ALGORITHMS, FALL 2018, HOMEWORK 9

1. To add a vertex to a graph and then calculate the minimum spanning tree, we have to consider the possibility that introducing the vertex will introduce a better path through the vertex.

   But we can also notice that there is no way we will select an edge from the graph that is neither part of the current MST or the new edges created by the vertex, so the set of edges we have to check is much smaller.

   The time complexity of this method would still be $O(E + E')$, where E' is the number of edges introduced by the vertex.

2. This problem is very similar to finding the MST of a graph. We can use Kruskal's algorithm to find the minimum spanning tree in $O(E \log V)$ time.

   Then we just find the path from x to y in $O(E)$ time, which makes the runtime in total $O(E \log V + E)$ time.

3. A solution to this issue is to use something similar to Djikstra's algorithm.

   We can create a min-heap of vertices, with each item in the heap holding both the min-distance from the infected node and the min-distance from the antidoted node. This way every time we pop off the heap we always get the next node to be touched in chronological order.

   This also allows us to detect collisions. If a node comes off the heap and both their infected distance and their antidoted distance are the same, we can then treat it as being an 'invalid' path through the graph, and not add update its adjacent nodes with more distances.

   Once we have updated all of the distances in the graph, we can then iterate across all of the nodes in $O(V)$, and check whether the distance to antidote or distance to infection is smaller. This tells us whether or not its fully infected or fully protected.

   This would run in $O(V \log V)$ time, the same runtime as Djikstra's algorithm.

   P.S. antidote is probably not the right word for this? I think vaccine makes a little more sense.