Problem 1 SCC

1. RUN DFS on this Graph

(a) **For each vertex, show values d (discovery), f (finish), and π (parent).**

	d	f	π
a	1	20	NIL
b	3	18	c
c	2	19	a
d	4	17	b
e	9	11	h
f	7	10	e
g	8	9	f
h	5	16	d
i	12	15	h
j	13	14	i
k	21	22	NIL
l	23	24	NIL

(b) **Write the vertices in order from largest to smallest finish time:**
From the largest to smallest

l,k,a,c,b,d,h, i, j, e, f, g

2. RUN DFS on Transpose Graph

Now run DFS on the transpose graph, visiting vertices in order of largest to smallest finish time from the DFS of step 1 (as required by the SCC algorithm). Again, show values d (discovery), f (finish), and π (parent).

	d	f	π
a	5	10	NIL
b	6	9	a
c	7	8	b
d	11	16	NIL
e	12	15	e
f	21	24	NIL
g	22	23	f
h	13	14	e
i	17	18	NIL
j	13	14	NIL
k	3	4	NIL
l	1	2	NIL

3. List the strongly connected components you found by first listing the tree edges in the transpose graph that define the SCC, and then listing the vertices in the SCC

- SCC 1: Tree edges: $\{\}$; Verticies $\{l\}$
- SCC 2: Tree edges: $\{k\}$
- SCC 3: Tree edges: $\{(a,b),(b,c)\}$; Verticies: $\{a,b,c\}$
- SCC 4: Tree edges: $\{(d,e),(e,h)\}$; Verticies: $\{d,e,h\}$
- SCC 5: Tree edges: $\{\}$; Verticies: $\{i\}$
- SCC 5: Tree edges: $\{\}$; Verticies: $\{j\}$
- SCC 6: Tree edges: $\{(f,g)\}$; Verticies: $\{f,g\}$

Problem 2) Bottom Up Longest-Path

(a) **Show your pseudocodes for Longest-Path-Bottom-Up**

Algorithm 1 Longest-Path-Memoized(G,u,t,dist,next)

```
if u = t
    dist[u] = 0
    return 0
else
    for each v in G.adj[u]
        alt = w(u,v) + Longest-Path-Memoized(g,v,t,dist,next)
        if dist[u] < alt
            dist[u] = alt
            next[u] = v
    return dist[u]
```

Algorithm 2 Longest-Path-Bottom-Up(G, s, t, dist, prev)

```
let dist[1 ... n] and prev[1...n] be called the new arrays to work with
Topological-Sort(G)
for i = 1 to $|G.V|$
    dist[i] == $-\infty$
dist[s] == 0
for each $u \in G.V$ in topological order starting from s
    for each edge (u,v) $\in$ G.adj[u]
        if dist[u] + w(u,v) > dist[v]
            dist[v] = dist[u] + w(u,v)
            prev[v] = u
return(dist, next)
```

(b) **Explain why your algorithm works; in particular, why topological sort is useful.**

At line 6, "**for** each $u \in G.V$ in topological order starting from s" solves the problem in smaller pieces before going to the larger problems and by the time it hits vertex "u", it has processed all verticies lower in the topological order which ensures on covering the edges that are incoming to u.

Hence, all the verticies we can reach the present vertex will have computed their longest path solution to the start vertex, s.

(c) **Analyze its asymptotic run time.**

From the CLRS book the runtime of Topological Sort is $\Theta(V + E)$ which occurs at the line 2. Form line 3 to 4 is at least takes $\Theta(V)$ and the loop at line 6 executes $|V|$ times.

Hence, the overall result is Θ(V+E).

Problem 3 ”Really Bad Networks”

(a) **Show your pseudocode.**

Algorithm 3 CHECK-NETWORK(G)

```
S = a random selection of vertex $\in$ G
DFS(S)
for each vertex v $\in$ G
    if v is not visited
        return FALSE
    else
        return true
```

(b) **Explain why it works.**
By using DFS search we can determine whether two nodes have a path between them. DFS looks at the all of the children nodes of x until it reaches y.
Furthermore, DFS has the properties as it tells that

- x and y are complete disjoint
- x is a descendant of y
- y is a descendant of x

Hence, by using DFS one can find a connection between two nodes.

(c) **Show the run-time**
From the CLRS book the runtime of DFS so at line 2 is O(V+E) and from line 4 to 7 the for loop runs O(V) so the overall runtime is O(V(V+E)).

Problem 4: "Counting Simple Paths in DAG"

(a) **Show the pseudocode.**

Algorithm 4 Simple-Path(G,s,t)

```
1: for each v ∈ G.V
2:     v.p = 0
3: t.p = 1
4: Compute $G^T = (V, E^T)$ //the transpose of G
5: TOPOLOGICAL-SORT($G^T$)
6: for each v ∈ the topological order of $G^T$
7:     if v == s
8:         return v.p
9:     for each u ∈ $G^T$adj[v]
10:        u.p = u.p + v.p
```

(b) **Explain why it works.**
We know that the path from vertex v to t is the sum of the number of paths from the neighbor that an edge to v.
Furthermore, we also know that sum of the neighbor values are correct since they are topologically closer to t so have already been processed.

(c) **Analyze its asymptotic run time.**
From the CLRS book Topological sort at line 5 takes $O(V + E)$.
From line 1 to 3 it takes $O(V)$ and from page 616 of our CLRS book to create the transpose takes $O(V + E)$.
Hence the overall run time is O(V+E).