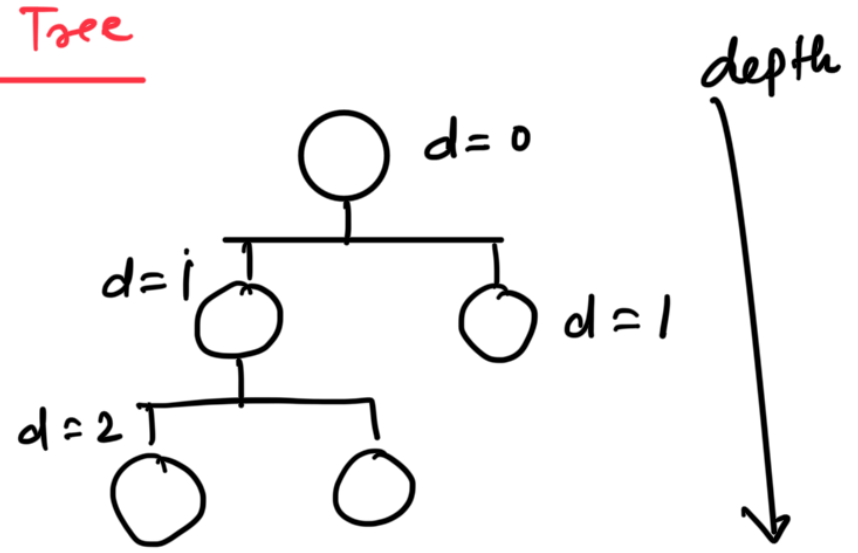
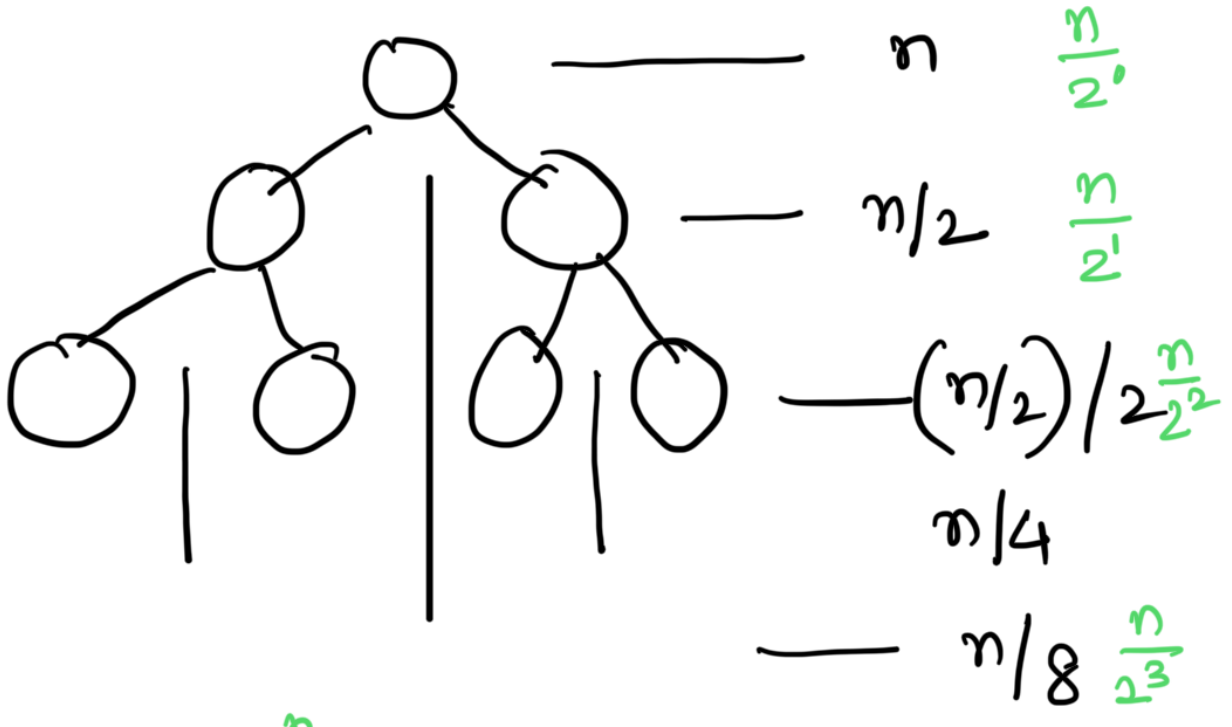
Balanced Binary Tree

