Warm-up

Problem 1. Give the full pseudocode for doing a trinode restructure at x, y, z where x is the left child of y and y is the left child of z.

Problem 2. Consider the implementation of a binary search tree on n keys where the keys are stored in the internal nodes. Prove using induction that the height of the tree is at least $\log_2 n$.

Problem 3. Consider the implementation of a binary search tree on n keys where the keys are stored in the internal nodes. Prove using induction that the number of external nodes is $n + 1$.

Problem solving

Problem 4. Consider the following algorithm for testing if a given binary tree has the binary search tree property.

```
1: function TEST-BST(T)
2:     for u ∈ T do
3:         if u.left ≠ nil and u.key < u.left.key then
4:             return False
5:         if u.right ≠ nil and u.key > u.right.key then
6:             return False
7:     return True
```

Either prove the correctness of this algorithm or provide a counter example where it fails to return the correct answer.

Problem 5. Consider the following operation on a binary search tree: LARGEST() that returns the largest key in the tree.

Give an implementation that runs in $O(h)$ time, where h is the height of the tree.

Problem 6. Consider the following operation on a binary search tree: SECOND-LARGEST() that returns the second largest key in the tree.

Give an implementation that runs in $O(h)$ time, where h is the height of the tree.

Problem 7. Consider the following operation on a binary search tree: MEDIAN() that returns the median element of the tree (the element at position $\lfloor n/2 \rfloor$ when considering all elements in sorted order).

Give an implementation that runs in $O(h)$, where h is the height of the tree. You are allowed to augment the insertion and delete routines to keep additional data at each node.

Problem 8. Consider the following binary search tree operation: REMOVE_ALL(k_1, k_2) that removes from the tree any key $k \in [k_1, k_2]$.

Give an implementation of this operation that runs in $O(h + s)$ where h is the height of the tree and s is the number of keys we are to remove.