Problem 1. Write pseudocode for Prim’s algorithm. In your own words, explain what data structures are used to implement the algorithm, and how they affect its running time complexity.

Problem 2. Consider the Minimum Spanning Tree Problem on an undirected graph $G = (V, E)$, with a cost $c_e \geq 0$ on each edge, where the costs may not all be different. If the costs are not all distinct, there can in general be many distinct minimum-cost solutions.

Suppose we are given a spanning tree $T \subset E$ with the guarantee that for every $e \in T$, $e$ belongs to some minimum spanning tree of $G$. Can we conclude that $T$ itself must be a minimum spanning tree of $G$? If yes, proof the statement. If no, give a counterexample with explanation.

Problem 3. Let us say that a graph $G = (V, E)$ is a near-tree if it is connected and has at most $n + 8$ edges, where $n = |V|$. You may assume that all the edge costs are distinct.

(a) Give an algorithm with running time $O(n)$ that takes a near-tree $G$ with costs on its edges, and returns a minimum spanning tree of $G$.

(b) Explain why the running time of your algorithm is $O(n)$.

Hint: Assume the following to be true. Let $G$ be a graph with distinct edge costs for each edge. Then, for any cycle $C$ in $G$, the highest cost edge in $C$ will not be part of any minimum spanning tree of $G$.

Problem 4. Insertion sort can be expressed as a recursive procedure as follows. In order to sort $A[1, \ldots, n]$, we recursively sort $A[1, \ldots, n - 1]$ and then insert $A[n]$ into the sorted array $A[1, \ldots, n - 1]$.

(a) Write pseudocode for this recursive version of Insertion sort.

(b) Give a recurrence for the running time of the algorithm, and use that to establish an upper bound for it.

Problem 5. Suppose you are choosing between the following three algorithms that solve the same problem:

- Algorithm $A$ solves the problem by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm $B$ solves the problem of size $n$ by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.

- Algorithm $C$ solves problems of size $n$ by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms, and which would you choose? Justify your answer.