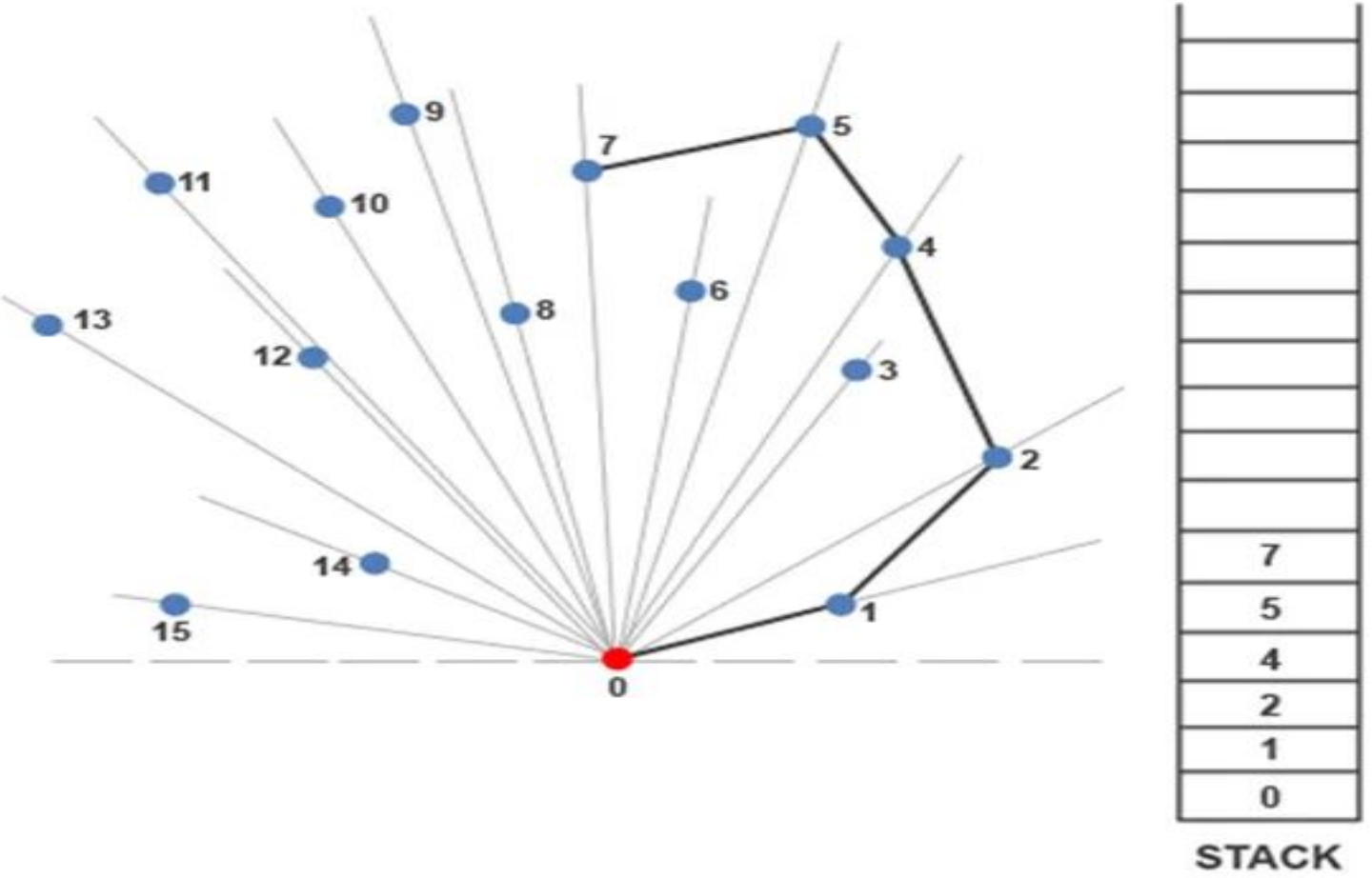
- To get to point 7, we must make a right-hand turn at 6.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

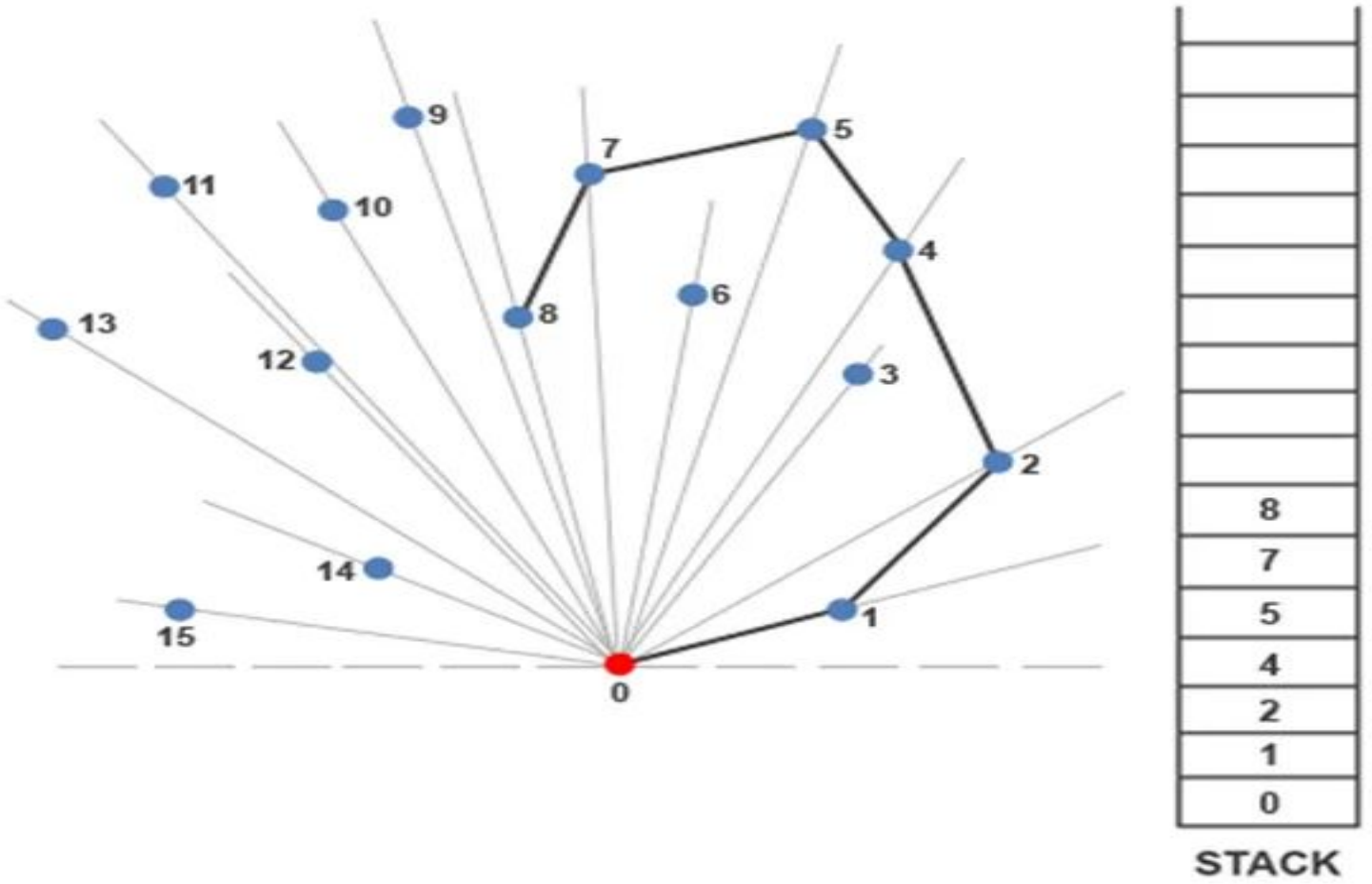
Point 6 is popped from the stack and the turn is examined from point 5 to point 7. Since we make a left hand turn from point 5 to point 7, we push point 7 onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

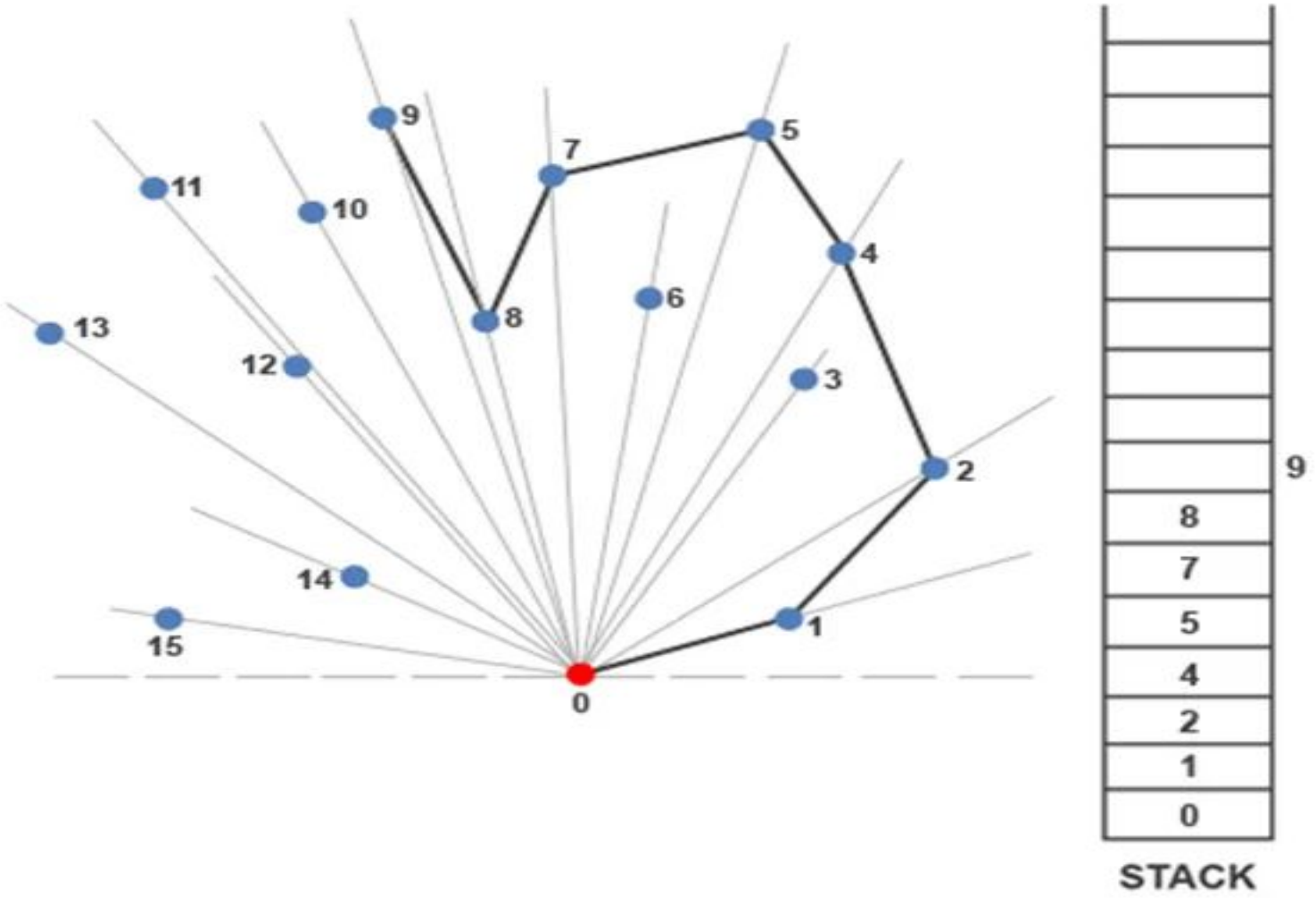
We attempt to push point 8 onto the stack. To get to point 8, we make a left at point 7, therefore point 8 is added to the stack. Point 8 is currently part of the convex hull.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

