Problem 1

Initially: \( \text{dist}(s) = 0, \text{dist}(v) = \infty \) for all \( v \neq s \), \( \text{pred}(v) = \text{NULL} \) for all \( v \).

Phase 1: Queue at the start of Phase 1: \( (s) \); Edges that are relaxed (in this order): \( (s, a), (s, b) \); New \( \text{dist}(\cdot) \) and \( \text{pred}(\cdot) \) values: \( \text{dist}(a) = 4, \text{dist}(b) = 5, \text{pred}(a) = s, \text{pred}(b) = s \).

Phase 2: Queue at the start of Phase 2: \( (a, b) \); Edges that are relaxed (in this order): \( (a, c), (b, a), (b, d) \); New \( \text{dist}(\cdot) \) and \( \text{pred}(\cdot) \) values: \( \text{dist}(c) = 6, \text{dist}(a) = 2, \text{dist}(d) = 4, \text{pred}(c) = a, \text{pred}(a) = b, \text{pred}(d) = b \).

Phase 3: Queue at the start of Phase 3: \( (c, a, d) \); Edges that are relaxed (in this order): \( (c, e), (a, c) \); New \( \text{dist}(\cdot) \) and \( \text{pred}(\cdot) \) values: \( \text{dist}(e) = 5, \text{dist}(c) = 4, \text{pred}(e) = c, \text{pred}(c) = a \).

Phase 4: Queue at the start of Phase 4: \( (e, c) \); Edges that are relaxed (in this order): \( (e, c) \); New \( \text{dist}(\cdot) \) and \( \text{pred}(\cdot) \) values: \( \text{dist}(e) = 3, \text{pred}(e) = c \).

Phase 5: Queue at the start of Phase 5: \( (e) \); No edges are relaxed and so no \( \text{dist}(\cdot) \) values or \( \text{pred}(\cdot) \) values are updated.

Problem 2

Instead of using a min-heap priority queue implementation of the “bag” data structure, we implement the “bag” as an array \( A[1, \ldots, (n-1)W] \) such that for any \( j, 1 \leq j \leq (n-1)W \), \( A[j] \) contains the set of all vertices in the bag with \( \text{dist}(\cdot) \) equal to \( j \). We also maintain an index called \( \text{current} \), that is initialized to 1. This index always points to the slot in \( A \) that we will look at next to find a vertex with smallest \( \text{dist}(\cdot) \) value in the “bag.”

We now need to describe two operations on this array:

- **Finding and removing a vertex with smallest \( \text{dist}(\cdot) \) value from the bag.** We scan \( A \) starting at index \( \text{current} \) until we reach a slot in \( A \) that is non-empty. We pick an arbitrary vertex from the set stored at this slot and remove it from the set. The vertex chosen in this manner has the smallest \( \text{dist}(\cdot) \) value among all vertices in the bag. Since our scan of \( A \) always moves to the right, the total amount of time we spending in pulling out all vertices from the bag is \( O(n \cdot W) \).

- **Relaxing edges.** When a vertex \( u \) is removed from the bag, we process all edges \( (u, v) \) outgoing from \( u \) and relax these if necessary. For each edge, \( (u, v) \) that is relaxed, \( \text{dist}(v) \) falls and so \( v \) has to be removed from its old slot in \( A \) and moved to a new slot. All this can be done in \( O(1) \) time because we know the old (larger) \( \text{dist}(\cdot) \) value of \( v \) and also the new (smaller) \( \text{dist}(\cdot) \) value and we can uses these \( \text{dist}(\cdot) \) values as indices in \( A \). Thus the total amount of time we spend relaxing edges outgoing from \( u \) is \( O(\text{degree}(u)) \). When this is summed over all vertices \( u \), we get a running time of \( O(m) \).

Thus the total running time of the algorithm is \( O(nW + m) \).
Problem 3
Let $G = (V, E)$ be the given, connected, edge-weighted graph. Let $w(e)$ denote the weight of an edge $e \in E$. Create a new edge-weighted graph $G'$ by replacing each edge weight $w(e)$ by $-w(e)$ (i.e., the negation of $w(e)$). Otherwise, $G$ and $G'$ are identical. Now compute an MST on $G'$ using your favorite MST algorithm. The claim is that the minimum weight spanning tree $T$ of $G'$ is a maximum weight spanning tree of $G$. This follows from the fact that if $T$ has total weight $W$ in $G'$, then it has weight $-W$ in $G$. Therefore, if there were a heavier spanning tree in $G$, then there would have been a lighter spanning tree in $G'$ that the MST algorithm did not find – a contradiction. Using any of the standard MST algorithms, we compute a maximum spanning tree in $O(m \log n)$ time.

Problem 4
Instead of a min-heap priority queue, we maintain an array $A[1, \ldots, n]$ to implement the “bag” data structure. In each slot $A[j]$ we maintain the $\text{dist}(\cdot)$ value of vertex $j$. Thus, the $n$ vertices of the graph serve as indices into this array. Then finding and removing a vertex with smallest $\text{dist}(\cdot)$ value from the bag simply requires a scan of the entire array. This takes $O(n)$ time per vertex that is removed and therefore takes $O(n^2)$ total time. When a vertex $u$ is removed from the bag, we process all edges $(u, v)$ outgoing from $u$ and relax these if necessary. For each edge, $(u, v)$ that is relaxed, $\text{dist}(v)$ falls and needs to be updated in $A$. Using $v$ as an index into $A$ allows us to do this in $O(1)$ time. Thus the total amount of time we spend relaxing edges outgoing from $u$ is $O(\text{degree}(u))$. When this is summed over all vertices $u$, we get a running time of $O(m) = O(n^2)$. Therefore, the total running time is $O(n^2)$. 