1. Draw a series of figures demonstrating the insertion of the values

   20, 9, 3, 7, 5, 8, 25, 30, 15, 6, 17

   into an initially ***empty*** AVL tree. Insert the values in the order they appear in the given sequence. You must:

   - Show the resulting AVL tree immediately before and after each insertion step that causes the tree's rebalancing.
   - Calculate the ***balance degree*** (as the difference between the heights of the left and the right subtrees rooted at a node) and label each node of the AVL tree before and after the necessary rebalancing.
   - Clearly indicate the node(s) at which the rotation is performed. Here, let $v$ be the first node you encounter in going up from the newly added node, say $z$, towards the root of the AVL tree $T$ such that $v$ is unbalanced. Then, let $x$ be the child of $v$ such that $x$ is the ancestor of $z$. Determine whether the subtree of $T$ rooted at $v$ is ***right*** or ***left heavy***. Similarly, indicate whether the subtree of $T$ rooted at $x$ is ***right*** or ***left heavy***.
   - Performing a ***single*** or a ***double rotation*** as a rebalancing (repairing) operation on the AVL tree, specify the type of the rotation that you apply: ***Single Left Rotation***, ***Single Right Rotation***, ***Left-Right Rotation***, or ***Right-Left Rotation***.
   - Each time a new value is added, ensure that both the ***Binary Search Tree Property*** and the ***Height-Balance (AVL Tree) Property*** are maintained.

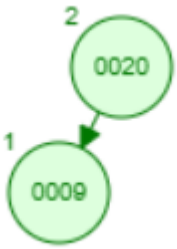
A :

BF : balance factor.

1. insert 20 :

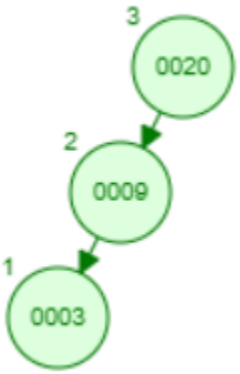

   BF = 0

2. insert 9



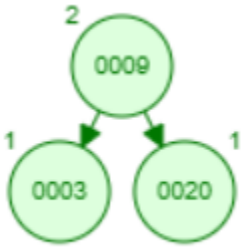
9 < 20, shift it to the left subtree .  20 BF = 1 ; 9 BF = 0

3. insert 3



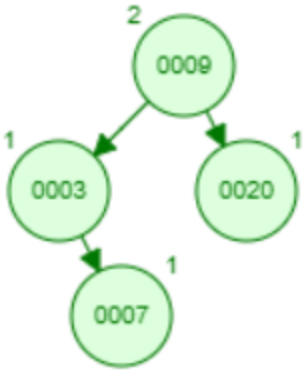

3 < 9 < 20, shift to the left subtree.

Because 20 BF = 2 , the left subtree of the left subtree is too high, so it need to single right rotation with 9 as axis. Now the BF of 9 and 20 are 0 .




Because 20 BF = 2 , the left subtree of the left subtree is too high, so it need to single right rotation with 9 as axis. Now the BF of 9 and 20 are 0 .

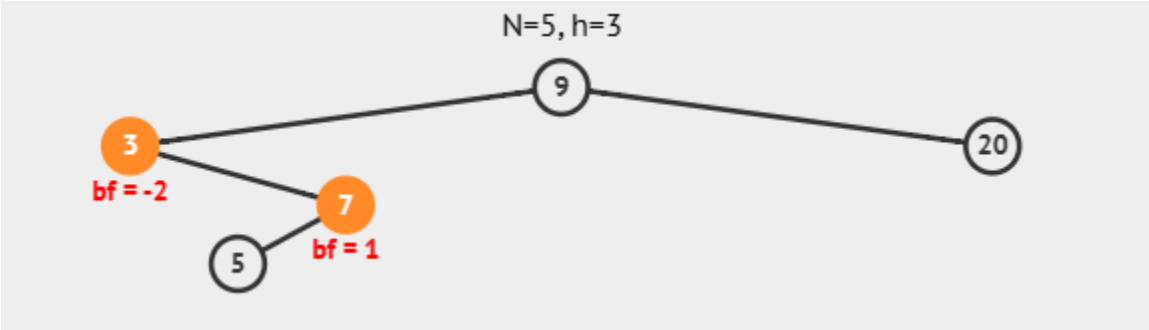
4. insert 7




Because 7 < 9 ; 7 > 3. So it will be on the right subtree of the 3.