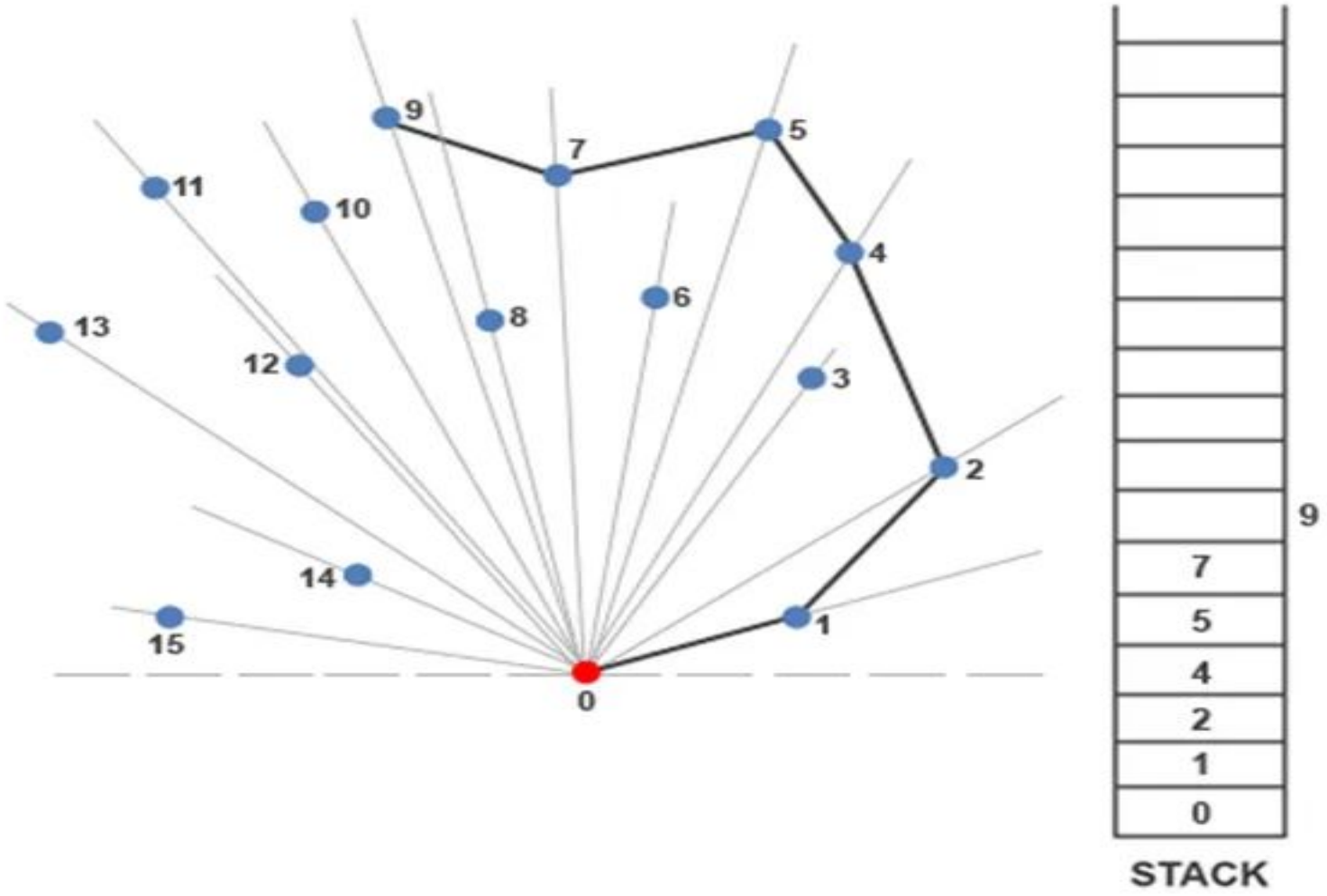
Going to point 9 requires a right-hand turn at point 8.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

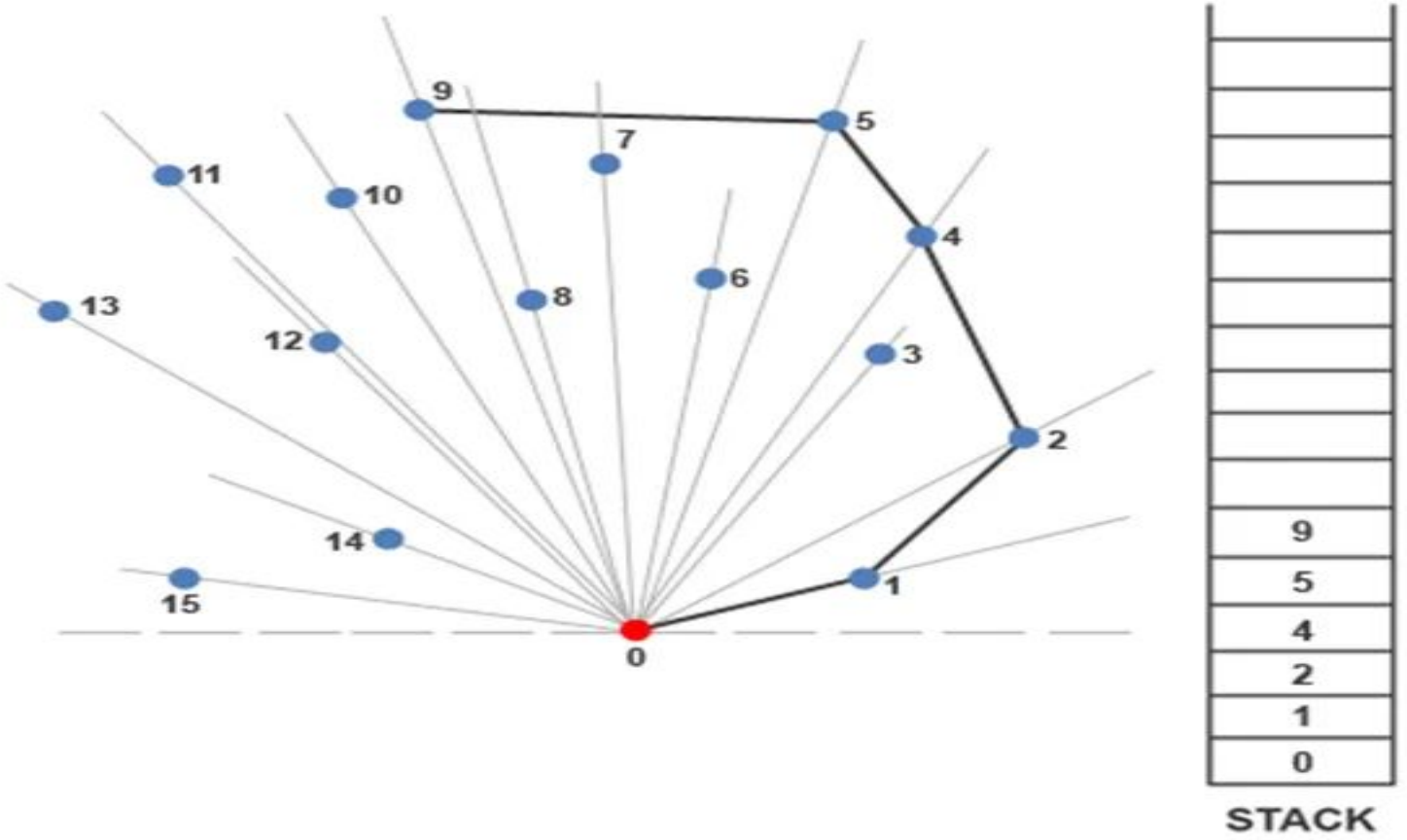
Since there's a right-hand turn, point 8 is popped from the stack and point 9 is compared with point 7.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

