7.5 DFS and why you mark your territory: to prevent explosions

a)
F(k) = {
  1, k = 1;
  1 + F(k-1) + 2 + 2F(k-2) + 4 + 4F(k-3) + ..., otherwise;
}

Incorrect. F(k) represents the number of times the print statement is executed, so a F(k-1) call already counts the 2(k-1) times print is called. In my answer, I counted the number of times the recursive call is executed instead.

b)
F(k-1) = {
  1 + F(k-2) + 2 + 2F(k-3) + 4 + 4F(k-4)+ ...
} (general case)

F(k) - F(k-1) = 2 + F(k-2) + 2 + 2F(k-3) + 4 + 4F(k-4)...
= 1 + F(k-1)

F(k) = 1 + 2F(k-1)

F(k) = 1 + 2 + 4 + ... + 2^(k-1) = 2^k - 1

Correct

c)
```
global seen [];
procedure DFS(k;;);
  seen.append(k);
  if k > 0 then
    for h <- k - 1 downto 0 do
      if h not in seen then
        print(k, h);
        DFS(h);
      end if;
    end for;
  end if;
end_DFS;
```

I did not write the recurrence equation, which is simply P(n) = n + P(n-1).

d)
Exactly once for every edge:
n - 1 + n - 2 + n - 3 + ... + 1 = n * (n + 1) / 2

Correct

e)
For the WWKWW problem, having a lookup table saves time when the same calculation is required. Instead of repeating the calculation, the algorithm simply looks for the value that is stored from the last calculation. In this case, it is similar that the seen array stores nodes that have been traversed, so the subsequence recursive calls can skip these nodes and avoid extra printing.

Correct

7.7 Topoligical sort

When a Tarjan algorithm processes through the first repeated node in a cycle, the node would be marked "started" but not yet "done". This would be a sign of a cycle, so the algorithm can execute its cycle processing subsequence.

Correct

7.8 Topoligical sort

When a BFS algorithm runs in a graph that has a cycle, the cycle part will not have a "handle" which can allow every node within the cycle to enter the "ready" queue to be sorted. For that reason, the BFS algorithm will fail when there is a cycle in the graph.

Correct. The BFS will still complete in the presence of a cycle, but it will not be correct.

7.10 Preorder and postorder vertex numberings

a)
All nodes from the root to u's children.

Correct

b)
If u.pre > v.pre and u.post > v.post, u is a descendant of v.

Correct

c)
Since a parent-child relationship is reversed in preorder and postorder, the two nodes will have a reversed positioning in these two orderings.

Correct

7.Xa

```
global G = (V,E);
global adj[1...n] stores all egdes starting from 1 to n;
global Ready q;

// prepare every node
for every node k, add its number of edges to its k.val;
push nodes with its val = 0 to q;

procedure BFS(G, k);
   // start from the ready q;
   while q is not empty,
      k <- q.pop;
      for all vertices m connected to k,
         m.val - 1;
         if m.val = 0 then q.append(m);
         recursively call BFS(m);
      end for;
   end while;
   // after the while loop, q should be empty.
   print(k);
end_BFS;
```

The line in red is extra, since there is a while loop.

7.Xb

```
// High-level Tarjan's Reverse Topoligical Sort
// prepare all nodes
mark all nodes as not done and not started

global G = (V,E);
procedure TarjanDFS(G;;w);
   mark w as started;
   for all neighbors m of w do
      if m is unstarted then TarjanDFS(m);
      else if m is undone then // there is a cycle
         start cycle processing;
      end if;
   end for;
   print(w);
   mark w as done;
end_TarjanDFS;
```

Correct