Answer questions 1 and 2, and any 20 points from the rest.

1. (10 points) Suppose that we are given a set of depth first intervals of the nodes of a graph G as follows:


Answer the following queries for graph G. **Please refer to the attached figure.**

(a) What are the descendant and ancestor nodes of \( v[6] \) in G?
   
   **Ans:** The ancestor node: 7; descendant nodes: 1, 2, 5

(b) How many components are there in G? **Ans:** There are two connected components.

(c) Identify a pair of nodes in a connected component of G which are not related (i.e. one is neither a descendant nor an ancestor of the other).
   
   **Ans:** nodes 1 and 2. Nodes 3 and 4; and 2 and 5 are also not related.
(d) Construct the depth first tree of $G$ which realizes the dfs intervals as given. **Ans:** Please see the attached figure.

(e) Remove one node from $G$ such that the number of connected components remains the same. (Note that $G$ may have many edges which we are not aware of.) **Ans:** Any leaf node of a tree of size at least two in the DFS forest can be removed without increasing/decreasing the number of components. We can, therefore, remove $1, 2, 3$ or $4$.

2. (10 points) Run the strongly connected components algorithm on the following directed graph $G$. Whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

![Directed Graph](image)

Answer the following questions.

(a) In what order are the strongly connected components (SCCs) found?  
**Ans:** The strongly connected components determined in order are: $E, B, A$; $HGI, CDFJ$; The algorithm I have used to find the SCCs is:

- Step 1: Perform DFS on $G$ and compute $[\text{pre}(v), \text{post}(v)]$ interval for all $v$.
- Step 2: Order the vertices in decreasing post(*) values.
- Step 3: Compute the reverse graph $G^R$ of $G$.
- Step 4: Run DFS on $G$ and during the DFS, process the vertices in decreasing order of their post numbers from step1.

(b) Which are source SCCs and which are sink SCCs?  
**Ans:** The source SCCs: $E$ and $B$; sink SCC: $CDFJ$.

(c) Draw the "metagraph".  
**Ans:** See the following figure.
(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

**Ans:** Two directed edges (one directed from E to B, and the other from C to E), when added, will make the graph $G$ with no source or sink node.

3. (10 points) It is easy to see that for any graph $G$, both DFS and BFS will take almost the same amount of time. However the space requirements may be significantly different.

(a) Give an example of an $n$-vertex graph for which the height of the recursion tree of DFS from a particular vertex $v$ is $n - 1$ whereas the queue of BFS will have at most one vertex at any given time if BFS is started from the same vertex.

**Ans:** When the graph is a path as follows,

```
 o-----o-----o-----o-----o-----o-----o-----o
 v(1)   v(2)   v(3)        v(n)
```

the height of the recursion tree of DFS starting from $v(1)$ will be $n - 1$. The queue size of BFS will be one.
(b) Give an example of an $n$ vertex graph for which the queue of BFS will have $n-1$ vertices at one time whereas the height of the recursion tree of DFS is at most one. Both searches are started from the same vertex.

**Ans:** The graph has the following edges:

\[(v(1), v(2)), (v(1), v(3)), (v(1), v(4)), \ldots, (v(1), v(n))\]

and both the searches start from $v(1)$.

4. (10 points)

(a) Suppose $G$ is a connected undirected graph. An edge $e$ whose removal disconnects the graph is called a bridge. Must every bridge $e$ be an edge in a depth-first search tree of $G$, or can $e$ be a back edge? Either give a proof or a counterexample.

**Ans:** The removal of a bridge $e = (u, v)$ disconnects the graph. Let $C_u$ and $C_v$ be the two connected components containing $u$ and $v$ respectively. The edge $(u, v)$ connects $C_u$ and $C_v$. Without any loss of generality suppose DFS of $G$ enters $C_u$ first. The search will enter $C_v$ from $u$. All the vertices of $C_v$ will be descendant nodes of $v$. There will be no back edge from $C_v$ to $C_u$. This forces all DFS of $G$ to use the edge $(u, v)$ as a tree edge.

(b) Using a DFS on $G$, can you identify a bridge edge of $G$, if it exists.

**Ans:** From the above discussion we notice that when we are performing DFS and when we are backing up along a tree edge $(u, v)$ where $\text{pre}(u) < \text{pre}(v)$, we check if there is any back edge from $v$ or any descendant of $v$ to an ancestor of $v$. The ancestor nodes of $v$ are $u$ and all the ancestor nodes of $u$. $(u, v)$ is not a bridge if there is any such back edge, otherwise it is a bridge edge. This can be determined during the DFS process. It is possible to identify all the bridge edges of $G$ in $O(|V| + |E|)$ time. Problem 3.31 of the text deals with this problem.

5. (10 points) A mother vertex in a directed graph $G = (V, E)$ is a vertex $v$ such that all other vertices $G$ can be reached by a directed path from $v$.

(a) Give an $O(n + m)$ algorithm to test whether a given vertex $v$ is a mother of $G$, where $n = |V|$ and $m = |E|$.

**Ans:** We start DFS from $v$ and check if the DFS is a tree with $|V| - 1$ tree edges.

(b) Give an $O(n + m)$ algorithm to test whether graph $G$ contains a mother vertex.
Ans: Identify a source SCC vertex in the metagraph. Pick any vertex in the selected SCC component. Then check if it is mother vertex.

6. (10 points **Bonus**) Finding the Topological Sort of a Directed Acyclic Graph

Let $G=(V,A)$ be a directed acyclic graph that has an edge between every pair of vertices and whose vertices are labeled 1, 2, ..., $n$, where $n = |V|$. To determine the direction of an edge between two vertices in $V$, you are only allowed to ask a query. A query consists of two specified vertices $u$ and $v$ and is answered with:

- "from $u$ to $v$" if $(u,v)$ is in $A$, or
- "from $v$ to $u$" if $(v,u)$ is in $A$.

Determine the number of queries required in the worst case, as a function of $n$, to find a topological sort of $G$.

**Ans:** Our directed graph $G = (V,A)$ is a DAG and there is a directed edge between every two vertices, i.e. for any $u$ and $v$, of $V$, either $(u,v)$ or $(v,u)$, but not both, is an element of $A$. Since $G$ is a DAG, we can topologically sort the vertices of $G$, i.e. we can order the vertices on a line from left to right such that for any $i, j$, $i < j$ in the ordered list, the arc $(i,j)$ is a element of $A$. We can treat the topologically ordering the vertices problem as a comparison-based sorting problem since for any two vertices $u$ and $v$, $u$ is to the left of $v$ (i.e. $u$ is smaller than $v$) in the sorted order if there is an arc from $u$ to $v$. If $(v,u) \in A$, $v$ lies to the left of $u$ in the sorted order.

We know that we can sort $n$ elements using $O(n \log n)$ comparisons (mergesort, heapsort). Therefore we require $O(n \log n)$ queries to topologically order the vertices of $G$. 