1 Consider an undirected graph with $n \geq 2$ vertices. What are the minimum and maximum number of different layers that the graph could have, respectively?

1. 1 and $n - 1$
2. 2 and $n - 1$
3. 1 and n
4. 2 and n

(4) In a graph with $n = 2$ we will have exactly two layers there cannot be any less because the source vertex is in layer 0. In a graph with n vertices we can have at most n layers. There cannot be any more than n layers because every layers is disjoint and must contain at least one element.

2 Consider an undirected graph with n vertices and m edges. What are the minimum and maximum number of connected components that the graph could have, respectively?

1. 1 and $n - 1$
2. 1 and n
3. 1 and $max\{m, n\}$
4. 2 and $max\{m, n\}$

(2) We'll have a connect component of size 1 if we have a connected graph. We'll have n number of connected components for a graph where no vertex has an edge with any other.

3 How many different topological orderings does the following graph have with the vertex set $V = \{1, 2, 3, 4\}$ and edge set $E = \{(1, 2), (1, 3), (2, 4), (3, 4)\}$? Use only the labels $\{1, 2, 3, 4\}$.

1. 0
2. 1
3. 2
4. 3

(3) We have two topological orderings. $\{1, 2, 3, 4\}$ and $\{1, 3, 2, 4\}$.

4 What happens when the *TopoSort* algorithm is run on a graph with a directed cycle?

1. The algorithm might or might not loop forever.
2. The algorithm always loops forever.
3. The algorithm always halts, and may or may not successfully compute a topological ordering.
4. The algorithm always halts, and never successfully computes a topological ordering.

(4) The algorithm will halt because we will not visit a vertex again that have already been marked. The algorithm will not compute a topological ordering because a topological ordering requires a DAG.

5 Consider a directed acyclic graph with n vertices and m edges. What are the minimum and maximum number of strongly connected components that the graph could have respectively?

1. 1 and 1
2. 1 and n
3. 1 and m
4. n and n

(4) A DAG will have a minimum and maximum of n vertices.

6 Let G be a directed graph and G^{rev} a copy of G with the direction of every edge reversed. How are the $SCCS$ of G and G^{rev} related?

1. In general, they are unrelated.
2. Every SCC of G is also an SCC of G^{rev}, and conversely.
3. Every source SCC of G is also a source SCC of G^{rev}
4. Every sink SCC of G becomes a source SCC of G^{rev}

(2) and (4).

7 Which of the following statements hold? as usual, n and m denote the number of vertices and edges, respectively, of a graph.

1. Breadth-first search can be used to compute the connected components of an undirected graph in $O(m+n)$ time.
2. Breadth-first search can be used to compute the lengths of shortest paths from a starting vertex to every other vertex in $O(m+n)$ time, where "shortest" means having the fewest number of edges.
3. Depth-first search can be used to compute the strongly connected components of a directed graph in $O(m+n)$ time.
4. Depth-first search can be used to compute a topological ordering of a DAG in $O(m+n)$ time.

(1), (2), (3) and (4) are all valid

8 What is the running time of depth-first search, as a function of n and m, if the input graph is represented by an adjacency matrix? You may assume the graph does not have parallel edges.

1. $\Theta(m+n)$
2. $\Theta(m+n \cdot \log n)$
3. $\Theta(n^2)$
4. $\Theta(m \cdot n)$

(3) Given an $n \times n$ adjaceny matrix we have to search the entire col of a vertex v in order to know all of its neighbors. Hence for depth-first in the worst case we first we have to search the rows of every vertex and then the columns of that vertex to finds its neighbors. this gives us order of n^2 and thus we conclude $\Theta(n^2)$.

9 This problem explores the relationship between two definitions concerning graph distances. In this problem, we consider only graphs that are undirected and connected. The *diameter* of a graph is the maximum, over all choices of vertices v and w, of the shortest-path distance between v and w. Next, for a vertex v, let $l(v)$ denote the maximum, over all vertices w, of the shortest-path distance between v and w. The *radius* of a graph is the minimum value of $l(v)$, over all choices of the vertex v. Which of the following inequalities relating the radius r to the diameter d hold in every undirected connected graph?

