Problem 03: Discuss time complexity of insertion, deletion, and search in Balanced binary search tree (BST). add Simulation steps ~~for~~ for every Operation.

Solution:

Time Complexity between BST insertion, delation and search is given below:

Operation	Best case	Average case	Worst case
Insertion	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$
Deletion	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$
Search	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$

Semulation steps for every Operations:

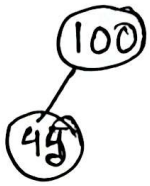
Insertion:

Step: 1: After insert root "100" the tree will be:

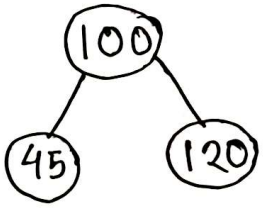
(100)

Step: 2: It will have two child element, Right and left. ~~The e~~ if the element is greater the Root then the element willbeinsent in right; Otherwise the element will be insent in left.

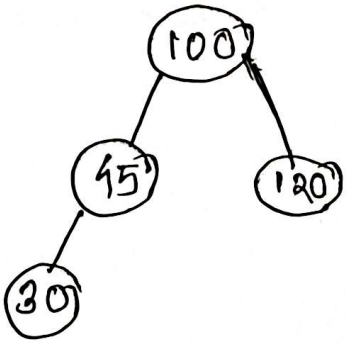
Now, Insert "45"; it is Smaller then "100"; so, It will be insert in left side of root node. (100 > 44).



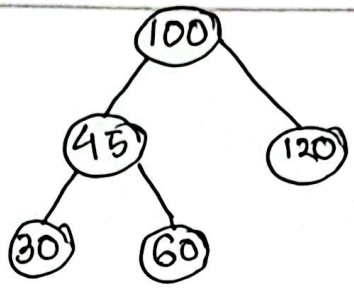
Step:3: Insert "120", Here "120" is greater then "100". So, It will be inserted in right node of root.



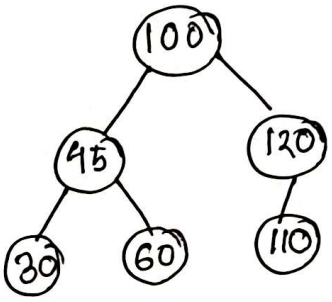
Step:4: Insert "30". "30" will be insert in the left Sub tree, as "100" has already a left node. So, "30" will compare with "45", as "45" is greater then "30" [45 > 30]; "30" will be "45"'s left node.



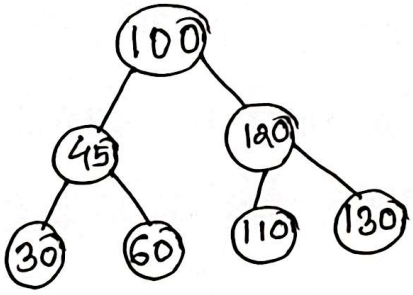
Step:5: Insert "60". "60" is Smaller then "100". (100 > 60). It will be insert in "100" left sub-tree. as "45" is Smaller then "60". So, "60" will insert in the right of "45".



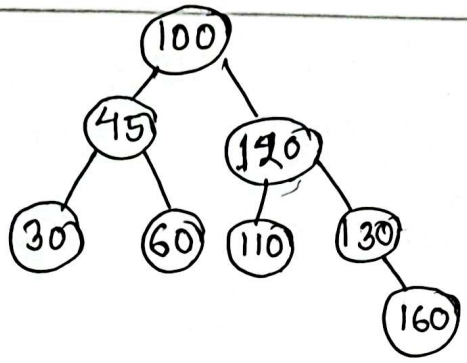
Step :6: Insert "110"; "110" is greater then "100" (110>100). So, "110" will be insert in the right sub tree. and "110" also ~~greater the~~ smaller then "120". (120 < 110). So, "110" will be "120" 's left node.