- A binary tree is balanced if the height of the tree is $O(\log n)$



where n = number of nodes



BT (height) = $\log n$

Some Special Trees

1. Binary Search Tree → prepare really well
2. AVL Tree
3. Red-Black Tree } → Get an idea these trees
4. B Tree
5. B^+ Tree } → Db ⇒ Database

Binary Search Tree: (BST)

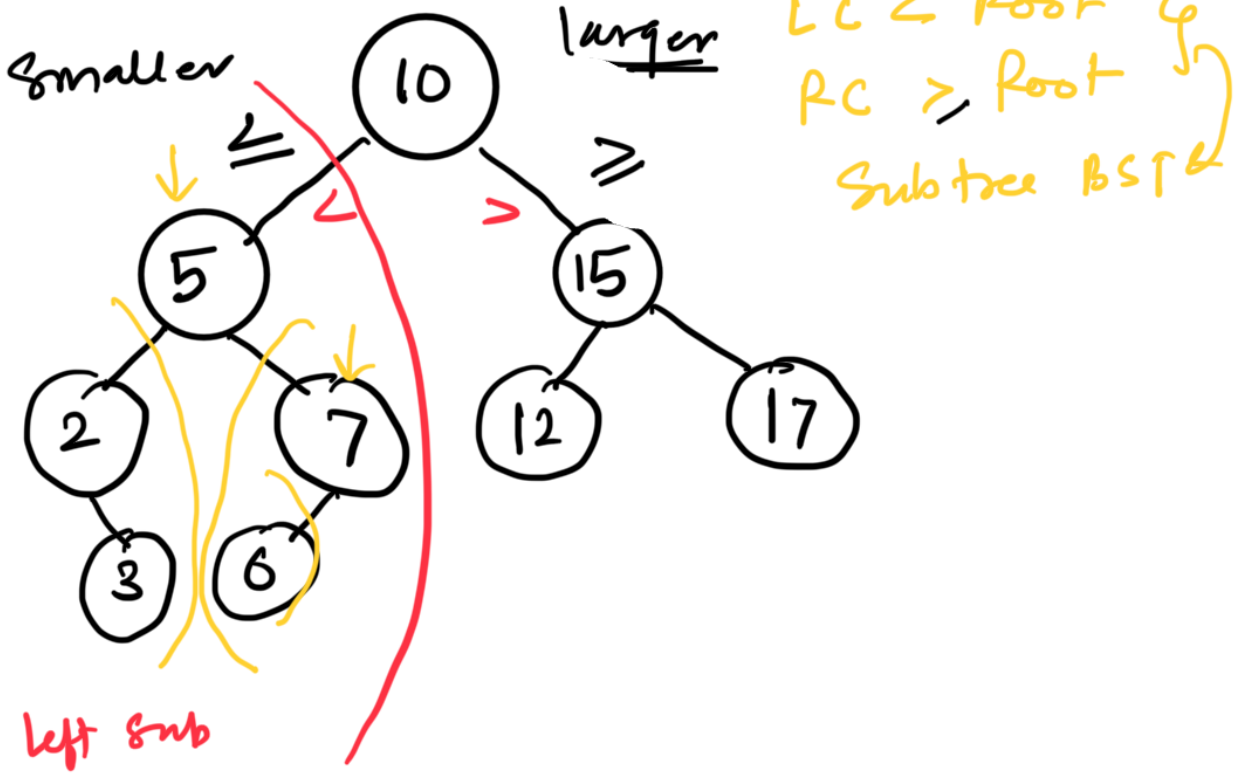
- A binary tree data structure in which each node has a key and satisfies the following properties

i) Left subtree of a node contains only nodes

with keys less than the root node keys.

ii) Right subtree of a node contains only nodes with keys greater than the root node key.

iii) Both left and right subtree must also be a binary search tree.



