Panther ID (No name, please):

COP 5407: Intro. to Algorithms FINAL EXAM; Spring 2019

2 pages; Max Score = 100 pts; Estimated Time = 2 hours;

Read This: This is a "open book" take-home exam. For algorithmic problems, provide a "Basic Idea" behind your solution first. Analyze the worst-case time complexity of all algorithms you are asked to design. Always attempt to argue why your algorithm is correct. Provide comments where needed. Make it easier for me to understand your pseudocode. Indentation is critical. Without indentation, I will not grade your solution. Whatever word processor you use, learn how to write subscripts, superscripts, and complicated mathematical formulae as needed. If you don't know how to do this, handwrite your solutions and send me a scanned copy.

1. [30]

   (a) [15] **Short Questions . . . 1**
   Dijkstra's single-source shortest path algorithm is a form of Dynamic Programming. We will explore this statement further in the following questions.

```
SIMPLIFIED-DIJKSTRA($G, w, s$)
for each vertex $v \in V[G]$ do
    $d[v] \leftarrow \infty$
end
$d[s] \leftarrow 0$
$Q \leftarrow V[G]$
while ($Q \neq \emptyset$) do
    $u \leftarrow$ EXTRACT-MIN($Q$)
    for (each $v \in Adj[u]$) do
        if ($d[v] > d[u] + w(u, v)$) then
            $d[v] \leftarrow d[u] + w(u, v)$
        end
    end
end
```

   i. [3] List the subproblems that are solved by this DP.
   ii. [1] What data structure stores the solutions to these subproblems?
   iii. [1] Is this a 1-dimensional or 2-dimensional DP?
   iv. [6] State clearly the recurrence that is used in this DP.
   v. [3] Explain how the priority queue is used to order the subproblems.
   vi. [1] Explain in plain English what $d[u]$ means at any given time during the execution of Dijkstra's algorithm.

(b) [15] **Short Questions . . . 2**

i. [8] Define the classes, $\mathcal{P}$, $\mathcal{NP}$, and $\mathcal{NP}$-Complete. State one open problem involving these complexity classes other than "Is $\mathcal{P} = \mathcal{NP}$?".

ii. [4] Pick any well-known $\mathcal{NP}$-Complete problem and write it down by providing the INSTANCE of the problem and the QUESTION that makes it $\mathcal{NP}$-Complete.

iii. [3] Any problem in $\mathcal{P}$ can be reduced in polynomial time to an $\mathcal{NP}$-Complete problem. True or False?

2. [45] **Profitable Road Trips** You are a salesman for N. Armada Corp., selling the empanada widgets. Your company has given you a highway map of the region in the form of a weighted graph with n cities and m highway stretches. Each highway stretch connects adjacent cities. A stretch connecting adjacent cities u and v satsifies the following conditions: (1) it can be traversed in both directions, (2) it has a non-negative real-valued length, denoted by $w(u, v)$, and (3) has a non-negative real-valued "toll and fuel" cost that is incurred when one uses that stretch, denoted by $t(u, v)$. When visiting a city u on the tour, you can earn a revenue of $r(u)$ by selling empanada widgets in that city. Your head office is in city s and you start your trip from s.

   Given the toll and fuel costs of every stretch of the highway map and the revenues possible from every city, design an efficient algorithm to choose one destination city x such that traveling from the head office to that destination city results in the **greatest net profit** (defined as total revenuw minus total cost). The only constraint is that whichever city you choose as your destination, you can only travel along a **shortest** path from s. If there is more than one shortest path to x, the algorithm picks the one with the higher net profit. You do not have to worry about the return trip since the company will fly you back from any destination at no cost. Analyze the time complexity of your algorithm. **Explain the basic idea behind your algorithm first. It is best to show how you would modify an existing algorithm.**

3. [25] Given an undirected, weighted graph $G(V, E)$ with n vertices and m edges, design an efficient **linear-time** ($O(m + n)$) algorithm to compute a graph $G'(V, E')$ on the same set of vertices, where $E' \subset E$ and E' contains the $\lceil m/2 \rceil$ edges with the smallest edge weights. For convenience you may assume that m is even. Note that the output needs to be a separate adjacency list for G' without destroying the adjacency list for G. Analyze the time complexity of your algorithm. **Explain the basic idea behind your algorithm first. If your algorithm is not a linear-time algorithm, it is necessary that you point it out.**