ALGORITHMS, FALL 2018, HOMEWORK 9

1. The problem here is straightforward, because we can use an early exit to make the algorithm faster (because we want to detect if the graph has *any* cycles). We can use depth first search here, to iterate over each node and its edges to see if any of the edges point at a node already traversed. The problem is $O(V + E)$ because the worst case is if the graph *doesn't* have any cycles, and so we have iterated over every node and all of its edges exactly once. Otherwise we would have exited early, only hitting one item twice. (the edge that traverses backwards).

2. This issue is very similar to the issue presented in the previous problem, but here we need to first generate levels of exclusion in the graph. A directed graph can also be treated as a tree, such that each level of nodes will not have an edge pointing at a higher level. We can use this to make sure that no nodes will create "cycles" by early exiting when a node is pointing at a node of a higher level. This is similar to depth-first search because we now keep track of visited nodes as well as nodes that are "above" the current nodes.

   The runtime is still $O(V+E)$ because we only traverse each node once, adding it to our trackers, and then moving on along its edges. Early exits prevent us from traversing the same node twice, limiting us to $V + E$.