BF of 9 is 2-1 = 1 so it's ok,

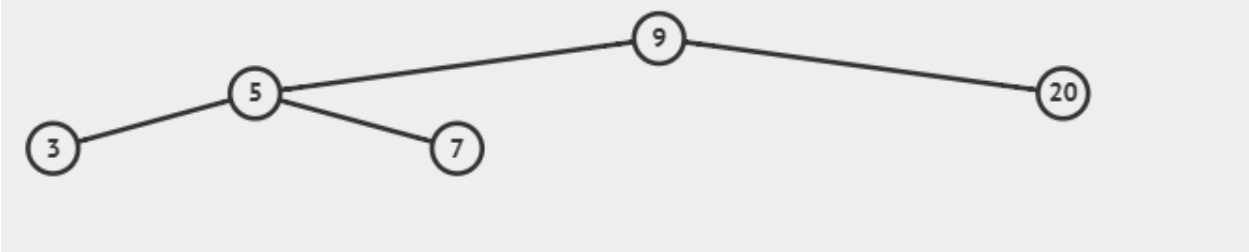
5. insert 5




Because 3 < 5 < 7 , it will belong to left subtree of 7.

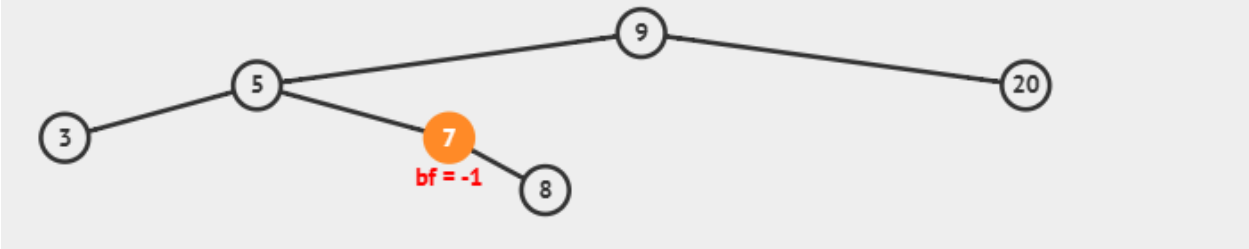
But the BF of 3 will be 0-2=-2, which is not balanced. BF of 7 is 1 .

Do a single right rotate with 7 axis, then do another single left rotation of 3 with 5 axis.




Now the BF of 5 is 0 and BF of 9 is 1.

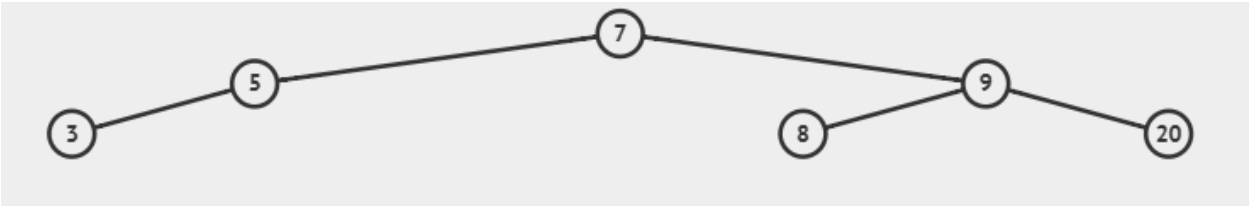
6. insert 8




Because 8 < 9 ; 8 > 5 , 7 ; so it belongs to right subtree of 7.

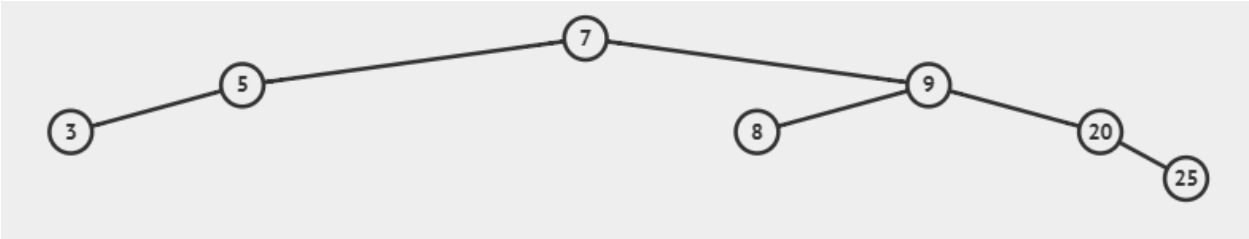
But the BF of 7 and 5 are 0-1 = -1 ; BF of 9 is 3-1=2 which is not balanced.

Do a single left rotation of 7 with 5 axis , then do another single right rotation of 7 with 9 axis.




