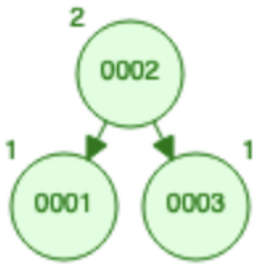
1 - Generate a Binary Search Tree based on some input string, e.g. 7, 5, 12, 4, 16, 9

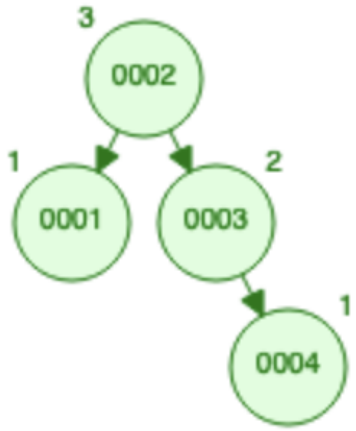
Simple, if less than the parent node, insert left. If the key is bigger than the parent node, insert right.

2- Generate an AVL Tree based on some input, e.g. 1, 2, 3, 4, 5, 6, 7

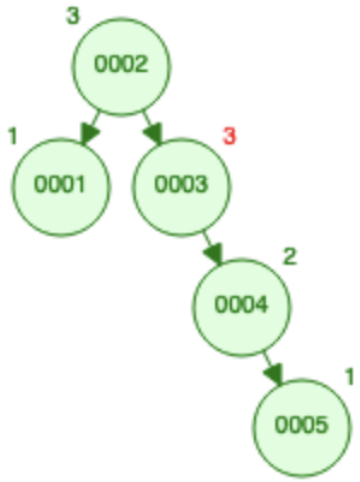
After inserting 3, the height of the tree is unbalanced because the difference is more than +-1 and a left rotation must take place to rebalance the tree.



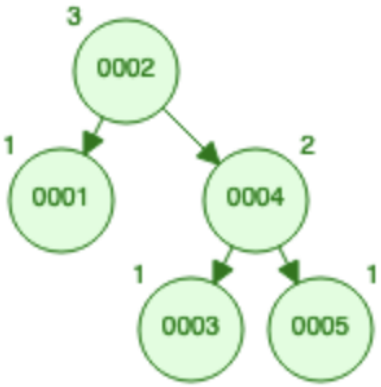
Inserting 4 does not make the tree unbalanced because the left and right subtress difference is only 1. (1 and 2)



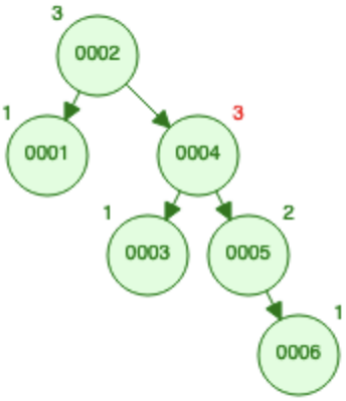
Inserting 5 will cause the tree to be unbalanced and needs a left rotation again. This will cause the difference for the left and right subtrees to be greater than +-1



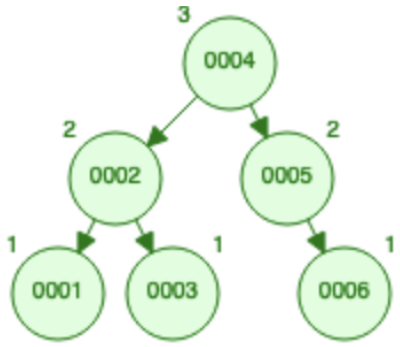
What we need to do is move 4 up one position and let the 3 key be the left node of 4, this will balance the tree.



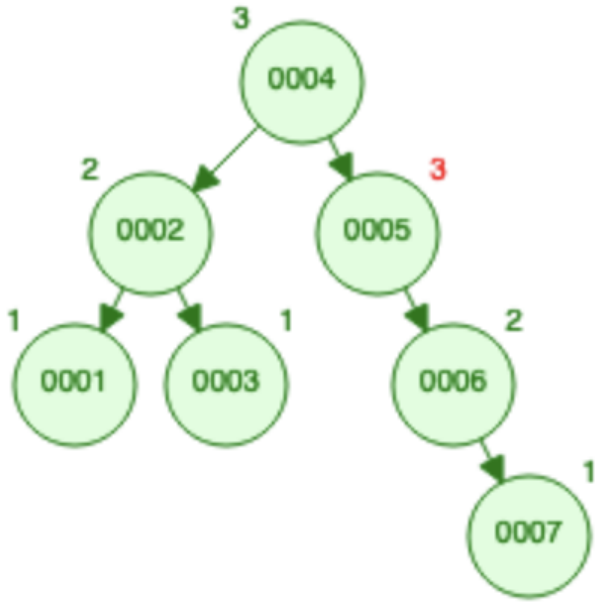
Inserting 6 will also cause the tree to become unbalanced as right side sub tree is now heavier than the left

