Solutions to Homework 1

Solution 1: Throughout, we assume that points are in general position.

(a) We assert that d lies within the interior of triangle $\triangle abc$ if and only if

$$\text{orient}(a, b, d) = \mathit{orient}(b, c, d) = \text{orient}(c, a, d)$$

Because a, b, and c are not collinear, not all three orientation tests can be zero. Hence, if the test succeeds, we may assume that all three are positive or all three are negative. Consider the orientations of the point d with respect to the three directed sides of the triangle. Clearly, if any of these orientations differ, d lies to one side of one directed edge and to the opposite side of another, so it cannot be inside the triangle. On the other hand, if all orientations are the same, then either d lies on the same side of all three edges (and hence is in the triangle) or outside all three edges. However, it's easy to see that the latter cannot happen.

(b) We assert that the line segments $\overline{pq}$ and $\overline{st}$ intersect if and only if s and t lie on opposite sides of the (infinite) line $\overline{pq}$ and p and q lie on opposite sides of the line $\overline{st}$. This gives the following condition for intersection:

$$\text{orient}(p, q, s) \neq \text{orient}(p, q, t) \text{ and } \text{orient}(s, t, p) \neq \text{orient}(s, t, q)$$

To see that this is correct, consider the point r where the two infinite lines $\overline{pq}$ and $\overline{st}$ intersect. (Assume for simplicity that they are not parallel.) If the segments intersect, then r lies between p and q on one line and between s and t on the other, which implies that the orientations of each pair differ with respect to the other pair (see Fig. 1(a)). On the other hand, if the segments do not intersect, then at least one of the pairs lies on the same side of r (see Fig. 1(b)), implying that this pair has an equal orientation, and the above test fails.

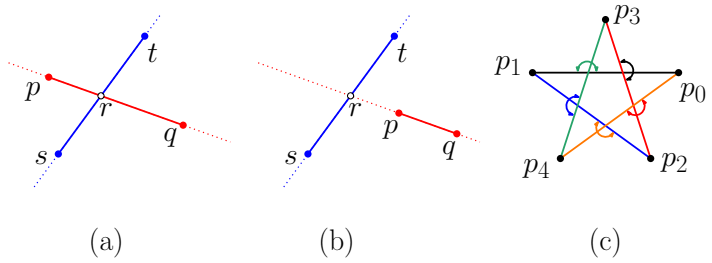

Figure 1: Using orientations for geometric properties.

Note that this does not necessarily work if the points are not in general position. For example, if all four points are collinear, all orientation tests return 0, regardless of whether the segments intersect.

(c) This is a long answer, but it raises some interesting issues. Henceforth, we assume that all indices are taken modulo 5.

20 orientations: To determine whether a sequence of five points forms a pentagram, we could naively invoke the solution to part (b) on each of the five pairs of edges that must intersect. This would lead to 20 orientation tests, which is excessive.

10 orientations: We can reduce this to just 10 orientation tests. (Note that this seems like a reasonable value, because there are $\binom{5}{3} = 10$ ways of forming triples.) There are many equivalent ways of doing it. The simplest is to compute them all, cache the answers, and then run the 20-orientation solution. This formally satisfies the requirements of the problem, but it's a bit of a cheat, since it still makes 20 comparisons. The answer that follows is closer to the spirit of the problem.

The pentagram can turn globally clockwise or counterclockwise. We start by computing one orientation, $\langle p_0, p_1, p_2 \rangle$, to determine the global orientation. We then enter a loop, which for each pair (p_i, p_{i+1}) determines whether the points p_{i+2} and p_{i+3} lie on opposite sides of the line $\overline{p_i, p_{i+1}}$ (see Fig. 1(c)).

- $o \leftarrow \text{orient}(p_0, p_1, p_2)$ // global orientation
- for $i \leftarrow 0, \ldots, 4$
  - if ($\text{orient}(p_i, p_{i+1}, p_{i+2}) \neq o$ || $\text{orient}(p[i], p[i+1], p[i+3]) \neq -o$) return "failure"
- return "success"

Although this appears to require 11 orientation tests, it can be implemented with only 10. The initial orientation test is actually the first one done in the loop, so we could incorporate it into the for-loop.

