Searching:
In a binary tree, the best-case for searching is, O(1). The reason being, there could be only one node (in this case N = 1, hence O(1) == O(N)), or the node could be located at the root node, thus making the time complexity O(1).
In a binary tree, the worst-case is, O(N). The reason being, the node could be located at the location "N", with N being the number of nodes in the binary tree. So, if the node is located at position "N", it will take "N" traversals across the binary tree, to get to the node.

For any kind of self-balancing binary tree / AVL Tree, the best-case time complexity will be O(1), because we will only have one node or the node that we are looking for is the root node.
The worst-case time complexity will be O(log(N)), because we will have to traverse the various sub-trees of the self-balancing binary tree, and there will mechanisms in place to order the elements, in such a way that we don't have to iterate through all N nodes, and instead we can do it in O(log(N)).

Insertion:
In a binary tree, the best-case for insertion is, O(1). That case for that is, if we need to insert the root node(in this case N = 1, hence O(1) == O(N)).
In a binary tree, the worst-case is, O(N), because we will have to traverse all the elements of the binary tree, to reach the final node, where we want to insert our new node.

For any kind of self-balancing binary tree / AVL Tree, the best-case time complexity will be O(1), if we want to create our initial binary tree with the root node. The worst-case time complexity for a self-balancing binary tree will be O(log(N)), because we will have to traverse through the various nodes of the binary tree to get to the last node, from where we will insert our new node.

Deletion:
In a binary tree, the best-case for deletion is, O(1). The reason being, there could be only one node (in this case N = 1, hence O(1) == O(N)), or the node could be located at the root node, thus making the time complexity O(1). Apart from the deletion in and of itself, there is also swapping of different elements, which will take extra time, because the binary tree could be out-of-order depending on the elements.
In a binary tree, the worst-case for deletion is, O(N). The reason being, there could be N nodes that we must traverse through to get to the node that we want to delete. The two cases where this could be a possibility are when the node we are looking for (root node) is the node we want to delete, and it's the only node in the binary tree. The second case is that, we might have to go through N – 1 different nodes, to get to the final node (1). After getting to the final node, we then delete it. So, to get to N-1 different nodes to get to the final node (1), will take a time-complexity of N-1+1 = N, different nodes. Hence, the time complexity in the worst-case is O(N).

For any kind of self-balancing binary tree / AVL Tree, the best-case time complexity will be O(1), which will be the case when we only need to delete the root node. The worst-case time complexity will be when we have to reach the leaf of the self-balancing binary tree, which will be O(log(N)), because we will have to traverse the various nodes of the self-balancing binary tree to get to the last node, from where we will insert our new node.