Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, and if applicable, efficiency, as well as correctness. Out of 80 points.

**Which version of Dijkstra do we use? 20 points each** We saw two implementations of Dijkstra’s algorithm in class, one using arrays and the other using a heap. For each of the following kinds of graph, explain which implementation you would use and give a time analysis for that version on the graph in question. When the graphs described as undirected, that means in the directed graph, there are edges in both directions. (I don’t specify the start node, because it should not change your answer.)

1. A “wheel and hub” graph: add to a directed cycle of length $n - 1$ a central node $v$ with edges from $v$ to all nodes in the cycle.
   The wheel and hub graph has $n$ nodes and $2(n - 1) = \Theta(n)$ edges, $n - 1$ around the cycle and $n - 1$ from the central node. So the heap implementation is $O((n + m) \log n) = O(n \log n)$ in this case, whereas the array implementation is $O(n^2)$, so the heap version is asymptotically faster.

2. A “barbell” graph: Start with two complete undirected graphs on two sets of nodes {$v_1, \ldots, v_{n/2}$} and {$u_1, \ldots, u_{n/2}$} (where there are edges between each $v_i$ and each $v_j$ with $j \neq i$ and similarly between $u_i$ and $u_j$) and add an undirected edge between $u_1$ and $v_1$.
   Each end of the barbell has $(n/2)(n/2 - 1) = \Theta(n^2)$ edges, so the $O(n^2)$ array implementation is better than the $O((n + m) \log n) = O(n^2 \log n)$ array implementation for this case.
Using algorithms to solve new problems—40 points Explain how we can modify or use one of the known graph search algorithms to solve the following problem in the given time: (10 points correct algorithm, 10 points correctness proof, 10 points efficiency, 10 points time analysis)

**Shortest non-trivial cycle containing a node** You are given a directed graph \( G \) with non-negative edge weights \( w[e] \) and a node \( s \in V \). You wish to find the minimum total length cycle containing \( s \) that is non-trivial, i.e., contains at least one node besides \( s \) itself. Show how to do this in \( O((n + m) \log n) \) time.

Solution 1:
A non-trivial cycle will be at some other node \( v \neq s \) immediately before returning to \( s \). So it will consist of some path from \( s \) to \( v \) followed by an edge from \( v \) to \( s \). The minimum length of such a cycle is the minimum path distance to \( v \), plus the cost of the edge from \( v \) to \( s \). So our algorithm is: First run Dijkstra’s algorithm to compute the shortest path distance to each \( v \), \( D[v] \). Then run through each node \( v \) and compute, if \( s \in N(v) \), \( D[v] + w((v, s)) \). Output the minimum such value.

The time for the second stage is \( O(m) \), since we check for each \( v \) whether \( s \) is a neighbor of \( v \). This is smaller than the time for Dijkstra’s algorithm, which is \( O((n + m) \log n) \) using the heap implementation. So the total time is \( O((n + m) \log n) \).

Solution 2: Create a new graph \( G' \) by taking \( G \) and replacing \( s \) with two nodes \( s_{\text{out}} \) and \( s_{\text{in}} \). \( s_{\text{out}} \) has all out-going edges from \( s \) and \( s_{\text{in}} \) has all in-coming edges. Otherwise, vertices, edges and weights are identical to \( G \).

Each path from \( s_{\text{out}} \) to \( s_{\text{in}} \) corresponds to a non-trivial cycle containing \( s \) in \( G \), since it starts and ends at \( s \) and must go along some edge out of \( s \) as its first step. Conversely, each non-trivial cycle is a path from \( s_{\text{out}} \) to \( s_{\text{in}} \). Therefore, the minimum length of such a cycle is the shortest path distance from \( s_{\text{in}} \) to \( s_{\text{out}} \) in \( G' \), which we compute using Dijkstra’s algorithm.

\( G' \) has one more node and the same number of edges as \( G \), so the time to compute \( G' \) is linear, and the time to run Dijkstra’s algorithm is still \( O((n + m) \log n) \). Therefore the total time is \( O((n + m) \log n) \).