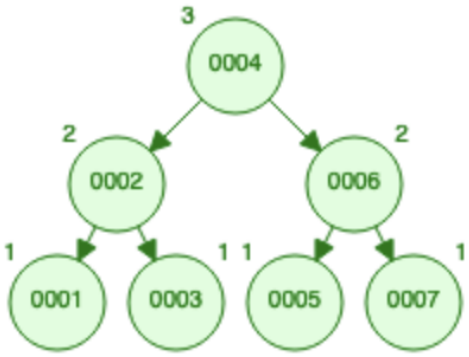


We can again move 4 up and do a single rotation left.

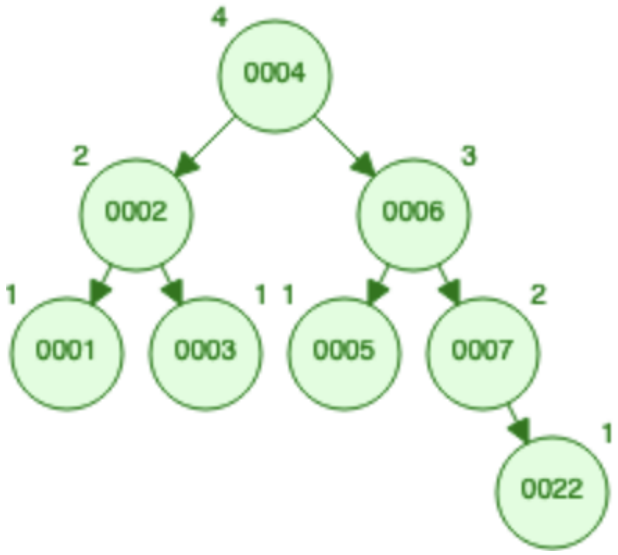


Insderting 7 will cause another left rotation because the lowest left of the sub child trees becomes unbalanced. I do not understand this part. This should be okay from my understanding of the rules. But why does it require a shift? Is it because the root node and right sub child tree contain the same height? Not because then the root will have the height of 4.





If I insert 22, the BST does what I expected it to do. Its like if I inserted 7, the root node has 4 and the right subtree has three and there is only a 1 height difference. Very confused.

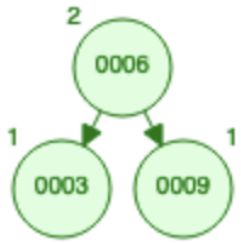


3 - Generate an AVL tree input that has 2 double rotations and a single rotation.

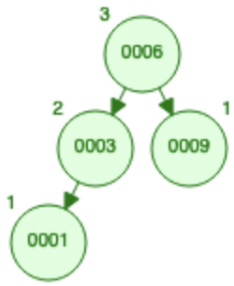
I can cause a double lef-right rotation if I insert 9,3,6

This causes a weird tree called a left-right imbalance. What has to happen is the 6 node has to be the root node. So I can rotate left once. So I have

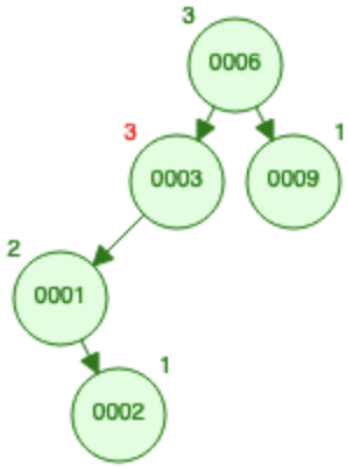
9 -> 6 -> 3 . but this causes the tree to be imbalance and not an avl. So then I can rotate right one time to get the following tree. Where 6 is the root.