1. $r \leq \frac{d}{2}$

2. $r \leq d$
3. $r \geq \frac{d}{2}$
4. $r \geq d$

(2), (3) is correct. (1), (4) can be shown to be wrong by counter-examples.

10 When does a directed graph have a unique topological ordering?

1. Whenever it is directed acyclic
2. Whenever it has a unique cycle.
3. Whenever it contains a directed path that visits every vertex exactly once.
4. None of the other options are correct.

(4) None of the other options are correct

11 Consider running the *TopoSort* algorithm on a directed graph G that is not directed acyclic. The algorithm will not compute a topological ordering. Does it compute an ordering that minimizes the number of edges that travel backward?

1. The *TopoSort* algorithm always compute an ordering of the vertices that minimizes the number of backward edges
2. the *TopoSort* algorithm never computes an ordering of the vertices that minimizes the number of backward edges.
3. There are examples in which the *TopoSort* algorithm computes an ordering of the vertices that minimizes the number of backward edges, and also examples in which it doesn't
4. The *TopoSort* algorithm computes an ordering of the vertices that minimizes the number of backward edges if and only if the input graph is a directed cycle.

(3) is correct.

12 If you add one new edge to a directed graph G, then the number of strongly connected components

1. might or might not remain the same
2. cannot decrease
3. cannot increase
4. cannot decrease by more than 1

(1) might or might not remain the same

13 Recall the *Kosaraju* algorithm from which uses two passes of depth-first search to compute the strongly connected components of a directed graph. Which of the following statements are true?

1. The algorithm would remain ccorrect if it used breadth-first search instead of depth-first search in both its passes
2. The algorithm would remain correct if we used breadth-first search instead of depth-first search in its first pass
3. The algorithm would remain correct if we use breadth-first search instead of depth-first search in its second pass.
4. The algorithm is not correct unless it uses depth-first search in both its passes

(3) is correct. We are required to use depth-first search in our first pass to compute the reverse postorder of the reversed graph. From that use the connected components algorithm on the original graph which can implemented via a breadth-first search or a depth-first search.

14 Recall that in the *Kosaraju* algorithm, the first pass of depth-first seach operates on the reversed version of the input graph and the second the original input graph. Which of the following statements are true?

1. The algorithm would remain correct if in the first pass it assigned vertex positions in increasing order and in the considered the vertics in decreasing order of vertex position
2. The algorithm would remain correct if it used the original input graph in its first pass and the reversed graph in its second pass.
3. The algorithm would remain correct if it used the original input graph in both passes, provided in the first pass it assigned vertex positions in increasing order.
4. The algorithm would remain correct if it used the origin input graph in both passes, provided in the scond pass it considered the vertices order of vertex position.

(1) is correct because we still consider the vertices in the same order as the original algorithm. (2) is correct because we recall that the $SCCS$ of a graph and its reversal are the same. Thus we can consider reversal of the graph as the input and run the algorithm and it would correctly compute the $SCCS$ of the graph.

15 In the $2SAT$ problem, you are given a set of clauses, each of which is the disjunction of two literals. You would like to assign a value "true" or "false" to each of the variables so that all the clauses are satisfied, with at least on true literal in each clause. For example, if the input contains the three clauses $x_1 \vee x_2$, $\neg x_1 \vee x_3$, and $\neg x_2 \vee \neg x_3$, then one way to satisfy all of them is to set x_1 and x_3 to "true" and x_2 to "false". Of the seven other possible truth assignments, only one satisfies all three clauses. Design an algorithm that determines whether or not a given $2SAT$ instance has at least one satisfying assignment Your algorithm should run in $O(m + n)$ time, where m and n are the number of clauses and variables, respectively.

We can form the implication digraph with $2 \cdot n$ vertices, one for the literal and the other for the negation. For each clause $x \vee y$, we include edges from $\neg y$ to x and $\neg x$ to y . We use the SCC algorithm to compute the $SCC's$ of the graph. If any literal x' has both it and its negation in the same SCC we have reached a contradiction and the equation cannot be satisfied, true otherwise.