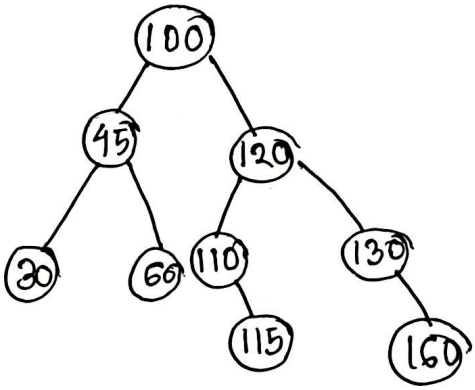
Step :7: Insert "130". "130" is greaten then "100" (100<130), and "130" is also greaten then "120" (130>120). So, "130" will insert in right sub-tree of "100" and "120".



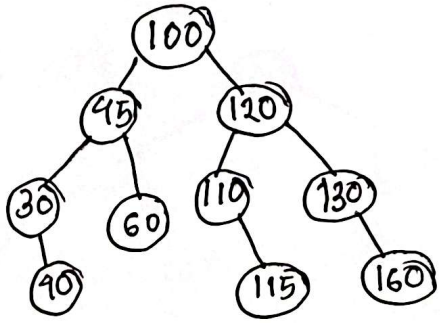
Step :8: Insert "160", "160" is greaten then "100", It is also greaten then "120" and also greatten then "130", [100 < 120 < 130 < 160]; So, Here "160" will be "100" and "120" 's right sub-tree and "130" 's right child.



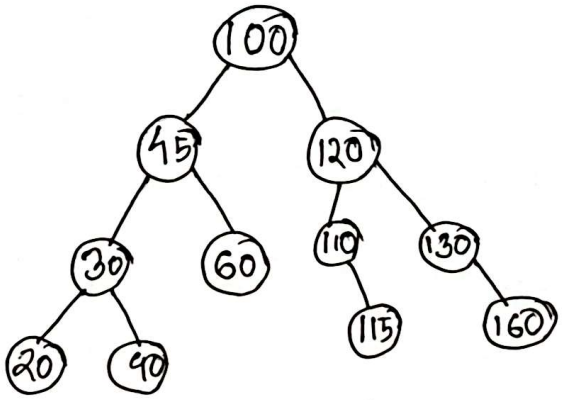
Step: 09: Insert "115", "115" will be insert is the same way. it will be insert is "110"s right child. [100 < 120 > 110 < 130]



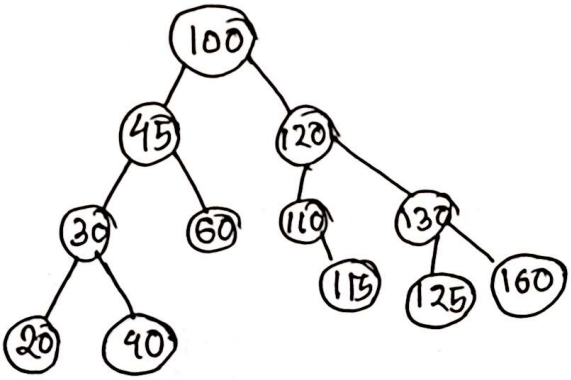
Step: 10: Insert "40"; "40" is smaller then "100" (100>40) also smaller then "45" ~~and "30". So, it will be insert in left sub tree of "100", "45" and~~ and greater then "30" (30 < 40). So, "40" will be "100", "45" left sub tree but "30"'s right node. [100 > 45 > 30 < 40]



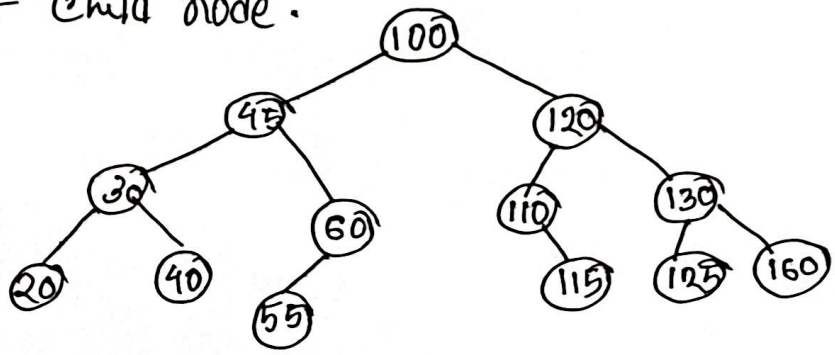
Step : 10: Insert "20"; "20" is smaller then "100", "45", and "30". It will be insert in "100"'s left Subtree, "45" and "30"'s left tree. [100 > 45 > 30 > 20]



Step : 11: Insert "125"; (100 < 120 < 130 > 125), so, "125" will be "100", "120"'s Right Subtree and "130"'s left Child.