Now the BF of 7 is 0, balanced.

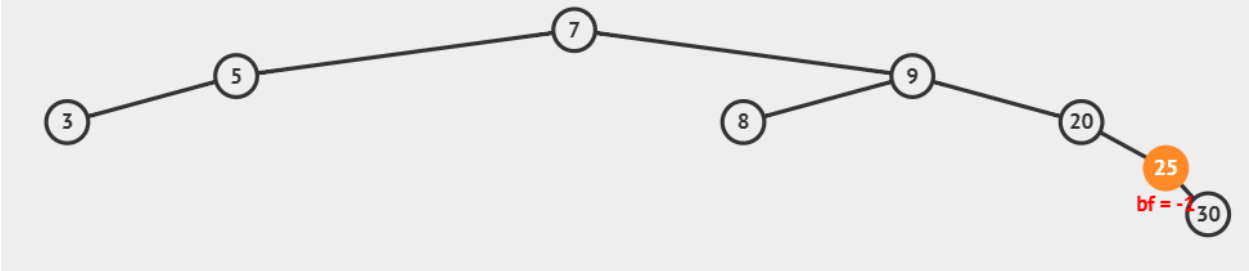
7. insert 25




Because 25 is the largest so far, so it will be on the right subtree of 20.

Because the BF of 7 is 2 – 3 = -1, it's balanced.

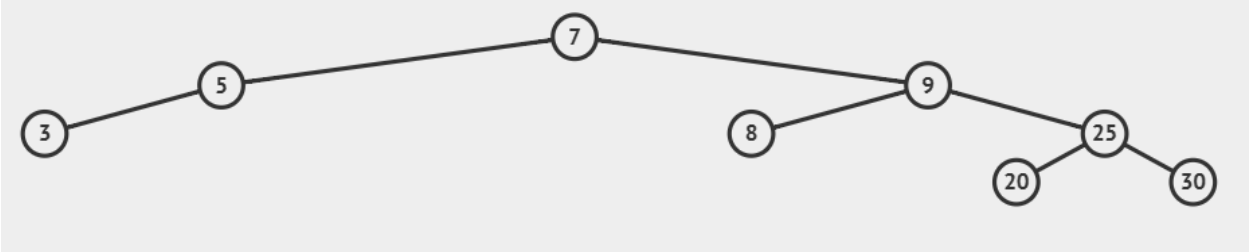
8. insert 30



Because 30 > 25 , so it will be on the right subtree of 25.

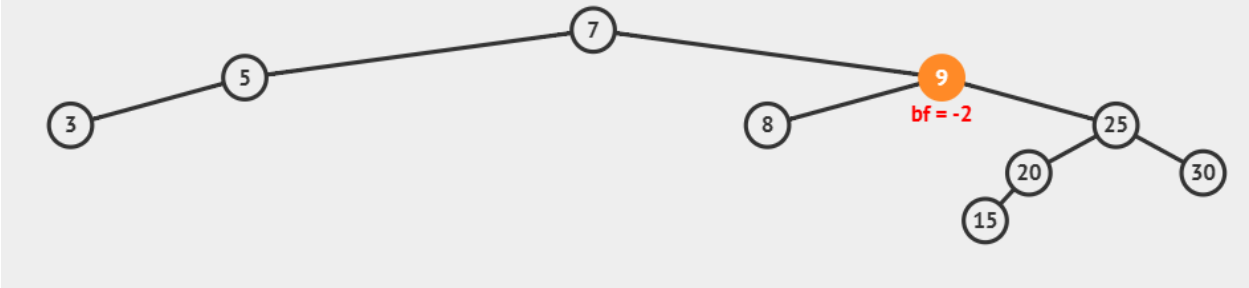
But the BF of 20 is 0 – 2 = -2 which is not balance.

So we need to do a single left rotation for 20 with 25 axis.




Now the BF of 25 is 0 ; BF of 9 is –1 ; BF of 7 is –1 . Balanced.