Why is it correct? Here is a formal proof.

Claim: The above algorithm succeeds if and only if all pairs of edges of the form $\overline{p_i p_{i+1}}$ (indices taken mod 5) intersect.

Proof: It is easy to see that if all the edges intersect, then the algorithm's tests agree with this result. The harder part is proving the converse, namely that if all the tests the algorithm performs are satisfied, then all the edges intersect.

Of course, consecutive edges intersect at their endpoints, so we need to check non-consecutive edges. There are five such pairs to check, namely, $\overline{p_i p_{i+1}}$ and $\overline{p_{i+2} p_{i+3}}$, for $0 \leq i < 5$. To simplify notation, we'll prove one special case; the others follow by symmetry. Let's first assume that `o = +1`, and the edge pair to check for intersection is $\overline{p_0 p_1}$ and $\overline{p_2 p_3}$. Based on the reasoning in part (a), and given the orientation `o`, the necessary tests are:

$$\begin{aligned}&(\text{orient}(p_0, p_1, p_2) = o) \text{ and } (\text{orient}(p_0, p_1, p_3) = -o) \text{ and}\\ &(\text{orient}(p_2, p_3, p_0) = -o) \text{ and } (\text{orient}(p_2, p_3, p_1) = o)\end{aligned}$$

The algorithm above explicitly validates the first two conditions when $i = 0$. It also validates the third condition when $i = 2$. It validates the fourth condition indirectly. Observe that $\text{orient}(p_2, p_3, p_1) = \text{orient}(p_1, p_2, p_3)$, and so the algorithm validates the fourth condition when $i = 1$.

This implies that 10 orientation tests suffice. See the Challenge Problem for a solution involving just 7 orientation tests.

Solution 2: We show that the point of tangency between the upper supporting line at q and the upper hull of a point set P can be computed in $O(\log n)$ time. Assume the upper-hull vertices of P are sorted left to right, and q lies to the right of all of them.

Let p_j denote the vertex of P's upper hull through which the supporting line passes. We will make the general-position assumption that the supporting line passes through exactly one point of P. By the same reasoning that we used in the correctness proof of Graham's algorithm, it follows that for all vertices i, where $i \leq j$, we have $\text{orient}(q, p_i, p_{i-1}) > 0$, that is, this triple makes a left-hand turn (see Figure 2(a) and (b)). Similarly, for all vertices i, where $i > j$, we have $\text{orient}(q, p_i, p_{i-1}) < 0$, that is, this triple makes a right-hand turn (see Figure 2(c)).

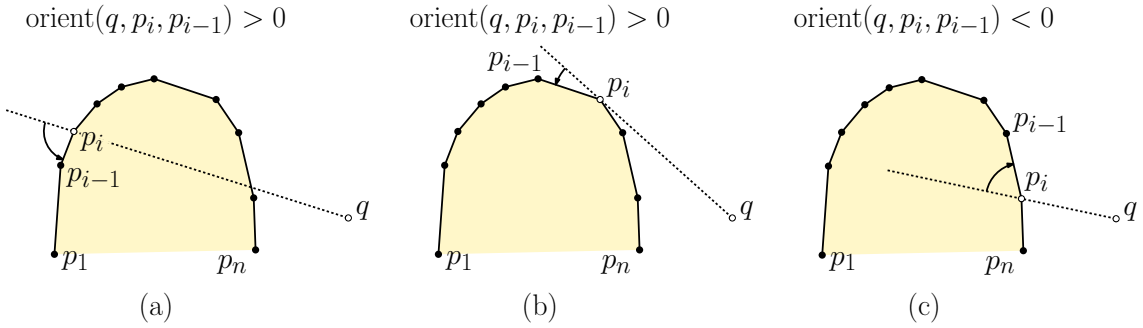

Figure 2: Computing the tangent by binary search

One minor technicality remains. If $i = 1$, then there is no point p_{i-1}. We can handle this by creating an imaginary sentinel point p_0 that lies infinitely below p_1, say with coordinates $p_0 = (0, -\infty)$. Alternatively, we can handle $i = 1$ as a special case in our algorithm.

