**Fundamental Algorithms** **Instructor: Amir Shpilka**
**Courant, Fall 2017** **Due Date: Wed Oct 18, 11:55pm**

# Homework 6

1. (a) (6 points) Give an algorithm that constructs a balanced binary search tree from an array of $n$ elements in time $O(n \log n)$. Briefly justify running time and correctness.

   (b) (6 points) Design an algorithm that takes as input an INORDER-TREE-WALK and POSTORDER-TREE-WALK of a binary tree $T$ on $n$ nodes (both as $n$-elements arrays) and outputs the PREORDER-TREE-WALK of $T$ (again, as $n$-element array). Notice, $T$ is not necessarily a binary *search* tree.

   Briefly justify correctness and running time of your algorithm.

   **Note**: you may assume the tree has distinct elements.

   (c) (3 points) Now assume that the tree $T$ is a binary *search* tree. Modify your algorithm in part (a) so that it works given only the POSTORDER-TREE-WALK of $T$.

2. (5 points) Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. For each of the following sequences, say whether or not it could be the sequence of nodes examined, and justify.

   1. 2,252,401,398,330,344,397,363.
   2. 924, 220, 911, 244, 898, 258, 362, 363.
   3. 925, 202, 911, 240, 912, 245, 363.
   4. 2,399,387,219,266,382,381,278,363.
   5. 935, 278, 347, 621, 299, 392, 358, 363.

3. (5 points) Professor Donald thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key $k$ in a binary search tree ends up in a leaf. Consider three sets: $A$, the keys to the left of the search path; $B$, the keys on the search path; and $C$, the keys to the right of the search path. Professor Donald claims that any three keys $a \in A, b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professors claim.

4. (5 points) We can sort a given set of $n$ numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

5. (5 points) Is the operation of deletion "commutative" in the sense that deleting $x$ and then $y$ from a binary search tree leaves the same tree as deleting $y$ and then $x$? Argue why it is or give a counterexample.