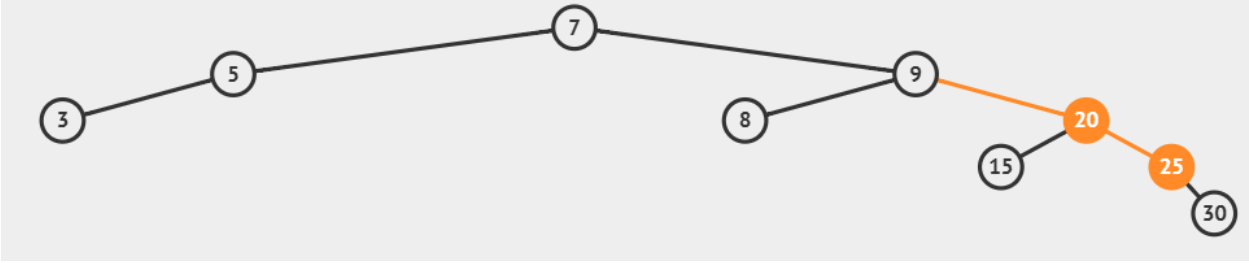
9. insert 15



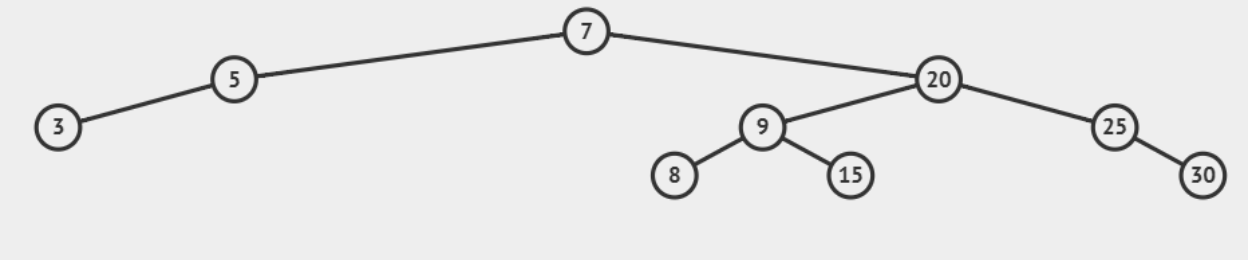

Because 20 , 25 >15 > 9 , so it will belong to left subtree of 20.

But the BF of 9 is 1 – 3 = -2 which is not balanced.

So we need to do a single right rotation of 20 with 25 axis .

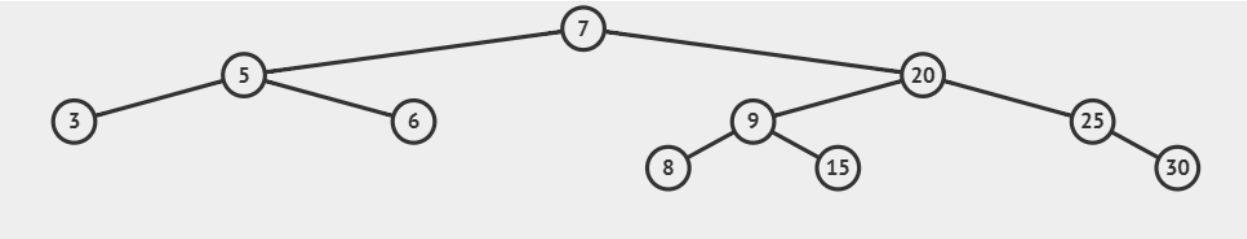



Then we need to do another  left rotation of 9 with 20 axis.




Now the BF of 25 is –1 ; BF of 20 is 0; BF of 7 is –1. Balanced.

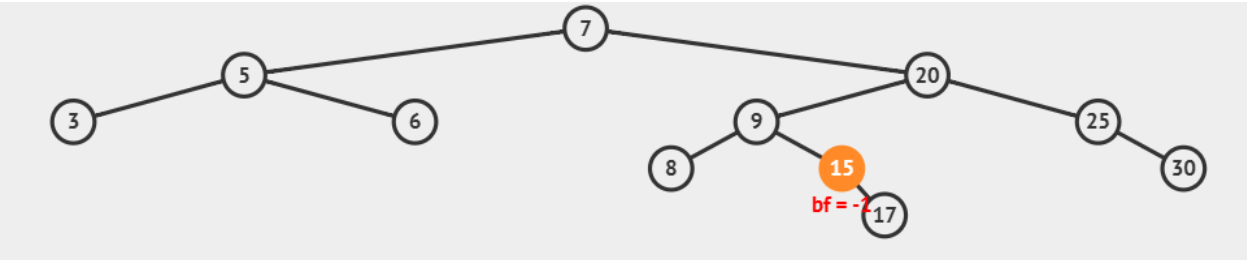
10. insert 6




Because 7 > 6 > 5, so it will be on the right subtree of 5.

The BF of 5 is 0 and BF of 7 is –1. Balanced .

