PSO 2 — Week 8: Solutions

CS381, Fall 2026

Problem 1

(a) Point in a convex polygon

The diagonals from v_0 cut the polygon into the triangles $v_0v_iv_{i+1}$ for $i = 1, \ldots, n-2$, fanned out around v_0. Binary search for the triangle whose wedge contains q, then test that one triangle.

```
if orient(v[0], v[1], q) < 0 or orient(v[0], v[n-1], q) > 0: return outside
lo := 1; hi := n-1
while hi - lo > 1:
    mid := floor((lo + hi) / 2)
    if orient(v[0], v[mid], q) >= 0: lo := mid
    else: hi := mid
if orient(v[lo], v[lo+1], q) >= 0: return inside
else: return outside
```

Where convexity is used. The binary search needs the test $\text{orient}(v_0, v_i, q) \geq 0$ — "q is on the left of the ray from v_0 through v_i, or on it" — to be true for an initial stretch of i and false afterwards. That holds because the rays from v_0 through $v_1, v_2, \ldots, v_{n-1}$ turn steadily counterclockwise without wrapping around. In a convex polygon listed counterclockwise, every vertex lies strictly on the left of each edge it is not an endpoint of; in particular v_0 is on the left of the edge v_iv_{i+1}, and since orient is unchanged by rotating its three arguments cyclically, $\text{orient}(v_0, v_i, v_{i+1}) > 0$. So each ray turns counterclockwise from the one before. They cannot wrap around, because all of them lie within the interior angle at v_0, which is less than 180°.

Correctness. The first line returns "outside" exactly when q lies outside the angle at v_0 formed by the rays through v_1 and v_{n-1}; the whole polygon lies within that angle, so this is right. Otherwise the loop keeps $\text{orient}(v_0, v_{lo}, q) \geq 0$, and keeps $\text{orient}(v_0, v_{hi}, q) < 0$ unless $hi = n-1$, so it ends with $hi = lo+1$ and q in the wedge between the rays through v_{lo} and v_{lo+1}. Within that wedge the polygon is exactly the triangle $v_0v_{lo}v_{lo+1}$, and q is in that triangle exactly when it is on the left of, or on, the edge $v_{lo}v_{lo+1}$. □

Time. $O(\log n)$ iterations with one orientation test each, plus three more tests.

Without convexity. On a polygon that is not convex, the rays from v_0 can turn back clockwise, so the test above can switch from false back to true and the binary search can stop in the wrong wedge. Worse, a fan triangle $v_0v_iv_{i+1}$ can stick out of the polygon, so even the final test can give the wrong answer. General polygons are usually handled by counting how many edges a ray from q crosses, which takes $O(n)$ per query.

(b) A lower bound for convex hull

The reduction. Given distinct numbers $x_1, \ldots, x_n$, form the points $p_i = (x_i, x_i^2)$ and run the hull algorithm on them. Find the hull vertex with the smallest x-coordinate in $O(n)$ time, and read the counterclockwise list starting there. We show this lists the x_i in increasing order, so sorting costs the hull algorithm's time plus $O(n)$.

Why it works. Relabel so that $x_1 < x_2 < \cdots < x_n$. For any $a < b < c$,

$$\text{orient}\big((a,a^2),(b,b^2),(c,c^2)\big) \;=\; (b-a)(c^2-a^2) - (b^2-a^2)(c-a) \;=\; (b-a)(c-a)(c-b) \;>\; 0,$$

so any three of the points, taken in increasing order of x, make a left turn. Since orient is unchanged by rotating its arguments cyclically, the same is true of any rotation of such a triple.

Now each of the segments $p_1p_2, \ldots, p_{n-1}p_n$ and p_np_1 has every other point strictly on its left. For p_ip_{i+1} and another point p_k, the triple (p_i, p_{i+1}, p_k) is increasing if $k > i+1$, and is a rotation of the increasing triple (p_k, p_i, p_{i+1}) if $k < i$. For p_np_1, the triple (p_n, p_1, p_k) is a rotation of (p_1, p_k, p_n). A segment between two of the points with all the others strictly on its left is a counterclockwise edge of the hull. So the hull is the polygon $p_1, p_2, \ldots, p_n$: every point is a vertex, and reading counterclockwise from the leftmost one, p_1, gives the sorted order. $\square$

What this says. A hull algorithm running in $o(n \log n)$ time would sort in $o(n \log n)$ time. One point needs care. The $\Omega(n \log n)$ bound for sorting from lecture counts comparisons between input numbers, while hull algorithms make orientation tests, and on these points an orientation test evaluates the sign of a polynomial in the x_i rather than comparing two of them. So the conclusion uses the stronger fact that sorting needs $\Omega(n \log n)$ time even for algorithms allowed to test the signs of low-degree polynomials in their input (a theorem of Ben-Or). In that model, computing the hull with its vertices in order takes $\Omega(n \log n)$ time. Graham's scan meets that bound, so its sorting step cannot be avoided.

What the reduction relies on. The counterclockwise order of the output. On these inputs every point is a hull vertex, so an algorithm that reported only *which* points are vertices would answer "all of them" and say nothing about the order. That version of the problem is known to need $\Omega(n \log n)$ time as well, but proving it takes a different argument.