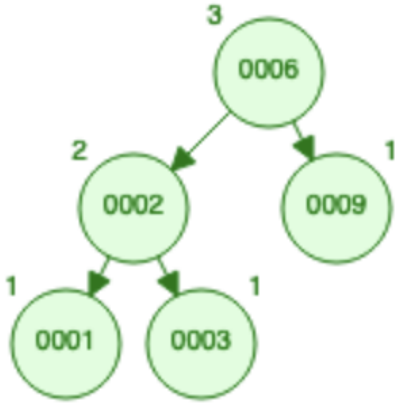
Now If I insert another number 1, this works just fine.



If I insert a 2, then this will cause the tree to be unbalanced as the left subtree and right subtree difference is greater than +1 and because the left subtree does not meet the binary search tree criteria, as 2 is not less than 1.

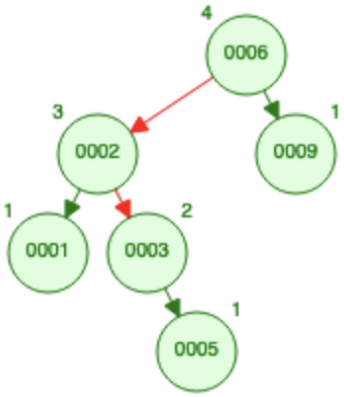


This will cause the 2 node to become the parent node of 1 and 3 and satisfy all requirements.

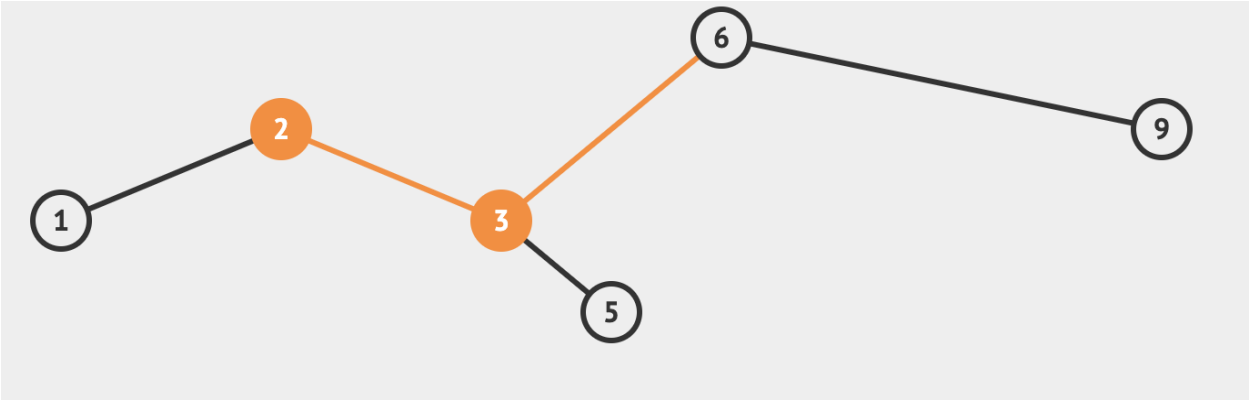


So far the tree 9,6,3,1,2 has given me a double right-left rotation and a single rotation

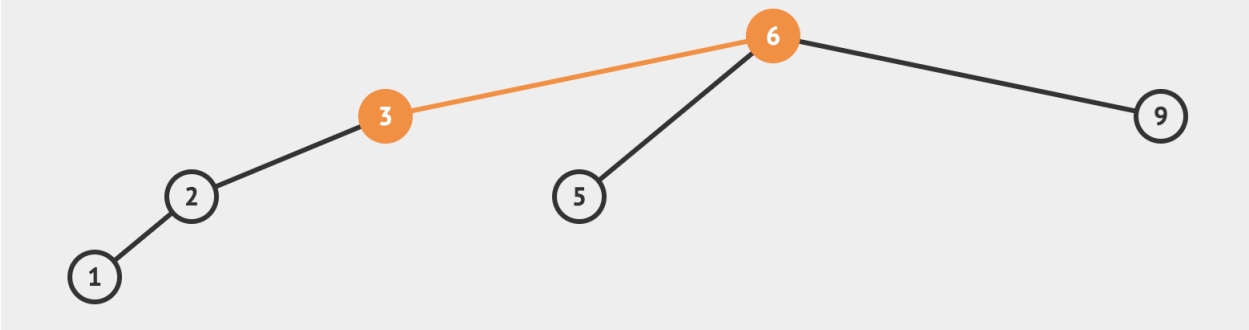
If I insert a 5, this will cause a double rotation again.