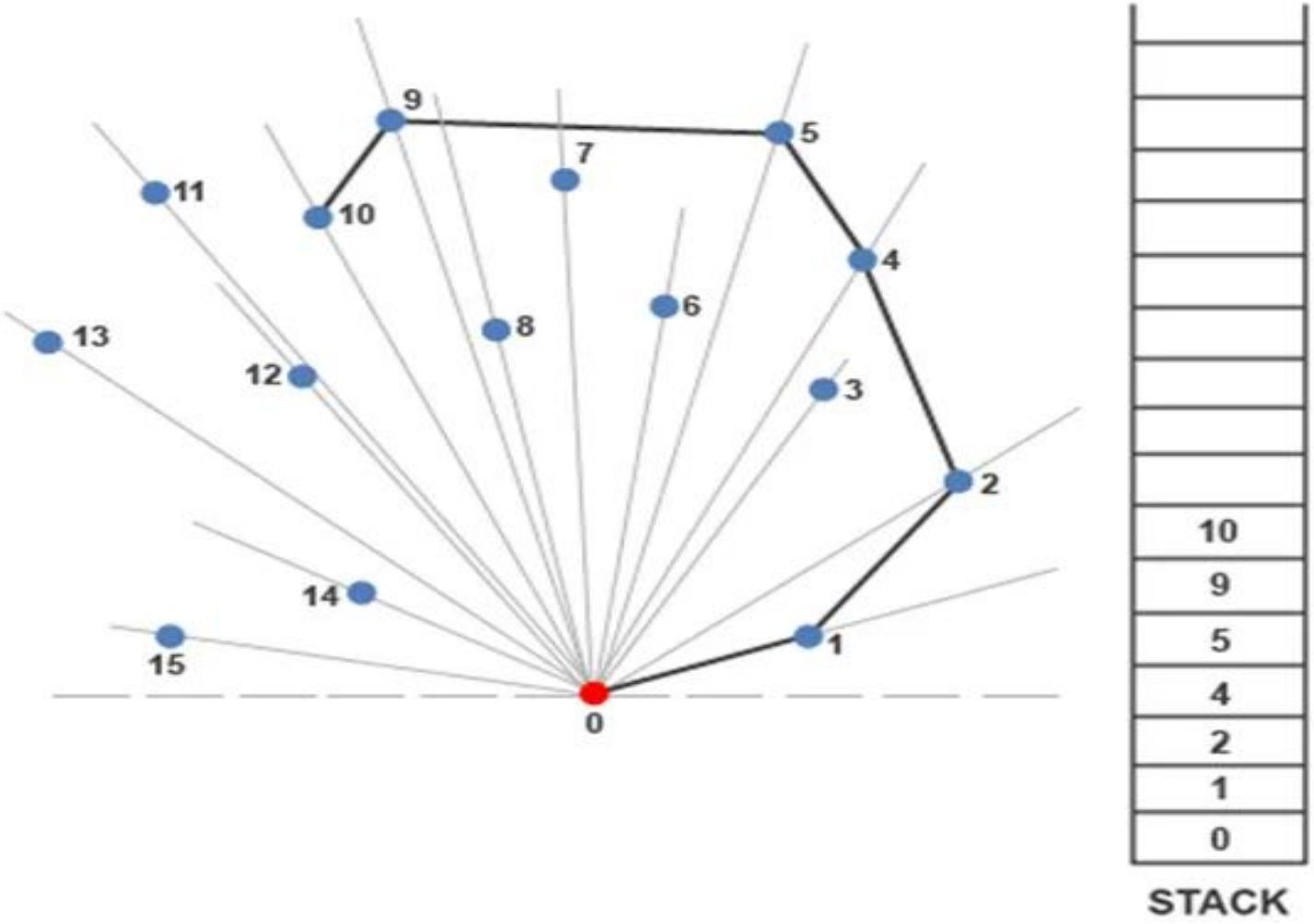
To get to point 9 from point 7 requires another right-turn, so we pop point 7 from the stack too and compare point 9 to point 5. We make a left-hand turn at point 5 to get to point 9, so 9 is pushed onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

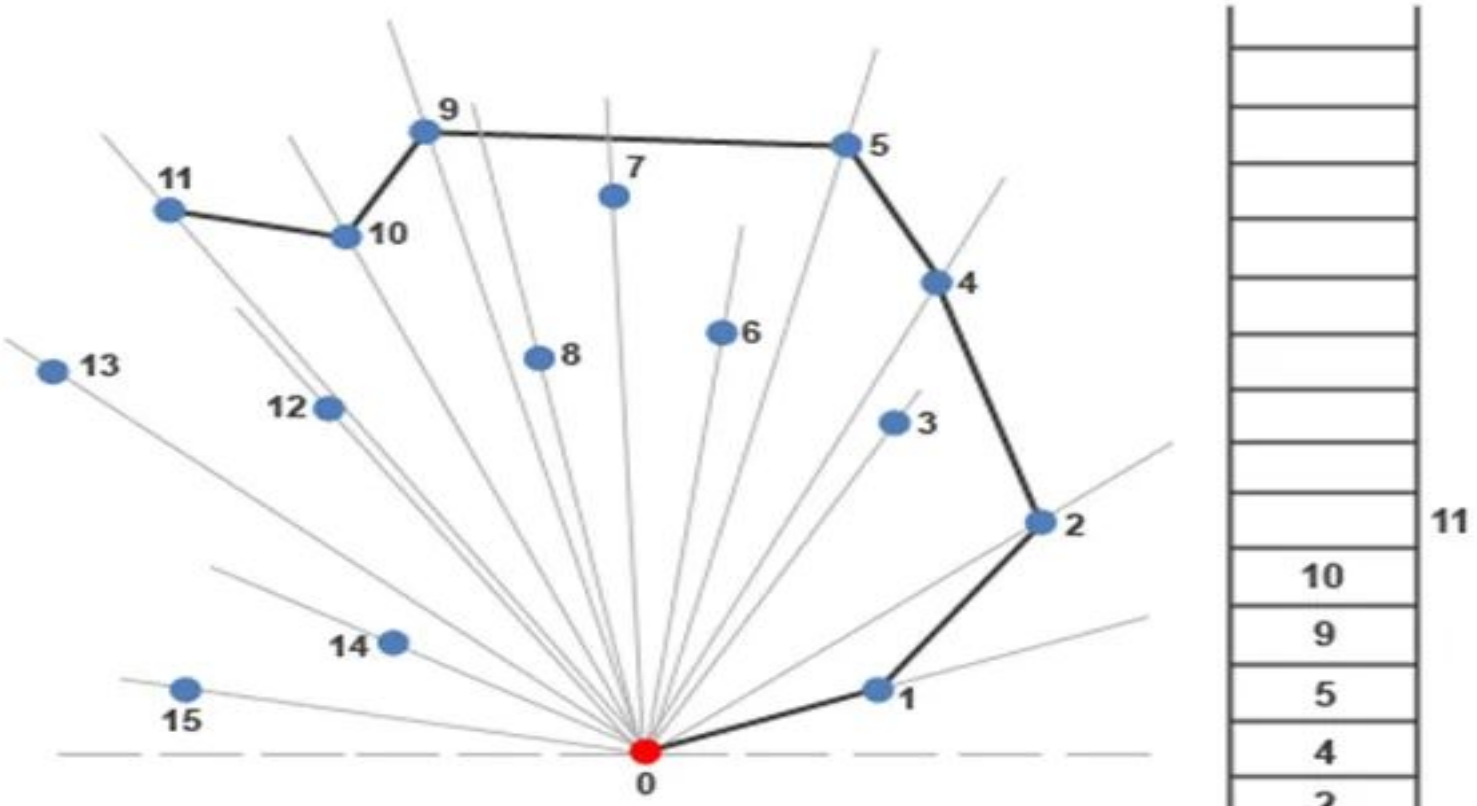
Next, we make a left turn to get to point 10. Point 10 is currently part of the convex hull.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

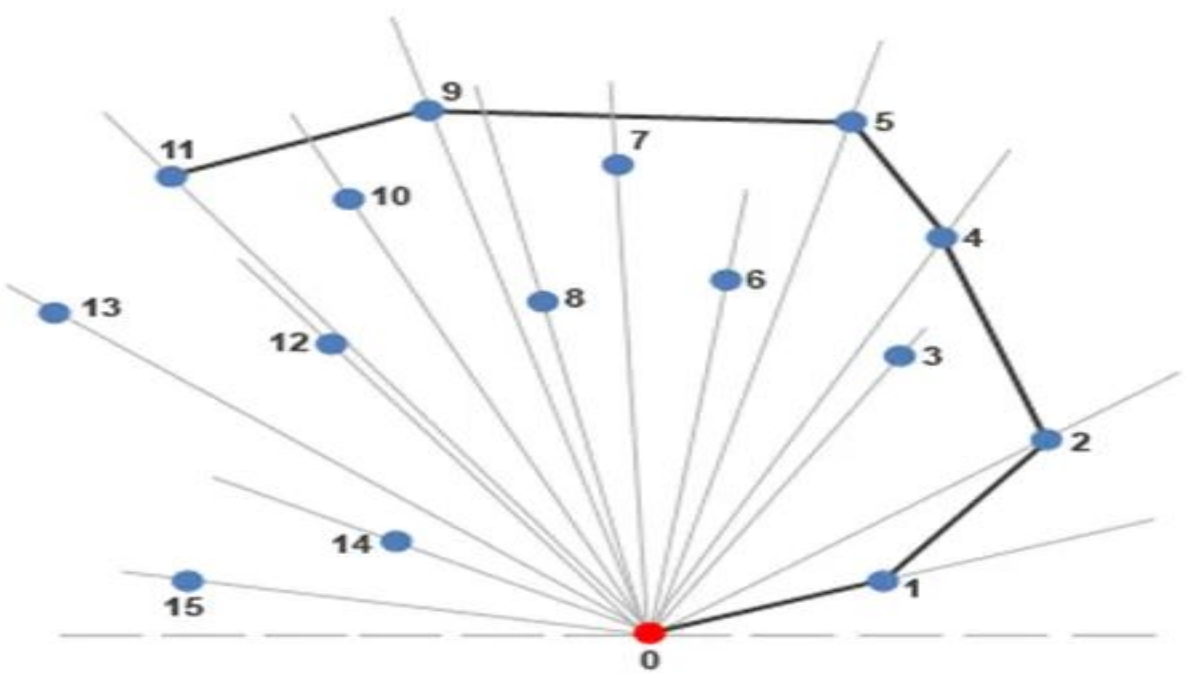
A right turn is required to get to point 11 from point 10.

