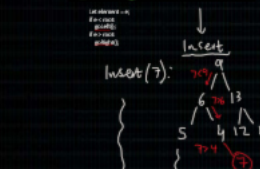
BST vs BT

BT: max 2 child

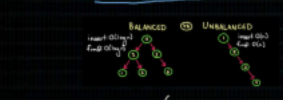
BST: - max 2 child
- left < root
- Right > root



Insert

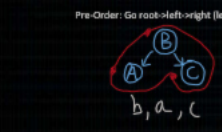
Insert(7):

BALANCED VS UNBALANCED



TRAVERSAL

Pre-Order: Go root->left->right (left pointer)

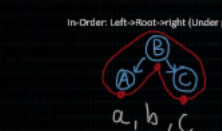


b, a, c

Pre-Order

(Root → Left → Right)

In-Order: Left->Root->right (Under pointer)

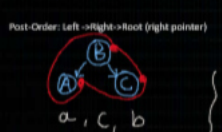


a, b, C

In order Used typically

(Left → Root → Right)

Post-Order: Left ->Right->Root (right pointer)

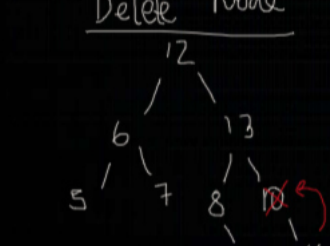


a, c, b

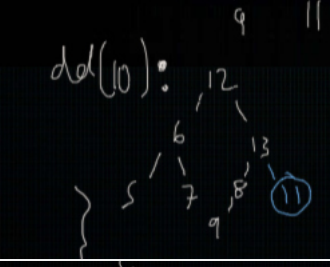
Post-Order

(Left → Right → Root)

Delete Node

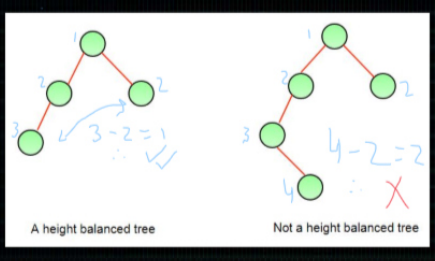


del(10):



Height Balanced Tree

A *height balanced binary tree* is a binary tree in which the height of the *left* subtree and right subtree of any node does not differ by more than 1 and both the left and right subtree are also height balanced.



Operations: Time Complexity

Binary Tree (max 2 child)

- ***Searching:*** *For searching element 2, we have to traverse all elements (assuming we do breadth first traversal). Therefore, searching in binary tree has worst case complexity of O(N).*
- ***Insertion:*** *For inserting element as left child of 2, we have to traverse all elements. Therefore, insertion in binary tree has worst case complexity of O(N).*
- ***Deletion:*** *For deletion of element 2, we have to traverse all elements to find 2 (assuming we do breadth first traversal). Therefore, deletion in binary tree has worst case complexity of O(N).*

Binary Search Tree

(left < root < right)

- ***Searching:*** *For searching element 1, we have to traverse all elements (in order 3, 2, 1). Therefore, searching in binary search tree has worst case complexity of O(n). In general, the time complexity is O(h) where* ***h*** *is the height of BST.*
- ***Insertion:*** *For inserting element 0, it must be inserted as the left child of 1. Therefore, we need to traverse all elements (in order 3, 2, 1) to insert 0 which has the worst-case complexity of O(n). In general, the time complexity is O(h).*
- ***Deletion:*** *For deletion of element 1, we have to traverse all elements to find 1 (in order 3, 2, 1). Therefore, deletion in binary tree has worst case complexity of O(n). In general, the time complexity is O(h).*

AVL / Height Balanced Tree

{ height = h
h(right) − h(left) = 1 }

- ***Searching:*** *For searching element 1, we have to traverse elements (in order 5, 4, 1) = 3 = $log_2 n$. Therefore, searching in the AVL tree has worst-case complexity of $O(log_2 n)$.*
- ***Insertion:*** *For inserting element 12, it must be inserted as the right child of 9. Therefore, we need to traverse elements (in order 5, 7, 9) to insert 12 which has the worst-case complexity of $O(log_2 n)$.*
- ***Deletion:*** *For deletion of element 9, we have to traverse elements to find 9 (in order 5, 7, 9). Therefore, deletion in a binary tree has worst-case complexity of $O(log_2 n)$.*