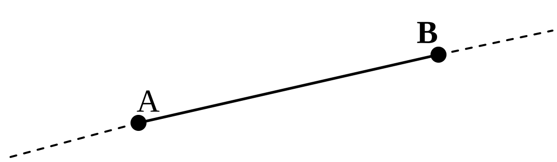
Chapter 7

Geometric Algorithms

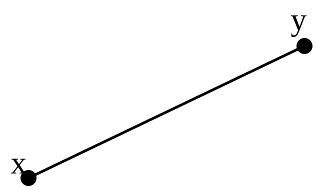
The easy explanation is that geometry algorithms are what a software developer programs to solve geometry problems. And we all know what geometry problems are, right? The simplest of such problems might be to find the intersection of two lines, the area of a given region, or the inscribed circle of a triangle. Methods and formulas have been around for a long time to solve such simple problems. But, when it comes to solving even these simple problems as accurate, robust, and efficient software programs, the easy formulas are sometimes inappropriate and difficult to implement. This is the starting point for geometry algorithms as methods for representing elementary geometric objects and performing the basic constructions of classical geometry.

Geometric Primitives

A line is a group of points on a straight path that extends to infinity. Any two points on the line can be used to name it.

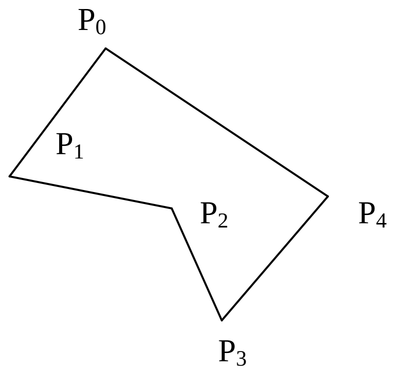


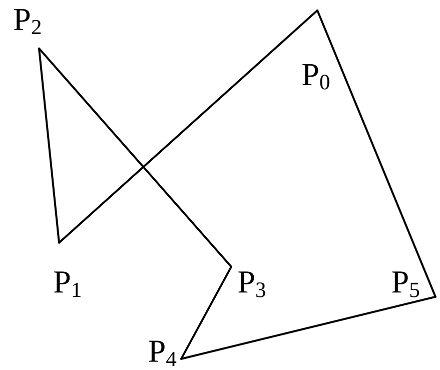
A line segment is a part of a line that has two end points. The two end points of the line segment are used to name the line segment.



Polygon

A closed figure of n line segments, where n ≥ 3. The polygon P is represented by its vertices, usually in counterclockwise order of traversal of its boundary, P = (p0, p1, ..., pn-1) or the line segments that are ordered as P = (S_0, S_1, ...,S_{n-1}) such that the end point of preceding line segment becomes starting point of the next line segment.

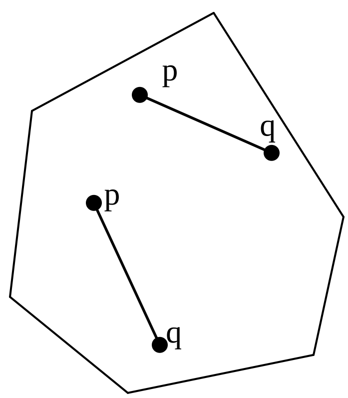


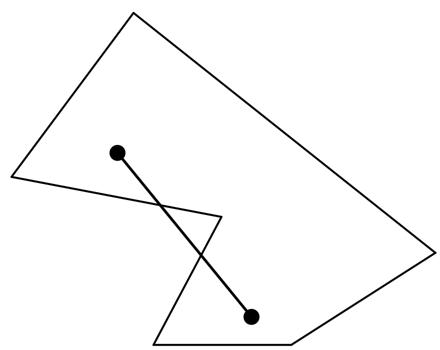


A Simple polygon is a polygon P with no two non-consecutive edges intersecting. There is a well-defined bounded interior and unbounded exterior for a simple polygon, where the interior is surrounded by edges.