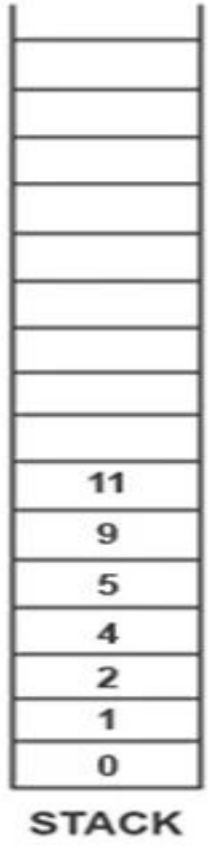


Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

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.

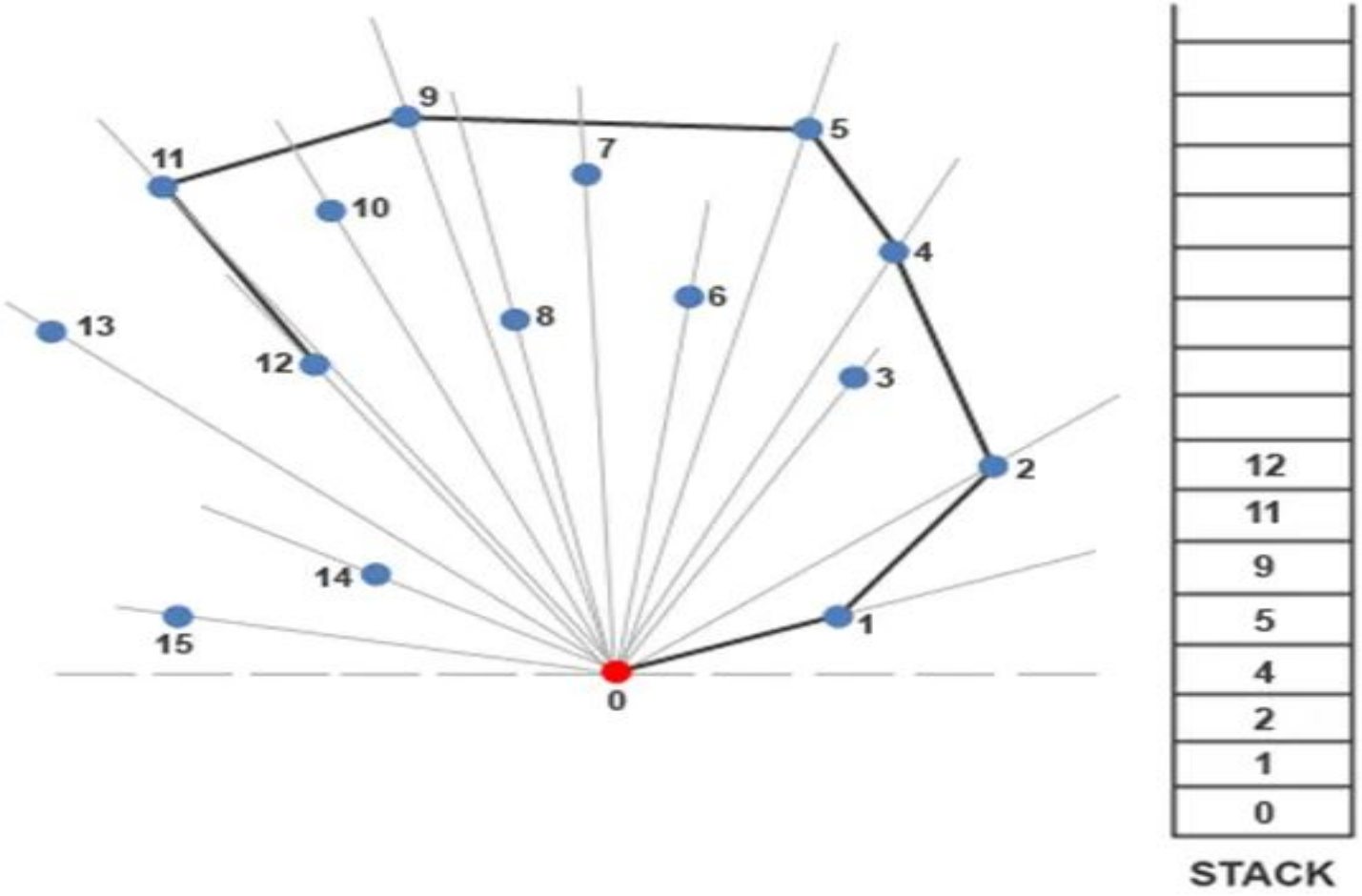




Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

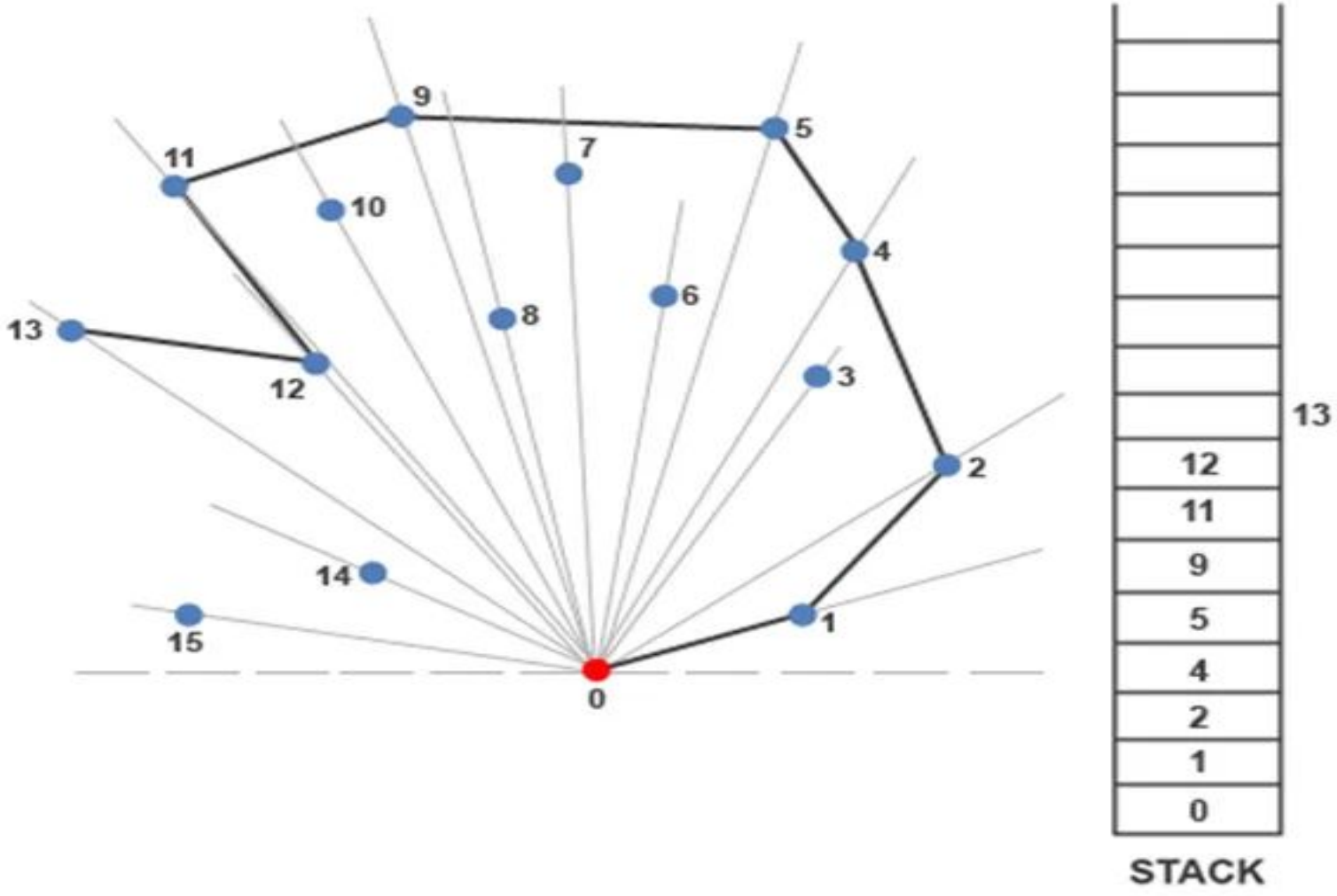
A left turn is made at point 11 to get to point 12. Point 12 is therefore pushed onto the stack and is currently considered part of the convex hull.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

