Computational Geometry

1. Point \textit{w.r.t.} a Line Segment

Three possibilities: (i) Point is on the left of the line segment (ii) Point is on the right of the line segment (iii) Point is collinear with the line segment.

Let \( P_0, P_1 \) define the line segment \( S_1 \).
Let \( P_2 \) be the point of interest.
Consider a line segment \( S_2 \) connecting points \( P_1 \) and \( P_2 \).
Now we can define the relations:
1. \( \text{IsLeft}: \text{slope}(S_2) > \text{slope}(S_1) \)
2. \( \text{IsRight}: \text{slope}(S_2) < \text{slope}(S_1) \)
3. \( \text{Collinear}: \text{slope}(S_2) = \text{slope}(S_1) \)

We can define another relation by combining \( \text{IsLeft} \) and \( \text{Collinear} \) to indicate that the point is either on the left or collinear.
4. \( \text{IsLeftOn}: \text{slope}(S_2) \geq \text{slope}(S_1) \).

Definition of slope = \( \frac{\text{rise}}{\text{run}} \)

Thus, \( \text{slope}(S_2) = \frac{y_2 - y_1}{x_2 - x_1} \), \( \text{slope}(S_1) = \frac{y_1 - y_0}{x_1 - x_0} \)

Substituting slopes with these equations we get:
1. \( \text{IsLeft}(P_0, P_1, P_2): \frac{y_2 - y_1}{x_2 - x_1} > \frac{y_1 - y_0}{x_1 - x_0} \)
   or \((y_2 - y_1)(x_1 - x_0) > (x_2 - x_1)(y_1 - y_0)\)
   or \((y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0) > 0\)
2. \( \text{IsRight}(P_0, P_1, P_2): (y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0) < 0\)
3. \( \text{Collinear}(P_0, P_1, P_2): (y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0) = 0\)
4. \( \text{IsLeftOn}(P_0, P_1, P_2): (y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0) \geq 0\)

(Note: Though the initial part of the derivation assumes that \( x_0 < x_1 < x_2 \), the final derived relationship 1 to 4 are valid for any points and line segment.)

Signed Area of a Triangle

Area of a triangle defined through points \( P_0, P_1 \) and \( P_2 \) =
\[ \frac{1}{2} \left[ (y_2 - y_1)(x_1 - x_0) - (x_2 - x_1)(y_1 - y_0) \right] \]

Thus relations above are actually using the signed area of the triangle.

If the points are in CCW orientation then the area is +ve, CW orientation then the area is –ve and collinear then the area is 0.
2. Segment-Segment Intersection

Two Issues.
1. Do they intersect?
2. What is their point of intersection?

Do the segments intersect?
Let \( P_0, P_1 \) define the line segment \( S_1 \).
Let \( P_2, P_3 \) define the line segment \( S_2 \).

\[
\text{Intersect}(S_1, S_2) = (\text{isLeftOn}(P_0, P_1, P_2) \oplus \text{isLeftOn}(P_0, P_1, P_3)) \land (\text{isLeftOn}(P_2, P_3, P_0) \oplus \text{isLeftOn}(P_2, P_3, P_1))
\]

where \( \oplus \) represents XOR operator.

What is their point of intersection?
Solve the equations of the line segment.
We can use analytical equations of line or parametric equations of line.

Here, we will use parametric equations of line.

Parametric equation of \( S_1 \): \( P = (1-t)P_0 + tP_1 \) where \( 0 \leq t \leq 1 \).

Similarly the parametric equation of \( S_2 \): \( P = P_2 + s(P_3 - P_2) \) where \( 0 \leq s \leq 1 \).

Point of intersection \( P \) must satisfy the equation: \( P_0 + t(P_1 - P_0) = P_2 + s(P_3 - P_2) \)

Solving this equation we get: \( t = \frac{(y_3 - y_2)(x_2 - x_0) - (x_3 - x_2)(y_2 - y_0)}{(y_3 - y_2)(x_3 - x_0) - (x_3 - x_2)(y_3 - y_0)} \)

(NOTE: The denominator can be zero when the segments are co-linear. So the co-linearity must be resolved before using this equation.)