Properties of BST -

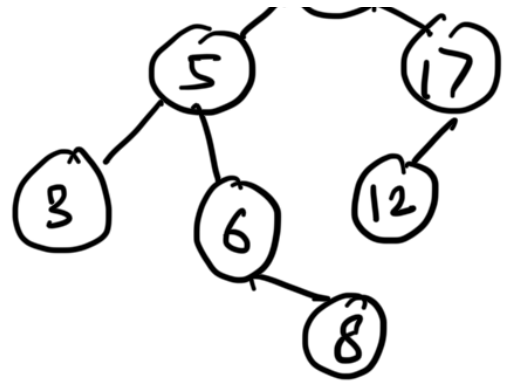
1. Binary Tree Structure -
   - Each node has at most 2 children

2. Ordering Property -
   - Left subtree ≤ Root
   - Right subtree ≥ Root

3. Search Property -
   - Search operation is the efficient with time complexity directly proportional to the height of the tree.

4. In-Order Traversal -

In Order - 3, 5, 6, 8, 10, 12, 17 → Ascending Order



- In Order traversal of BST results into ascending sorted order

5. Efficiency -

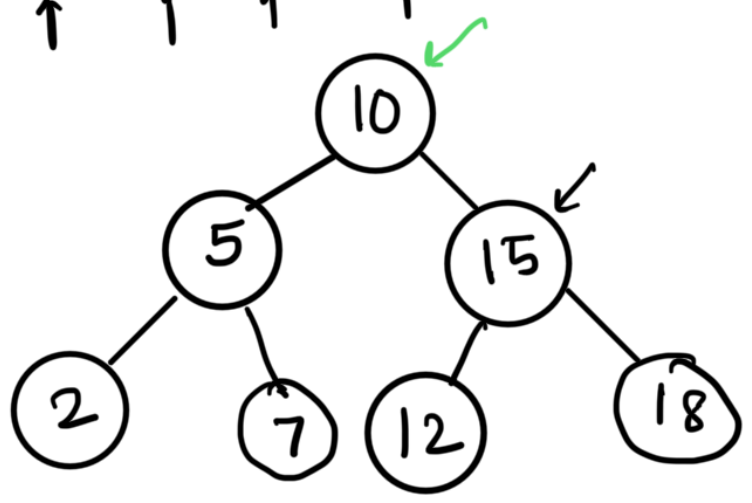
Operations like insertion & deletetion & search

Best case $O(\log n)$ → when tree is balanced

Worst case - $O(n)$ → when tree is skewed

Ex: 10, 5, 15, 12, 18, 2, 7 (Root)

