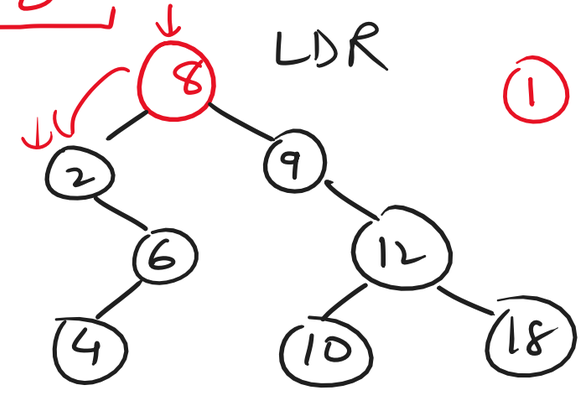
Introduction to Search Trees: Binary Search Tree

This is a special kind of Binary Tree where each node of the tree follows a very unique property:

⇒ L < Node < Right :→ int arr = [8, 2, 6, 9, 4, 12, 18]



(I) insert( )
(II) search( )
(III) delete( )

In-order: → 2, 4, 6, 8, 9, 10, 12, 18

① When we search for any data, we skip one part/subtree of the full tree.
∴ TC = O(log n)

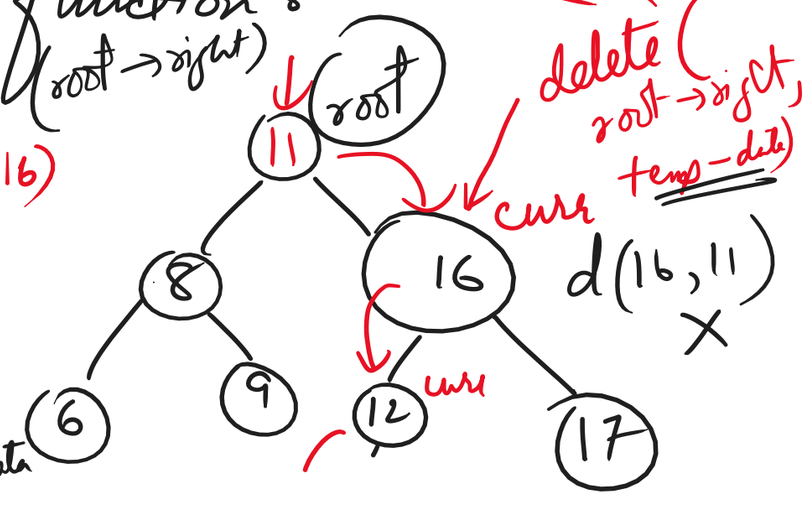
Three important checks for the delete function: L<N<R

Only 1 children

root→right == null
Temp = r.left
delete root
return temp;

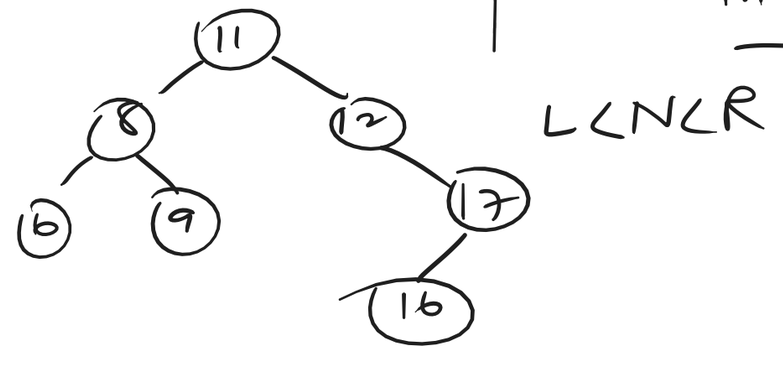
root→left == null
Temp = root.right
delete root
return temp

find min (root→right)
find min(16)
↳ 11
temp = 11
r→data = temp→data



In-order
6, 8, 9, 10, 11, 12, 16, 17

In-order Successor:

