Name: ________________________________ Student ID: ________________________________

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
Total:	40	

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 class[1]:

- `explore`(G, s)

  This returns three arrays of size $|V|$: `pre`, `post`, and `visited`.

- `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.

- `scc`(G)

  This returns an array `scc` of size $|V|$. If the graph has k strongly connected components, then the `scc` array assigns each node a number in the range 1 to k.

- `bfs`(G, s), `dijkstra`(G, ℓ, s), `bellman-ford`(G, ℓ, s)

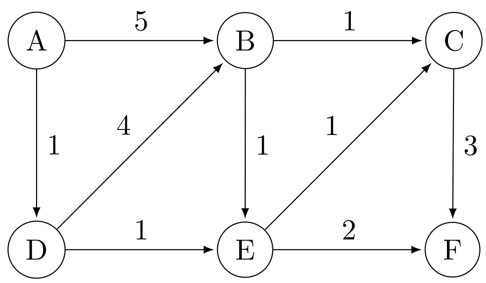
  These all return two arrays of size $|V|$: `dist` and `prev`.

- `dag-sp`(G, ℓ, s)

  This returns two arrays of size $|V|$: `dist` and `prev`. The array `dist` contains the shortest paths from s to all other reachable nodes in G. The algorithm is similar to `dag-lp` which instead returns the longest paths. These only work on directed acyclic graphs with and without negative edges.

[1] We recall from class/text the following time complexities. (1) `dfs/explore`: $O(|V| + |E|)$. (2) `scc`: $O(|V| + |E|)$. (3) `bfs`: $O(|V| + |E|)$. (4) `dijkstra`: $O((|V| + |E|) \log |V|)$ assuming a simple binary heap implementation of the priority queue.(5) `bellman-ford`: $O(|V| \cdot |E|)$ (6) `dag-sp`: $O(|V| + |E|)$.

1. (10 points) For the directed graph below with non-negative edges, list the order in which nodes are processed by each of the following algorithms. Start all algorithms from node A and ignore edge lengths if they are not commonly used by an algorithm (e.g., `dfs`). Break any ties alphabetically (alphabetically-lowest first).



`dfs` ______________________________

`bfs` ______________________________

`dijkstra` ______________________________

`dag-sp` ______________________________

2. **Short answer**. For true/false questions state whether the claim is true or false. If true, give a brief justification. If false, justify by providing a counterexample. *No points will be given for simply writing "true" or "false" without any justification!*

   (a) (2 ½ points) The running-time of a divide-and-conquer algorithm is characterized by the following recurrence: $T(n) = 4T(n/2) + \log n$. Provide a tight big-O bound for the running-time of this algorithm.

   (b) (2 ½ points) True/False: For any directed graph $G = (V, E)$ if all edge lengths are distinct (no two edge lengths are the same) then the shortest path between two nodes s and t is unique.

(c) (2 ½ points) True/False: Given two nodes s, t, the shortest (simple) cycle containing s and t must also contain a shortest s-t path.

(d) (2 ½ points) True/False: Given a directed graph $G = (V, E)$ and node $s \in G$, with all nodes in V reachable from s. We run the Bellman-Ford algorithm in G, starting from node s, and the stored `dist` values do not change from the $(|V|/2 - 1)^{st}$ iteration to the $(|V|/2)^{st}$ iteration. Then, G cannot have any negative cycles.

3. (10 points) You are given a nonempty array A with n distinct integer-valued elements. The values in the array increase monotonically from A_1 to an element A_i, and then decrease monotonically from A_i to A_n. The element A_i is called the *peak* element of A. In other words:

$$A_1 < A_2 < \ldots < A_i > A_{i+1} > A_{i+2} > \ldots > A_n$$

In the following example the *peak* element is $A_4 = 5$:

$$[-7, -1, 4, 5, 2, 0, -10, -23]$$

Design a divide-and-conquer algorithm to find the *peak* element A_i. Briefly explain your algorithm, give a recurrence characterizing its time complexity, apply the master theorem to provide a big-O running-time, and provide pseudo-code.

4. Given a connected, directed graph $G = (V, E)$, we say that $v \in V$ is a *root node* in G if, for all $u \in V, u \neq v$, there exists a directed v-u path in G.

   (a) (2 points) Give an example **with no more than four (4) nodes** of a connected, directed *acyclic* graph (DAG) with no root node.

   (b) (4 points) Suppose that you are given a connected, directed graph G and a known root node r. Give an efficient algorithm to find all other root nodes in G. (Possibly useful observation: if there is a directed path in G from a node u to r, then u is also a root node.) Briefly explain and justify your algorithm, analyze the time complexity, and provide pseudo-code.

(c) (4 points) Suppose that you are given a connected, directed graph G that may or may not have any root nodes. Explain (with pseudo-code if you wish) how to determine all of G's root nodes, or indicate that no root nodes exist, in time $O(|V| + |E|)$. Clearly state any algorithms from class that you use in your solution.

SCRATCH PAGE. DO NOT REMOVE.