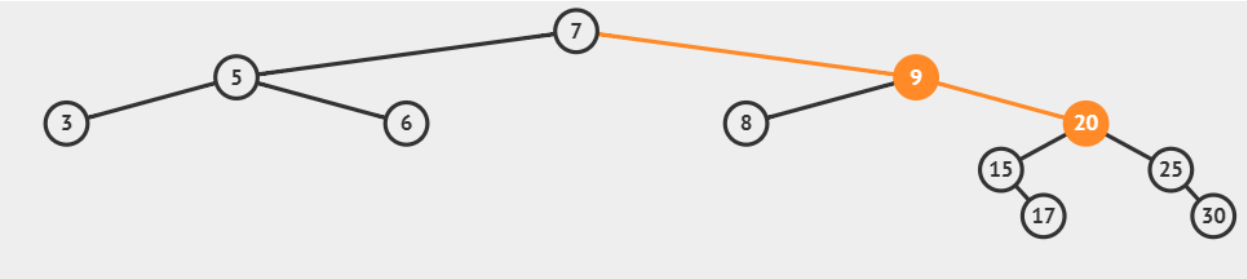
11. insert 17



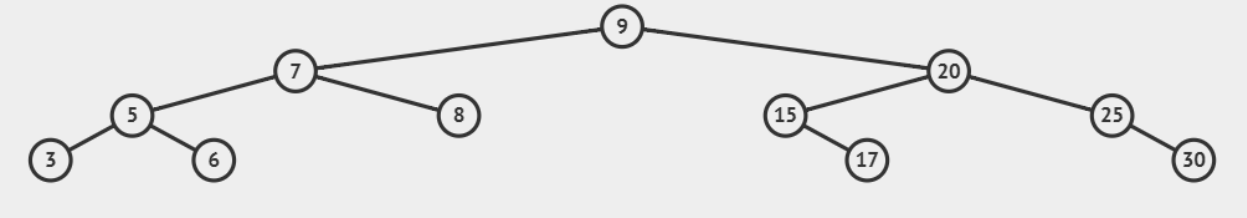

Because 20 < 15 > 9, so it belongs to right subtree of 15.

But the BF of 7 is –2 , not balanced.

Make it single right rotation of 9 with 20 axis.




Then do another single left rotation of 9 with 7 axis.




The BF of 9 ; 5 ;20 are 0 .

BF of 7 is 1; BF of 25 is –1. Balanced.

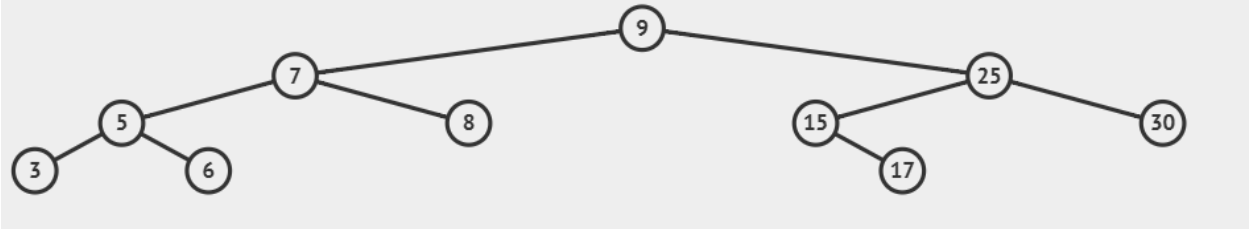
2. Now, draw a series of figures showing the deletion of the values:

   20, 15, 8, 25, 30, 9, 17, 5, 6, 3, 7

   from the AVL tree built in the previous part of the task. Delete the values in the order they appear in the given sequence. Draw the AVL tree after each deletion and rotation (if any) and complement each of the figures with all the aforementioned details.

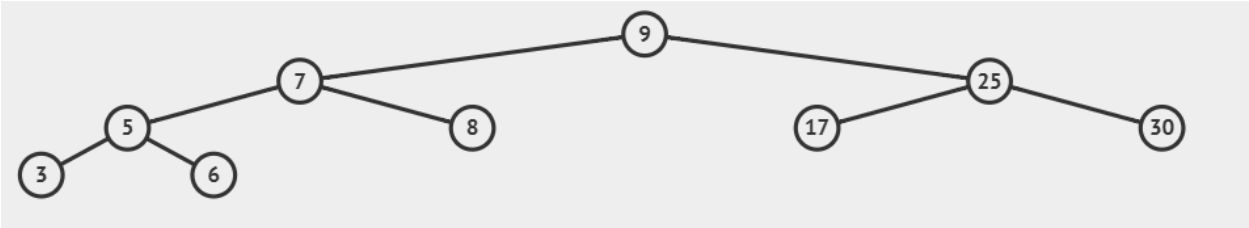
   When you delete a node with two children, you must always replace it with ***'the largest element smaller than the key of the deleted node'*** (as opposed to the other possible rule: 'the smallest element larger than the key of the deleted node'). **Following this rule is important** to let us check your solution quickly.

