Insert_element

The worst case for insert_element is simply log(n). The reason that the answer is log n is caused by the way my recursive helper function for inserting elements work. The functions checks whether the value we're looking for is greater than or less then the current node, then recurs to insert the element onto whatever node the functions calls next. On the way out of the recursive loop, the insert_element updates the heights of the affected nodes and then balances the node, both which are done in O(1) time. Since the insert helper function is called log(n) times at worst case and one run of insert function has linear time complexity, the Big O is 1 * log(n) = log(n).

Remove_element

The worst case performance of remove_element is also log n. The reason that is is log n, is that to find the node, we recur log(n) times at most to find the node to remove. After going through the code whether to go left or right, and calling the recursive function, all we have to do after removing the element is to fix the heights and then check if the nodes are balances after the recursive calls are done. Since everything else is O(1), we know that calling an O(1) function a maximum of O(logn) times gives us an O(logn) worst case performance for remove element.

To_list

My way of creating to_list was to have a recursive function which called to_list of the left node + a list with the value of the current node + to_list of the right node. Since adding lists together is done in linear time, and to_list gets called n times (n is number of nodes), this means the big O is n*n, which equals O(n^2).