Solutions to Homework 1

Solution 1: It is tricky to answer exactly correct. The answer I am looking for is $2n - h - 2$ (and this is good enough for full credit), but this is not quite correct.

Here is the justification. Consider the main while loop in Graham's scan. If there are at least two points on the stack, then the orientation evaluation is performed at least once. For each additional evaluation, one point is being popped off the stack. Ignoring the first two points, there are $n - 2$ instances where there are two points on the stack. For each of the $n - h$ points that do not appear on the hull, we perform an additional orientation test, which leads to the desired total of $(n - 2) + (n - h) = 2n - h - 2$.

There is a subtlety, however. This assumes that the stack always has at least two points. Unfortunately, there are values of n and h, where this does not hold. In particular, if $n \geq 3$ and $h = 2$, we know that first and last points must be on the upper hull, which implies that at some iteration of the while loop, the number of points in the stack falls to just one. In this case we do not perform an orientation test. So, I believe that to get the formula perfectly correct, there needs to be an additional term that is -1 whenever $n \geq 3$ and $h = 2$.

Solution 2:

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

```
orient(a, b, d) == orient(b, c, d) == orient(c, a, d)
```

Because a, b, and c are not collinear, it cannot be that all three orientation tests are 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 the three directed sides of the triangle. Clearly, if any of these orientations differ, d lies to one side of one directed edge on the opposite side of the other, so it cannot be inside the triangle. On the other hand, if all the orientations are all the same, then either d lies inside 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:

```
orient(p, q, s) != orient(p, q, t) && orient(s, t, p) != 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.

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

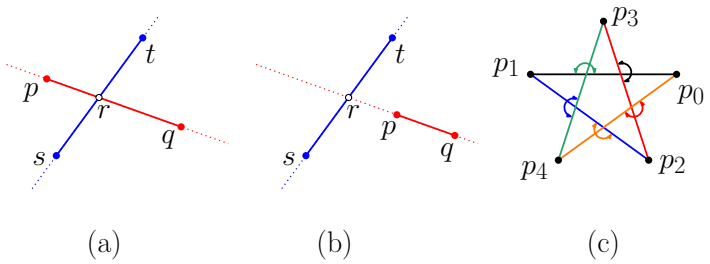

Figure 1: Using orientations for geometric properties.

(c) This is a rather long answer, but there are some interesting issues here. 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 just compute them all, cache the answers, and then run the 20-orientation solution. This formally satisfies the requirements of the problem, but is a bit of a cheat, since there are still 20 comparisons that are made. The answer that follows is a bit more true 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 = orient(p[0], p[1], p[2])
for (int i = 0; i < 5; i++)
    if (orient(p[i], p[i+1], p[i+2]) != o ||
        orient(p[i], p[i+1], p[i+3]) != -o) return failure
return success
```

While this appears to make 11 orientation tests, it can be implemented using only 10. The reason is that the initial orientation test is actually the first done in the loop, so we could have incorporated it into the code of 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 tests given by the algorithm will agree with this result. The harder part is to prove the converse, namely that if all the tests made by the algorithm 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 the notation, we'll just prove one special case, and the

others follow by symmetry. Let's first assume that `o = +1`, and the edge pair to check for intersection is $\overline{p_0p_1}$ and $\overline{p_2p_3}$. Based on the reasoning in part (a), and given the orientation `o`, the necessary tests are:

```
if (orient(p[0], p[1], p[2]) ==  o && orient(p[0], p[1], p[3]) == -o &&
    orient(p[2], p[3], p[0]) == -o && orient(p[2], p[3], p[1]) ==  o)
```

Observe that the above algorithm will explicitly validate the first two conditions when $i = 0$. Also, it validates the third when $i = 2$. It validates the fourth condition, indirectly. Observe that `orient(p[2], p[3], p[1]) == 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 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. Note that the base of the logarithm does not matter for the asymptotic running time, so we will assume that it is the logarithm base 2, which we denote 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 will make use of a standard fact based on Stirling's approximation that $\sum_{i=1}^{n} \lg i = \lg n! \approx \lg(n/e)^n = \Theta(n \lg n)$. 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: Let us assume that P includes the lower-left and lower-right corners of the region, $(0, 0)$ and $(1, 0)$. Let U denote the sequence of points on the upper-hull of P', sorted from left to right. Let ℓ denote the segment that bounds the minimum-area enclosure. All the points of P lie on or below ℓ, and at least one point of P must lie on ℓ (for otherwise we could move the line lower and make the area smaller). Therefore, ℓ is a supporting line for U (see Fig. 3(a)). Our objective is to compute the supporting line that produces the minimum area.

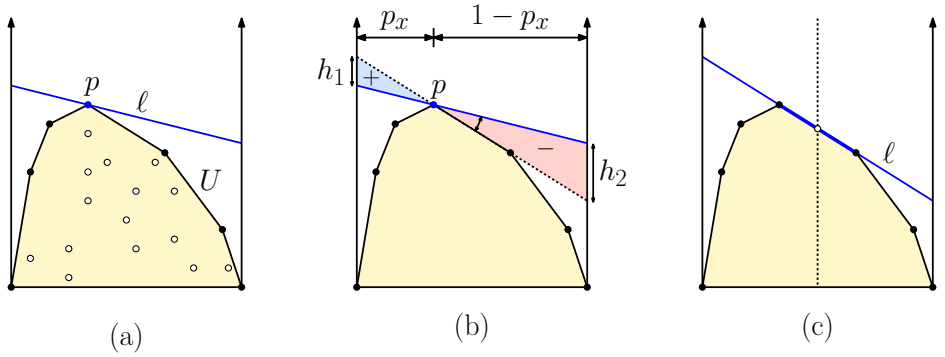

Figure 2: Minimum-area enclosure.