Step : 12: Insert "55"; (100 > 45 < 60 > 55). So, "55" will be be "100"'s a left Subtree, "45"'s right Sub-tree. and "60" left child node.



Deletation:

Step:01: ~~Delete~~ Delete "115"; "115" is compare with root "100". Though 115 > 100, So, "115" go to "100"'s Right Sub-tree.

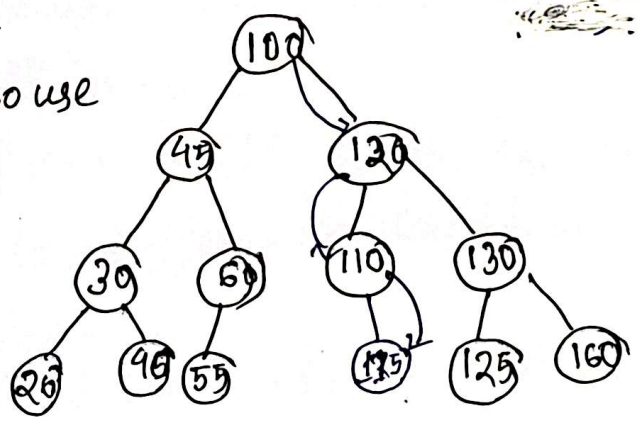
Step:02: "100"'s Right child node is "120". Here "115" is Compare with "120", Though 115 < 120, so, It will go to "120" left child node to compare.

Step:03: "120"'s left child node is "110". Here "110" is Compare with "115", Though 110 < 115, So, It will go to "110"'s ~~left child node~~ Right node to compare.

Step:04: "110"'s right node is "115", it will compared with the "115", (115 == 115). So, 115 will be delete, the "110"'s right ~~null~~ node's pointer will be "NULL".

The tree:

[Here we also use Searching]

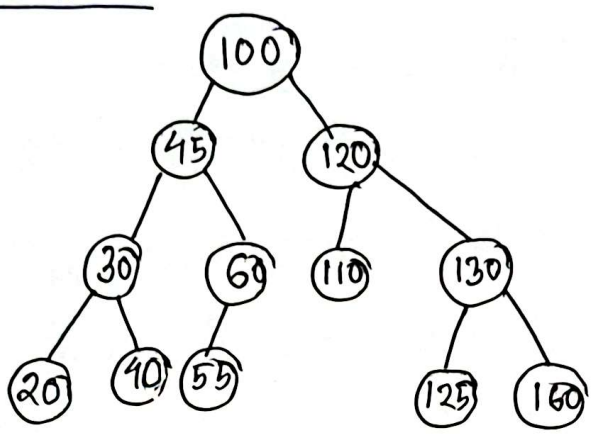


[100 ≤ 115]

[120 ≥ 115]

[115 > 100]

The new tree:



Before Deletetion, the tree in Pre-Order traversal:
100, 45, 30, 20, 40, 60, 55, 120, 110, 115, 130, 125, 160

After Deletation, the tree is Pre-order traversal:
100, 45, 30, 20, 40, 60, 55, 120, 110, 130, 125, 160

Search:

Process: If we want to search any element,

Step 1: the search element will compare will the root. if search element and ~~key~~ tanget matched then we will return root.

Step 2: Otherwise, we will search the element in left or right based one two condition.

The condition:

(i) if (root -> tanget < key), then search in left;

(ii) if (root -> tenget > key), then search in right;

Simulation:

