INSTRUCTIONS. Be clear and concise. Write your answers in the space provided. Use the backs of pages, and/or the scratch page at the end, for your scratchwork. All graphs are assumed to be simple (no loops or multiple edges). Good luck!

You may freely use or cite the following subroutines from class:

- `dfs(G)`. This returns three arrays of size $|V|$: `pre`, `post`, and `cc`. If the graph has $k$ connected components, then the `cc` array assigns each node a number in the range 1 to $k$
- `bfs(G, s), dijkstra(G, l, s), bellman-ford(G, l, s)`. Each of these returns two arrays of size $|V|$: `dist` and `prev`. 
QUESTION 1. True/False, Short Answer.

Briefly justify your answers.

(a) (2 points) True or False: Depth-first search on a graph $G = (V, E)$ can be implemented to run in time $O((|V| + |E|)^3)$.

(b) (4 points) Give the big-O solution to the recurrence relation $T(n) = 27T(n^3) + O(\log n)$.

(c) (4 points) Give a recurrence relation for the running time $T(n)$ of the following pseudocode:

```plaintext
def dqContains([a_1, ..., a_n], k)
1 if n == 1
2 return a_1 == k
3 return dqContains([a_1, ..., a_{\lfloor n/2 \rfloor}], k) || dqContains([a_{\lfloor n/2 \rfloor} + 1, ..., a_n], k)
```
QUESTION 2. DFS, SCCs.

Refer to the graph $G$ shown above.

(a) (2 points) In the right side of the figure, draw all edges of the reverse graph $G^R$, then execute depth-first search on $G^R$, starting at vertex $A$, writing pre and post labels next to each vertex. Break all ties lexicographically (i.e., according to alphabetical order).

(b) (2 points) List the strongly connected components (SCCs) of $G$, in the order that they are found using the algorithm given in class.

(c) (2 points) Draw the metagraph of strongly connected components (SCCs) for the above graph $G$.

(d) (2 points) List the source and sink SCCs of the graph $G$.

(e) (2 points) The original graph $G$ can be made strongly connected by adding edges. Determine a minimum-cardinality set of directed edges that, if added to the original graph $G$, would make the graph strongly connected. If there are multiple solutions, any will do.
QUESTION 3. BELLMAN-FORD.

(a) (6 points) Suppose that the Bellman-Ford algorithm is executed on the graph shown below, with S as the source vertex. Fill in the table with the distance values of each vertex, at each iteration of the algorithm.

![Graph Diagram]

<table>
<thead>
<tr>
<th>iteration</th>
<th>s</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) (2 points) If we continue to run Bellman-Ford on the given graph, will the stored dist values change between iterations 15 and 16? Answer Yes or No, and give a one-sentence justification.

(c) (2 points) Suppose that an edge from E to A with weight -4 is added to the given graph. If we continue to run Bellman-Ford on the modified graph, will the stored dist values change between iterations 15 and 16? Answer Yes or No, and give a one-sentence justification.
QUESTION 4. D/Q.

You are given a list of \( n \) intervals \([x_i, y_i]\), where \( x_i \) and \( y_i \) are integers with \( x_i \leq y_i \). The interval \([x_i, y_i]\) represents the set of integers between \( x_i \) and \( y_i \), inclusive. For instance, the interval \([3, 6]\) represents the set \( \{3, 4, 5, 6\} \).

Define the overlap of two intervals \( I, I' \) to be \( I \cap I' \), i.e., the cardinality of their intersection (the number of integers that are included in both intervals).

Devise an \( O(n \log n) \) D/Q algorithm that, given \( n \) intervals, finds and outputs a pair of intervals that have the maximum overlap. Hint: In HW2 #4, we split a set of intervals by their left endpoints.

(Note: you will not receive credit for the naive, non-D/Q algorithm that tests the overlap of each pair of intervals in \( O(n^2) \) time.)

(a) (3 points) Describe your algorithm in English. Be very clear in describing your “merge” step.

(b) (5 points) Write down pseudocode of your algorithm.

(c) (2 points) Give an analysis of your algorithm’s runtime, \( T(n) \). (As applicable, you may invoke the Master Theorem.)
QUESTION 5. PATH-COUNTING IN A DAG.

Given a directed acyclic graph (DAG) $G = (V, E)$ and a source vertex $s \in V$, design an efficient algorithm to obtain the number of distinct paths from $s$ to all other vertices of $V$. [Hint: recall the discussion of finding longest / shortest paths in a DAG.]

In the example graph below, there are three simple paths from source vertex $S$ to vertex $D$: $S \rightarrow B \rightarrow D$, $S \rightarrow C \rightarrow D$, $S \rightarrow C \rightarrow B \rightarrow D$.

(a) (3 points) Give an English description of your algorithm.

(b) (4 points) Give pseudocode of your algorithm.

(c) (3 points) Analyze the runtime of your algorithm.
procedure dag-shortest-paths(G, l, s)
Input: DAG $G = (V, E)$;
      edge lengths $(l_e : e \in E)$; vertex $s \in V$
Output: For all vertices $u$ reachable from $s$, dist($u$) is set to the distance from $s$ to $u$.

for all $u \in V$:
  dist($u$) = $\infty$
  prev($u$) = nil

dist($s$) = 0
Linearize $G$
for each $u \in V$, in linearized order:
  for all edges $(u, v) \in E$:
    dist($v$) = min(dist($v$), dist($u$) + $l(u, v)$)