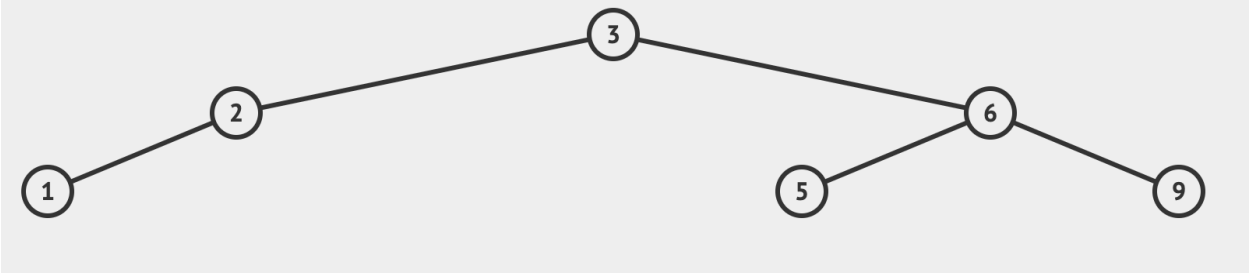


As 5 is not greater than 6 and the height of difference is greater than 1. So we gotta rotate 2 to the left



And then rotate 6 to the right

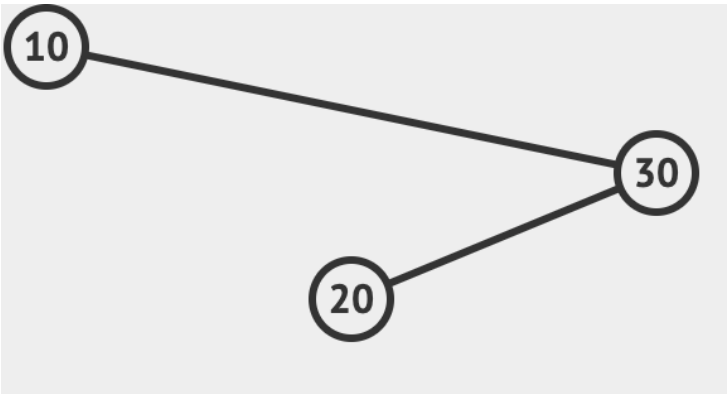




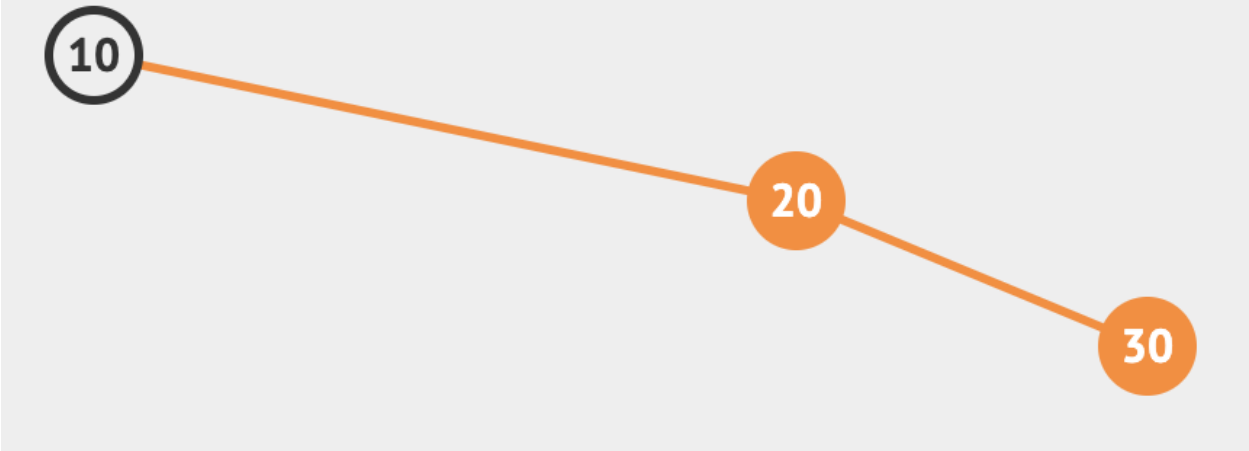
So the tree 9,6,3,1,2 ,5 satisfies the 2 double rotations and 1 single rotation criteria.

