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).

This process is done n times, once for every value in the list that is being sorted. So the runtime is O(n log n)