Q1)

(g) True

Justification: Because whether we are using Prim or Kruskal, the main idea of both algorithm is pick the least weight safe edge. If we negated the weights of the graph, the edges of the largest value, becomes the edge of the least value and vice versa, and so, the algorithm will pick the least weight edges (were the largest weights in the original graph), thus it constructs the largest weight MST.

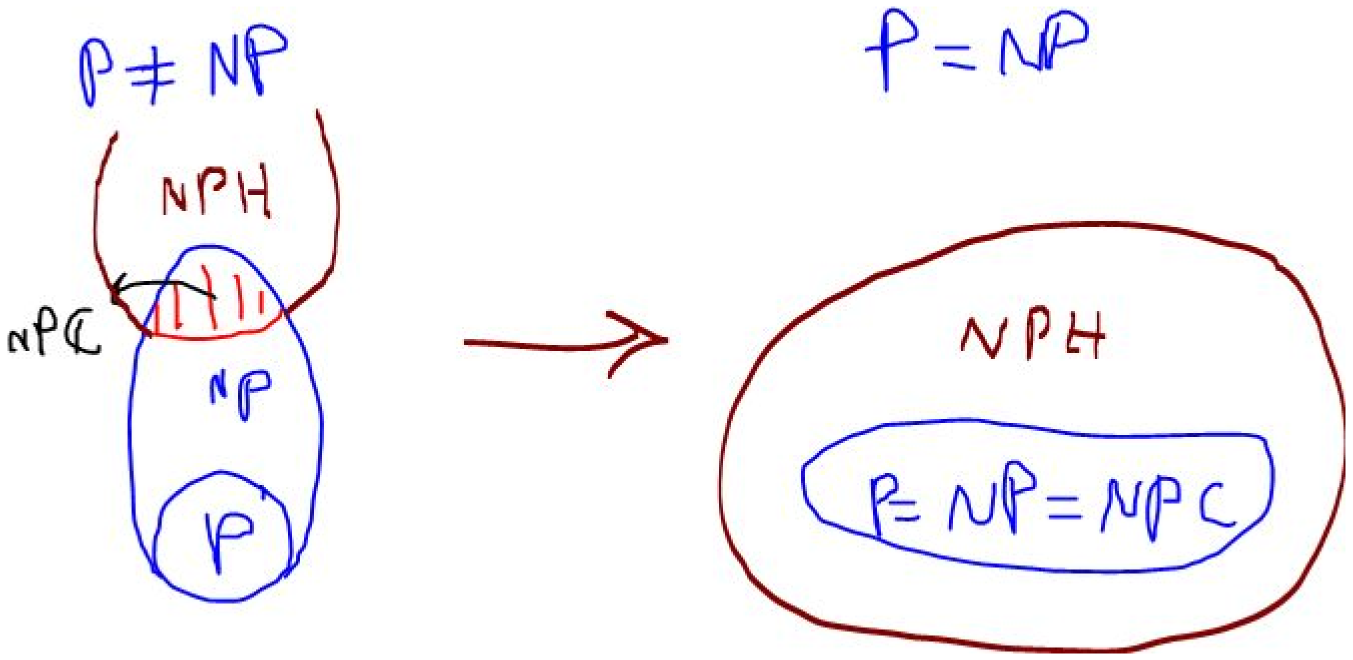
(b) I am not sure what's T. But if it's the MST, then it's false, and we can use the counterexample used in the exam. The MST in the example has a cost of 2+2 = 4. In that MST, we can go from s to t with a cost of 4. But the shortest path between s and t is actually 3 using the direct edge. If we tried to include the direct edge between s and t in the MST, it will be of cost 5, and thus not a "Minimum" spanning tree anymore.

(3) True.

Justification: Let's consider Prim for example, the main idea is to pick the least weight edge that connects a node from the set vertices currently in the MST to a node that belongs to the set of vertices currently not in the MST. A similar reasoning may follow for Kruskal as well but in terms of connected components.