4 - Generate an AVL tree that has 1 double rotations and 2 single rotation.

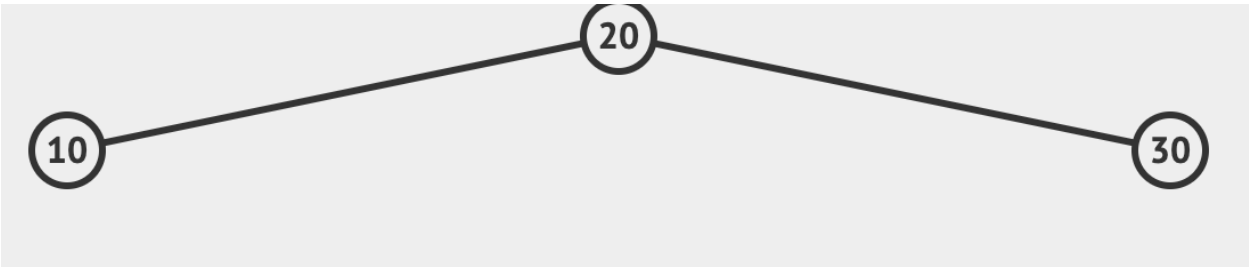
If I start with insert 10,30,20 this will give me a right-left imbalance tree and will need a single rotation to start off with as the order of the right subtree does not satisfy a BST and the weight og the right subtree is greater than the left.



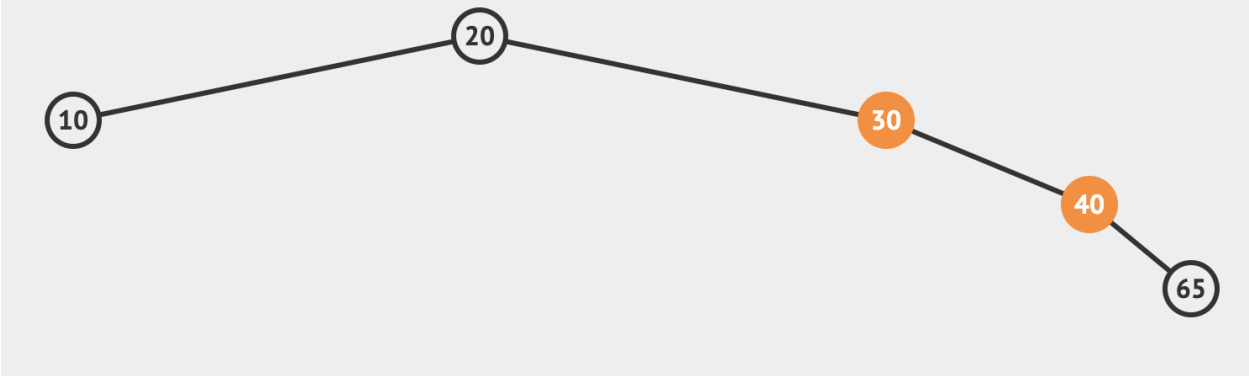
A rotation of 30 to the right will put this in order

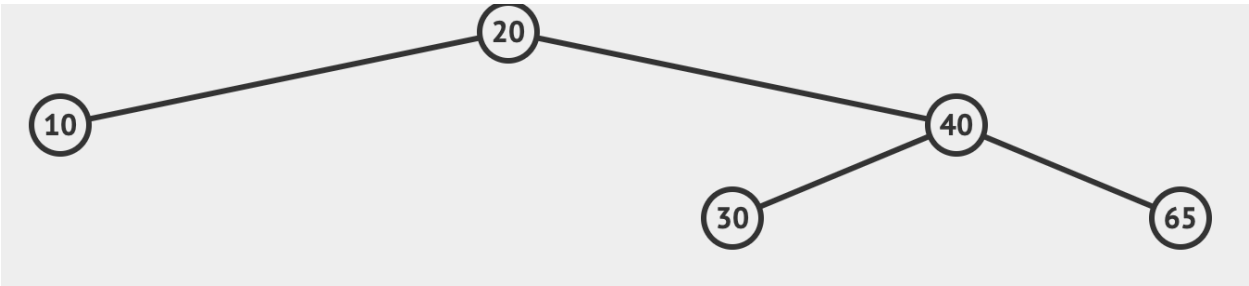


A rotation of 10 to the left will fix this.

