Written Assignment 5

1. Algorithm to find number of trees in a forest

```
DFS-Count(G)
1. For each vertex u
2.         u.color = WHITE
3.         u.π = NIL
4. time = 0 , count = 0
5. For each vertex u
6.         if u.color == WHITE
7.             count = count + 1
8.             DFS_VISIT(G, u)

DFS_VISIT(G, u)
1. time = time + 1 , u.d = time , u.color = GRAY
2. For each v in adjacency list of u
3.         if v.color == WHITE
4.             v.color = GRAY , v.π = u
5.             DFS_VISIT(G, v)
6. u.color = BLACK
7. time = time + 1
8. u.f = time
```

Proof :-

By definition a forest is a disconnected graph whose components are all ~~trees~~ trees. Therefore, if we use the algorithm DFS to find each tree we can find the number of disconnected components or trees.

Running time = is same as DFS = $O(|V| + |E|)$

2. Reference :- Kahn's Algorithm
en.wikipedia.org/wiki/Topological_sorting

Kahn's Algorithm.

```
1. L ← Empty list
2. S ← Set of all nodes with InDegree == 0
3 While S is not empty do
4.          n = S.pop()
5.          L.append(n)
6.          For each m in adjacency list of n with edge e
7.                  remove edge e from the graph
8.                  if m has no other edges.
9.                          S.append(m)
10 return L
```

Input :- The Graph must be a DAG
Output :- A Topologically sorted list L.

Time Complexity :-
1. Calculation of In Degrees = $O(|E|)$
2. Outer loop runs V times and inner loop runs E times. As we are removing edges as we encounter them the complexity is $O(|V| + |E|)$

∴ Total Time Complexity = $O(|V|+|E|)$

Algorithm to find set S with nodes having In Degree == 0

```
1. S ← Set of all ~~nodes~~ vertices in graph G
2. for each edge e in G
3.                u = ~~node~~ vertex1 , v = vertex2
4.                If v is in S
5.                        S.remove(v)
6. return S.
```

3 We can use Topological ~~sorting~~ to test if G is a DAG.

We can modify Kahn's algorithm to check if any edges are left in the graph. If edges are present then the graph will have at least one cycle. Therefore, it will prove if it is a DAG or not

Kahn's Algorithm.

```
1. L ← Empty list
2. S ← Set of all nodes with InDegree == 0
3 while S is not empty do
4.                n = S.pop()
5                 L.append(n)
6.                For each m in adjacency list of n with
                                        edge e
7.                        remove edge e from the Graph
8.                        if m has no other edges
9.                                S.append(m)
10. if graph has edges
11                return 'Not a DAG'
```

12. else
13 return "It is a DAG"

Time Complexity :-
1. Calculation of InDegrees = $O(|E|)$
2. Kahn's algorithm = $O(|V|+|E|)$
[as described in Q27]
∴ Time Complexity = $O(|V|+|E|)$