(4) True. (I am not sure)

Justification:

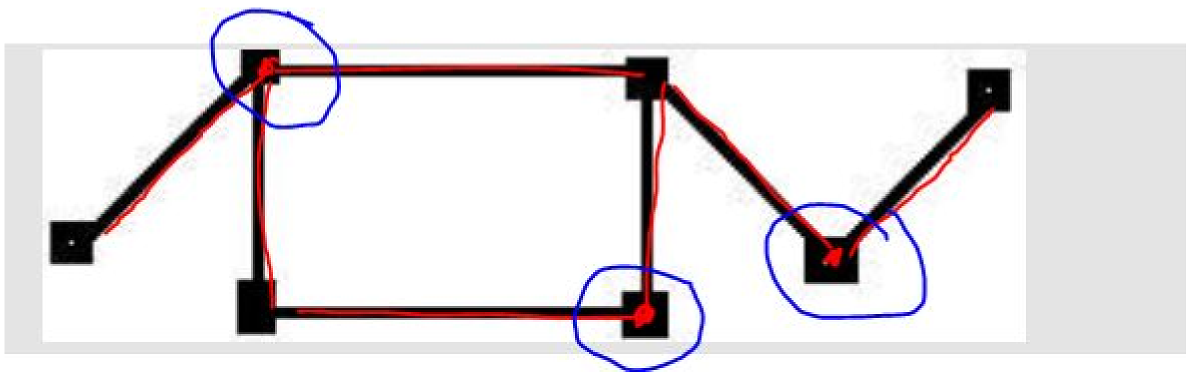


Q2)

(1) W(9) Because we do V-1 iterations (longest possible number of edges in a shortest path).

(2) From the Clique problem.

(3) 3 vertices.



(5) Kosraju's Algorithm (Strongly Connected Components). But we do produce at least 1 set not exactly 1 set.

Q3)

(1) False
Justification: Because MST algorithms only depend on sorting the edges according to their weights. So if we added a constant to all weights, the sorting will not change.

Q4)

Here: https://ideone.com/xcLzTE

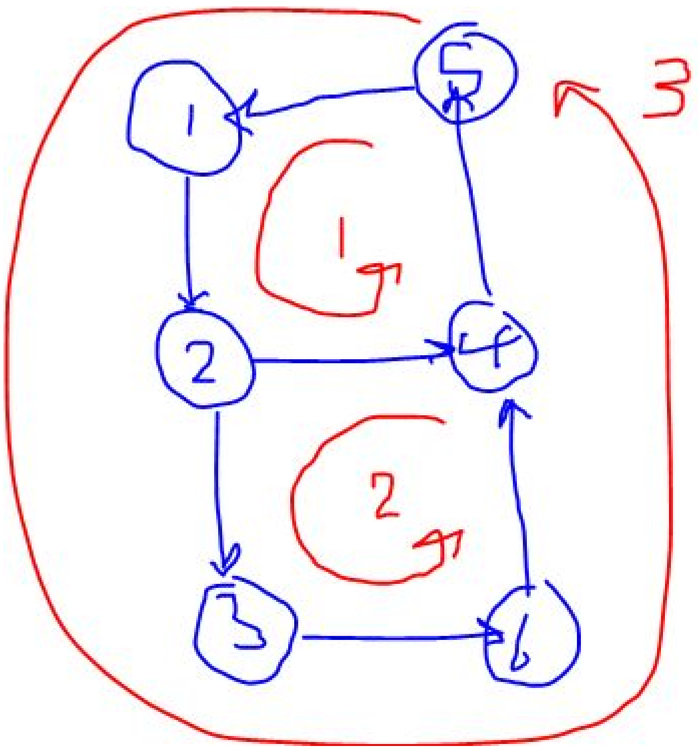
Q5)

Assuming he only needs the length of the LIS not the actual LIS.
Recursion only: https://ideone.com/32RnaF O(2^n)
DP: https://ideone.com/3l0u85 O(n^2)
Within the course content, we can't do an optimal greedy solution [Just describe any greedy approach and give a counterexample]. But in fact, there is an O(N.log(n)) solution that can obtain the length of the LIS (but no the LIS itself). Meanwhile, the DP solution can get the LIS itself.

