Question 1:

To tell whether or not a graph is in fact a tree, two conditions must be met: 1. The graph has no cycles. 2. The graph is entirely connected. In order to make sure that these two conditions are satisfied we can run a BFS on the adjacency list and mark nodes as visited. If all the nodes are visited in the end then condition 2. has been met. To check for a cycle, for a visited node, if that node has an adjacent visited node and is not a parent of the other node, then we know that there must be a cycle in the graph. To test these two conditions would take $O(n + m)$ time.

Question 2:

2.1 A

The spreadsheet of names to people not liked is essentially an adjacency list for a graph. If there is a cycle in the graph we would have a problem, but as long as the shortest cycle is longer than three then it is fine. If the shortest cycle is 3, it is not possible to place them onto 2 floors. E.g. For the cycle $A \to B \to C \to A$, that means that person A dislikes B and so they need to be on different floors so $\frac{A}{B}$, but person B dislikes C so they also need to be on different floors, so $\frac{AC}{B}$, but this is not possible since C dislikes A. To detect these potential issues, we can run the algorithm discussed in Question 1 to detect whether or not there is a cycle in the graph and the length of that cycle. Time and space complexity would be the same as in Question 1, $O(n + m)$ where n is the number of vertices, aka the number of people and k is the number of edges/values in the spreadsheet.

2.2 B

Modifying the above algorithm for f floors instead of 2 I noticed that, for a group of three people to fit on two floors, they obviously can't all dislike each other, so extrapolating upwards, for a group of 4 people to fit on 3 floors they also can't all dislike each other. So a simple modification to the algorithm above can be that if there exists a cycle in which more people all dislike each other than is the total number of floors, then it wouldn't be possible to organize them all correctly on the floors. This would pattern would be the same regardless of the number of floors and people due to the pigeonhole principle.

Question 3:

We can modify a given graph to contain info on whether it has been reached by the infection or the antidote first. We can then modify Dijkstra's shortest path algorithm to ignore paths through nodes of opposite type, i.e. ignore paths through infected nodes if you are the antidote and vice-versa if you're the infection. We can then, for each node that is not x nor y, run the algorithm and see if that node is reached first by x or y, i.e. if the path from x to n is shorter than from y to n then mark the node with an x, following the rules above for not traveling through a node you're not allowed to. Continuing in this way we can mark every node in the graph with either $infected$, *antidote*, or *neither* (*neither* meaning neither can use that node, a tie). You would need to run Dijkstra's twice for every node, once starting at x, once starting at y, excluding nodes x and y so the runtime would be the two times number of vertices minus 2, $2(v-2)$, times the expected time complexity of Dijkstra's, so $O(2(v-2)) \cdot O(\text{Dijkstra's})$.