Traversing the tree in insert element is O(log N) runtime because the tree is cut in half every time recursive insert calls itself. For example, a perfect tree of height 3 has seven elements. In any insertion case, 3 nodes must be traveled to until the insertion can occur, assuming the value is not already in the tree. The insert element method calls the balance method when it is returning, which in turn may call the rotate and double rotate functions. The balance function just contains conditionals and function calls to rotate and double rotate. Double rotate just calls the rotate function twice and has one conditional. Rotate itself, is where the actual rotations are done, this is done with the use of many conditionals. The rotate function operates in constant time as the size of the tree or subtree it is rotating does not affect the performance. Because rotate operates in constant time, double rotate, and balance runs in constant time. The only part of insertion that is not constant time is the recursive part of traversing the tree, so the total runtime for insert element is O(log N).

The worst-case for the remove element method is removing the root node of a tree or subtree. This is because removing the node of a tree calls the replacement value method which returns the value of the smallest leaf node on the right side of the tree, which could be the deepest node. It is important to note that at worst this part is O(log N) runtime. For example, if the root of a tree or subtree is removed the worst case is that it goes to the farthest leaf node to get the replacement value. This would require traversing the longest path of that tree or subtree, which is O(log N) as the tree is cut in half everytime recursive remove is called. The remove element method calls the balance method when it is returning, which in turn may call the rotate and double rotate functions. The balance function just contains conditionals and function calls to rotate and double rotate. Double rotate just calls the rotate function twice and has one conditional. Rotate itself, is where the actual rotations are done, this is done with the use of many conditionals. The rotate function operates in constant time as the size of the tree or subtree it is rotating does not affect the performance. Because rotate operates in constant time, double rotate, and balance runs in constant time. Therefore, remove element has O(log N) runtime as the only part which is not constant time in the worst case is recursive remove.

The to list method has a runtime of O(n). To list has to visit every node in order to put its value in the list. If the tree contains n nodes to list will visit n nodes, if the tree has n+1 nodes to list will visit n+1 nodes. Therefore the amount of steps is purely dependent on the size of the tree and the number of steps increases proportionally to the increase in the size of the tree.