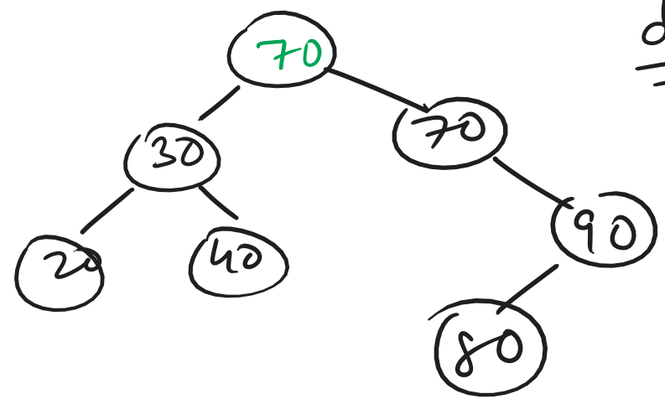


L<N<R

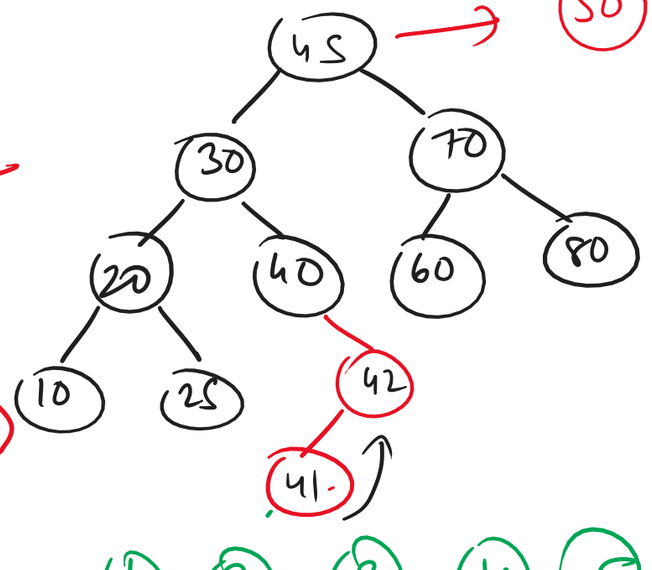
{ Red Black Tree
{ AVL Tree

50, 30, 20, 40, 70, 90, 80

delete 50



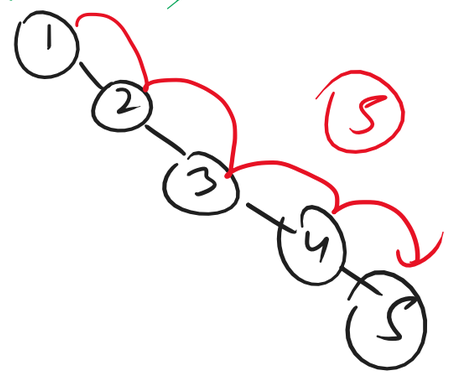
RB Tree
AV Tree
↗ logN



(50) delete (STL)

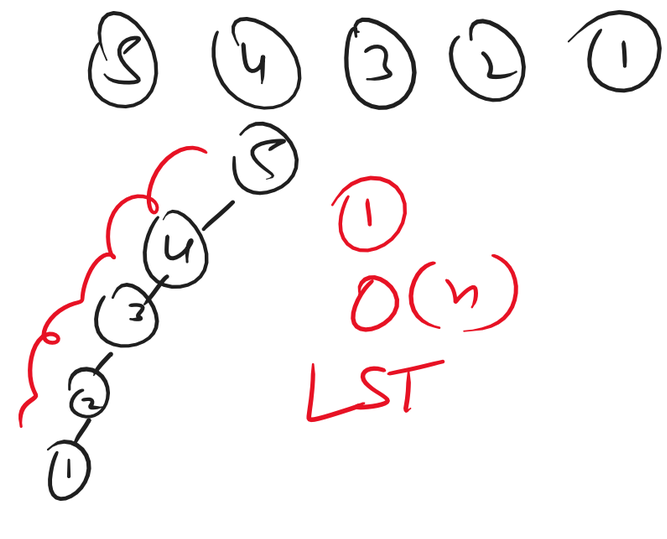
* Less Restructuring
↳ In Order Successor
* More Restructuring
↳ In Order Predecessor

Strictly Balanced BST →



RST

① ② ③ ④ ⑤
⑤ O(n) Skewed Trees



① O(n)
LST