Q6) I am sorry doctor, but there is not. The number of (all) cycles in a graph is exponential. Enumerating all of them is an NP-Hard problem (I guess :V). The only cycles we can count in polynomial time is simple cycles, but we haven't studied the algorithm to do so.

I honestly think that he tried to modify the this problem from GeekforGeeks
https://www.geeksforgeeks.org/print-all-the-cycles-in-an-undirected-graph/?fbclid=IwAR0IyjmiEq3QeocrUzFmeqPZ1mjx1NhCFFVFoVlAv55u_cn72yffYk2HSI4

The thing is, the solution presented in the article is incorrect. It doesn't count all cycles, it doesn't count all simple cycles, it's totally messed up. For example consider cases that has composite cycles, like so

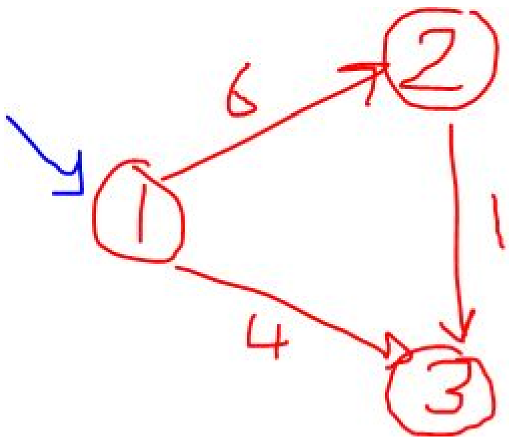


The solution in the article will get the bigger cycle (count = 1), or maybe the two small cycles (by luck, depends on the order at which the dfs processes the nodes, so count = 2), but never all 3.

Q7)

(a) No. Because Prim tries to choose the least weight safe edge. It's not guaranteed that such choice will produce MST.

Counter example.
The algorithm starts at node 1. Picks the least weight edge that connects 1 to node outside the MST, so it picks the edge (1, 3) with cost 4. And then it picks another edge, not it picks (1,2) with cost 6, so the total is 4+6=10. But a better solution is to take edge (1,2) and the edge (2,3) with cost 6+1=7. So it didn't produce and MST.



(b) The question seems to be considering it space wise. The adjacency matrix takes O(V^2) space, and the adjacency list takes O(V+E) space. And since 1 entry in the matrix occupies 1 word, while 1 entry in the list occupies 2 words, then we can favor the matrix to the list only if

1 * V^2 < 2 * (V+E)

V < 2 * (1 + E/V)

This usually happens when the graph is dense, i.e. E is closer to V(V-1)/2 (V^2), and thus E/V is closer to V.

(c) If the graph is tree, that means that E = V - 1. So the graph is sparse. E << V^2. So we can use Johnson's algorithm. Just explain Johnson and compare the complexities.

(d) Because the shortest path might change in the re-weighted graph. Check this counterexample.
https://youtu.be/1O_x5-1gBuY?list=PLN4dVDLBrOps0wo6u4lBf298OpI7-V5Oq&t=529

(e) Because having cycles will violate the sort property. Having a cycle (backedge) implies that I need to visit node u before node v, and at the same time, I need to visit node v before u. We can use an example like the one in the video about courses. If math1 is a prerequisite for math2, and math2 is a prerequisite for math1, then there is no order at which I can take a course without violating the dependencies (sorting order).

Other:

Prime:	Greedy
Floyed:	DP
Dijikstra:	Greedy
Kruskal:	Greedy
Bellman Ford:	DP