This means we can run a binary search among the points of P until we find the largest index j such that $\text{orient}(q, p_i, p_{i-1}) > 0$. Since each test takes constant time, this will take $O(\log n)$ time in total.

Solution 3: Recall that if h_i^* denotes the ith guess, the ith iteration of Chan's algorithm runs in $O(n \log h_i^*)$ time. The base of the logarithm does not affect the asymptotic running time, so we assume base 2, denoted by lg.

The algorithm terminates as soon as $h_i^* \geq h$. We cannot assume that the algorithm will stop exactly when $h_i^* = h$, which means that we will need to round i up to the next higher value to guarantee that $h_i^* \geq h$ by taking ceilings. (Usually floors and ceilings do not matter when giving asymptotic analyses, but when the growth rates are quite high we need to be careful.) Throughout, we will use the fact that $\lceil x \rceil \leq x + 1$.

(a) $h^* \leftarrow \min(i^2, n)$: The algorithm terminates when $i = \left\lceil \sqrt{h} \right\rceil \leq \sqrt{h} + 1$. We use a standard fact based on Stirling's approximation: $\sum_{i=1}^{n} \lg i = \lg n! \approx \lg(n/e)^n = \Theta(n \lg n)$. Thus, up to constant factors, the running time is at most

$$n \sum_{i=1}^{\sqrt{h}+1} \lg(i^2) \;=\; n \sum_{i=1}^{\sqrt{h}+1} 2 \lg i \;=\; \Theta(n(\sqrt{h}+1) \lg(\sqrt{h}+1)) \;=\; \Theta(n\sqrt{h} \lg \sqrt{h}) \;=\; \Theta(n\sqrt{h} \lg h),$$

which is larger than Chan's algorithm by a factor of $\sqrt{h}$.

(b) $h^* \leftarrow \min(2^i, n)$: The algorithm terminates when $i = \lceil \lg h \rceil \leq (\lg h)+1$. Using the well-known fact that $\sum_{i=1}^{n} i = \Theta(n^2)$, the running time is at most

$$n \sum_{i=1}^{(\lg h)+1} \lg 2^i \;=\; n \sum_{i=1}^{(\lg h)+1} i \;=\; \Theta(n \lg^2 h),$$

which is larger than Chan's algorithm by a factor of $\lg h$.

Solution 4: To determine whether the point p_i lies strictly to the left of the directed line $\overrightarrow{q_0q_1}$, we test whether $\text{orient}(q_0, q_1, p_i) > 0$. Among all the points satisfying this condition, the point p_j that we seek has the property that it lies to the right of the directed line $\overrightarrow{q_1p_i}$ for all points p_i that satisfy the first condition, that is, $\text{orient}(q_1, p_j, p_i) \geq 0$, for all $i \neq j$. Whenever we find a point that violates this condition, we update j. Thus, we have the following:

- $j \leftarrow 0$
- for $i \leftarrow 1, \ldots, n$
  - if ($\text{orient}(q_0, q_1, p_i) > 0$) // if p_i is left of directed line $\overrightarrow{q_0q_1}$
    - if ($j = 0$ || $\text{orient}(q_1, p_j, p_i) < 0$) // if p_i right of directed line $\overrightarrow{q_1p_j}$
      - $j \leftarrow i$ // update j

Solution 5: Throughout this problem we will make the general-position assumption that no two points of P have the same x- or y-coordinate. The terms "left" and "right" refer to ordering according to x-coordinates and "above" and "below" to the y-coordinate.

(a) A point $p_i = (x_i, y_i)$ is in the Pareto set hull of a set P if and only if the solid "L"-shaped wedge of points such that $x \geq x_i$ and $y \geq y_i$ contains no other point of P (see Fig. 3(a)). If a point p_j exists within this region, we say that p_i is *dominated* by p_j.