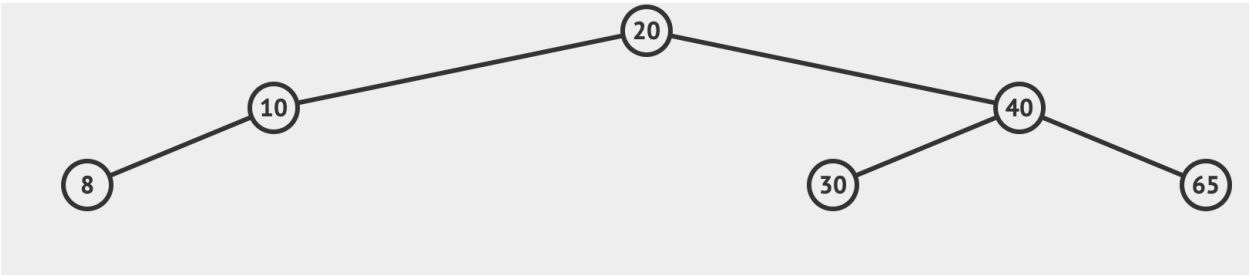


If I insert 40 everything works out. But if I add a 40 and 65 then a single rotation happens when and 30 and 40 need to be rotated to the left

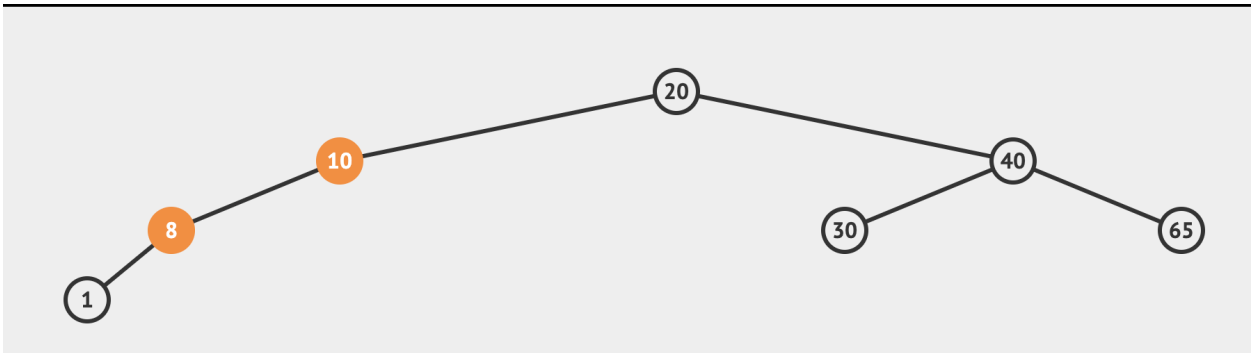


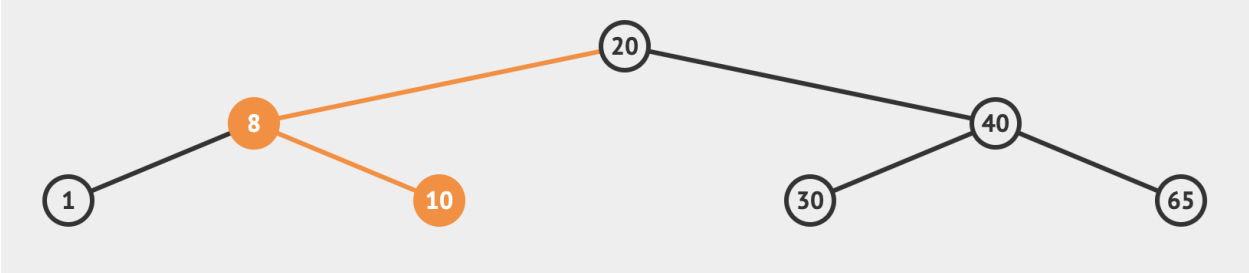


If I insert 8, everything works out. Nothing really changes



But if I add 1, this will cause another single rotation where 8 and 10 will rotate to the right twice.





Thus the tree 10,30,20,40,65,8,1 will give me a a double rotation and two single rotations

5 - Generate an input string for a 2-3 tree that results in 2 nodes at the root, and at least 3 nodes with only one value.