1. Delete 20



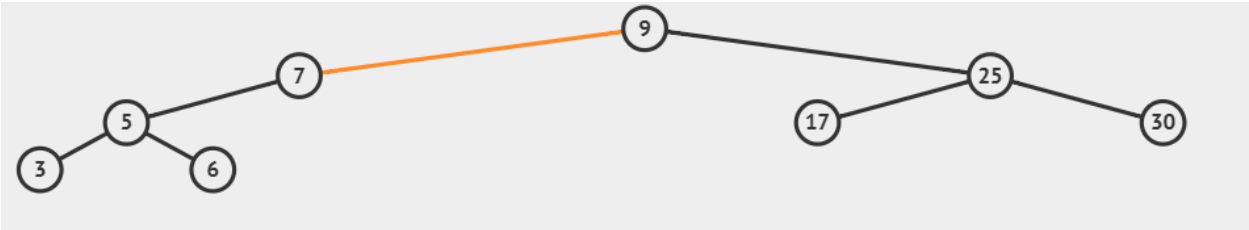

The BF of 9 , 25 , 7 are 1; Balanced.

2. Delete 15



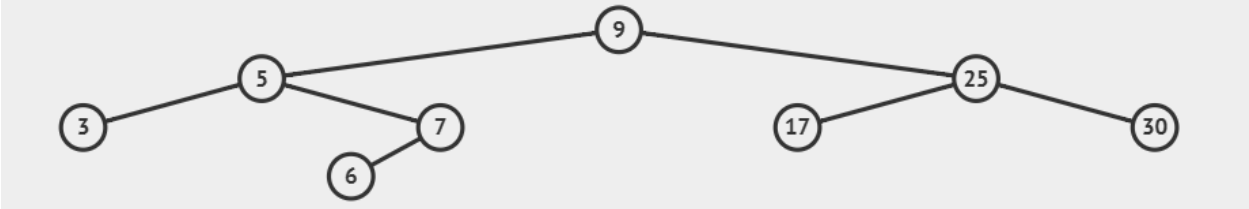

BF of 9 is 1 ; BF of 25 is 0. Balanced.

3. Delete 8



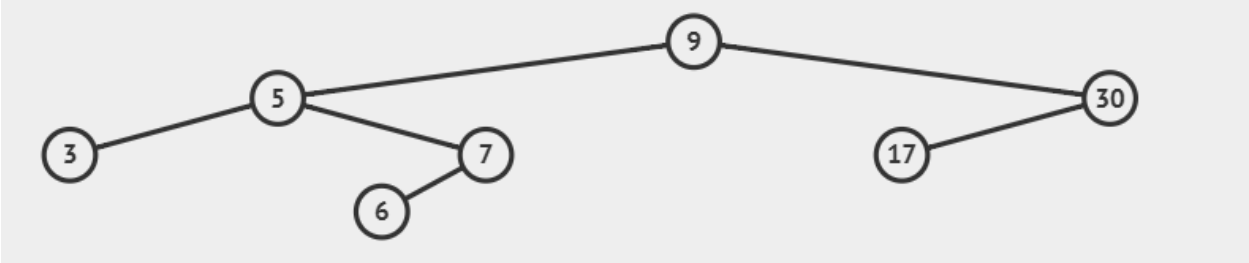

Because the BF of 7 is 2, which is not balanced.

Need to do a single right rotation of 5 with 7 axis.



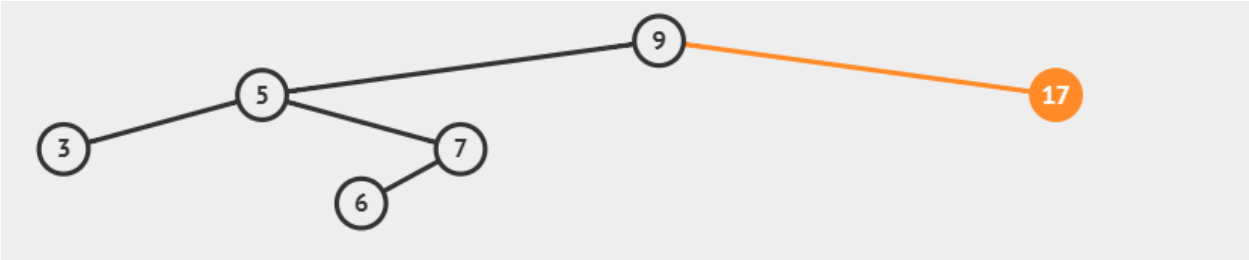

Now the BF of 5 is –1 ; BF of 9 is 1. Balanced.

