Homework 3 Solutions

Problem 1 Analyze the Preorder Traversal. You have a list of n real numbers, and you want to form a binary search tree with them. What is the tree formation complexity? Would your answer change, if I tell you that when your tree has k nodes then its depth varies from that of a complete binary tree of k nodes by at most a constant?

Preorder traversal has complexity $\theta(n)$. Let $T(n)$ be he complexity for a n-node tree rooted at a particular node. $T(n) = 1 + \sum_{i=1}^{\text{children of the node}} (C_i)$. Note that C_i s are mutually disjoint. Complexity with a leaf node is 1. Doing this recursively, we see $T(n) = cn$. Thus the complexity is $\theta(n)$.

The tree formation complexity depends on the original order of the data. If the original list is completely sorted (or sorted in reverse order), the resulting tree appears as a sequence of only right (or left) links. This is the worst case scenario which will provide an upper bound.

To create a new node requires constant time, so the creation of n nodes requires cn time.
We will focus on the number of comparisons that is needed during the formation. The first node requires no comparisons, the second requires one comparison, the third node requires two comparisons, and so on. Thus the total number of comparisons is

$$1 + 2 + ... + n - 1 = \frac{n(n-1)}{2}$$
$$\Rightarrow 1 + 2 + ... + n - 1 = O(n^2)$$

So, the formation complexity is $O(n^2)$.

Let's take the second case. Consider for simplicity a complete binary tree with n nodes. It is known that the $n = 2^{d+1} - 1$, where d is the depth of the tree. From this relation we conclude that

$$d = \log(n+1) - 1$$

The number of nodes at any level l is 2^l and the number of comparisons necessary to place a node at level l is l. Thus the total number of comparisons is

$$\sum_{l=1}^{d} 2^l l$$

Using the formula of geometric series it can be shown that the resulting sum is $O(n \log n)$.
If we add the overhead that when the tree has k nodes then its depth varies from that of a complete binary tree of k nodes by at most a constant c, it will not change the $O(n \log n)$ complexity.

Problem 2 Sort a list of real numbers using a binary search tree. Analyze the complexity of your algorithm. Your grade depends on the complexity of your algorithm.

The first step is to build the tree. As it was proved in Problem 1, the worst case requires $O(n^2)$. The second step is to incorporate *inorder traversal* of the tree. This recursive algorithm allows to print the keys in sorted order. It takes $\Theta(n)$ time to do the traversal.
Note that it holds that the basic operations Find, FindMin, FindMax, Insert and Delete can be made to run in $O(h)$ time on a binary search tree of height h. So, the complexity of the sorting depends

on efficient building of the tree. If we build a binary search tree with a *balance* condition like AVL tree then the above operations take $O(\log n)$ time in the worst case. Thus, the tree formation will cost $O(n \log n)$.
In that case, sorting also will take $O(n \log n)$.

Problem 3 You have a binary search tree. Consider a leaf l. B is the set of keys in the path of l including l and the root. A is the set of keys to the left of this path. C is the set of keys to the right of the path. Is the following statement true or false? Given any element a in A, b in B, c in C, $a \leq b \leq c$. Justify your answer.

The statement is false. We will provide a small counter-example.
Consider a tree with root 6 and children node 2 and node 7.Node 2 respectively has children nodes 1 and 4. Node 4 has children nodes 3 and 5.
Now,consider the leaf 3. Set B includes nodes $6, 2, 4, 3$. Set A includes node 1 and set C includes nodes $5, 8$. You can see that 5 belongs to C but is smaller than 6 which is member of set B.

Problem 4 Node A is a leaf and node B is its parent in a binary search tree. Show that either B is the smallest element larger than A in the tree, or the largest element smaller than A.

The proof is based in contradiction. We consider 2 cases:
First case: let $B < A$ and K be the predecessor of A different than B. K cannot exist to the left subtree of B since that would imply it is smaller than B. If K has in its left subtree B, then K is greater than A, so it cannot be predecessor of A. So it remains that K has in its right subtree B. But then K is smaller than B, so again it cannot be a predecessor of A which is a contradiction.
So, B must be the predecessor of A.
Similar arguments exist in case that $B > A$.

Problem 5: A strict binary tree is one where every node has 0 or 2 children. Prove that if there are n leaf nodes in a strict binary tree then the total number of nodes is $2n - 1$.

The proof is based on induction.
Base case (n=1):
When there is just one leaf node, at the same time it is the only one node in the tree, which verifies $2n - 1$.
Induction Hypothesis: let the statement hold for n
We will prove that the statement holds for $n + 1$: Consider the tree T with $n + 1$ leafs.
There is one node C that has 2 leafs A, B for children. If we omit A, B from T then the new tree T' has n leafs since C is a leaf in T'. By induction hypothesis T' has $2n - 1$ nodes. But tree T includes tree T' and nodes A, B. So the total number of nodes in T is $2n - 1 + 2 = 2n + 1$ which proves the statement.