Convex Polygon

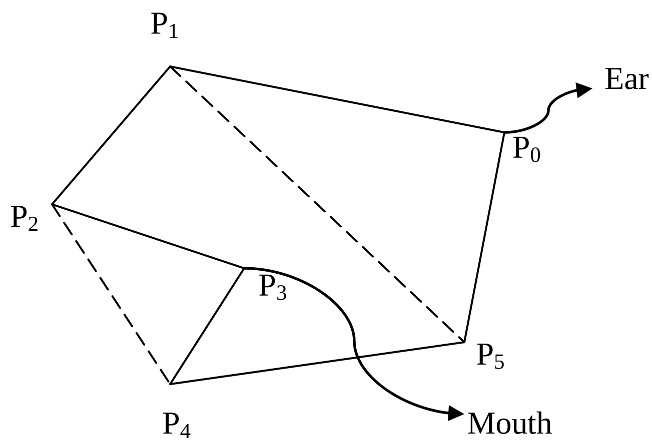
A simple polygon P is convex if and only if for any pair of points x, y in P, the line segment joining x and y lies entirely in P. We can notice that if all the interior angle is less than 180^0, then the simple polygon is a convex polygon.





Ear and Mouth

A vertex p_i of a simple polygon P is called an ear if for the consecutive vertices $p_{i-1}, p_i\ p_{i+1}$ (p_{i-1}, p_{i+1}) is a diagonal. A vertex p_i of a simple polygon P is called a mouth if the diagonal (p_{i-1}, p_{i+1}) is an external diagonal.



Convex Hull

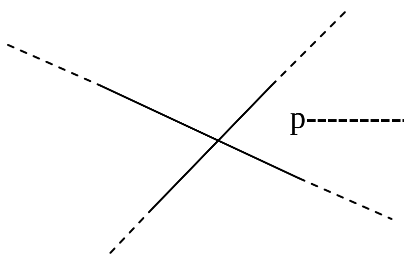
The convex hull of a polygon P is the smallest convex polygon that contains P. Similarly, we can define the convex hull of a set of points R as the smallest convex polygon containing R.

Computing point of intersection between two line segments

We can apply our coordinate geometry method for finding the point of intersection between two line segments. Let S1 and S2 be any two line segments. The following steps are used to calculate point of intersection between two line segments. We are not considering parallel line segments here in this discussion.

- Determine the equations of line through the line segment S1 and S2. Say the equations are L1 = (y = m1x + c1) and L2 = (y = m2x + c2) respectively. We can find the equation of line L1 using the formula of slope (m1) = (y2-y1)/ (x2-x1), where (x1,y1) and (x2,y2) are two given end points of the line segment S1. Similarly we can find the m2 for L2 also. The values of ci's can be obtained by using the point of the line segment on the obtained equation after getting slope of the respective lines.

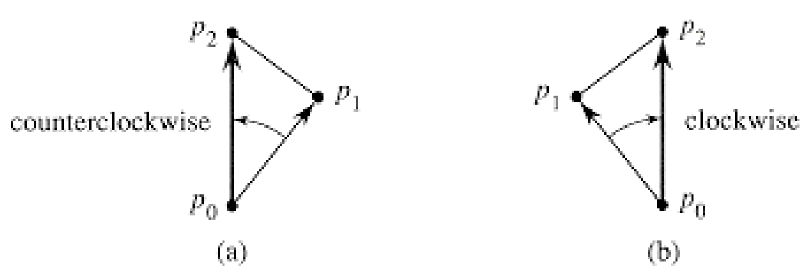
- Solve two equations of lines L1 and L2, let the value obtained by solving be p = (xi, yi). Here we confront with two cases. The first case is, if p is the intersection of two line segments then p lies on both S1 and S2. The second case is if p is not an intersection point then p does not lie on at least one of the line segments S1 and S2.



p----------------------->Line Segment intersect with each other

Determining Intersection between two Line Segments

Whether two consecutive line segments turn left or right at point p_l. Cross products allow us to answer this question without computing the angle. We simply check whether directed segment is clockwise or counterclockwise relative to directed segment . To do this, we compute the cross product $(p_2 - p_0) X (p_1 - p_0)$. If the sign of this cross product is negative, then is counterclockwise with respect to P_1, and thus we make a left turn at P_1. A positive cross product indicates a clockwise orientation and a right turn. A cross product of 0 means that points p_0, p_1, and p_2 are collinear.