(a) In this part we will prove the following claim, which characterizes the optimal choice for ℓ.

Claim: Let ℓ be the segment of a support line for U passing between the two vertical sides of the enclosing region. Let p be the point of ℓ that lies on U and is closest to ℓ's midpoint. Then ℓ is the area-minimizing supporting line if and only if p coincides with ℓ's midpoint.

Proof: Since ℓ is assumed to hit both the left and right sides, its midpoint is at $x = 1/2$. Let p_x be p's x-coordinate. Suppose to the contrary that $p_x \neq 1/2$. We will show that ℓ does not yield the minimum-area enclosure. There are two cases, $p_x < 1/2$ and $p_x > 1/2$. Let's assume the former, since both cases are symmetrical.

Observe that p is a vertex of the upper hull and ℓ is not collinear with the edge that lies immediately to the right of p (since otherwise, we could have moved p even closer to $x = 1/2$.) As such, we can perform an infinitessimal clockwise rotation of ℓ about p while keeping all the points of P below ℓ. By doing so, we sweep out two similar triangles on either side of p (shaded in Fig 2(b)). Because $p_x < 1/2$ lies to the left of the midpoint, the triangle on the left has smaller area than the triangle on the right. (The two triangles are similar, and clearly the one on the left is smaller.)

By applying the rotation, the area of the enclosed region decreases by the area of the triangle on the right, and increases by the area of the triangle on the left. Thus, there is a net decrease in the overall area, implying that ℓ is not the area-minimizing support line, completing the contradiction.

(b) From the above claim, our problem reduces to computing the support line who midpoint lies on U. This is equivalent to determining the edge of U that is intersected by a vertical line at $x = 1/2$, which lies midway between the two vertical sides (see Fig. 2(c)).

Assuming that the vertices of the upper hull U are given from left-to-right order, this can be accomplished by performing simple binary search on this sequence in $O(\log m)$ time, where m is the number of vertices on the upper hull. Given the vertices to the left and right of the midpoint, we define ℓ to be linear extension of this edge.

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. 3(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. 3(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. 3(c)) that forms an edge on boundary of the convex hull. No other edge can cross this one, implying that it is not a pentagram.

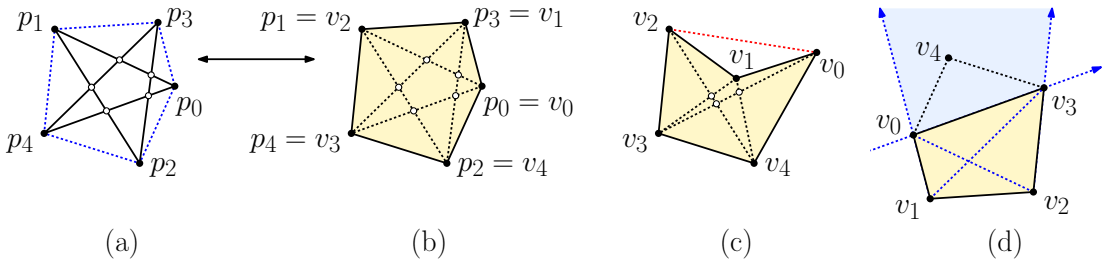

Figure 3: 7-orientation solution.

So this leads to the question of, what is the fewest number of orientation tests 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), this can be done using the four orientation tests.

So this leaves the question of where to place v_4. The answer depends on the global orientation of the pentagon, 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. 3(d)). This can be done with three orientation tests, for a total of $4 + 3 = 7$ orientation tests.

Solution to Challenge Problem 2: Recall that the sequence is $h^* \leftarrow \min\left(\sqrt{2}^{\sqrt{2}^i}, n\right)$. We will use the fact that $\lg_{\sqrt{2}} x = \lg x / \lg\sqrt{2} = 2\lg x$. The algorithm terminates when

$$i \;=\; \left\lceil \lg_{\sqrt{2}} \lg_{\sqrt{2}} h \right\rceil \;=\; \lceil 2\lg(2\lg h)\rceil \;\le\; (2\lg(2\lg h)) + 1.$$

Thus, the running time is at most

$$n \sum_{i=1}^{(2\lg(2\lg h))+1} \lg \sqrt{2}^{\sqrt{2}^i} \;=\; n \sum_{i=1}^{(2\lg(2\lg h))+1} \sqrt{2}^i \lg\sqrt{2} \;=\; \frac{n}{2} \sum_{i=1}^{(2\lg(2\lg h))+1} 2^{(i/2)}.$$

This is a geometric series, and hence its value is asymptotically dominated by its largest term. Thus, up to constant factors, the running time is at most

$$n2^{\frac{(2\lg(2\lg h))+1}{2}} \;\le\; n2^{(\lg(2\lg h))+1} \;=\; 2n2^{\lg(2\lg h)} \;=\; 2n(2\lg h) \;=\; 4n\lg h.$$

This is asymptotically the same as Chan's algorithm. By the way, I don't believe that there is anything particularly special about $\sqrt{2}$. This analysis should hold for c^{c^i}, for any $c > 1$.