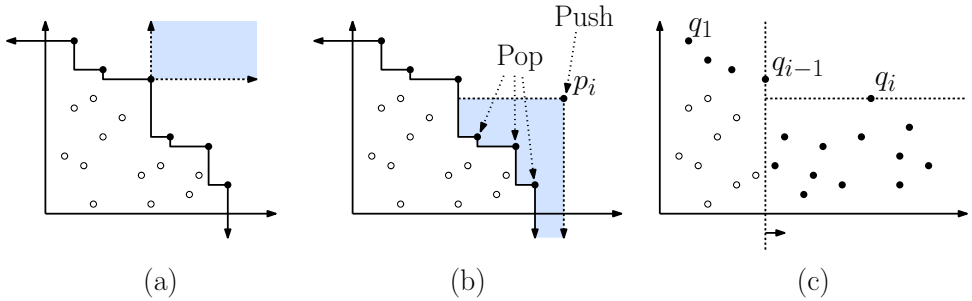

Figure 3: Pareto sets

(b) The Graham-like approach to computing the Pareto set first sorts the points from left to right. Let us renumber the points in this order as $\langle p_1, \ldots, p_n \rangle$. As we process the points, we maintain a stack S containing the Pareto set of the first $i-1$ points, with the rightmost point on top.

We begin by pushing p_1 onto an empty stack. Generally, when processing point $p_i = (x_i, y_i)$, we pop all the points from the stack whose y-coordinate is not greater than that of p_i (see Fig. 3(b)). We then push p_i onto the stack. After processing all points, we output the stack contents from bottom to top.

To show this is correct, we claim that the stack contents (bottom to top) equal the Pareto set sorted left to right. If p_i is in the Pareto set, then its y-coordinate is larger than the y-coordinate of any point to its right. Therefore, once we push p_i, we never pop it. Conversely, if p_i is not in the Pareto set, there is a point p_j to its right that has a larger y-coordinate. Consider the next such point after p_i. When p_j is processed, it pops all points on the stack with smaller y-coordinates, so p_i is popped and not on the final stack. The order of points on the stack follows from our left-to-right processing.

The point set can be sorted in $O(n \log n)$ time, and since each point is pushed once and popped at most once, the subsequent processing runs in $O(n)$ time. Thus, the overall running time is $O(n \log n)$.

(c) The Jarvis-like algorithm successively adds a new point to the Pareto set, working from left to right. The leftmost point of the final Pareto set is the point in P with the maximum y-coordinate, which we denote by q_1. Starting with q_1, our algorithm will successively add one additional point to the Pareto set, working from left to right. Let us assume inductively that $\langle q_1, q_2, \ldots, q_{i-1} \rangle$ have already been selected. It follows that all the remaining contenders for the Pareto set must have x-coordinates that exceed $q_{i-1,x}$ (for otherwise they cannot dominate q_{i-1}), and all such points must have y-coordinates smaller than $q_{i-1,y}$ (for otherwise q_{i-1} would have been dominated by any such point).

We scan all points in P, and among those whose x-coordinates exceed $q_{i-1,x}$, we select the one with the largest y-coordinate. We set q_i to be this point (see Fig. 3(c)). If no such point exists, we conclude that no points remain in the Pareto set, output the current sequence $\langle q_1, \ldots, q_{i-1} \rangle$, and terminate.

To show this is correct, we show that the output sequence of points $\langle q_0, q_1, \ldots, q_h \rangle$ equals the Pareto set from left to right. If not, consider the first point q_i where the two sets differ. Observe that (by its definition) q_i is not dominated by any point of P, and is therefore in the Pareto set. Therefore, we must have skipped some Pareto point between q_{i-1} and q_i. Let p be the leftmost such point. By definition of q_i, we have $p_y < q_{i,y}$ and since p is the first such failure, we have $p_x < q_{i,x}$. But then p is dominated by q, and so p is not on the Pareto set, a contradiction.

To derive the running time, note that each step of the algorithm runs in $O(n)$ time and adds one additional point to the Pareto set. The algorithm stops after $h + 1$ iterations, and so the running time is $O((h + 1)n) = O(hn)$, as desired.

