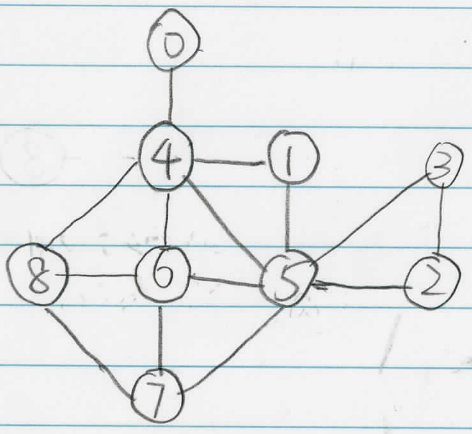
1. (a).



(b).

0 → 4 /

1 → 4 → 5 /

2 → 3 → 5 /

3 → 2 → 5 /

4 → 0 → 1 → 5 → 6 → 8 /

5 → 1 → 2 → 3 → 4 → 6 → 7 /

6 → 4 → 5 → 7 → 8 /

7 → 5 → 6 → 8 /

8 → 4 → 6 → 7 /

(c)

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	1	0	0	0
2	0	0	0	1	0	1	0	0	0
3	0	0	1	0	0	1	0	0	0
4	1	1	0	0	0	1	1	0	1
5	0	1	1	1	1	0	1	1	0
6	0	0	0	0	1	1	0	1	1
7	0	0	0	0	0	1	1	0	1
8	0	0	0	0	1	0	1	1	0

2. (a) 0, 4, 1, 5, 2, 3, 7, 6, 8
(b) 0, 4, 1, 5, 6, 8, 2, 3, 7

3. Each of the k components is a tree, say component i has V_i vertices and $V_i - 1$ edges. In total there are $\sum_i^k (V_i - 1) = n - k$ edges, that is $m = n - k$.

4. Input: a graph G,
Output: Determine whethere there is an unique topological order.

Initialize an empty list L1;
Initialize an empty list L2;
Add all vertices with no incoming edges into L2;
check whethere there are duplicate vertices;
while L2 is not empty:
    v ← Remove the last in L2;
    L1.add(v);

for all the vertices w with an edge e from v to w do
remove edge e from G;
if w has no other incoming edges then
push w into L2;
if G has edges left then
return false; (there is no topological order)
else
return L1;

8

5\. Input: A graph g with n vertices.
Output: Return true if g can be colored in 2 colors

```
int i<-0;
while i<n do:
        g.setVisted(i, false);
        i++;
boolean color = true;
Queue q = new Queue;
g.setColor(0, color)
q.enqueue(0);
while (q.empty() == True) do:
        i<-q.dequeue();
        while(g.getVisted()==True) do:
                g.setVisted(i, true);
                color <- g.getColor(i)==True;
                for each j ∈ g.getNeighbors(i) do:
                        if (g.getVisted(j) and (g.getColor(j)=color)==True
                                return False;
                        if (g.getVisted(j)==True){
                                g.setColor(j)color);
                                q.enqueue(j);
                        }

return
```