Insert



10 < 18 R

15 > 18 & R

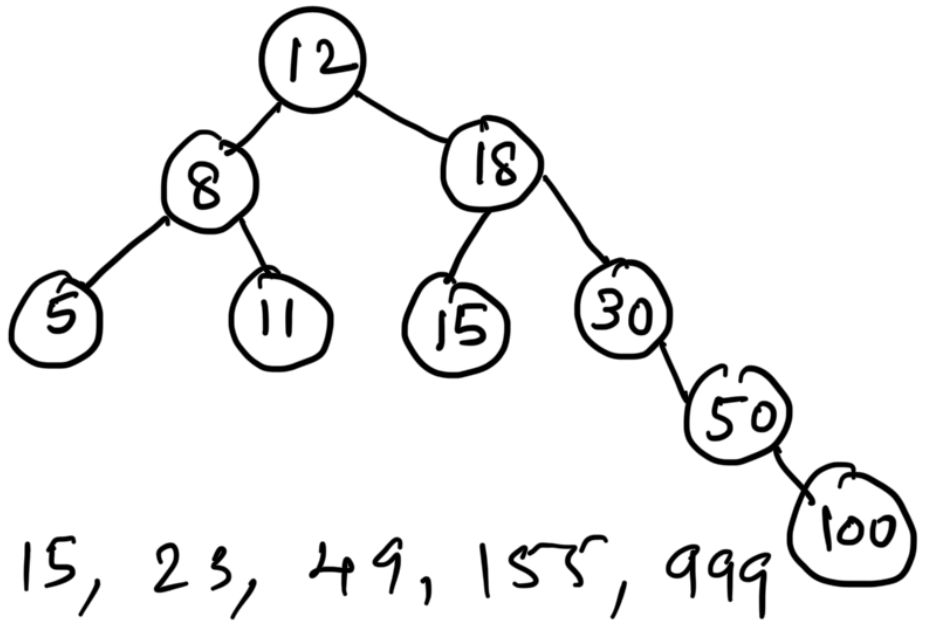
2 < 10 L

2 < 5 L

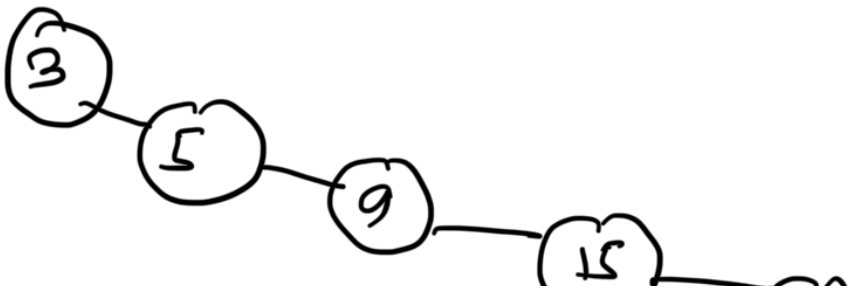
7 < 10 L

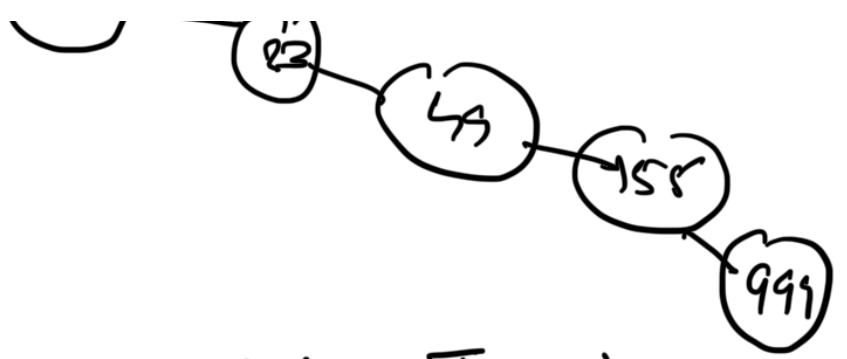
5 < 7 R

12, 8, 18, 5, 11, 15, 30, 50, 100



3, 5, 9, 15, 23, 49, 155, 999





Drawback of BST:

In order traversal sorted → Tree is growing only one direction

Result ⇒ Unbalanced Tree (BST)

