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 are undirected, that means in the directed graph, there are edges in both directions.

1. The log \( n \) dimensional hypercube, where we think of nodes as all bit strings of length \( k = \log n \), and two nodes are adjacent if they differ in exactly one bit.
   
   **Answer:** Every node \( b_1...b_k \) has \( k = \log n \) neighbors, with each neighbor being of the form \( b_1...b_i-1\overline{b_i}b_i+1...b_k \) for \( 1 \leq i \leq k \). Thus, the number of edges \( m \) is \( n \log n \). Using a heap would take time \( O((n+m) \log n) = O(n(\log n)^2) \) for this graph, whereas using an array takes time \( O(n^2) \) regardless of the number of edges. Since \( n(\log n)^2 \in o(n^2) \), it is better to use the heap implementation.

2. A randomly chosen graph on \( n \) nodes, where each edge is present with probability \( 1/2 \)
   
   **Answer:** Since of \( \Omega(n^2) \) possible edges, each is present with constant probability, we expect about half the possible edges to be present, i.e, \( m = \Theta(n^2) \). Then the array implementation would take time \( O(n^2) \), whereas the heap implementation would take time \( O((n+m) \log n) = O(n^2 \log n) \) in this case. So the array implementation would be better.

**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:

**k Closest points** You are given a directed graph \( G \) with non-negative edge weights, and a node \( s \). You wish to find the \( k \) nodes that are closest in distance to \( s \) (breaking ties arbitrarily). Show how to do this in \( O(nk) \) time.

**Answer:** As shown in the textbook, Dijkstra’s algorithm enumerates points in order of the shortest path distance to the start node. Thus, we run Dijkstra’s algorithm but stop after \( k \) nodes have been put into \( R \), and return the set \( R \). Using the array data structure, each iteration takes time \( O(n) \), and adds one node to \( R \), so the total time will be \( O(nk) \), as required. (If we used the heap implementation, each iteration takes time \( O(\log n + \text{degree of the node added to } R) \) In the worst case, even for a sparse graph, we could have a few nodes with degree \( n \), and these could be the ones closest to \( s \). So the time could be up to \( O(n \log nk) \), which is slightly more than the problem asks for.)