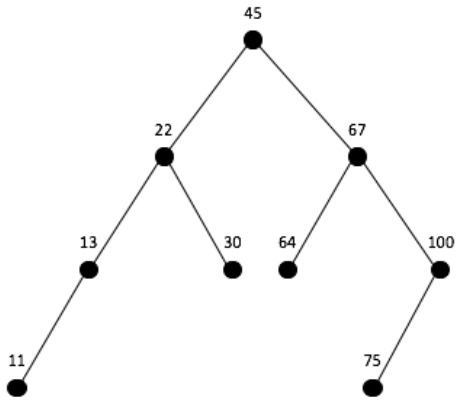
Practice Quiz key: Binary Trees and Binary Search Trees

1. A **full** binary tree is a binary tree in which every interior node has exactly two children.
2. If a perfect binary tree has height h, then we know:
    a. It has $\mathbf{2^h}$ leaves.
    b. It has $\mathbf{2^{h+1} - 1}$ total nodes.
3. If we know that a perfect binary tree has n nodes, then we know its height is approximately **log(n).**

4. Add the following numbers in the order given to a binary search tree: 45, 67, 22, 100, 75, 13, 11, 64, 30

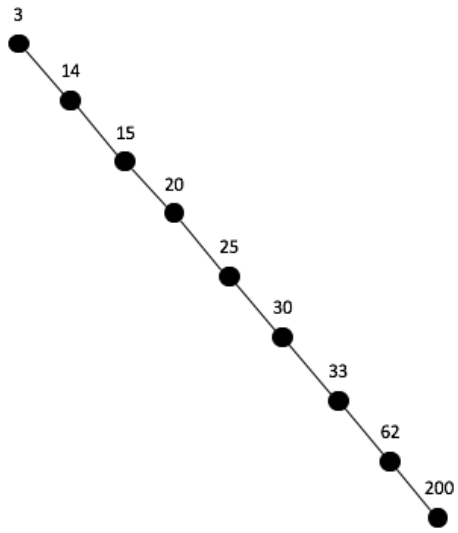


5. What is the height of the tree from #4? What is the height of the subtree rooted at the node holding the value 22? What is the depth of the node holding the value 22?

Tree Height = 3
Height of Subtree rooted at 22 = 2
Depth of the Node holding the value 22 = 1

6. Add the following numbers in the order given to a binary search tree: 3, 14, 15, 20, 25, 30, 33, 62, 200.

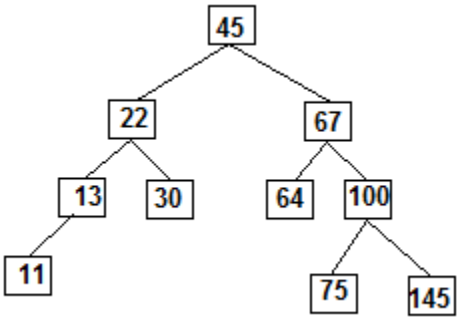
(See next page)



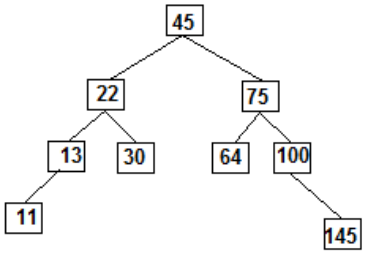
7. Is the tree from #6 balanced? Why or why not? What is the execution time required for searching for a value in this tree?

No, it is not balanced because it has all of the child nodes on the right side of the root node and no nodes on the left side. Balanced trees should be able to maintain O(log n) for all operations, however, the tree in #3 would yield an execution time of O(n) when searching for a value.

8. Add a new value, 145, to the tree from #4:



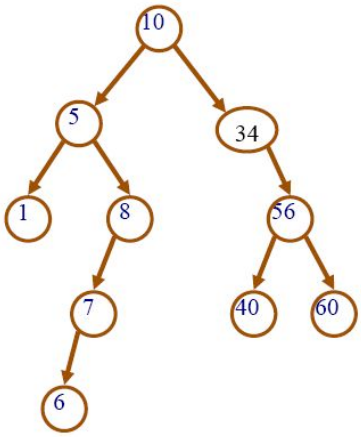
9. Remove the value 67 from the tree from #4. What value did you replace it with and why?



67 is replaced with 75 because 75 is the leftmost child of the right child of 67.

10. For the following Binary Search Tree, list the nodes as they are visited in a pre-order, in-order, and post-order traversal.

-- Pre-order traversal - visit the parent first and then left and right children
-- In-order traversal - visit the left child, then the parent and the right child
-- Post-order traversal - visit left child, then the right child and then the parent



Pre-order:
10 – 5 – 1 – 8 – 7 – 6 – 34 – 56 – 40 - 60

In-order:
1 – 5 – 6 – 7 – 8 – 10 – 34 – 40 – 56 - 60

Post–order:
1 – 6 – 7 – 8 – 5 – 40 – 60 – 56 – 34 – 10

11. Why is the time required to find a key in a binary search tree O(n) in the worst case?

The worst case occurs when the longest path in the tree is followed in search of the target and the longest path is based on the height of the tree. Binary trees come in many shapes and sizes and their heights can vary. We read that a tree of size n can have a minimum height of roughly log n when the tree is complete and a maximum height of n when there is one node per level. If we have no knowledge about the shape of the tree and its height, we have to assume the worst and in this case that would be a tree of height n. Thus, the time required to find a key in a binary search tree is O(n) in the worst case.