Solution ⇒ Unbalanced BST ⟶ Balanced BST

1. AVL Tree
2. Red-Black Tree

} Balanced Tree

Deletion Operation:-

Imp

Case 1:- Deleting a leaf node

- Simply remove the node

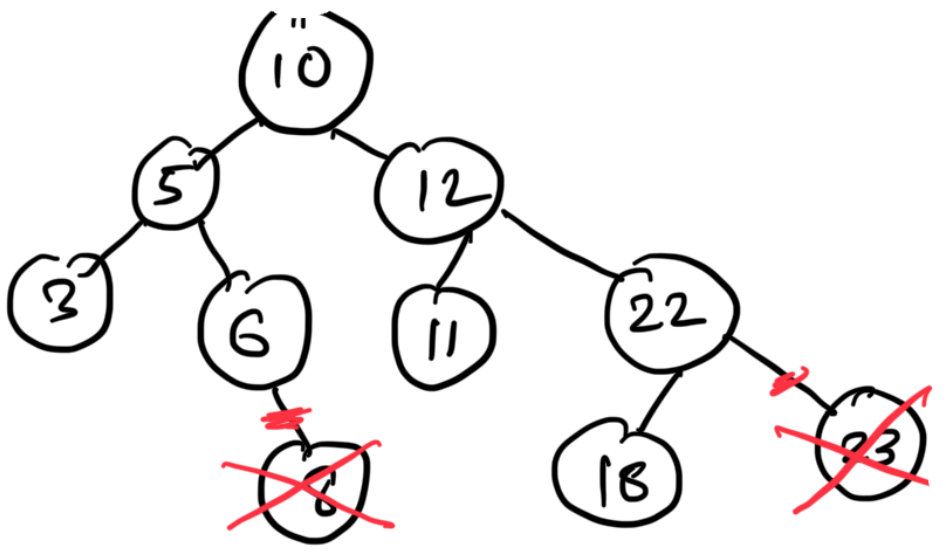
Case 2:- Deleting a node with one child

- Replace the node with its child

Case 3: Deleting a node with two children

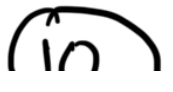
Special Interview - Replace the node with its inorder successor (Smallest node in the right subtree)

Case 1

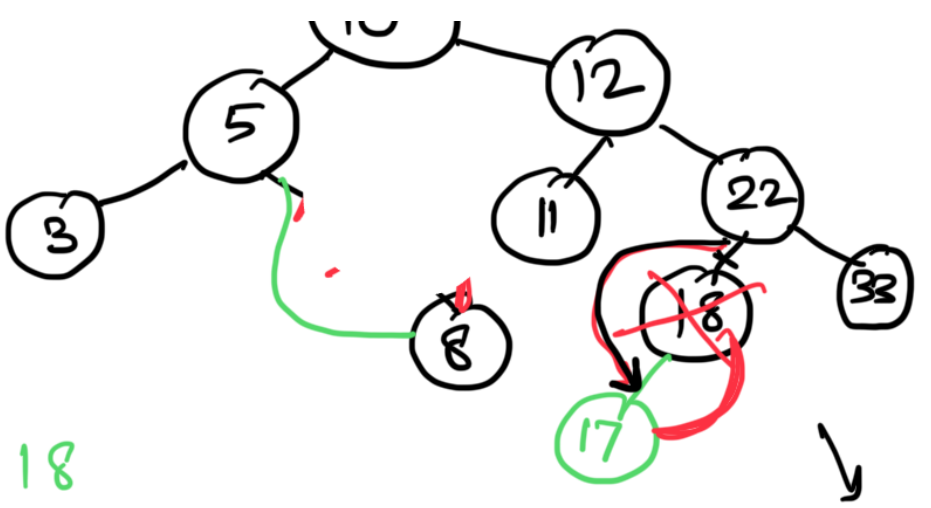


Delete - 8, 23

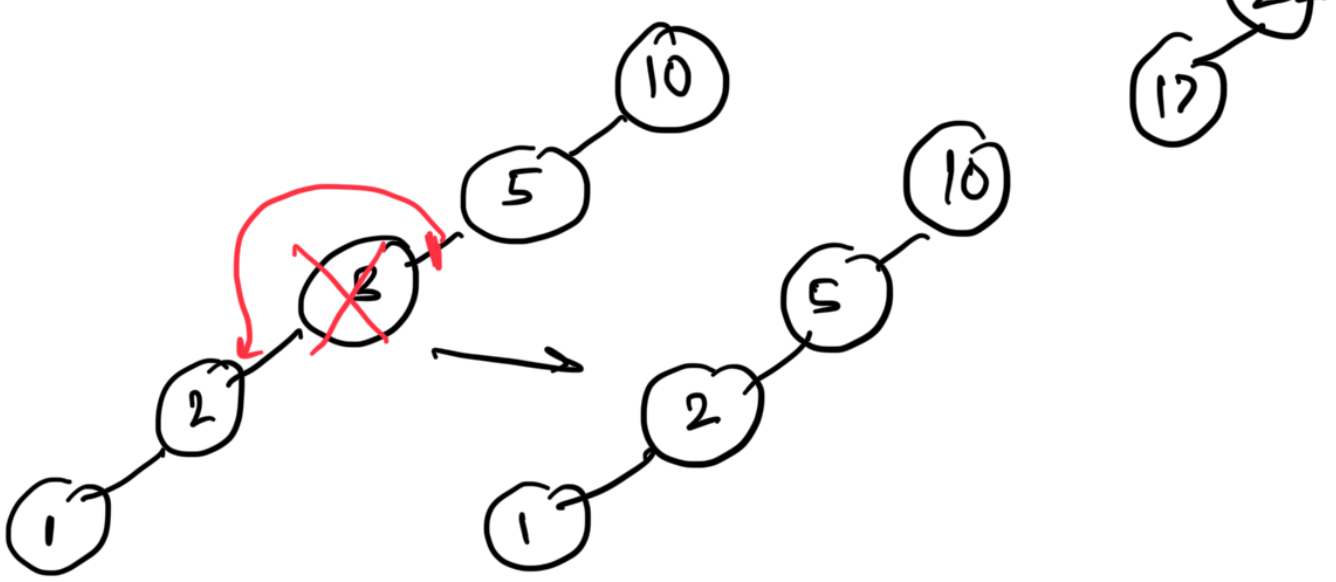
Case 2:



