• Homework 10 due **Saturday**, May 2 at 5pm
• Note for Problem C, there are two typos in the sample input
  Two lines say
  50 0
  Should be:
  5
  0 0
• Final Exam: Thursday, May 14 from 4pm to 5:50pm in WWH 101
  • Must bring a Laptop! (Email me **asap** if you do not have one!)
• Course evaluations
  • We’ll complete the CS course evals right now: deliver to Rm 305 upon completion
  • Be sure to complete the CAS course evals online!
A polygon is **convex** if any line segment drawn inside the polygon does not intersect any edges of the polygon.

- Otherwise, it is **concave**
• To check if a polygon is concave or convex, check if all three consecutive vertices in the polygon form the same turns
  • Recall that polygons are represented as an ordered (CW/CCW) list of points
• If one triple is found that is not consistent with the others, then the polygon is concave
Exercise: the textbook presents the following code for determining if a polygon is concave or convex. Identify two potential errors/ambiguities. The function $ccw$ returns true iff the three points make a counterclockwise turn.

```cpp
bool isConvex(const vector<point> &P) {
    int sz = (int)P.size(); // consecutive vertices of P form the same turns
    if (sz <= 3) return false; // a point/sz=2 or a line/sz=3 is not convex
    bool isLeft = ccw(P[0], P[1], P[2]); // remember one result
    for (int i = 1; i < sz-1; i++) // then compare with the others
        if (ccw(P[i], P[i+1], P[(i+2) % sz]) != isLeft)
            return false; // different sign -> this polygon is concave
    return true; // this polygon is convex
}
```
• How do you determine if a point is inside a polygon?
• Given a polygon defined by points $P_i$ and a candidate point $t$
• **Winding number algorithm**
• Consider all angles formed by consecutive polygon vertices and the point: $\{P_i, t, p_{i+1}\}$
• Sum up all angles subtended by each side of the polygon
  • Add up left turns
  • Subtract right turns
• If the final sum is $2\pi$, then the point is inside the polygon
Winding Number Algorithm
• Another algorithm: **ray casting**
  • Draw a line from the point in any fixed direction so that the line intersects edges of the polygon
    • If there are an odd number of intersections, the point is inside the polygon and outside otherwise
  • **Exercise:** what are the pros and cons of ray casting vs. winding number?
• Given a convex polygon and a line, you can cut the polygon into two convex polygons
• Algorithm intuition (i.e., return one of the sub-polygons):
  • Iterate through polygon vertices, identify if they are to the left or the right of the line
  • If a polygon edge intersects the line, then find the intersection and add it to the resultant polygon
    • Ignore all subsequent points until we return to the starting side of the line
• Recall the definition of the **convex hull**: for a set of points P, the convex hull is the smallest set of points for which each point of P is either on the boundary of the convex hull or is in its interior.
• Pick a **pivot** point: e.g., the bottom-most, right-most point
• Sort all other points based on their angles with respect to the pivot
• A stack is maintained
  • All points from the polygon will be pushed onto the stack once
  • All points that don’t end up on the convex hull will eventually be popped off
• The algorithm maintains that the top three elements of the stack always make a left turn
**Exercise:** Given up to 1M points, find the closest pair.
• Idea: sweep a vertical line across the plane keeping track of:
  • Closest pair out of all points encountered and the distance $d$ between them
  • All points within $d$ units to the left of the vertical line in an ordered set $D$, sorted by $y$-coordinate

• Every time the line hits a point $p$:
  • Remove all points further than $d$ units away from the line from $D$
  • Find the closest point to $p$ in $D$, update $d$ if necessary
• Finding the closest point to \( p \) in \( D \):
  • Recall that \( D \) is sorted by y-coordinates, so you can further restrict the search (the **active region**)
  • How does this step not make the algorithm \( O(n^2) \)?

• Note that the horizontal and vertical separation between any two points in \( D \) is at least \( d \), so the points in the active region has a constant upper bound

• Running time: \( O(n \log n) \)
**Exercise**: Given a set of horizontal and vertical line segments on a 2D plane, return the **number** of intersections. Assume the number of intersections is fewer than 1M and there are no overlapping lines.
Let an **event** be an x-coordinate at which something “interesting” happens:
- Start / end of a horizontal line segment
- A vertical line segment

Initialize a priority queue Q with all of the events
Initialize an empty binary search tree T of line segments s that cross sweep line L

While Q is nonempty, pop event p with minimum x-coordinate and:
- If p is the left endpoint of a line segment, insert into s into T
- If p is the right endpoint of a line segment, remove s from T
- If p is a vertical line, search T for overlapping line segments
Exercise: given up to 1M lines on a 2D plane, what is the union of their area?
Competitive Programming 7.3—7.5