4. Delete 25

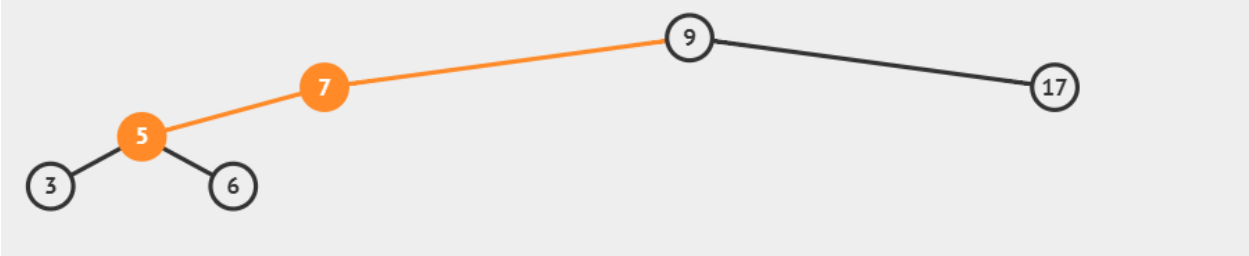



BF of 30 is 1 ; BF of 9 is 1. Balanced.

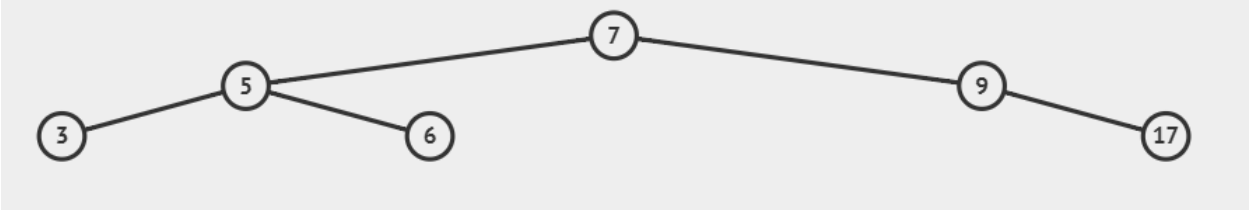
5. Delete 30




After deleted 30; the BF of 9 is 2. The BF of 5 is –1 .
Need to do a single left rotation of 7 with 5 axis.

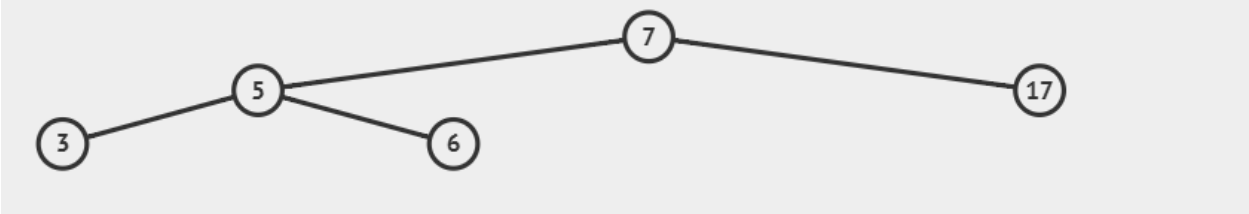



Then do another right rotation of 7 with 9 axis.



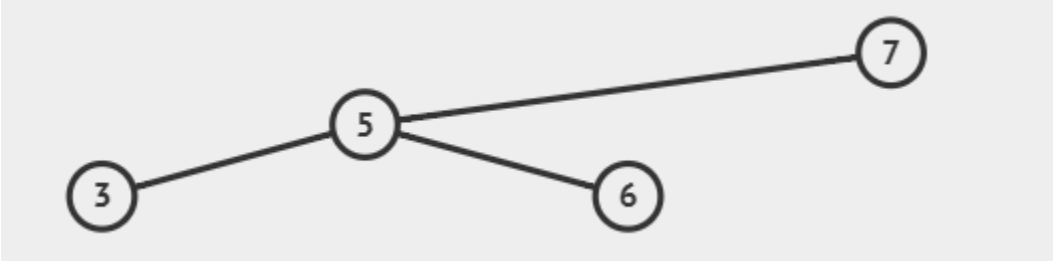

Now the BF of 7 is 0 ; BF of 9 is –1; BF of 5 is 0. Balanced .