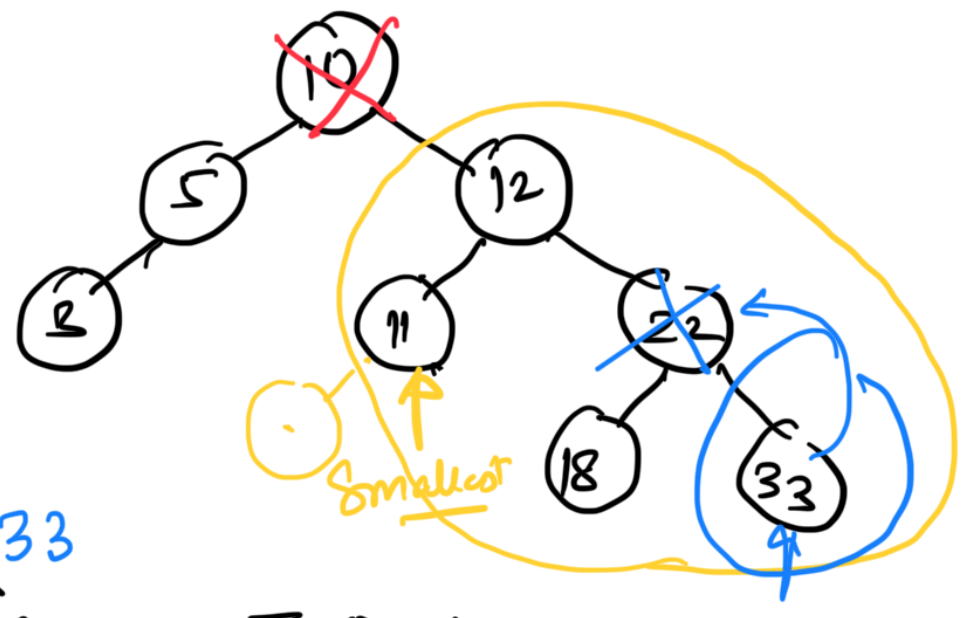
Case 3:



