34.2-3:

So, for this algorithm it would first need to find the Hamiltonian cycle. If there is a Hamiltonian cycle in it would proceed to producing the order of the Hamiltonian cycle.

Algorithm for checking for a Hamiltonian cycle:

1. First select a node which would be N
2. Edges(E) of the selected node
3. Then calculate the pair of the edges E1,E2 ϵ E_n and construct a graph that is G'(N,(E-E_v)U{E1, E2}) contains a Hamiltonian cycle.
4. Then this would return true if the cycle exists
5. If not then return false.

If the Hamiltonian cycle is true then we would end up printing the Hamiltonian cycle of G in O(N) time and this would be be put together in polynomial time.

34.2-7:

The best thing to start doing with a directed acyclic graph is to do a topological sort. This can be done in O(n+m) time. Once this sort is done then we know now that the edges are going from lower index vertices to the higher index vertices. This would mean that there would exists a Hamiltonian cycle because there would be edges between the adjacent vertices. In a Hamiltonian path we can't go back, and we have to visit all the nodes, so we are able to not skip by having them sorted.

35.1-4:

With a vertex cover we are needing to have at least one vertex for each edge. In a tree we can assume that there are at least two leaves. So, the tree will always have an edge that is adjacent to a leaf. So, we basically trimming the tree tell we have an isolated edge and picking that vertex. Identifying the leaves can be done in linear time by just using the breadth first search. This greedy algorithm would be adding one vertex at each iteration.

GreedyVertexCover(G)

1. Let C = 0
2. While V cannot equal 0 then
3. Finding a leaf vertex V by using a breadth first search
4. Locate u which is the parent of v
5. C = C U {u}
6. Then remote all the edges incident to u
7. Return C

Non deterministic algorithm for K-clique:

For a k-clique we are seeing whether a set of k vertices will for m a complete graph by checking for all the solutions. This will give us the non-deterministic approach because we can possibly have a different correct solution in each set of a k vertices.

1. S= null
2. For(i=1; I <= k; i++)
3. V = set of choices(n)
4. If v is a set of elements in A then
5. Return false
6. V = to all the nodes in V U all nodes in A
7. For all (I,j) where the I is a element of V and J is a elemnt of V and I cannot equal J then
8. If (I,j) is not the edge of a graph then
9. Return false
10. Else return true