QUESTION 1. Short Answer.

Briefly justify your answers.

(a) (4 points) Give the big-O solution to the recurrence relation $T(n) = 3T(\frac{n}{2}) + O(n)$.

\begin{align*}
  a &= 3, \quad b = 2, \quad d = 1, \\
  a &> b^d \\
  \text{By M.T.,} \quad T(n) &= O(n \log_2 3) \quad (3 > 2^1)
\end{align*}

(b) (9 points) Match the following for an edge $(u, v)$ in a directed graph $G$, when depth-first search is executed. (Match the column entries on the left to the ones on the right.)

i. Tree Edge or Forward Edge \quad a. $\text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u)$

ii. Back edge \quad b. $\text{pre}(v) < \text{post}(v) < \text{pre}(u) < \text{post}(u)$

iii. Cross edge \quad c. $\text{pre}(v) < \text{pre}(u) < \text{post}(u) < \text{post}(v)$

(Note: There was another variant with minor reordering.)
(c) (4 points) Dijkstra's algorithm does not necessarily give correct results when the input graph has negative edge weights. Bob proposes that we add a large constant to each edge weight so that all edge weights are made positive, and then use Dijkstra's algorithm to find shortest paths from a given source to all reachable vertices.

Bob's idea unfortunately is flawed. Explain why, and show a small counterexample with at most five vertices.

Flaw: A shortest path in Bob's transformed (re-weighted) graph is not necessarily a shortest path in the input graph.

Counterexample:

```
A
 o-------
  
  o-------
  
  o-------
  
  o-------
```

True SP from S to C is S - A - B - C.

Adding a large positive constant $M (> 5)$ to each edge weight doesn't prevent Dijkstra from returning S - B - C in the shortest-paths tree.

(d) (4 points) Prove or disprove: If $f(n)$ is $O(g(n))$ then $2^{f(n)}$ is $O(2^{g(n)})$.

- Consider $f = \log n^2$, $g = \log n$
  
  $f = 2 \cdot g = O(g)$.
  
  But $2^{\log n^2} = n^2 \not\in O(2^{\log n}) = O(n)$
(c) (5 points) Consider the following directed graph $G$, with source vertex $s$. The labels on the edges represent their weights.

![Graph Diagram]

Execute the Bellman-Ford algorithm on this graph, with $s$ as the source vertex. (Recall that $d_j^{(k)}$ denotes the shortest $s \rightarrow v_j$ pathlength using $\leq k$ edges.) Fill in the table below with the distance values $d_j^{(k)}$ of all the vertices $v_j$ for the first three iterations of the algorithm ($k = 1, 2, 3$).

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$s$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</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>0</td>
<td>10</td>
<td>$\infty$</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$k = 2$</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>3</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$k = 3$</td>
<td>0</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>
QUESTION 2. DFS, BFS, SCC-Finding.

For this question, refer to the graph above.

(a) (4 points) List the strongly connected components (SCCs) in the given graph $G$, in the order that they are found using the algorithm given in class. Show your work, including $G^R$ and pre/post numbers.

- $d$ has highest post number in DFS of $G^R$.
- $\text{explore}(d)$ in $G^R$ finds the SCC $\{d\}$.
- $a$ has next highest post number.
- $\text{explore}(a)$ in $G^R$ finds the SCC $\{a, b, c, e, s\}$.

List of SCCs in order found: $\{d\}, \{a, b, c, e, s\}$
(b) (2 points) What is the minimum number of edges that, when added to the given graph $G$, will make it strongly connected?

Adding any one of the edges $(d,a)$, $(d,b)$, $(d,c)$, $(d,e)$, $(d,s)$ suffices to make the graph strongly connected.

