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. Good luck!

You may freely use or cite the following subroutines from class¹:

- `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)`
  This returns two arrays of size |V|: dist and prev.

¹Recall from class/text the time complexities (1) `dfs`: O(|V| + |E|); (2) `bfs`: O(|V| + |E|).
QUESTION 1.

(a) (5 points) Consider the following pseudocode:

```python
function example(n):
    x = 0
    if n = 1:
        return
    endif

    for i = 1 to n:
        for j = 1 to n:
            x = x + 1
        endfor

    return example(n/3)
```

State the recurrence relation for the running time $T(n)$ of the function `example(n)`. Solve the recurrence using the Master Theorem.

(b) (5 points) An algorithm solves problems of size $n$ by recursively solving two subproblems of size $n - 3$ and then combining the subproblems in constant time. Write the recurrence relation for the running time $T(n)$ of this algorithm. Give the running time in big-$O$ notation. (Show your work.)
QUESTION 2.

Refer to the graph $G$ shown below and answer the following questions.

(a) (5 points) In the right side of the figure, draw the reverse graph $G^R$ and execute DFS in $G^R$ starting from vertex $A$, breaking all ties in lexicographic order. Write the pre and post labels for each vertex in $G^R$.

(b) (3 points) In the following table, write down the SCC(s) of $G$ according to whether they are source SCC(s), sink SCC(s), or neither source nor sink SCC(s).

<table>
<thead>
<tr>
<th>SCC(s) that are source SCC(s) in $G$</th>
<th>SCC(s) that are sink SCC(s) in $G$</th>
<th>SCC(s) that are neither sink nor source SCC(s) in $G$</th>
</tr>
</thead>
</table>

(c) (2 points) What is the minimum number of edges that, when added to $G$, will make it strongly connected?
QUESTION 3.

Consider the following directed graph $G$, with source vertex $S$ and negative-weight edges only present from the source vertex.

![Graph Diagram]

(a) (5 points) Suppose Dijkstra's algorithm is executed on the graph, with $S$ as the source vertex. Fill in the table with the intermediate distance values of all the vertices at each iteration of the algorithm. [Note: You know from Homework #2, Problem 7 that Dijkstra’s algorithm correctly finds all source-sink shortest path lengths when any negative-weight edges in the graph are only from $S$, as in this case.]

<table>
<thead>
<tr>
<th>Iteration $l(S)$</th>
<th>$l(A)$</th>
<th>$l(B)$</th>
<th>$l(C)$</th>
<th>$l(D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) (5 points) Suppose the Bellman-Ford algorithm is executed on the same graph, with $S$ as the source vertex. Fill in the table with the intermediate distance values of all the vertices at each iteration of the algorithm.

<table>
<thead>
<tr>
<th>Iteration $k$</th>
<th>$l_S^k$</th>
<th>$l_A^k$</th>
<th>$l_B^k$</th>
<th>$l_C^k$</th>
<th>$l_D^k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 0$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$k = 1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k = 4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
QUESTION 4.

You are given an array $A[1...n]$ of $n$ elements. A *majority element* of $A$ is any element that occurs strictly more than $n/2$ times (so, if $n = 6$ or $n = 7$, any majority element will occur in at least four positions). For example, in the following array, 5 is a majority element

$[1, 5, 5, 5, 2, 6, 5, 1, 5]$

Assume that elements *cannot be sorted*, but *can only be compared for equality*.

(Thus, for example, you aren’t allowed to say “Sort the array in $O(n \log n)$ time, then go through the sorted array once in $O(n)$ time to see if an element occurs more than $n/2$ times”.)

(a) *(4 points)* Describe in words a divide-and-conquer algorithm to find a majority element in $A$ (or determine that no majority element exists) in $O(n \log n)$ time. Note: You **MUST** use a DQ algorithm for this problem.
(b) \textit{(4 points)} Give pseudocode for your algorithm in (a).

(c) \textit{(2 points)} Write the recurrence \( T(n) \) that characterizes the running time of your algorithm.
QUESTION 5.

A shortest path between two vertices $s \in V$ and $t \in V$ in an undirected, unweighted graph is a path from $s$ to $t$ that contains the least number of edges. Often there are multiple shortest paths between two vertices of a graph. Give a linear-time algorithm for the following task (i.e., your algorithm must run in $O(|V| + |E|)$ time).

**Input:** Undirected, unweighted graph $G = (V, E)$; vertices $s, t \in V$.

**Output:** The number of distinct shortest paths from $s$ to $t$.

For the following graph, your algorithm should return a value of 2, as there are two distinct shortest $s$-$t$ paths (of length 2 edges: $s \to a \to t$ and $s \to b \to t$). The path $s \to a \to b \to t$ has length 3 edges, and is not a shortest $s$-$t$ path.

![Graph Diagram](image)

(a) *(4 points)* Describe your algorithm in words.
(b) (4 points) Give pseudocode for your algorithm in (a).

(c) (2 points) Provide a time-complexity analysis in big-O notation based upon your pseudocode.