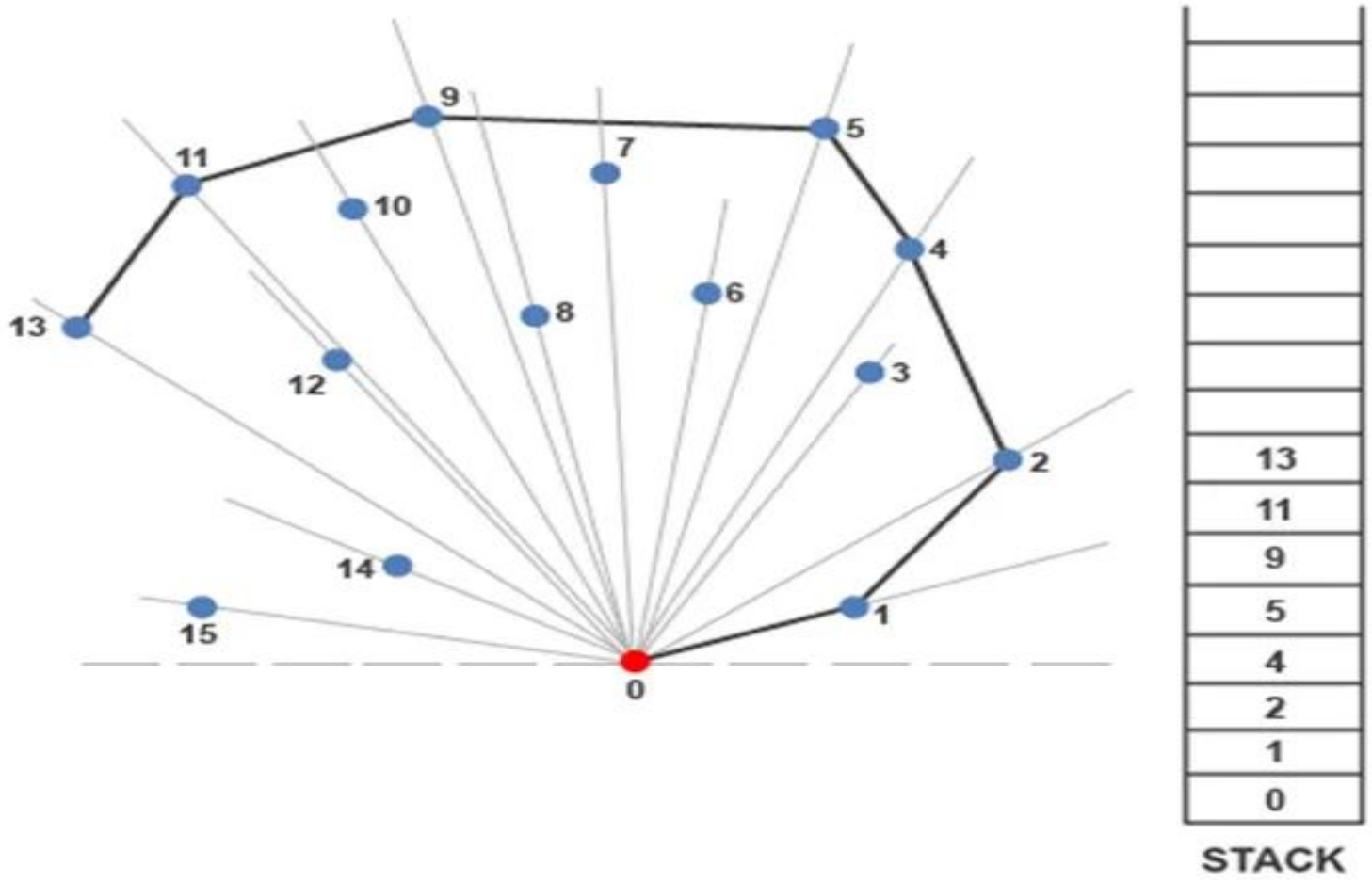
A right turn is required to go to point 13 from point 12.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

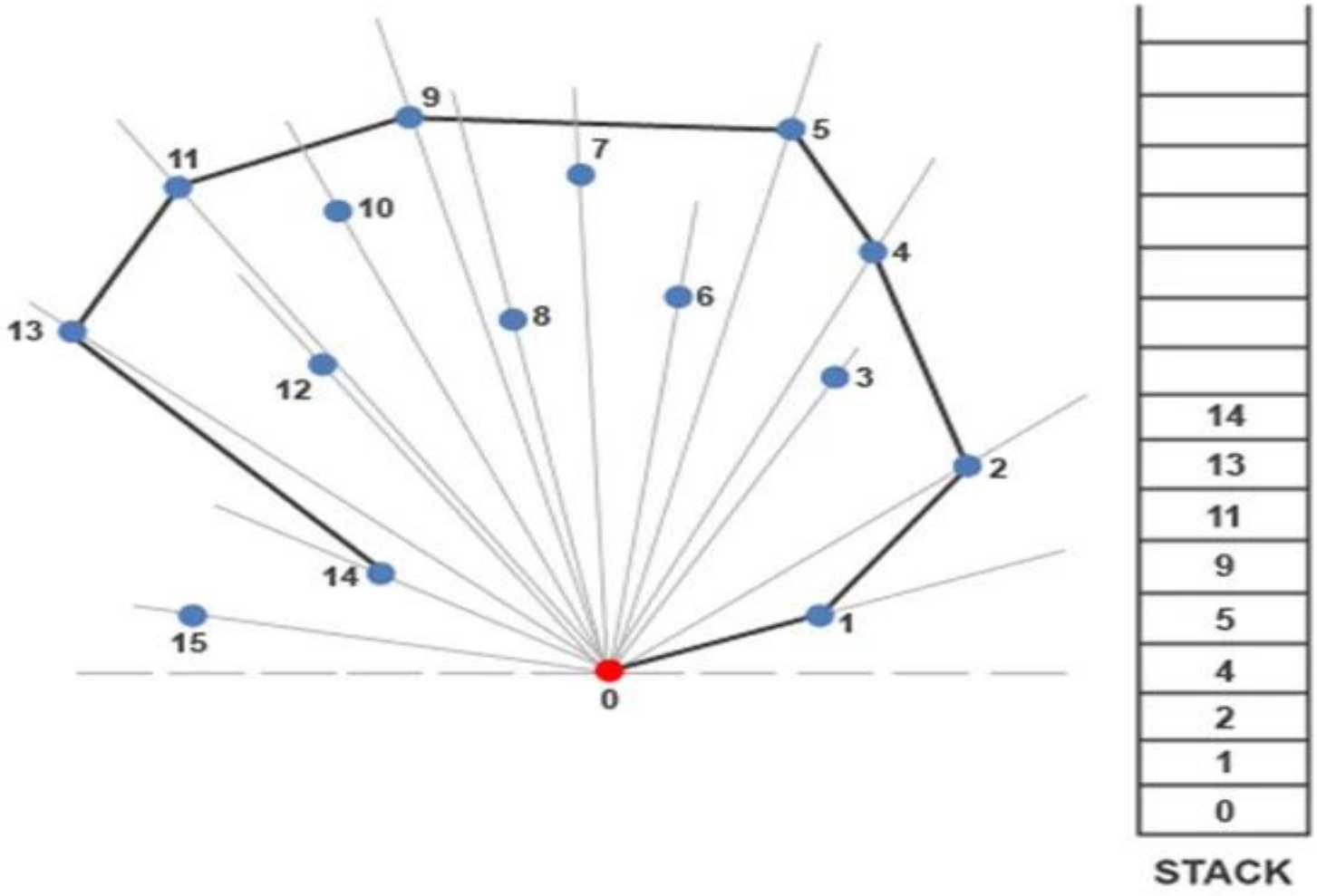
Point 12 is popped from the stack and the path to point 13 from point 11 is examined. Since a left turn is made at point 11, point 13 is pushed onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

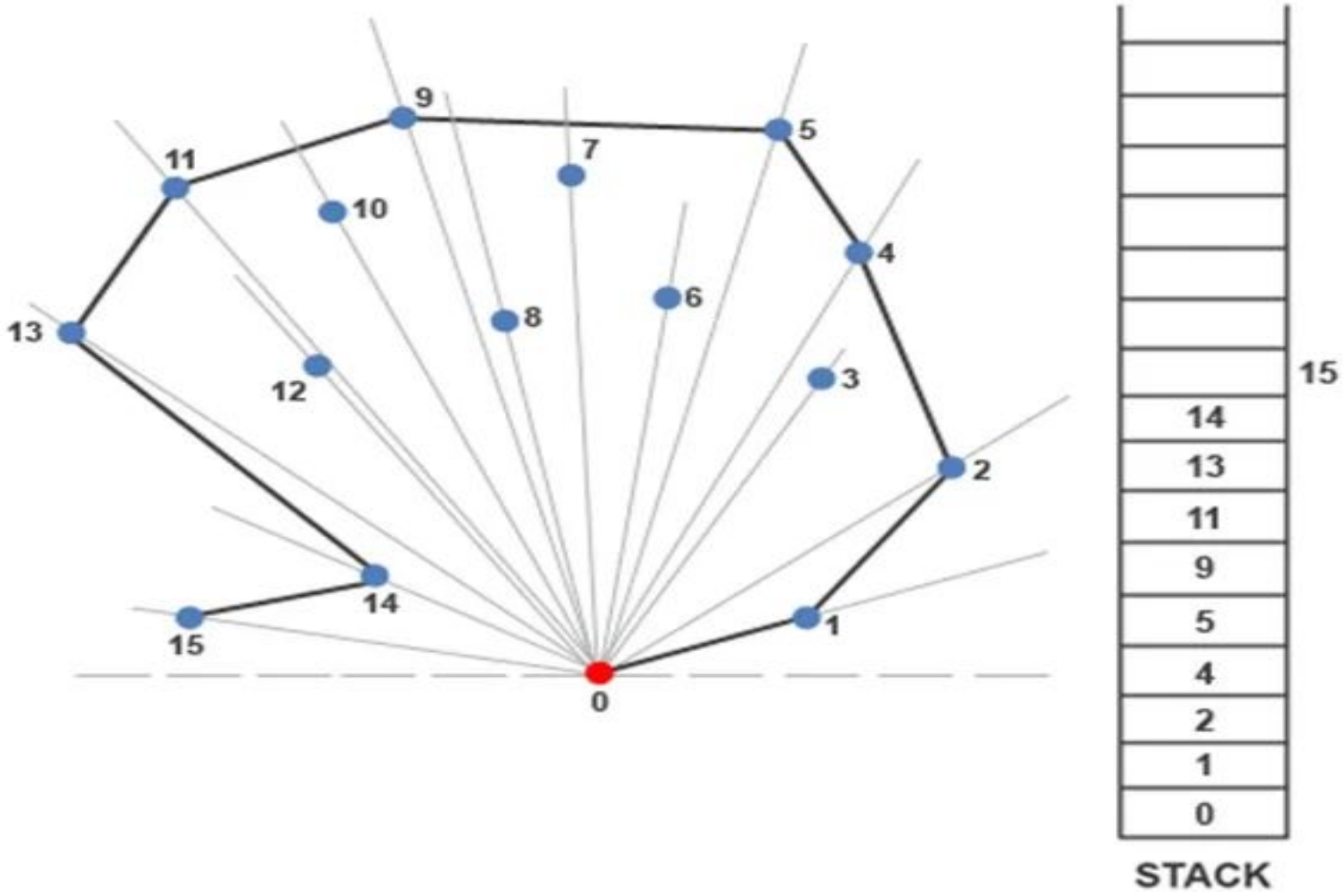
A left turn is made at point 13 to get to point 14, so point 14 is pushed onto the stack.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

