Below are some notes and sketches of correct answers for the first homework exam. We have also included the most common mistakes that were made. If anything is still unclear, feel free to ask Marc about Questions 2 and 4, or Wouter about Questions 1 and 3.

1. (1 point) Write pseudo-code for the following operation on a doubly-connected edge list: Given a vertex (object) \( v \), return the number of different faces that are incident to \( v \). Use the proper, agreed names of the doubly-connected edge list as in the textbook. Before giving the pseudo-code, first describe in a few sentences what your code does. Then give the pseudo-code.

**Answer:** The idea is to loop around \( v \)'s incident edges in clockwise order by following \texttt{Next} and \texttt{Twin} pointers, and to use \texttt{IncidentFace} to find all incident faces during this loop. You need to take extra care to prevent faces from being counted twice. In Algorithm 1 below, we use a mark bit that denotes whether a face has been visited. (To revert these bits afterwards, you could run the algorithm again and set the same flags back to \texttt{false}.)

There are multiple ways to prevent faces from being counted twice. If you use mark bits (as above), this algorithm runs in \( O(d) \) time. Alternatively, you can use a data structure that keeps track of the reported faces in sorted order (if you specify on what criterion you can sort them). The algorithm then takes \( O(d \log d) \) time. Either answer deserved a full point, as long as you did not say anything that was incorrect.

**Algorithm 1 GETNUMBEROFINCIDENTFACES(\( v \))**

\begin{verbatim}
1: \( e_0 \leftarrow \text{IncidentEdge}(v) \)
   \{Count the first face\}
2: \( F_0 \leftarrow \text{IncidentFace}(e_0) \)
3: \( F_0.\text{visited} \leftarrow \text{true} \)
4: \( \text{result} \leftarrow 1 \)
   \{Loop over the half-edges with origin \( v \}\}
5: \( e \leftarrow \text{Next}(\text{Twin}(e_0)) \)
6: \( \text{while } e \neq e_0 \)
7: \( F \leftarrow \text{IncidentFace}(e) \)
8: \( \text{if } F.\text{visited} = \text{false} \)
9: \( F.\text{visited} \leftarrow \text{true} \)
10: \( \text{result} \leftarrow \text{result} + 1 \)
11: \( \text{end if} \)
12: \( e \leftarrow \text{Next}(\text{Twin}(e)) \)
13: \( \text{end while} \)
14: \( \text{return } \text{result} \)
\end{verbatim}

There are multiple ways to prevent faces from being counted twice. If you use mark bits (as above), this algorithm runs in \( O(d) \) time. Alternatively, you can use a data structure that keeps track of the reported faces in sorted order (if you specify on what criterion you can sort them). The algorithm then takes \( O(d \log d) \) time. Either answer deserved a full point, as long as you did not say anything that was incorrect.
Notes: Overall, your solutions for preventing duplicate faces were divided about fifty-fifty between mark bits and some type of sorting. Some students used hash sets and assumed that their operations always run in constant time; this is a dangerous claim if you don’t specify what makes a face hash-able. If you forgot to check for duplicate faces altogether, you could earn at most 0.5 points.

Most of you interpreted ‘first describe in a few sentences what your code does’ as ‘first write the pseudocode in plain text’, which does not really help ;) We were expecting a short high-level description such as the one given at the beginning of this answer.

2. (1 point) Write pseudo-code for the following operation on a doubly-connected edge list: Given a half-edge \( \vec{e} \), report the coordinates of the vertex with the lowest \( x \)-coordinate in the DCEL which \( \vec{e} \) is part of. Again, first describe in a few sentences what your pseudo-code does.

Notes: It is easy to see that any correct answer must contain Twin and InnerComponents somewhere, otherwise it can never be correct. You could use this for a sanity check.

Some students assumed that the graph formed by the vertices and edges of the subdivision is connected. This made the question much easier, and therefore no points were given.

Some students assumed that the Twin of a half-edge incident to a face \( f \) is always incident to a different face than \( f \). This is not true: a half-edge and its twin can be incident to the same face. This can even hold for an edge that is incident to the leftmost vertex of that face. In this case only half the points were given for this question.

Some students thought that IncidentEdge is a whole list of pointers to half-edges that have the vertex as the origin. Others thought that InnerComponents contains pointers to all half-edges in the InnerComponents of a face, instead of just one per cycle. This costs points too.

3. Let \( P \) be a set of \( n \geq 3 \) points in the plane. No two points of \( P \) coincide, and no three points of \( P \) lie on a single line; you may assume this in this whole question. We define the concept of a dented convex hull: a dented convex hull of \( P \) is a simple polygon \( Q \) whose vertices are points of \( P \), all points of \( P \) lie in \( Q \) or on the boundary of \( Q \), and at most one vertex of \( Q \) is a point of \( P \) that does not lie on the convex hull.