Delete : 6, 18



Case 3



Delete: 10 → 33

Root, LC, RC

InOrder

3, 5, 10, 11, 12, 18, 22, 33

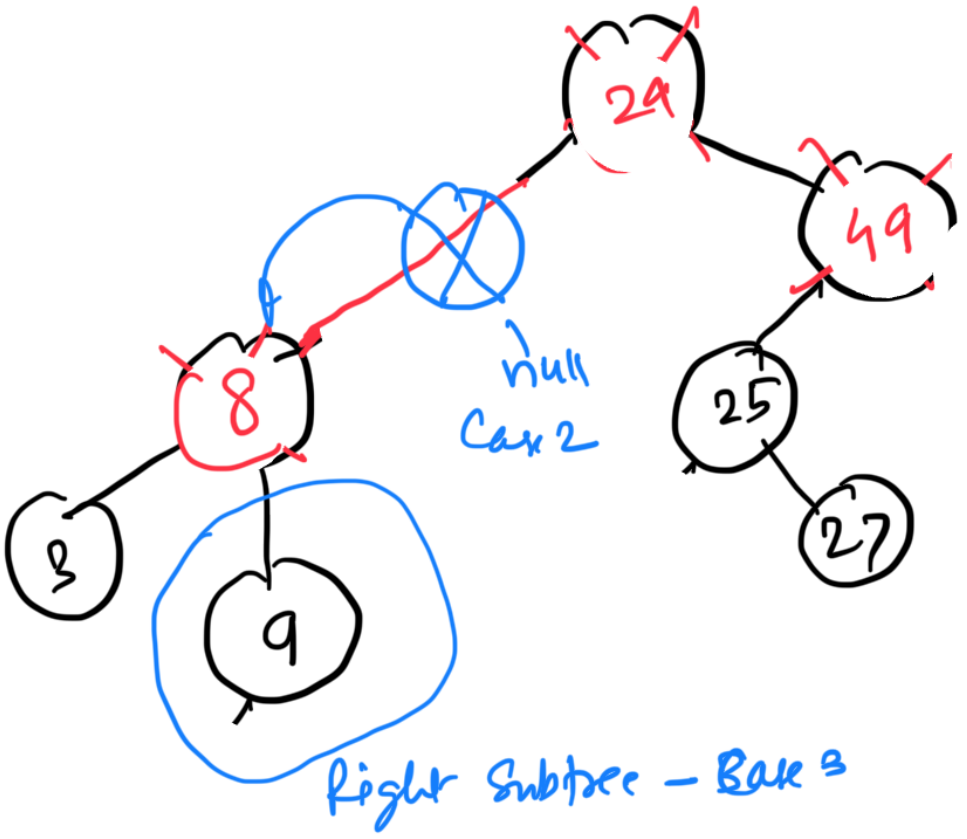
Step 1

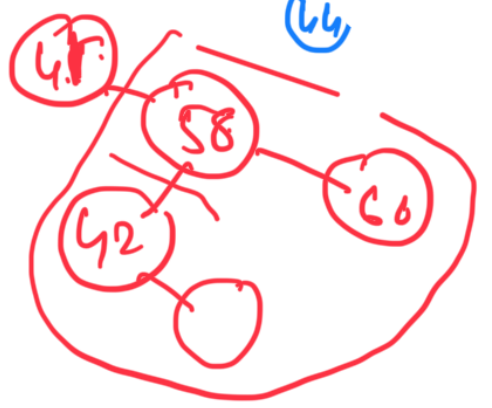
Step 2 → Successor

By

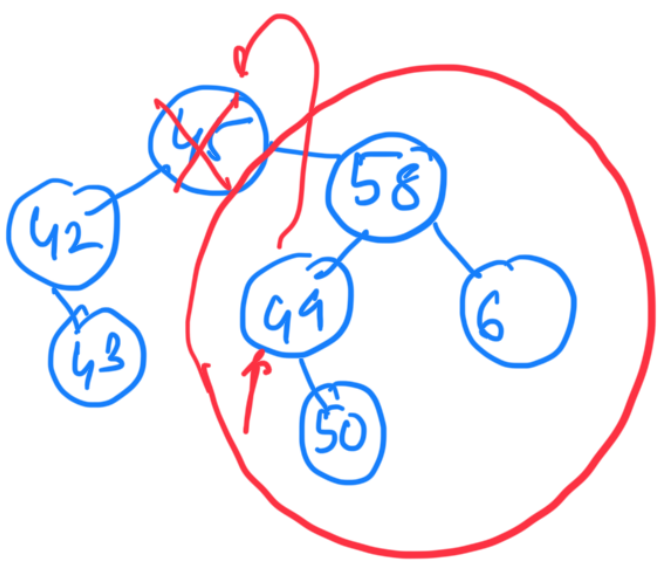
Any side duplicate

Case with duplicate value





Right Subtree – Base 3



2, 3, 5

Inorder → Successor