(c) (4 points) Execute breadth-first search on the given graph starting at vertex $s$. List the vertices along with their distances from $s$ in the order that the vertices are enqueued during the BFS.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance from $s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>$e$</td>
<td>2</td>
</tr>
</tbody>
</table>
QUESTION 3. Peak Finding

Let \( A[1..n] \) be an array of \( n \) distinct numbers. The sequence of numbers in \( A \) has a peak at index \( i \) if the sub-array \( A[1...i] \) is increasing (i.e., \( A[j] < A[j+1] \) for \( 1 \leq j \leq i-1 \)) and the sub-array \( A[i...n] \) is decreasing (i.e., \( A[j] > A[j+1] \) for \( i \leq j < n \)). The index \( i \) is called the peak of \( A \).

For example, the array \([4,6,9,5,1]\) has a peak 3 since the array elements from index 1 to index 3 are in increasing order, and the array elements from index 3 to index 5 are in decreasing order.

Design an efficient algorithm that, when given an array \( A \) of \( n \) distinct numbers that has a peak, returns the index \( i \) of the peak. (NOTE: A linear-time algorithm will not receive points.)

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

If an element is peak, it will be greater than its previous and next element.


If \( A[mid-1] < A[mid] < A[mid+1] \) then peak will lie on right subarray (between mid+1 and high) since mid element is in increasing part of the array.


If subarray size is 1 then that index will be peak since \( A \) is guaranteed to have a peak.

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

```plaintext
PeakFind (A, 1, n).

Procedure: PeakFind (A, low, high) {
    if (low > high)
        return low;
    mid = low + high; 2
        return mid;
        return PeakFind (A, mid+1, high);
    else return PeakFind (A, low, mid-1);
}
```
(c) (3 points) Analyze the runtime of your algorithm.

I divide array into two equal-sized subarray.
Decide in constant time which subarray to solve and solve only one subarray.

Hence, $T(n) = T(n/2) + O(1)$.

Using master's theorem ($a=1, b=2, d=0$; $d=\log_2(a)$)

$T(n) = O(\log n)$. 
QUESTION 4. Minimum-Weight Cycle Finding

Let $G = (V,E)$ be a positively-weighted, directed graph. Design an efficient algorithm to find the weight of the minimum-weight simple cycle in the graph. To be considered “efficient”, your algorithm’s big-O runtime complexity must be no more than a factor of $|V|$ larger than some runtime complexity given on the cover sheet of the exam.

A simple cycle is a cycle in which no vertex or edge is repeated, except the starting or the ending vertex. The weight of a cycle is the sum of the weights of the edges in the cycle.

[Hint: A cycle may be viewed as being composed of an edge $u \rightarrow v$ and a path $v \rightarrow u$.]

For example, in the graph shown above, $A \rightarrow B \rightarrow C \rightarrow A$ is the minimum-weight simple cycle, with weight = 4. For the above graph, your algorithm should return the weight value 4.

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

A simple cycle is composed of an edge $u \rightarrow v$ and a path $v \rightarrow u$.

We find shortest paths for all pairs $(v,u)$ by running Dijkstra’s algorithm starting from each vertex $v \in V$. We will label the length of shortest path $\text{dist}(v,u)$.

Next, over all edges $(u',v') \in E$, we find the cycle which has

$$\text{min} \left( l(u',v') + \text{dist}(v',u') \right)$$
(b) (4 points) Give pseudocode of your algorithm.

\[
\text{Min \_simple\_cycle} \ (G, s) \\
\text{for every } v \in V:
\text{dist}(v, u) = \text{dijkstra}(G, l, v)
\]

\[\min = \infty\]

\text{for every } (u', v') \in E:
\[\text{if } l(u', v') + \text{dist}(v', u') < \min:
\text{\quad min} = l(u', v') + \text{dist}(v', u')\]
\[\text{return } \min\]

(c) (3 points) Analyze the runtime of your algorithm.

Dijkstra's algorithm takes \(O((1V + 1E) \log 1V)\) time.

Running time on 1V vertices = \(1V \times O((1V + 1E) \log 1V)\)

Finding min = \(O(1E)\) time

Total running time = \(O((1V + 1E) 1V \log 1V) + O(1E)\)
\[= O\left((1V + 1E) 1V \log 1V\right)\]