1. If the black-box algorithm returns “no” on the call to the original 3-CNF expression, then no satisfying assignment can be found. Otherwise if “yes” is returned, iterate over each variable x_i found in Φ. At each iteration, replace all its regular occurrences in Φ with True and all its negation occurrences with False, then call the black-box algorithm on this new version of Φ. If “yes” is still returned, that variable can be set to True in the final satisfying assignment, otherwise False. Revert Φ back to its original state and repeat iterations until you have a satisfying assignment that contains every variable. Note that the iterations and the calling of the black-box algorithm can be done in polynomial time.

2. Currently, the black-box algorithm can be used to solve some instances of the Independent Set Problem, however it does not return a valid decision when the graph G is not connected. To solve this problem, we can add an arbitrary node u to the vertex set of G and create an edge from u to every other node of the vertex set, creating G*. Given an arbitrary undirected graph G, and a number k > 1 (if k is 1, you can alternatively solve the problem by simply determining whether G contains a node), G contains an independent set of size at least k if and only if the black-box algorithm returns “yes” for G* and k. Note that any independent set found in G* by the black-box algorithm can also be found in G, as it cannot contain the node u if k > 1. Adding the extra node and calling the black-box algorithm both can be done in polynomial time and thus the Independent Set Problem can be solved in polynomial time. This approach assumes that G contains at least 1 node. If G’s vertex set is empty, then return “yes” if k = 0 and “no” if k > 1 instead of calling the black-box algorithm.

3.
To prove that Hitting Set is NP-complete, we will first show that it is NP. Consider the following certificate and verifier:
Certificate: a set $H \subseteq A$ for $B_1, B_2, \ldots, B_m$ so that the size of H is at most k (note that this certificate is of polynomial size)
Verifier: checks that the size of H is at most k, checks that $H \subseteq A$, and checks that H contains at least 1 element from each B_i (note that the verifier only needs time polynomial in the input size)

To show that Hitting Set is NP-hard, we will show that Vertex Cover (VC) $\leq_p$ Hitting Set (HS). We are given an instance of the VC problem, consisting of graph G = (V, E) and number k. We want to convert this into an instance of the HS problem. We will do so as follows: Let A correspond to the vertex set V of G. For each edge in E, we create a set in the collection that contains the 2 vertices that are attached to the edge. Observe that this mapping only requires time polynomial in the size of the input.
Claim 1. G has a vertex cover of size at most k if and only if A has a hitting set of size at most k.
Proof of claim. If a vertex cover VC with a size of at most k forms a vertex cover of G, then for any edge in the graph, one of its vertex endpoints is in VC. Therefore, this would also form a hitting set because the collection is made from edges, and for each edge VC “hits” at least 1

endpoint. Conversely, if A has a hitting set H of size at most k, then for any edge in the graph, one of its vertex endpoints is in H, which would form a vertex cover.

This proves that Hitting Set is NP-complete.

4a. I enjoyed learning about dynamic programming because it was interesting to see how complex problems could be simplified to an O(n) time complexity. Also, out of all the algorithms they are probably the most practical in the real world.
4b. Greedy algorithms were extremely easy to understand as the implementation was often very self-explanatory (just keep taking the locally optimal choices).
4c. NP-completeness and NP-hardness were slightly difficult to understand as the proofs were different to the ones we have learned so far.