We compute the cross product of the vectors given by two line segments as:

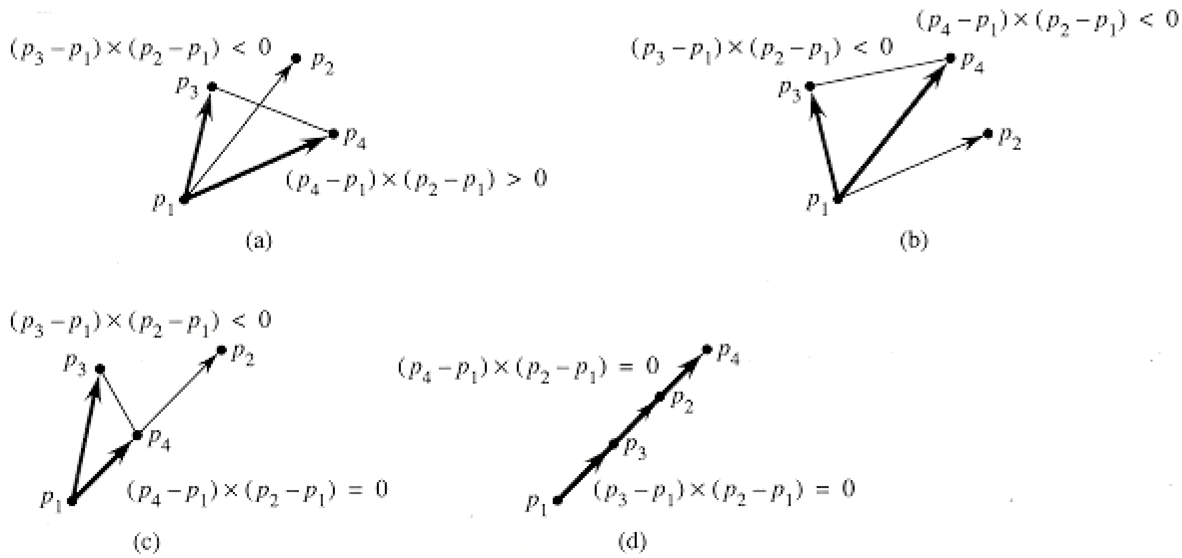
(p1- p0) X (p2- p0) = (x1- x0, y1- y0) ´(x2- x0, y2- y0) = (x1- x0)(y2- y0)-(y1- y0) (x2- x0), this can be represented as

$$\Delta = \begin{vmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \end{vmatrix}$$

Here we have,

- ✓ If $\Delta = 0$ then p_0,p_1,p_2 are collinear
- ✓ If $\Delta > 0$ then p_0,p_1,p_2 make left turn i.e. there is left turn at p_1. (p_0,p_1 is clockwise with respect to p_0,p_2)
- ✓ If $\Delta < 0$ then p_0,p_1,p_2 make right turn i.e. there is right turn at p1, (p_0,p_1 is anticlockwise with respect to p_0,p_2)

Using the concept of left and right turn we can detect the intersection between the two line segments in very efficient manner. Two segments S1 = (P, Q) and S2 = (R, S) do not intersect if PQR and PQS are of same turn type or RSP and RSQ are of same turn type.