Notes: For this question, we first introduce some extra terminology. We call a point of \( P \) an interior point if it does not lie on the boundary of the convex hull \( Q \). A really dented convex hull (RDCH) is a dented convex hull (DCH) that is not equal to the convex hull \( Q \) itself. For any RDCH \( D \), let the dent vertex \( v_D \) be the single interior point of \( P \) on the boundary of \( D \). Let the dent be the triangle \( \Delta abv_D \) that is ‘cut out’ of the convex hull by replacing an edge \((a, b)\) of \( Q \) with two edges \((a, v_D)\) and \((b, v_D)\).

By definition of a dented convex hull, no other points of \( P \) can lie inside this dent.

(a) (1 point) Prove that if not all points lie on the convex hull, then there always is a dented convex hull with smaller area than the convex hull itself.

Answer: Using the extra definitions above, we can rephrase the question to this: prove that if \( P \) has at least one interior point, then there is at least one RDCH with an area smaller than the area of \( Q \).
Proving the ‘smaller area’ part of this question is relatively easy. For any RDCH $D$, the dent has a non-zero area (because no three points lie on a line). Also, because $v_D$ lies inside $Q$, the entire dent lies inside $Q$, so the area of $D$ is the area of $Q$ minus the area of the dent. A short argument such as this was already enough.

The ‘there is at least one RDCH’ part was a bit more difficult. One possible answer is the following: For any edge $(a, b)$ on the boundary of $Q$, let $p^*$ be an interior point of $P$ that is closest to $(a, b)$. The triangle $\Delta abp^*$ does not contain any other points of $P$, because otherwise $p^*$ would not be closest to $(a, b)$. Therefore, this triangle is a valid dent, and we have found an RDCH.

Notes: For the second part, many of you used a recursive argument that is also correct. Let $(a, b)$ be any edge on the boundary of $Q$, and let $p$ be any interior point of $P$. If $\Delta abp$ is empty, then it is a valid dent. Otherwise, replace $p$ by an arbitrary point inside $\Delta abp$ and recurse. Because the triangle contains fewer points in each iteration, we will eventually end up with a valid dent.

Some students focused only on the ‘smaller area’ part and forgot that a dent should be empty.

(b) (1 point) Show that a set $P$ of $n$ points admits $\Theta(n^2)$ different dented convex hulls in the worst case. Recall that a worst-case $\Theta(\cdot)$ bound represents a worst-case $O(\cdot)$ bound and a worst-case $\Omega(\cdot)$ bound.

Answer: In general, to prove a worst-case upper bound, you need to argue that the problem can never be more complex than a certain bound. To prove a worst-case lower bound, you need to supply an actual example in which this bound occurs, in which the trick is that $O$s and $\Omega$s don’t care about constants. Most of you made this distinction correctly.

Upper bound: There are at most $O(n)$ edges on the convex hull boundary, and at most $O(n)$ internal points. If each edge can form a dented convex hull with each internal point, this gives $O(n^2)$ RDCHs in total, plus the convex hull itself. We cannot create more options because there are not more pairs of edges and internal points to choose from. (In this argument, it does not yet matter whether an edge-point pair actually yields a valid dent.)

Lower bound: Remember that we are looking for a concrete example now. A good starting point for creating quadratically many DCHs is to have a point set $P$ with $\Theta(n)$ points (e.g. one half of $P$) on the boundary of $Q$ and $\Theta(n)$ internal points (e.g. the other half). You can lay-out the points in such a way that $\Theta(n)$ boundary edges can each be replaced by $\Theta(n)$ valid dents. One example is shown below: for each of the boundary edges of $Q$ between two red points, we can create a valid DCH by creating a dent with any of the blue points. This gives $\Theta(n^2)$ DCHs for this particular example, proving that $\Omega(n^2)$ is the worst-case lower bound in general.
Notes: We received many different correct examples for the $\Omega(n^2)$ lower bound. Note that we officially expected a class of examples, parameterized by $n$. If you used concrete numbers that did not depend on $n$ (e.g. ‘place 10 points on a circle and 20 points inside it’), you did not receive the full points.

Some of your examples were degenerate cases with three or more points on a line. This situation was explicitly excluded from the question, but we still accepted such an answer.

Some students only proved the upper bound, and they used a $\Theta$ without realizing that it was actually just an $O$ (but, admittedly, a very fancy-looking one).

(c) (2 points) The optimal dented convex hull is the one with minimum area. Give an algorithm to compute the optimal dented convex hull and analyze its running time. Efficiency is good, but it is more important to make sure that your solution is correct.

Answer: There is a simple naive algorithm that takes $O(n^3)$ time and is already worth the full 2 points. Start by computing the convex hull $Q$. For each edge $(a, b)$ on the boundary of $Q$, try all interior points $p$ of $P$, and check if the triangle $\triangle abp$ does not contain any other (interior) points of $P$. If so, then $\triangle abp$ is a candidate dent. Throughout this algorithm, keep track of the candidate dent with the largest area. Of course, this largest dent corresponds to the RDCH with the smallest area. At the end, return this RDCH as the answer. If there are no valid dents at all, return the convex hull $Q$ itself.

This algorithm takes $O(n)$ time per candidate triangle, so $O(n^2)$ time per boundary edge of $Q$, so $O(n^2)$ time in total.