Solution to Challenge Problem 1: Here is a solution involving 7 orientation tests. It is based on the following observation: The sequence $\langle p_0, p_1, \ldots, p_4 \rangle$ forms a pentagram if and only if the sequence $\langle p_0, p_3, p_1, p_4, p_2 \rangle$ forms a convex pentagon (see Fig. 4(a) and (b)). Why is this so? Let's first relabel the points in this order as $\langle v_0, v_1, v_2, v_3, v_4 \rangle$ (see Fig. 4(b)). If the sequence forms a convex pentagon, then clearly, every pair of chords must intersect within the pentagon's interior. Conversely, if the sequence does not define a convex pentagon, then there must be some

nonconsecutive pair (e.g., the pair (v_0, v_2) in Fig. 4(c)) that forms an edge on the boundary of the convex hull. No other edge can cross it, so it is not a pentagram.

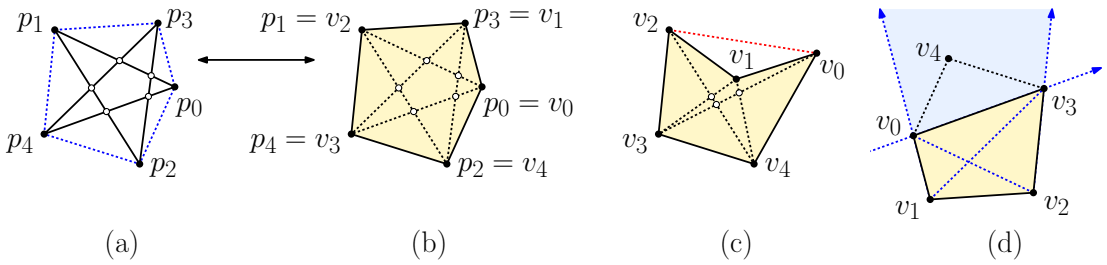

Figure 4: 7-orientation solution.

This leads to the question: what is the fewest number of orientation tests needed to determine whether a sequence of five points forms a convex pentagon? We can do this with seven orientation tests. Clearly, the first four points must form a convex quadrilateral. This is equivalent to saying that the segments $\overline{v_0v_2}$ and $\overline{v_1v_3}$ intersect. Based on solution 2(a), we can do this using the four orientation tests.

So this leaves the question of where to place v_4. The answer depends on the pentagon's global orientation: clockwise or counterclockwise. We can determine this from any one of the orientation tests performed so far. Let's assume it's counterclockwise. Observe that v_4 must lie (1) to the left of the directed line $\overrightarrow{v_0v_3}$, (2) to the left of the directed line $\overrightarrow{v_2v_3}$, and to the right of the directed line $\overrightarrow{v_1v_0}$ (see Fig. 4(d)). This requires three orientation tests, for a total of $4 + 3 = 7$ orientation tests.

Solution to Challenge Problem 2: The Chan-variant algorithm operates similarly to Chan's algorithm. First, it will employ the same squaring process to "guess" the approximate size of the final Pareto set. Call this value h^*. Also, like Chan's algorithm we will partition the point set into roughly n/h^* subsets, each of size at most h^*. We run the Graham-like algorithm from part (a) to obtain the mini-Pareto sequences (sorted from left to right) for each subset of the partition. Let $Q_1, \ldots, Q_m$ denote these mini-Pareto sequences.

The new element is the Jarvis-like merging process. We need a analog to the Tangent Lemma, which allows us to compute the next point of the Pareto set for each mini-hull in time $O(\log h^*)$. As described in part (c), the objective is to compute the point with the largest y-coordinate that lies to the right of q_{i-1}. To do this, we apply binary search to each mini-Pareto sequence to find the leftmost point whose x-coordinate is greater than that of q_{i-1} (if such a point exists). Because Pareto sets are monotonically decreasing, this point has the largest y-coordinate, as desired. Among the (at most) m of these points, we take the one that has the largest y-coordinate as q_i.

The binary search takes $O(\log h^*)$ per sequence, and so this takes $O((n/h^*) \log h^*)$ time overall. As in Chan's algorithm, we terminate the search after at most h^* iterations, and thus the running time (for one value of h^*) is $O(h^*(n/h^*) \log h^*) = O(n \log h^*)$.

The algorithm's correctness follows from the above remarks. The running-time analysis is exactly analogous to Chan's, so we omit it.