Q 1

1. All 3 possibilities are correct. Justification is what matters.
2. Any correct example will do. If {u.d, u.f] labels are not given, but you can reconstruct the DFS take 2 marks off.
3. Any correct example will do along with the justification. Source should be marked.

(d)

The suggested algorithm, do single source shortest path on DAG by reversing signs of edge weights is the desired solution. Another solution is to run Bellman Ford single source shortest paths after reversing signs of edge weights. Then, -(shortest path length in the reversed sign graph) = longest path length in the original graph.

Note that that Bellman Ford over DAGs needs only one pass of edge relaxations in the topological order, but the question just asks whether it can be done, which is true.

Single source shortest path on DAG was done in the class. However, direct solutions are possible and acceptable.

For example, here is a direct solution.

1. Sort vertices of G in topological order.
2. Set u.cost = -infty for all vertices u.
3. s.cost = 0
4. For vertices u starting from s in increasing topolog. order do
   1. for every vertex v adjacent to u do
      1. If v.cost < u.cost + w(u,v)
         1. v.cost = u.cost + w(u,v)
         2. v.parent = u

Q2

Algorithm is 13 marks, proof of correctness is 12 marks. Of 12 marks proof, 3 marks are for the sufficiency of bipartiteness and 9 for necessary condition.

Just correct algorithm with statement of sufficiency gets 16 marks.

Q3

Several correct algorithms are possible.

1. Run DFS and pick the vertex with highest finish time.
2. Run another DFS from this vertex and check if no white vertices are left upon termination.

Obvously algo is linear time. This much deserves full credit. Correctness is rather obvious. Someone who has got this far clearly knows what he/she is doing.

Another algorithm. Compute the strong connected component graph G^{SCC}.

Pick a source component in the G^{SCC}. How? By constructing the reverse graph of G^{SCC} and scanning the adjacency list to find a vertex with no outgoing vertices. If there is exactly one such vertex return YES else return NO. Some argument is needed for this algorithm, so 5 marks for the argument and 20 for the algorithm. This is a neat algorithm though. Linear time is obvious.

Q4. The model soln gives the only solution I understand that is efficient. If someone uses Bellman Ford instead and is otherwise correct, give 21/22 marks, since time complexity is not so good, but is correct.