Problem 1:
Write recursive versions of tree-min and tree-max.

Solution. The pseudo-code is given:

\begin{verbatim}
Algorithm 3 TREE-MINIMUM(x)
    if x.left ≠ NIL then
        return TREE-MINIMUM(x.left)
    else
        return x
    end if
\end{verbatim}

\begin{verbatim}
Algorithm 4 TREE-MAXIMUM(x)
    if x.right ≠ NIL then
        return TREE-MAXIMUM(x.right)
    else
        return x
    end if
\end{verbatim}

Problem 2
Number of elements smaller than root using preorder traversal of a BST Given a preorder traversal of a BST. The task is to find the number of elements less than root.

Examples:
Input: preorder[] = 3, 2, 1, 0, 5, 4, 6. Output: 3
Input: preorder[] = 5, 4, 3, 2, 1. Output: 4

Solution. Here is the straightforward approach:
1. Traverse the given preorder.
2. Check if the current element is greater than root.
3. If yes then return \textit{indexOfCurrentElement} – 1. The number of elements smaller than root will be all the elements that occur before the first element of the right subtree minus the root.

Problem 3
Let T be a binary search tree whose keys are distinct, let x be a leaf node, and let y be its parent. Show that y.key is either the smallest key in T larger than x.key or the largest key in T smaller than x.key.

Solution. If \( x = y.left \) then calling successor on \( x \) will result in no iterations of the while loop, and so will return \( y \). Similarly, if \( x = y.right \), the while loop for calling predecessor will be run no times, and so \( y \) will be returned. Then, it is just a matter of recognizing what the problem asks to show is exactly that \( y \) is either predecessor(x) or successor(x).
Problem 4
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.

Solution. Deletion is not commutative. In the following tree, deleting 1 then 2 yields a different from the one obtained by deleting 2 then 1.

![Tree Diagram]

Problem 5
We can sort a given set of $n$ numbers by first building a binary search tree containing these numbers (using 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?

Solution. The in order walk takes $O(n)$ time no matter what, so our only concern is how long it takes to build the tree. The worst case is that the tree formed has height $n$ because we were inserting them in already sorted order. This will result in a runtime of $\Theta(n^2)$ (think insertion sort). In the best case, the tree formed is approximately balanced. This will mean that the height doesn’t exceed $O(lg(n))$. Note that it can’t have a smaller height, because a complete binary tree of height $h$ only has $\Theta(2^h)$ elements. This will result in a runtime of $O(nlg(n))$. The high level argument for this is that the bottom level of the tree would have $\approx \frac{n}{2}$ nodes, and it takes $O(lg(n))$ to insert each of these nodes. We can also prove this claim by the fact that sorting has a lower bound of $\Omega(nlg(n))$, so we cannot use this algorithm to sort faster than that lower bound.