25. Trees :: Binary Search Trees :: Time Complexity

All of the basic operations, find, add/insert, and remove, have time complexity which is proportional to the height of the tree, i.e, they are $O(h)$. For example, consider $findNode()$. During each recursive call, we will either move to the left child of $root$ or the right child of $root$:

If the height of the tree is h and the tree is balanced (all nodes are at levels h or h - 1), the number of recursive calls will be proportional to h. The number of nodes in a perfect tree is $n = 2^{h+1}$ - 1:

25. Trees :: Binary Search Trees :: Time Complexity

If n is the number of elements in the tree, then it can be shown without a great deal of effort that h is proportional to $lg\ n$:

Therefore, the basic operations all have average time complexity of $O(lg\ n)$. The major issue with BST's is that depending on the order in which elements are added to the tree and the operations that are performed, a BST can degenerate into what is essentially a linked list:

which means that the time to find, add/insert, and delete becomes $O(n)$. For this reason, there are other types of tree data structures—similar to BST's—which attempt to maintain balance as elements are added and removed from the tree. One of the more popular data structures is the red-black tree, but that is a topic beyond the scope of this course.