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.

   Each node is a $k = \log n$ bit string, so there are $k = \log n$ neighbors (flipping each bit of the string) for each of $n$ nodes. So every node has degree $\log n$, meaning that $m = \Theta(n \log n)$ (whether the constant is 1 or 1/2 depends on whether we view the graph as directed or undirected). If we use the array implementation, the time is $O(n^2)$, whereas the heap implementation is $O((n + m) \log n) = O(n \log^2 n)$. Any polynomial in $\log n$ is smaller asymptotically than a factor of $n$, so the heap implementation is preferable.

2. A randomly chosen graph on $n$ nodes, where each edge is present with probability $1/2$

   Since each of $n(n - 1)/2$ edge positions is chosen with probability $1/2$, we expect around $m = n(n - 1)/4$ edges total. So with high probability, $m = \Theta(n^2)$. In this case the array implementation takes time $O(n^2)$, whereas the heap implementation takes time $O((n + m) \log n) = O(n^2 \log n)$. So the array implementation saves a factor of $\log n$ over the heap implementation, so is preferable.

**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)

Graph with node weights You are given a directed graph $G$ with non-negative weights $w[u]$ for each node $u \in V$. You are also given nodes $u$ and $v$, and wish to find the smallest total weights of nodes along any path that starts at $u$ and ends at node $v$. Show how to do this in $O((n + m) \log n)$ time.

Construct a graph $G' = (V', E')$ where $V' = \{u_{in}, u_{out} | u \in V\}$, and edges are as follows: There is an edge from each $u_{in}$ to $u_{out}$ of weight $w[u]$ for each $u \in V'$, and for every edge $(u, v) \in E$, there is an edge $(u_{out}, v_{in})$ in $E'$ of weight 0. We claim there is a path from $u$ to $v$ of total node weight $W$ in $G$ if and only if there is a path from $u_{in}$ to $v_{out}$ in $G'$ of total edge weight $W$ in $G'$. If $u, a^1, a^2, \ldots a^k, v$ is a path in $G$, then $u_{in}, u_{out}, a^1_{in}, a^1_{out}, \ldots a^k_{in}, a^k_{out}, v_{in}, v_{out}$ is a path in $G'$, and the weight of the edges from each $a_{in}^i$ to $a_{out}^i$ are $w[a^i]$, and other edges have weight 0. So the total weights of edges in $G'$ equals the total weight of nodes in $G$. Conversely, each path in $G'$ from $u_{in}$ to $v_{out}$ must have the form given above, $u_{in}, u_{out}, a^1_{in}, a^1_{out}, \ldots a^k_{in}, a^k_{out}, v_{in}, v_{out}$, because each node $a_{in}^i$ only has one edge leaving it, to $a_{out}^i$. Just as above, the total edge weights in $G'$ along this path is the sum of the node weights along the path $u, a^1, \ldots a^k, v$ in $G$.

So we can run Dijkstra’s algorithm (using heaps) on $G'$, with start node $u_{in}$, and return the minimum distance $D[v_{out}]$.

$|V'| = 2n$ and $|E'| = n + m$, so the total time for Dijkstra’s algorithm is $O((|V'| + |E'|) \log n) = O((2n + n + m) \log n)) = O((n + m) \log n)$. The time to construct $G'$ is $O(n + m)$ which is smaller, so does not affect the overall asymptotic time.