Question 1:

We can decide whether the graph is a tree using the following algorithm:

NumberOfEdges ← 0
NodeVisited ← {}
For all nodes n:
    For all m in adjacency list of n:
        NumberOfEdges ← NumberOfEdges + 1
        NodeVisited.addm
If (NodeVisited == Nodes of original graph) and (NumberOfEdges/2 == Number of nodes -1):
    Then the graph is tree

Correctness: What my algorithm do is basically check if every node is connected with some other nodes and compute the total number of edges. We then use the fact that a tree will have one less edge than the number of nodes to check if it is a tree.

Runtime Analysis: We need to iterate through every node once and every edge once, so the algorithm will have run time $O(m + n)$

If we are given G in the adjacency matrix format instead,
Change in algorithm: for node n, we will have to go through the corresponding row to find out the number of edges and the nodes it is connected to

Change in runtime: we will have to go through each row to count the number of zeros and ones to determine the number of edges. Thus. the algorithm will have runtime $O(n^2)$

Question 2:

(a) Runtime of Bellman-Ford Algorithm is O(mn), since it relax every pair of nodes that is connected by an edge on each iteration and repeat for n-1 times.

(b) If we are given k, then we only need to repeat for k times, so the new runtime is O(mk)

(c) We could first pretend the graph is not weighted and run the BFS on the graph, we can then find the the maximum length of all shortest paths (measured in sum of weights) from s in G. Let this number be k, we could then use our algorithm in (b)

New runtime = runtime of BFS + runtime from part(b) = O(m+n) + O(km) = O(km + n + m)

Question 3:

We will try to modify Dijkstra's algorithm to solve this problem.
Redefine ExtractMin(Q) so that it returns all items with minimum value.

Using v.dx, v.dy to denote Currently known weight of minimum-weight path from x and y respectively to v. And $v.\pi x, v.\pi y$ to represent v's parent in the shortest path from x and y respectively.

Dijkstra(G = (V,E),x, y, w):
Initialize x.dx = 0,all other v.dx = ∞
Initialize y.dy = 0,all other v.dy = ∞
All parents None.
$Q \leftarrow$ min-priority queue of all vertices (using score min{dx,dy} as key).
$S_{infected} \leftarrow$ empty set of infected elements.
$S_{vaccinated} \leftarrow$ empty set of vaccinated elements.
$S_{damaged} \leftarrow$ empty set of damaged elements.
While Q is not empty do:
    L $\leftarrow$ ExtractMin(Q)
    For u in L:
        if $u.dx < u.dy$ and $u.\pi x$ is not in $S_{infected}$:
            $S_{infected} \leftarrow S_{infected} \cup \{u\}$
        else if $u.dx > u.dy$ and $u.\pi y$ is not in $S_{damaged}$:
            $S_{vaccinated} \leftarrow S_{vaccinated} \cup \{u\}$
        else if $u.dx = u.dy$ and $u.\pi y$,$u.\pi x$ are not in $S_{damaged}$:
            $S_{damaged} \leftarrow S_{damaged} \cup \{u\}$
        else:
            $u.dx = u.dy = \infty$
        For all edges (u,v) E:
            if $(v.dx > u.dx + w(u,v))$ and u is not in $S_{damaged}$::
                $v.dx \leftarrow u.dx + w(u,v), v.\pi x \leftarrow u$,Float v in Q
            if $(v.dy > u.dy + w(u,v))$ and u is not in $S_{damaged}$:
                $v.dy \leftarrow u.dy + w(u,v), v.\pi y \leftarrow u$,Float v in Q

nodes in set $S_{infected}$, $S_{vaccinated}$ and $S_{damaged}$ are nodes that got infected, vaccinated and damaged respectively.

Note that some node could still be in none of the three set after running the algorithm. They are just separated with the rest of the graph by some damaged nodes.

runtime analysis and space complexities: Since this algorithm is basically Dijkstra's algorithm except it takes multiple minimum element at once, it should have same runtime and space complexity as Dijkstra's algorithm. So runtime:θ(n + m log m), space complexity:θ(n)