1. To compute the array $D^k$ we only need the values in $D^{k-1}$. After fully computing $D^k$ we can free up the space used for $D^{k-1}$. In this way we only use $O(n^2)$ space. This is similar to how we turn the $O(n)$ space dynamic programming solution for computing Fibonacci numbers into an $O(1)$ space algorithm.

2. Answers will depend on how you drew your graph. See Figure 7.15 in the textbook for an example.

3. Suppose that a graph $G$ has two distinct minimum spanning trees, call them $T$ and $S$. Let $e$ be an edge in $T$ that is not in $S$. If we add $e$ to $S$, then it forms a cycle $C$. Since $T$ is a tree (and therefore acyclic) there must be an edge $f \in C \setminus \{e\}$ which is not in $T$. If the weight of $f$ is less than that of $e$, then we can remove $e$ from $T$ and replace it with $f$ to get a spanning tree of lower weight, a contradiction. On the other hand, if the weight $f$ is greater than the weight of $e$, then we can remove $f$ from $S$ and replace it with $S$ to get a spanning tree of lower weight, a contradiction. Thus, we must what that the weights of $f$ and $e$ are the same. We have show that if a graph has two distinct minimum spanning tree, then its weights are not unique. The result in the problem now follows by taking the contrapositive.

4. Negate all the edge weights and run Algorithm 7.9 from the book. Note: This trick does not work in general. I am using the fact that Algorithm 7.9 works for graphs with negative weights.

5. Use your favorite algorithm to compute a maximum spanning tree for $G$. A maximum bandwidth path between any two vertices is the path in a maximum spanning tree. The problem does not ask for a runtime, but this method’s runtime is dominated by finding the maximum spanning tree.