6. Delete 9



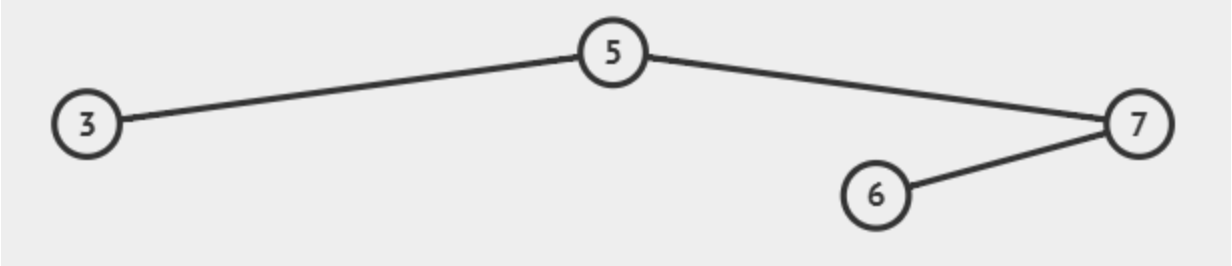

The BF of 7 is 1 ; BF of 5 is 0 ; Balanced.

7. Delete 17



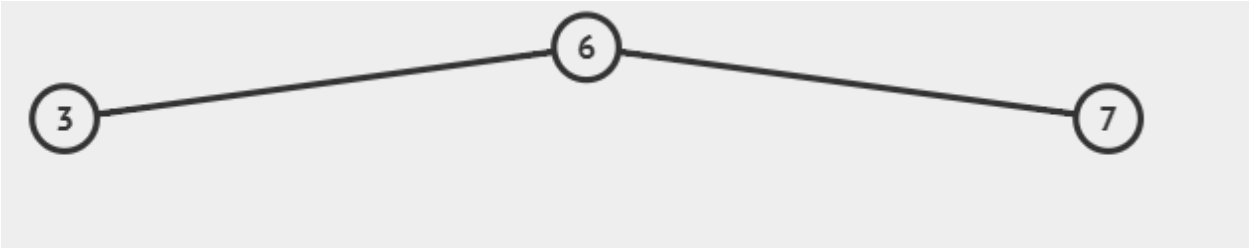

Because the BF of 7 is 2, it's not balance.

Need to do a single right rotation of 5 with 7 axis.




Now the BF of 5 is –1, which is balanced.

8. Delete 5




The BF of 6 is 0, balanced.

9. Delete 6


BF of 7 is 1, balanced.

10. Delete 3


Only 7 left , the BF is 0.

11. Delete 7

Nothing left .

3. Develop a linear $O(n)$ time algorithm that accepts an arbitrary binary search tree as an input and creates a **balanced** binary search tree whose height is $O(\log n)$. As the answer to this question, provide its pseudocode and description. Explain why you believe that your algorithm takes a linear time.

A: the pseudocode is :

Algorithm BalanceBsT(root):

// In-order traversal to get an ordered list of nodes (0(n))

nodes = InOrderTraversal(root)

// Recursively construct a balanced tree (0(n))

return BuildBalancedBsT(nodes,0, len(nodes)-1)

Function BuildBalancedBST(nodes, start,end):

if start > end:

return null

mid =(start +end)//2

root = nodes[mid]

root.left = BuildBalancedBST(nodes, start, mid-1)

root.right =BuildBalancedBST(nodes,mid +1,end)

return root

The reason is :

1. Time complexity of in-order traversal: O(n)
   In-order traversal requires visiting every node in the binary search tree once. For a tree containing several nodes, each node is visited once, and the time complexity is strictly O(n).

2. Time complexity of recursively building a balanced tree: O(n) The core of recursive construction is the divide-and-conquer strategy:

Select the middle element as the root: by calculating the midpoint of the interval mid = start+end/2, the time is O(1).

Recursively process the left and right halves: each recursion divides the array into two halves, but each node is only visited once.

For the total time complexity is O(n) + O(n) = O(n)

Reference:


1. https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
2. https://www.youtube.com/watch?v=zP2xbKerIds
3. https://tedwu1215.medium.com/avl-tree-%E9%AB%98%E5%BA%A6%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%85%83%E6%90%9C%E5%B0%8B%E6%A8%B9%E4%BB%8B%E7%B4%B9%E8%88%87%E7%AF%84%E4%BE%8B-15a82c5b778f ; Jul 4, 2023 ; 碼農思考中 - Ted;
4. https://blog.csdn.net/qq_28205153/article/details/51547495 ; 2016-05-31 17:20:44 published; TimeShatter
5. https://visualgo.net/en/bst