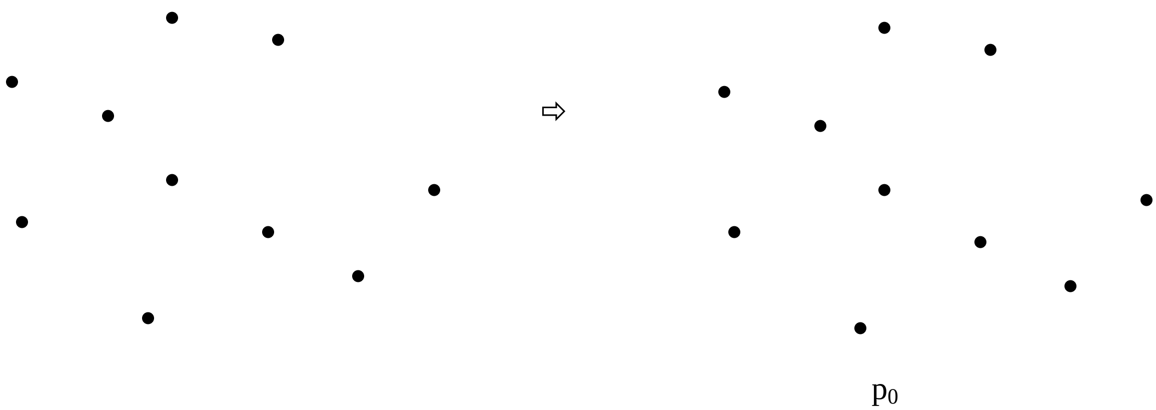


Grahm Scan Algorithm

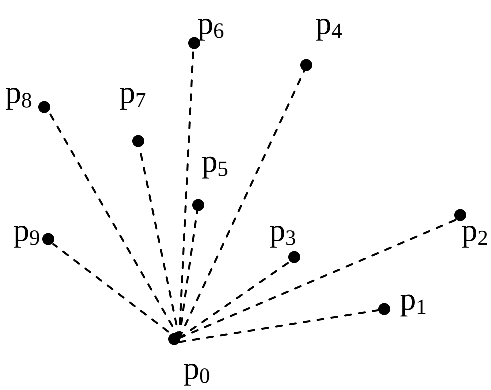
The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. Graham's scan solves the convex-hull problem by maintaining a stack S of candidate points. Each point of the input set Q is pushed once onto the stack, and the points that are not vertices of convex hull are eventually popped from the stack. When the algorithm terminates, stack S contains exactly the vertices of convex hull, in counterclockwise order of their appearance on the boundary.

The algorithm starts by picking a point in Q known to be a vertex of the convex hull. This can be done in O(*n*) time by selecting the rightmost lowest point in the set; that is, a point with first a minimum (lowest) *y* coordinate. Having selected this base point, call it p_0, the algorithm then sorts the other points p in **Q** by the increasing counter-clockwise angle the line segment p_0p makes with the *x*-axis. If there is a tie and two points have the same angle, discard the one that is closest to p_0.

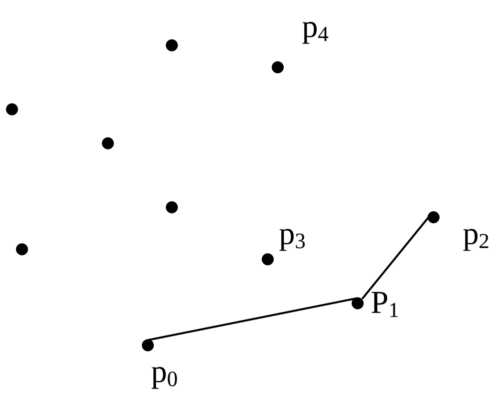
Given Points: Select point with lowest y-coordinate



Sort points in increasing order of angle in counter clockwise order, say sorted set of points is S

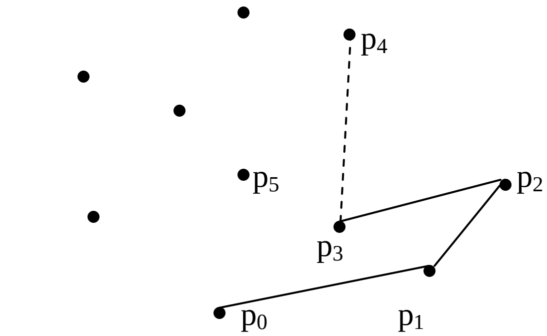


$S=\{p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8,p_9\}$



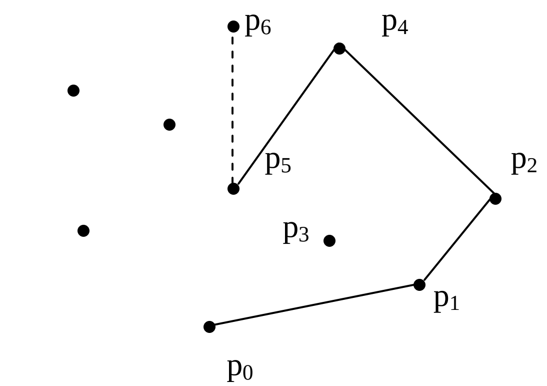
push (p1)

push(p2)