Compute the point by substituting \( t \) in equation \( P = P_0 + t(P_1 - P_0) \).
3. **Point inside a Polygon**

**Point inside a Convex Polygon**

Let P be the point.
Assume the points of the polygon are in CCW orientation:

Algorithm Inside(Polygon, P)

for every edge segment S of the Polygon

if ¬isLeftOn(S.firstEndPoint, S.secondEndPoint,P)
  return FALSE

return TRUE

**Point inside a Simple Polygon**

Half Line test:
Step1: Draw a half line (say parallel to X-axis) from the point.
Step2: Intersect the half line with the edges of the polygon
Step3: Count the intersections.
  Odd number of intersection: Point is inside
  Even number of intersection: Point is outside

Algorithm Inside(Polygon, P)

N_intersections ← 0

for every edge segment S of the Polygon

if HalfLineIntersect(P, S.firstEndPoint, S.secondEndPoint)
  return FALSE

return TRUE

Intersection of a half-line parallel to the X-axis with an edge of the polygon defined by two endpoints P_k and P_{k+1}:

HalfLineIntersect(P,P_k,P_{k+1})

// P Represents the half line.
// Let P be (x,y), P_k = (x_k,y_k) and so on.
if ((y_{k+1} - y) ≥ 0) ⊕ ((y_k - y) ≥ 0) // end points are on opposite side of halfline
  //then Possible intersection
  // Compute actual intersection of the edge with the half line.
  y_{intersection} ← y
  t ← (y_{intersection} - y_0) / (y_1 - y_0)
  x_{intersection} ← x_i + t(x_{i+1} - x_i)
  if (x_{intersection} < x) return FALSE //Point is on the line but not on half line.
  else return TRUE // Actual intersection
else return FALSE // The end points are on one side of the half line.
  // So can not intersect the half line.
4. Problem: Segment intersection
Given a set of ‘n’ line segments, compute the points of intersection.

Solution 1: Brute force algorithm
Test all segment pairs \((s_i, s_j), i \neq j\).
If \( s_i \) intersects \( s_j \), then report the point of intersection.

Complexity is \(O(n^2)\).

Solution 2: Scan-line algorithm or Plane Sweep algorithm
Idea: Two line segments \( s_i \) and \( s_j, i \neq j \) can intersect only if they are neighbors.

Neighbor relation: Line segments are neighbors only if their intersection points with the scan line are adjacent.

The neighbor relationship changes only:
- at end points of a segment
- at intersections of segment.

PlaneSweep:
- initialize event queue by adding all the end points of the segments in increasing ‘x’ order.
- initialize **scan line status list** to null. Note: This list will contain the segments in an decreasing (or increasing) order of their intersection points (y) with the scan line. Thus will define the neighborhood relationship.

- While (**event queue** is not empty)
  - e = next event
  - ProcessEvent (e)
- End while.

**ProcessEvent (e)**
- if e is a top end point
  - s = segment to which e belongs
  - insert s into the **scan line status list** according to y order.
  - Intersect s with its left neighbor in the **scan line status list**
  - Insert intersection point if any into the **event queue** in sorted order
  - Intersect s with its right neighbor in the **scan line status list**
  - Insert intersection point if any into the **event queue** in sorted order
- If e is bottom end point
  - s = segment to which e belongs
  - Intersect the left neighbor of s in the **scan line status list** with the right neighbor of s.
  - Insert intersection point if any into the **event queue** in sorted order
  - Remove s out of the **scan line status list**
- If e is the intersection point
  - (s\text{left}, s\text{right}) be the left segment and right segment pair intersecting at e
  - intersect left neighbor of s\text{left} in the **scan line status list** with s\text{right} and intersect s\text{left} with the right neighbor of s\text{right} in the **scan line status list**.
  - Insert intersection point(s) if any into the **event queue** in sorted order
  - Swap the order of (s\text{left}, s\text{right}) in the **scan line status list**.

**Data Structure for the scan line status list:**
Balanced search tree ordered with y value as the key.
It should support: insert, remove, left neighbor, right neighbor operations in O(log n) time.
So total cost of maintaining this tree will be O(n log n).

**Data Structure for the event queue:**
Priority queue with decreasing x order.
It should support: Insert operation and next operation in O(log (n+k)) = O(log n) time.
So total cost is O((n+k) log n) time.

**Total number of intersections:**
Each segment may generate at most 3 intersections. 2 at its top end point and 1 at its bottom end point. So at most 3 intersections.
So total number of intersections 3n.