Notes: In the running time analysis, some students made a distinction between the number of interior points (say $m$) and the number of points on the boundary of $Q$ (say $h$). This gives a more informative bound of e.g. $O(n \log h + hm^2)$. Note that the $O(n \log h)$ part is still needed in case $m = 0$; if you overlooked that, we did not subtract any points (this time).

Some of you computed the area of each candidate RDCH from scratch, overlooking the fact that we only need to know the area of a triangle for each candidate option. This did not cost you points unless you wrote it in such a way that the overall running time was harmed.

Most students forgot that the convex hull $Q$ itself is also a valid DCH. If there are no interior points, then there is no RDCH, and $Q$ is actually the only DCH. Overlooking this special case led to a small deduction of 0.2 points.
If you forgot that a dent should be empty, then the problem/algorithm becomes considerably easier, and you received at most 1 point for this subquestion. There are faster algorithms than this naive solution. You could earn up to 0.5 bonus points by describing one in a convincing manner. For any interior point \( p \), you can find all valid RDCHs with dent vertex \( p \) in \( O(n \log n) \) time by using a rotational sweep around \( p \). Alternatively, for any boundary edge \( (a, b) \) of \( Q \), a rotational sweep can help you find all valid RCDHs in which \( (a, b) \) is replaced. Both options yield a total running time of \( O(n^2 \log n) \). Most students who went this far chose the second option, but the first option is probably easier to explain. It was sometimes hard to see why an algorithm was correct; please take extra care to convince us whenever your solution is particularly complicated. You could even have solved the problem in \( O(n^2) \) time, e.g. by translating the problem to its dual setting as introduced in Chapter 8 of the book. However, this would have been very ambitious at this point in the course ;) We are not yet aware of any faster solutions.

4. Let \( S \) be a set of \( n \) line segments in the plane. Let \( k \) denote the number of intersection points among line segments in \( S \). Assume that for every intersection point there are just two line segments of \( S \) that pass through it, and there is no pair of line segments that have more than a single point in common. We are interested in computing a shortest horizontal 3-stabber: a horizontal line segment that intersects at least three line segments of \( S \) and is as short as possible.

(a) (1 point) Analyze what different cases can occur for the shortest horizontal 3-stabber, including degenerate ones that were not excluded in the problem statement above.

Notes: Several students did not refer in their answer to the shortest horizontal 3-stabber. They only described situations for the input line segments from \( S \), but not where the shortest horizontal 3-stabber would be in these cases. The question explicitly asked for cases about the shortest horizontal 3-stabber, so an answer not mentioning it cannot be correct.

Other students referred to horizontal line segments and vertical line segments only. Many students forgot to mention that it can be that a shortest horizontal 3-stabber does not exist (or they had this case only when \( S \) contains only one or two line segments). If this was the only problem with this subquestion, the answer would still get full points, but any additional mistake(s) leads to subtraction of a half point or a full point.

(b) (1 point) Imagine a plane sweep algorithm to solve this problem. The initialization, the status, the status structure, and the event list can be exactly the same as in Chapter 2 of the textbook. Only the events need to be handled differently. Describe in sufficient detail how the events are handled.

Notes: If your answer handled the cases with multiple events on the same y-coordinate correctly and extensively enough, you could get 0.5 point bonus here.

(c) (1 point) Assume that we are interested in the shortest horizontal \( m \)-stabber, where \( m \) is a parameter between 3 and \( n \). Explain how the algorithm should be adapted
to solve the shortest horizontal $m$-stabber problem, and analyze the running time, expressing it in $n$, $k$, and $m$ (or possibly a subset of these).

**Notes:** Most students had the general idea correct: you need to check $O(m)$ horizontal line segments at an event to be sure that you find the shortest horizontal 3-stabber.

Handling an event now does not take $O(\log n)$ time, it also takes time depending on $m$. If you wrote that $O(m \log n)$ time is needed for an event, and multiplied this with $O(n + k)$ events, your answer is correct. In fact, you can find $m$ consecutive leaves in any balanced binary search tree in $O(m + \log n)$ time, which gives a slightly better bound.

If you were really clever, you can notice that upper and lower endpoint events indeed require you to spend $O(m + \log n)$ time, but at an intersection point event, there are only two possibilities for shortest horizontal $m$-stabbers. Namely, where this intersection is leftmost or rightmost. This means we would have to find the leaves that are $m$-2 to the left and $m$-2 to the right of the intersection point, which can be done in $O(\log n)$ time if the tree stores the size of each subtree in each internal node. Now, the $O(n)$ endpoint events take $O(m + \log n)$ time and the $O(k)$ intersection point events take $O(\log n)$ time. Multiply and add, and get a full bonus point.

We awarded the 10th point for proper terminology and notation. You could fail to get it if your formulations were vague, if you used ‘line’ instead of ‘line segment’, if you omitted the $O(\ldots)$ where it was needed, if your notation was inconsistent, if you used the same notation for two different things in the same answer, etc. In case of only one violation, you could still get the full extra point. With two types of violations you would get only 0.5 point, and if it was particularly poor, you could lose the full point here. This applies only to violations that have not been counted already in subtractions of points in the answer (you won’t be missing points twice for one reason).