$p_1p_2p_3$:-Left Turn=>push(p3)

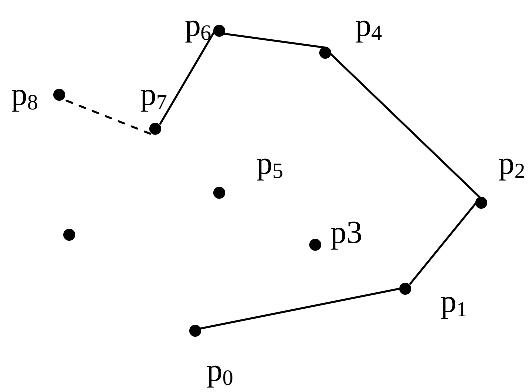
$p_2p_3p_4$:-Right Turn=>pop()



$p_1p_2p_4$:-Left Turn=>push(p_4)

$p_2p_4p_5$:-Left Turn=>push(p_5)

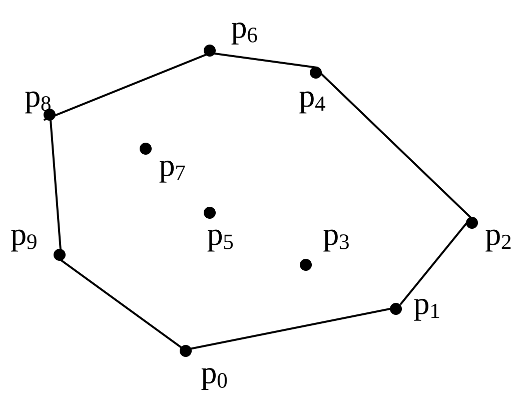
p4p5p6:-Right Turn=>pop()



$p_2p_4p_6$:-Left Turn=>push(p_6)

$p_4p_6p_7$:-Left Turn=>push(p_7)

$p_6p_7p_8$: Right Turn=>pop()



$p_4p_6p_8$:-Left Turn=>push(p_8)

$p_6p_8p_9$:-Left Turn=>push(p_9)

Algorithm

```
GrahamScan(P)
{
    p0 = point with lowest y-coordinate value.
    Angularly sort the other points with respect to p0
    Push(S, p0); // S is a stack
    Push(S, p1);
    Push(S, p2);
    For(i=3;i<m;i++)
    {
        a = NexttoTop(S);
        b= Top(S);
        while (a,b,pi makes non left turn)
                Pop(S);
        Push(S,qi);
}
    return S;
}
```

Analysis

It requires O(n) time to find p_0. Sorting of points require O(nlogn) time,. Push operation takes constant time i.e., O(1). We can understand that the while loop is executed O(m-2) times in total. Thus we can say that while loop takes O(1) time. So the worst case running time of the algorithm is T(n) = O(n) + O(nlogn) + O(n) = O(nlogn), where n = |P|.

Polygon Triangulation

Triangulation of a simple polygon P is decomposition of P into triangles by a maximal set of non-intersecting diagonals. Diagonal is an open line segment that connects two vertices of P and lies in the interior of P. Triangulations are usually not unique. Any triangulation of a simple polygon with n vertices consists of exactly n-2 triangles.

Triangulation is done by adding the diagonals. For polygon with n-vertices, the candidate diagonals can be $O(n^2)$. Check intersection of $O(n^2)$ segments with $O(n)$ edges it costs $O(n^3)$.There can be total of n–3 diagonals on triangulation. So total cost in triangulation is $O(n^4)$.