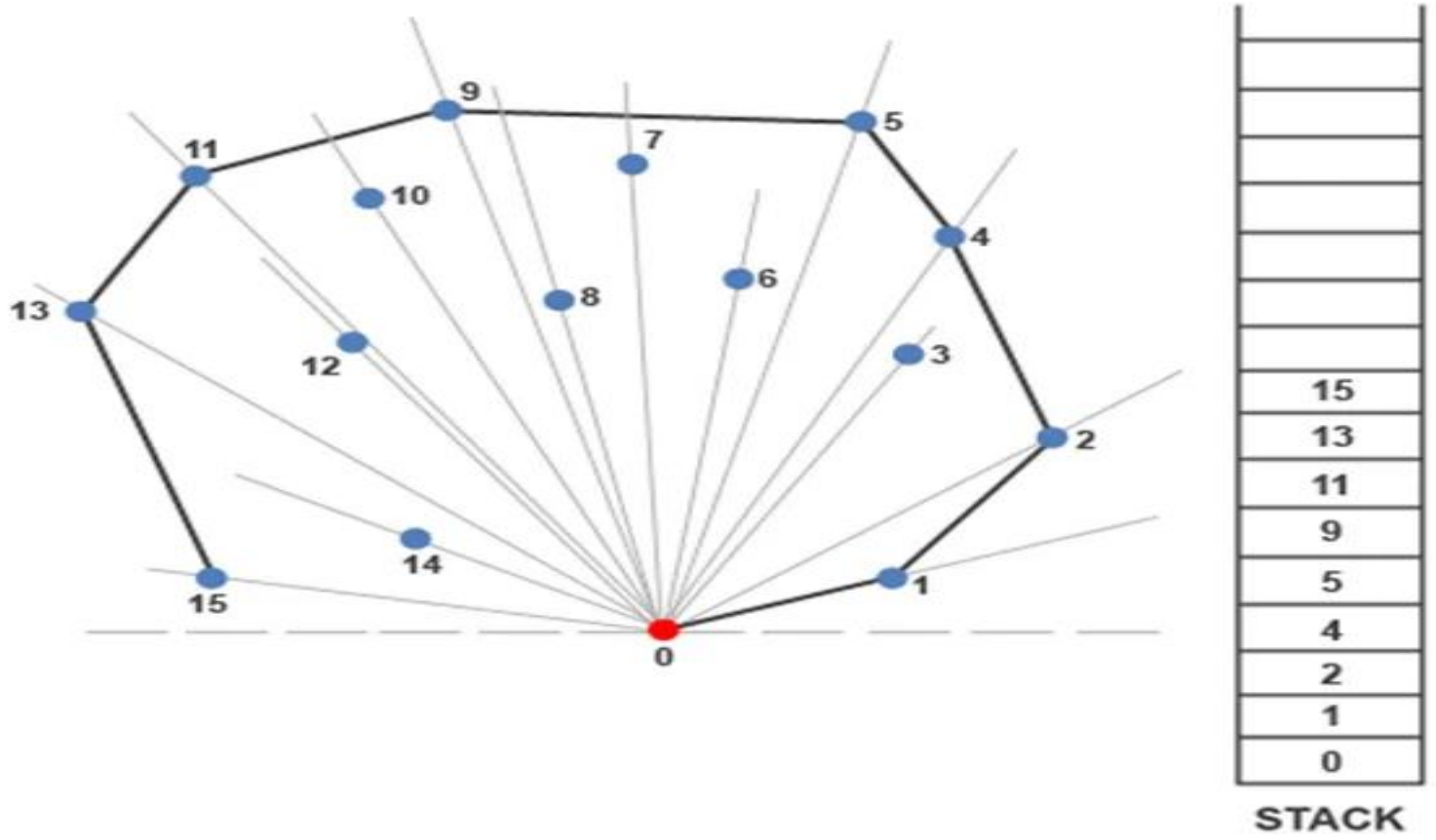
A right turn is required to go from point 14 to point 15.



Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

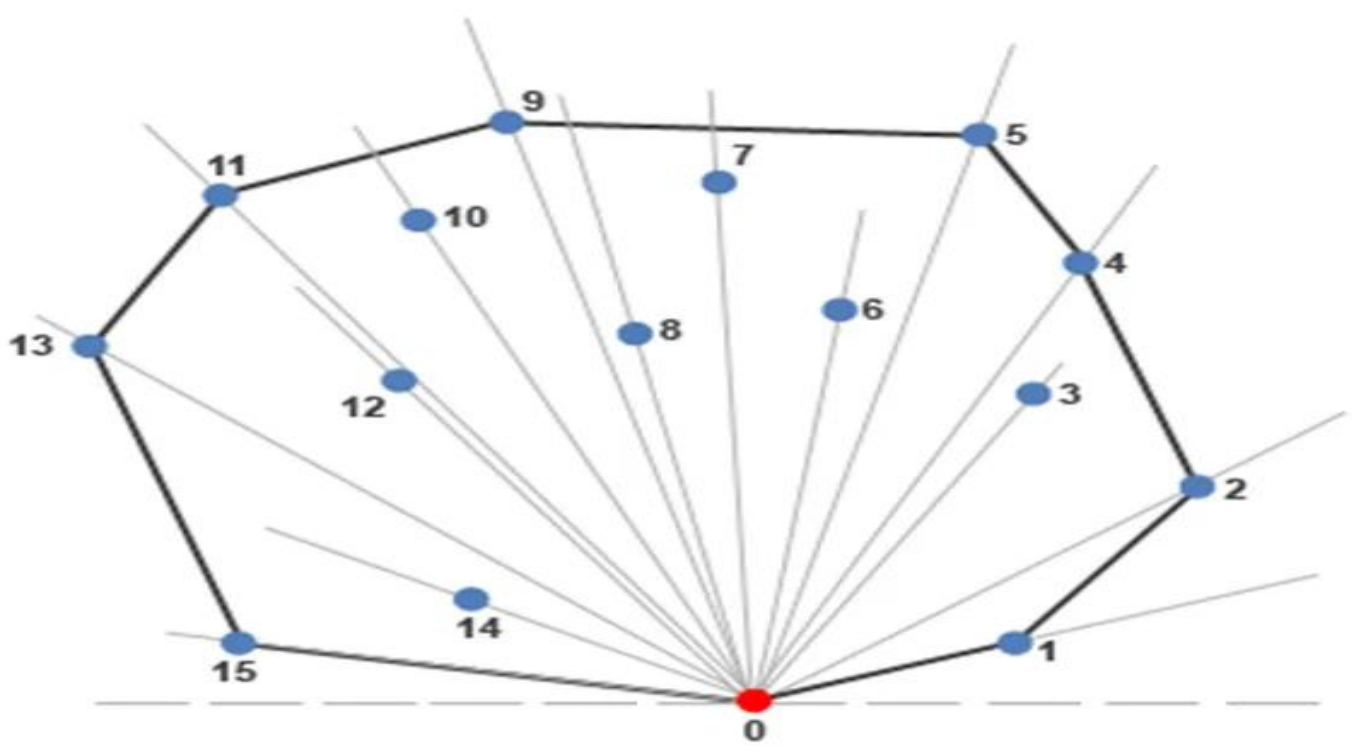
Since a right turn was made at point 14, point 14 is popped from the stack. The path to point 15 from point 13 is examined next. A left turn is made at point 13 to get to point 15, so point 15 is pushed onto the stack.

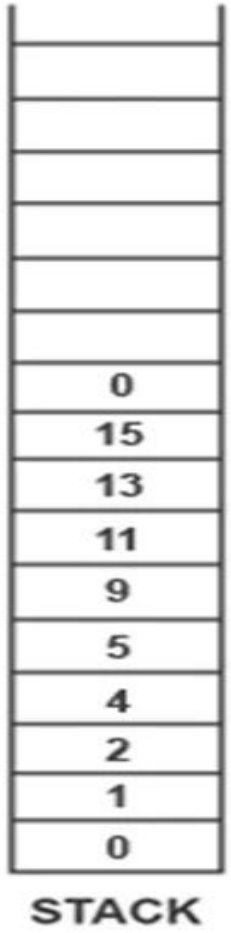


Computational Geometry (Convex Hull)

Graham scan Algorithm (Example)

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.





The points that are needed to create the convex hull are 0–1–2–4–5–9–11–13–15.

Computational Geometry (Convex Hull)

Graham scan Algorithm (Complexity)

Complexity:

- Sorting points by polar angle: O(nlogn)
- Constructing the hull: O(n)

Total Complexity: **O(nlogn)**

Jarvis's March (Gift Wrapping Algorithm) – Explore yourself

Jarvis's March is a straightforward algorithm that computes convex hull for a set of points. It relies
on the following two facts:

1. The leftmost point must be one vertex of the convex hull.
2. If point p is a vertex of the convex hull, then the points furthest clockwise and counter-clockwise
are also vertices of the convex hull.