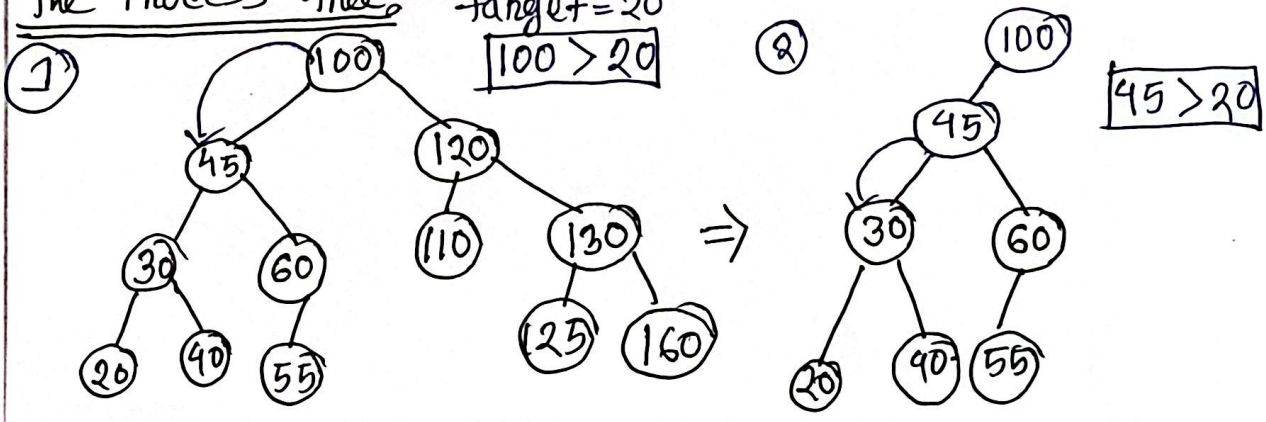
Step:01: target = "20"; Here, 20 is compare with 100, Though (100 > 20), So, the Search will start in 100's left sub-tree. and go to "100" is left node to compare.

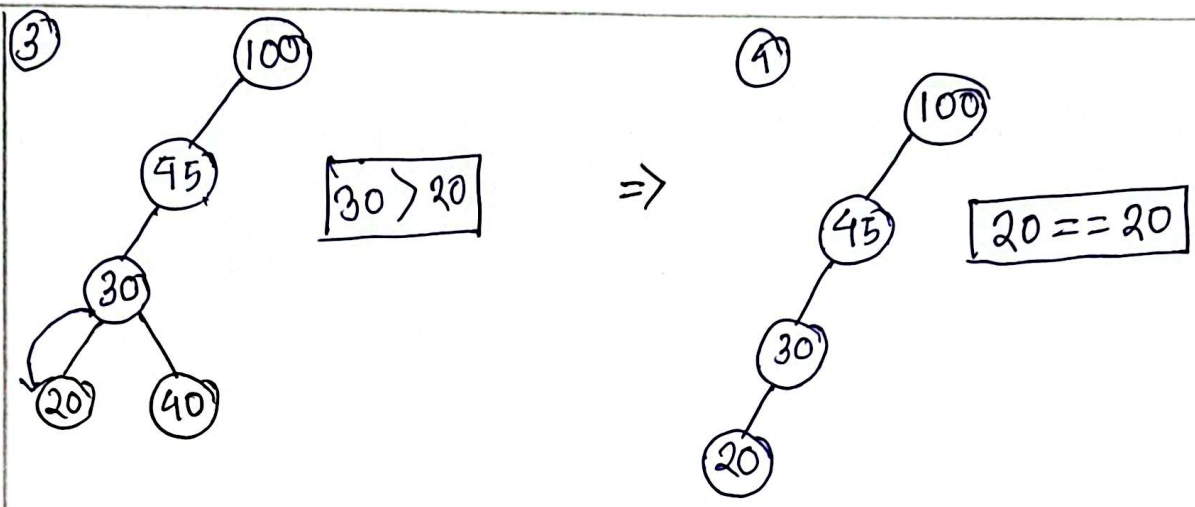
Step:02: "100" left child node is "45". 45 will be compare will "20". Though (20 < 45). So, the Search will start in 45's left sub-tree. And go to "45"'s left node to compare.

Step:03: "45" left child node is "30", so. "30" will be compare with "20". Though (20 < 30); the Search will start in left sub-tree. And go to "30"'s left child node to compare.

Step:04: "30"'s left child is "20", so, "20" will compare will target = "20". though (20 == 20). So, then it will return the index of node "20".

The Process tree:





Here in this, Searching function we only search of only half of the sub-tree, so, other side we don't check or search.

This is how the BST (Binary Search tree)'s insertion, deletion and search worked.