Assignment 3

Grades: each assignment is 5% in the final score and there is a 2% bonus in each assignment for bonus problems. We use the following scale for the simplicity of grading: the total score is 50 (additional 20 for bonus problems) for each assignment.

Problem 1 [Full Score: 10]. Show the contents of an array $A$ implementing a binary min-heap. Repeatedly insert the following values into an initially empty heap: 7, 6, 5, 4, 3, 2, 1. Use the up-heap bubbling insertion of the items one-by-one. (Show the content after each insertion.)

Problem 2 [Full Score: 10]. Show the contents of the array $A$ as a binary min-heap, initially with entries [7, 6, 5, 4, 3, 2, 1], as it changes during the linear time bottom-up heap construction. (Show the content after each down-heap bubbling.)

Problem 3 [Full Score: 10]. C-2.32 (on page 135). (You need to describe your algorithm and show why its complexity is $O(k)$.)

Problem 4 [Full Score: 10]. Given $k$ sorted queues (i.e., the front of each queue stores the minimum element in each queue) containing a total of $n$ elements, describe how to merge them into a single sorted queue in time $O(n \log k)$. You need to describe your algorithm and prove its complexity. (Hint: use a heap.)

Problem 5 [Full Score: 10]. Suppose we changed the height-balance property, so that the heights of two subtrees may differ by at most 2 (instead of 1). Prove that this modified AVL tree must still have a height that is $O(\log n)$. Please make your proof clear and formal. HINT: I recommend using the proof of Theorem 3.2 (page 153) as a starting point.

Problem 6 [Bonus Problem: 20]. Assume there is an implementation of binary tree ADT such that one can access its root, the key stored in each node, and the left and the right child of each node with $O(1)$ complexity. Let $T$, given as input, be such an object with $n$ nodes.

(1) [10] Describe an algorithm that checks whether $T$ is a valid binary search tree. Analyze the worst-case complexity of your algorithm.

(2) [10] Assume $T$ is a binary search tree and let $k$ be another input. Describe an algorithm that finds one of the closest-to-$k$ keys in the binary tree $T$. Analyze the worst-case complexity of your algorithm. (Assume all the keys are integers and the distance between two keys is the absolute value of their difference.)