Computational Geometry

Voronoi Diagram

Computational Geometry (Voronoi Diagram)

Voronoi Diagram

- A **Voronoi diagram** is a partitioning of a plane into regions based on the distance to a specific set of points.
- Each region, called a Voronoi cell, contains all the points closer to one particular point (called a site or generator) than to any other site

Key Concepts:

- **Voronoi Cell**: The region containing all points closer to a specific site than to any other site.
- **Duality with Delaunay Triangulation**: The Voronoi diagram is closely related to the Delaunay triangulation, where each edge of the triangulation is perpendicular to the edge of the Voronoi diagram

Applications:

- **Geographic Information Systems (GIS)**: Used for spatial analysis and mapping.
- **Computer Graphics**: Helps in texture mapping and mesh generation.
- **Urban Planning**: Assists in determining the optimal placement of facilities like schools, hospitals, and stores.
- **Biology**: Models the growth patterns of cells and tissues

Computational Geometry (Voronoi Diagram)

- A Voronoi diagram encodes proximity information, that is, what is close to what. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane (or in any dimensional space), which we call sites. Define V(p_i), the Voronoi cell for p_i, to be the set of points q in the plane that are closer to p_i than to any other site. Tha $\mathcal{V}(p_i) = \{q \mid \|p_i q\| < \|p_j q\|, \forall j \neq i\}$, is define $\|pq\| = \left(\sum_{i=j}^{d} (p_j - q_j)^2\right)^{1/2}$ where denotes the Euclidean distance between points p and q.
- The Voronoi diagram can be defined over any metric and in any dimension.
- Another way to define *V(pi)* is in terms of the intersection of halfplanes. Given two sites p_i and p_j, the set of points that are strictly closer to p_i than to p_j is just the open halfplane whose bounding line is the perpendicular bisector between p_i and p_j. Denote this halfplane h $\mathcal{V}(p_i) = \bigcap_{j \neq i} h(p_i, p_j)$. to see that a point q lies in V(p_i) if and only if q lies within the inter $h(p_i, p_j)$ for all $j \neq i$.
- In otherwords,

Computational Geometry (Voronoi Diagram)

- Since the intersection of halfplanes is a (possibly unbounded) convex polygon, it is easy to see that $V(p_i)$ is a (possibly unbounded) convex polygon.
- Finally, define the Voronoi diagram of P, denoted $Vor(P)$ to be what is left of the plane after we remove all the (open) Voronoi cells.
- The Voronoi diagram consists of a collection of line segments, which may be unbounded, either at one end or both.
- Voronoi Diagram can be use for answering nea neighbor queries, computational morphology and shape analysis, clustering and data minin facility location, multi-dimensional interpolati

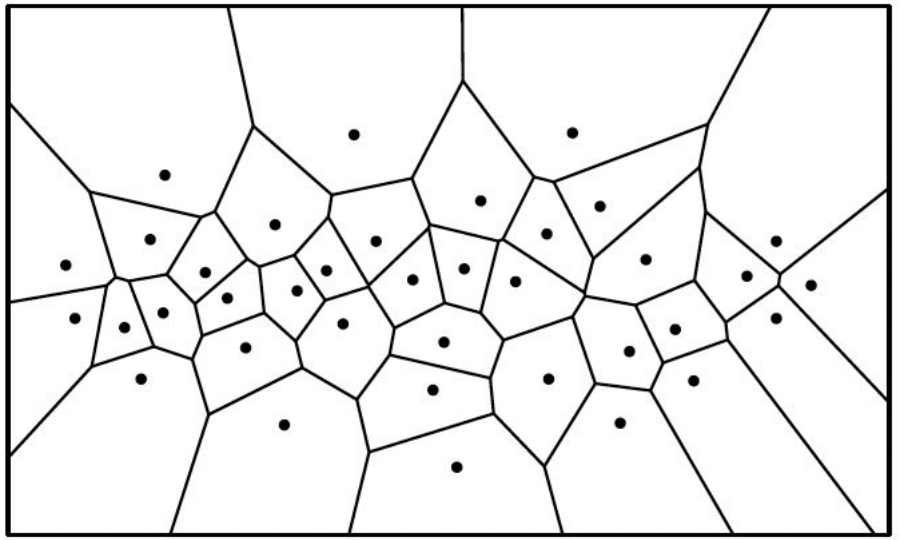
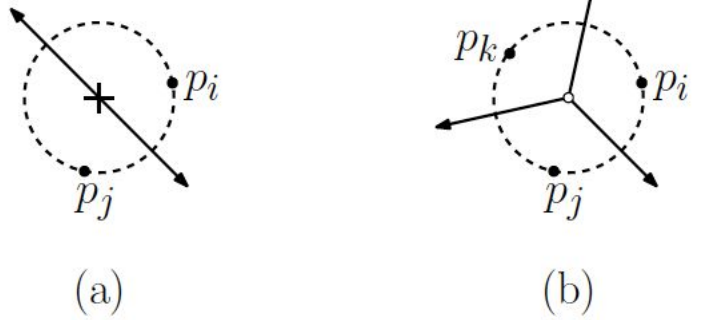


Fig: Voronoi diagram

Computational Geometry (Voronoi Diagram)

Properties of Voronoi Diagram:

- Voronoi complex:
  - Each point on an edge of the Voronoi diagram is equidistant from its two nearest neighbors p_i and p_j. Thus, there is a circle centered at such a point such that p_i and p_j lie on this circle, and no other site is interior to the circle (Fig. a)
- Voronoi vertices:
  - It follows that the vertex at which three Voronoi cells $V(p_i)$, $V(p_j)$, and $V(p_k)$ intersect, called a Voronoi vertex is equidistant from all sites (Fig. b).
  - Thus it is the center of the circle passing through these sit and this circle contains no other sites in its interior.



- Degree:
  - Generally three points in the plane define a unique circle. If we make the general position assumption that no four sites are cocircular, then the vertices of the Voronoi diagram all have degree three.

Computational Geometry (Voronoi Diagram)

- Convex hull:
  - A cell of the Voronoi diagram is unbounded if and only if the corresponding site lies on the convex hull.
  - Thus, given a Voronoi diagram, it is easy to extract the convex hull in linear time.
- Size:
  - If n denotes the number of sites, then the Voronoi diagram is a planar graph (if we imagine all the unbounded edges as going to a common vertex infinity) with exactly n faces. It follows from Euler's formula that the number of Voronoi vertices is roughly *2n* and the number of edges is roughly *3n*.

Computational Geometry (Voronoi Diagram)

Construction of Voronoi Diagram [Most common algorithms are]

1. **Incremental Construction Algorithm**
   - Complexity: $O(n^2)$ (average case O(nlogn) with proper data structures)
2. **Fortune's Sweep Line Algorithm**
   - Complexity: O(nlogn)
3. **Divide and Conquer Algorithm**
   - Complexity: O(nlogn)
4. **Lloyd's Algorithm (Iterative Refinement)**
   - Complexity: Dependent on iterations; often $O(n^2)$
5. **Brute Force Algorithm**
   - Complexity: $O(n^2)$
6. **Dual Delaunay Triangulation Approach**
   - Complexity: O(nlogn)

Construction of Voronoi Diagram(Incremental Approach)

- **Input :** A set of points $P=\{p_1,p_2,\ldots,p_n\}$ in a 2D plane.
- **Output :** A Voronoi diagram representing the partitioning of the plane based on the input points.

Algorithm

Step 1: Initialization

1. **Start with the first point** p_1:
   1. Since there is only one point, the entire plane belongs to p_1.
   2. Initialize the diagram as a single cell for p_1.
2. **Add the second point** p_2:
   1. Compute the perpendicular bisector of the line segment p_1p_2.
   2. This bisector divides the plane into two regions, one for $p_1$and one for p_2.
   3. Update the diagram with these two cells.

Construction of Voronoi Diagram(Incremental Approach)

Step 2: Iterative Addition

- For each subsequent point p_k (k=3,4,…,n):

1. **Identify affected cells**:
   - Determine the Voronoi cells that will be modified by the addition of p_k. These are the cells for points closer to p_k than to their current boundaries.
2. **Compute new edges**:
   - For each affected cell, compute the perpendicular bisector of the line segment between p_k and the cell's generating point.
   - The intersection of these bisectors defines the boundary of the new cell for p_k.
4. **Clip the new cell**:
   - The new Voronoi cell for p_k is bounded by:
     - The edges from the perpendicular bisectors.
     - Any existing Voronoi edges that are closer to other points.
4. **Update affected cells**:
   - For each affected cell, modify its boundaries to include the new edges from p_k.

Construction of Voronoi Diagram(Incremental Approach)

Step 3: Edge Removal

- Remove edges that no longer belong to the Voronoi diagram (i.e., those replaced by new edges from p_k).

Step 4: Final Diagram

- After all points have been added, finalize the diagram by trimming or extending unbounded edges as necessary (e.g., using bounding boxes).

- **How to calculate perpendicular bisector ? (Next Slide)**

1. **Find the midpoint** of p_1 and p_2:

$$\text{Midpoint} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

2. **Calculate the slope of the line** connecting p_1 and p_2:

$$\text{slope of line } p_1p_2 = \frac{y_2 - y_1}{x_2 - x_1}$$

(This is the **direct slope** between the two points.)

3. **Find the slope of the perpendicular bisector**: The slope of the perpendicular bisector is the **negative reciprocal** of the slope of the line joining p_1 and p_2:

$$\text{slope of perpendicular bisector} = -\frac{1}{\text{slope of line } p_1p_2}$$

If the line connecting p_1 and p_2 is vertical (i.e., $x_1 = x_2$), the slope of the perpendicular bisector will be **horizontal**, i.e., 0. Similarly, if the line is horizontal, the perpendicular bisector will be vertical.

4. **Equation of the perpendicular bisector**: Using the midpoint and the slope of the perpendicular bisector, the equation of the perpendicular bisector can be written in point-slope form:

$$y - \frac{y_1 + y_2}{2} = m_\perp \left(x - \frac{x_1 + x_2}{2}\right)$$

where $m_\perp$ is the slope of the perpendicular bisector.

