Problem 7-1 (Constructing Tree from Treewalks) 7 points

(a) (5 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.

(b) (2 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 $T$.

Problem 7-2 (Computing Less) 18 Points

Assume you are given a binary search tree $T$ on $n$ elements, whose height is $h$. As usual, let $v.key$ denote the key value of node $v$, and recall than the left sub-tree of every node $v$ only contains elements less than or equal to $v.key$.

(a) (5 points) Using a slight modification of the POSTORDER-TREE-WALK procedure, argue that in time $\Theta(n)$ you can compute, for every node $v$, the number of nodes (call it $v.less$) in $v$’s sub-tree which are strictly less than $v.key$. Write a pseudocode for the resulting recursive procedure FILL-LESS($T,v$), which will fill in the values $w.less$ for all the nodes $w$ of $T$’s subtree rooted at $v$ in time $O(n)$.

(Hint: Remember that some elements in the left sub-tree of $v$ might be equal to $v.key$, and should not be included in the count for $v.less$. Also do not forget to cover the base case.)

(b) (5 points) Now that each node $v$ contains the value $v.less$, show that you keep maintaining this value for each successive INSERT operation. Namely, write the pseudocode for the INSERT($T,$ value) procedure, which will insert a new (leaf) node $w$ into $T$ with $w.key = value$. The running time of your procedure should be $O(h)$, and it should correctly maintain all the $v.less$ values.

(Hint: When going down the tree from a node $v$, when does the insertion of $w$ increases the value $v.less$?)

(c) (5 points) Assume now that we successfully maintain the $v.less$ field for both the INSERT and the DELETE Operation, and also that all the key values are distinct. Show how to implement an $O(h)$-time procedure MYRANK($v$), which will return the rank of a node $v$ in the BST (i.e., if $v.key$ is the minimum of all the values it will return 1, or if it is the median it will return $n/2$). Why can’t we simply return $v.less + 1$ in time $O(1)$?
(d) (3 points) Instead of running the procedure MyRank as in part (c), is it possible to simply maintain — in time $O(h)$ — the field $v.rank$ which will correctly compute the rank of $v$ after each insertion and deletion? If yes, show how, if not, explain why not.

Problem 7-3 (Find the Josephus Permutation) 10 points

According to Josephus’ account of the siege of Yodfat, he and his $n$ comrade soldiers were trapped in a cave, the exit of which was blocked by Romans. They chose suicide over capture and decided that they would form a circle and start killing themselves using a “step” of size $m$; i.e., starting the count with some fixed person (called “number 1”), every $m$-th person is killed, after which the suicides continue with the remaining people on the (now smaller) circle. For example for $n = 8$ and $m = 3$, the order in which the people are killed is $(3, 6, 1, 5, 2, 8, 4, 7)$. We want an $O(n \log n)$ algorithm to find the order in which the soldiers were killed. (Hint: Use augmented 2-3 trees.)

Problem 7-4 (Non-Commutation of 2-3 Trees) 10 points

Consider the following 2-3 tree $T$:

```
    9
   / \  / \
 5   9  1  5  7  8  9
```

(a) (5 points) Show an element $x \notin T$ such that applying in sequence $\text{INSERT}(x)$ and $\text{DELETE}(x)$ will result with a tree $T'$ that is different from $T$.

(b) (5 points) Show an element $x \in T$ such that applying in sequence $\text{DELETE}(x)$ and $\text{INSERT}(x)$ will result with a tree $T'$ that is different from $T$.