Source: Chatgpt

Construction of Voronoi Diagram(Incremental Approach)

Example

Example

Let's say we have two points $p_1(1, 1)$ and $p_2(4, 5)$. To find the perpendicular bisector:

1. **Midpoint**:

$$\text{Midpoint} = \left(\frac{1+4}{2}, \frac{1+5}{2}\right) = (2.5, 3)$$

2. **Slope of p_1p_2**:

$$\text{slope of } p_1p_2 = \frac{5-1}{4-1} = \frac{4}{3}$$

3. **Slope of the perpendicular bisector**:

$$m_\perp = -\frac{1}{\frac{4}{3}} = -\frac{3}{4}$$

4. **Equation of the perpendicular bisector**: Using the midpoint $(2.5, 3)$ and the slope $-\frac{3}{4}$, the equation is:

$$y - 3 = -\frac{3}{4}(x - 2.5)$$

↓

Now, this bisector can be used to divide the plane into two regions: one for p_1 and one for p_2.

Construction of Voronoi Diagram(Incremental Approach)

Illustration

P={(2,3),(5,8),(1,7),(6,2)
}

Step 1: Start with the first point $(2, 3)$

- Initially, we have only one point, $p1 = (2, 3)$, and the entire plane is assigned to this point.
- **Diagram**: The diagram will consist of a single region that covers the entire plane and belongs to $p1$.

Step 2: Add the second point $(5, 8)$

- **Step 2.1: Compute the perpendicular bisector** between $(2, 3)$ and $(5, 8)$.
  - Midpoint:
$$\left(\frac{2+5}{2}, \frac{3+8}{2}\right) = (3.5, 5.5)$$
  - Slope of line $p1p2$:
$$\frac{8-3}{5-2} = \frac{5}{3}$$
  - Slope of the perpendicular bisector (negative reciprocal):
$$-\frac{3}{5}$$
  - Equation of the perpendicular bisector:
$$y - 5.5 = -\frac{3}{5}(x - 3.5)$$
    Simplifying this equation:
$$y = -\frac{3}{5}(x - 3.5) + 5.5$$
- **Step 2.2: Update the diagram** by adding the perpendicular bisector.
  - The bisector divides the plane into two regions: one for $p1 = (2, 3)$ and one for $p2 = (5, 8)$.

Construction of Voronoi Diagram(Incremental Approach)

Step 3: Add the third point $(1, 7)$

- **Step 3.1: Compute the perpendicular bisector** between $(2, 3)$ and $(1, 7)$.
  - Midpoint:

$$\left(\frac{2+1}{2}, \frac{3+7}{2}\right) = (1.5, 5)$$

  - Slope of line p_1p_3:

$$\frac{7-3}{1-2} = \frac{4}{-1} = -4$$

  - Slope of the perpendicular bisector:

$$\frac{1}{4}$$

  - Equation of the perpendicular bisector:

$$y - 5 = \frac{1}{4}(x - 1.5)$$

    Simplifying this equation:

$$y = \frac{1}{4}(x - 1.5) + 5$$

- **Step 3.2: Compute the perpendicular bisector** between $(5, 8)$ and $(1, 7)$.
  - Midpoint:

$$\left(\frac{5+1}{2}, \frac{8+7}{2}\right) = (3, 7.5)$$

  - Slope of line p_2p_3:

$$\frac{7-8}{1-5} = \frac{-1}{-4} = \frac{1}{4}$$

  - Slope of the perpendicular bisector:

$$-4$$

  - Equation of the perpendicular bisector:

$$y - 7.5 = -4(x - 3)$$

    Simplifying this equation:

$$y = -4(x - 3) + 7.5$$

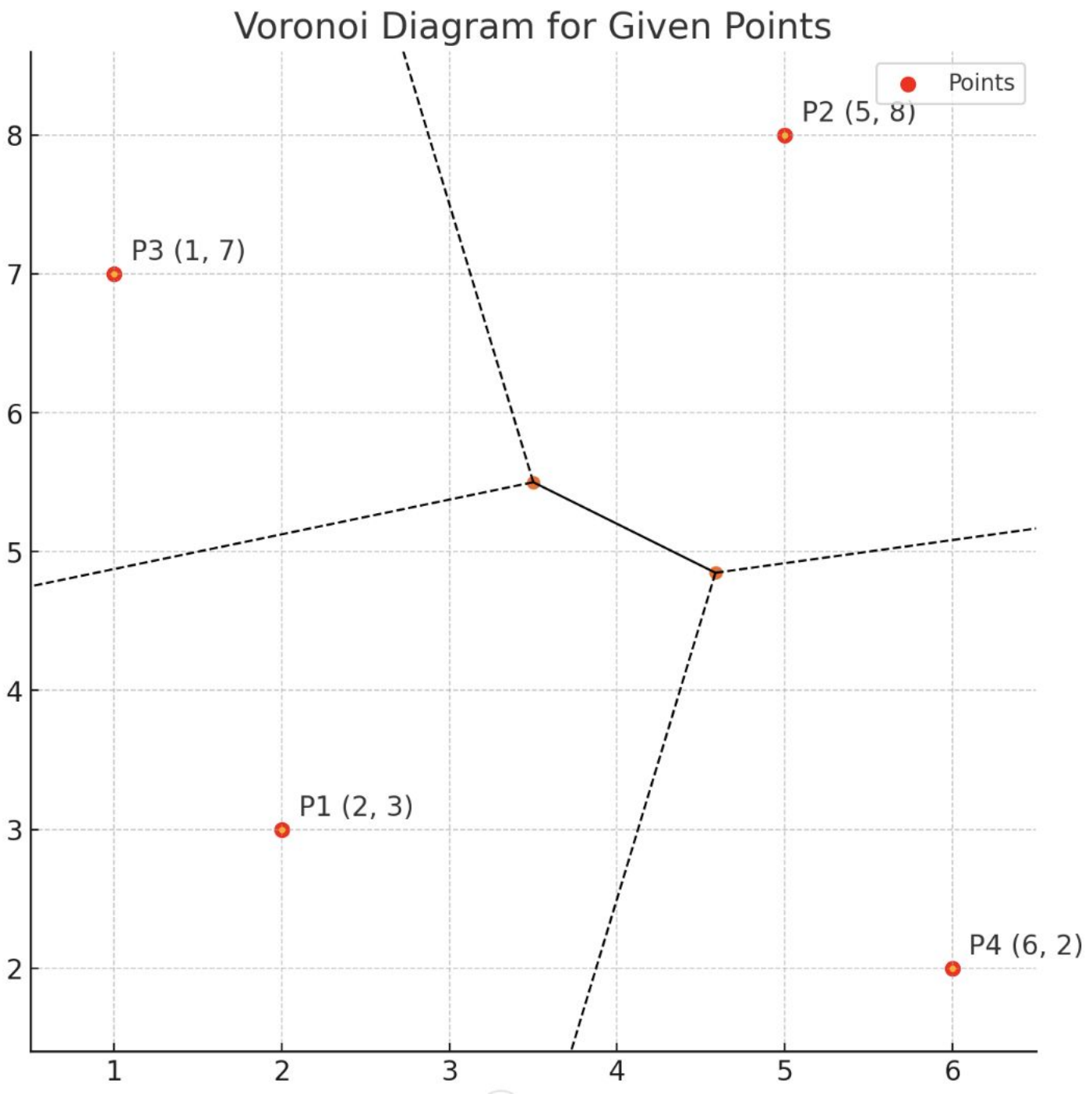
- **Step 3.3: Update the diagram** by adding the two new bisectors.
  - The plane is now divided into three regions: one for each of $p_1 = (2, 3)$, $p_2 = (5, 8)$, and $p_3 = (1, 7)$.

Construction of Voronoi Diagram(Incremental Approach)

Step 4: Add the fourth point $(6, 2)$

- **Step 4.1: Compute the perpendicular bisector** between $(2, 3)$ and $(6, 2)$.
  - Midpoint:
$$\left(\frac{2+6}{2}, \frac{3+2}{2}\right) = (4, 2.5)$$
  - Slope of line $p1p4$:
$$\frac{2-3}{6-2} = \frac{-1}{4}$$
  - Slope of the perpendicular bisector:
$$4$$
  - Equation of the perpendicular bisector:
$$y - 2.5 = 4(x - 4)$$
    Simplifying this equation:
$$y = 4(x - 4) + 2.5$$
- **Step 4.2: Compute the perpendicular bisector** between $(5, 8)$ and $(6, 2)$.
  - Midpoint:
$$\left(\frac{5+6}{2}, \frac{8+2}{2}\right) = (5.5, 5)$$
  - Slope of line $p2p4$:
$$\frac{2-8}{6-5} = \frac{-6}{1} = -6$$
  - Slope of the perpendicular bisector:
$$\frac{1}{6}$$
  - Equation of the perpendicular bisector:
$$y - 5 = \frac{1}{6}(x - 5.5)$$
    Simplifying this equation:
$$y = \frac{1}{6}(x - 5.5) + 5$$
- **Step 4.3: Compute the perpendicular bisector** between $(1, 7)$ and $(6, 2)$.
  - Midpoint:
$$\left(\frac{1+6}{2}, \frac{7+2}{2}\right) = (3.5, 4.5)$$
  - Slope of line $p3p4$:
$$\frac{2-7}{6-1} = \frac{-5}{5} = -1$$
  - Slope of the perpendicular bisector:
$$1$$
  - Equation of the perpendicular bisector:
$$y - 4.5 = 1(x - 3.5)$$
    Simplifying this equation:
$$y = (x - 3.5) + 4.5$$
$$y = x + 1$$
- **Step 4.4: Update the diagram** by adding the three new bisectors.
  - The plane is now divided into four regions, each associated with one of the four points: $p1 = (2, 3)$, $p2 = (5, 8)$, $p3 = (1, 7)$, and $p4 = (6, 2)$.

Fig: Voronoi Diagram of Given Points



Note: Demo in console

Reference

Graham scan Algorithm (Example)

- Convex Hull: Mark de Berg · Otfried Cheong, Marc van Kreveld · Mark Overmars: Computational Geometry: Algorithms and Applications,Third Edition Springer
- Convex Hull Example (Graham Scan): https://www.dinocajic.com/grahams-scan-visually-explained/
- Voronoi Diagram: Computational Geometry Lecture Notes Voronoi Diagrams, Valerie Barr, Hava Siegelmann, Gabor Sarkozy (1990) Michael Horn, Julie Weber (2004)

End of Section 2.6