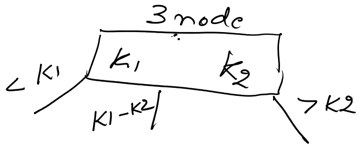
2-3- Trees

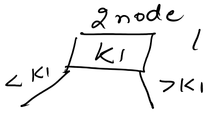
- Each node has 1 or 2 keys
- Pointer to child nodes



$K_1 < K_2$

- Each internal node must have 2 or 3 children

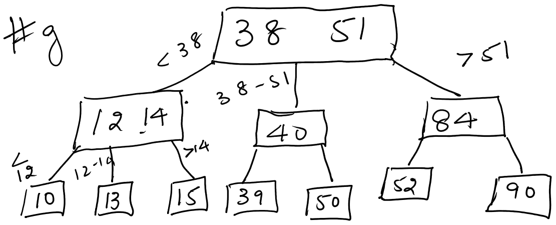
2 key values
-3 children



1 key value
2 children

- Keys are ordered from small to large
  $\Rightarrow k_1 < k_2$
- height of 2-3 tree varies from $O(\log_3 n)$ to $O(\log_2 n)$

#g



□ INSERTION 2-3 Search Tree [BOTTOM UP]

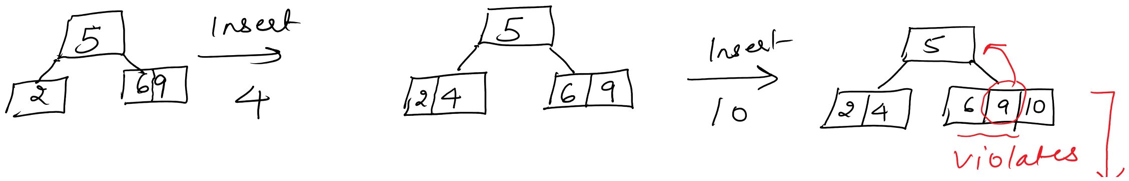
1. Find the correct "leaf node" to insert the new element
2. Insert the new element in the leaf node

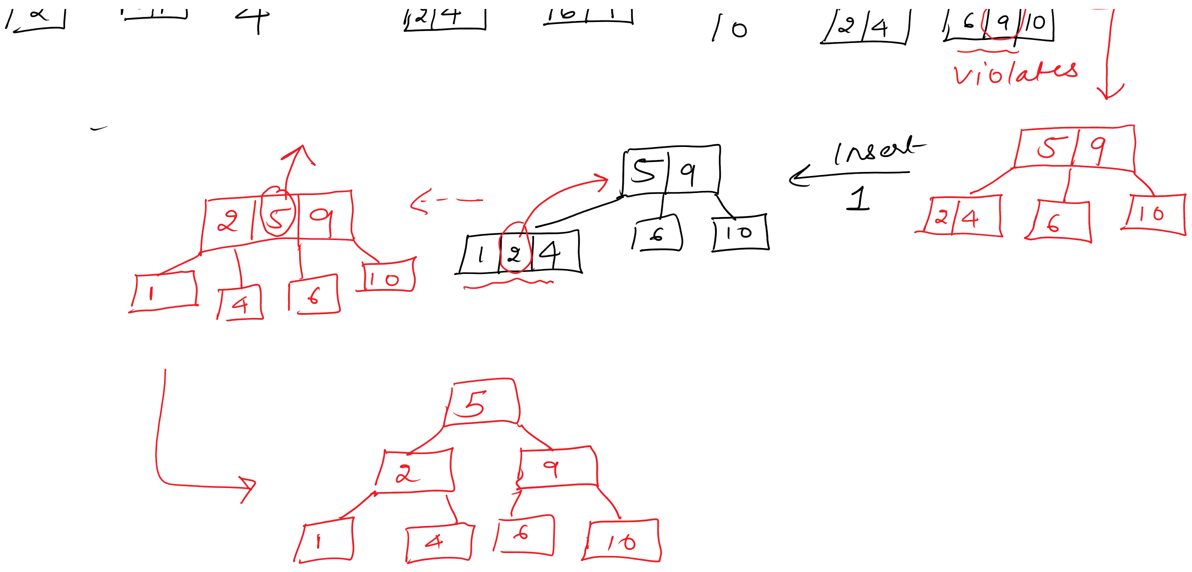
i) If there are 3 values in that node, split that node and push the middle value to the parent

- If the parent node has 3 values, split the parent & push middle value to parent's parent node.

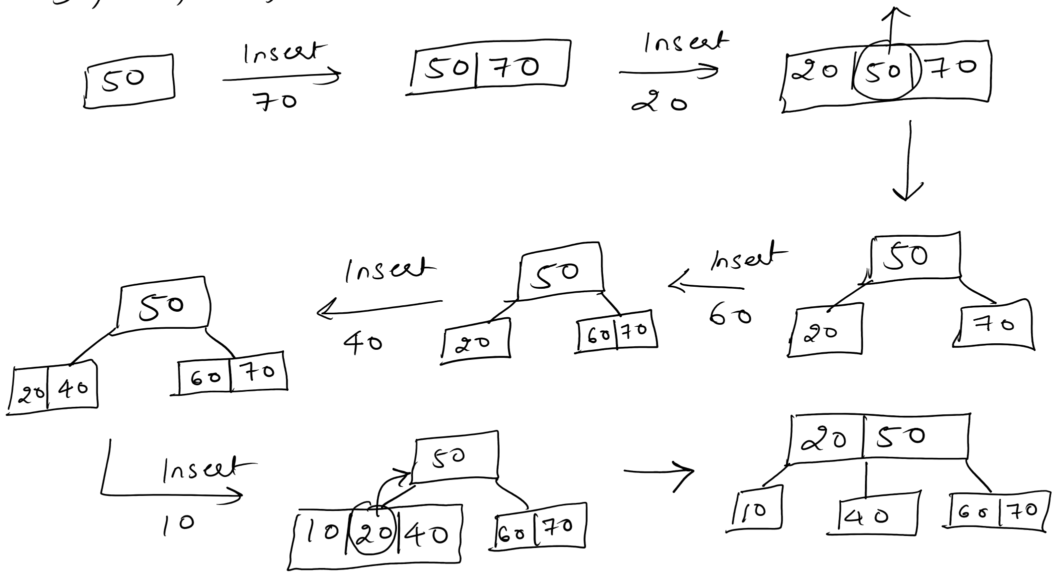
✗ At some point, the root node will eventually split & the height of the tree will increase by 1.

#g

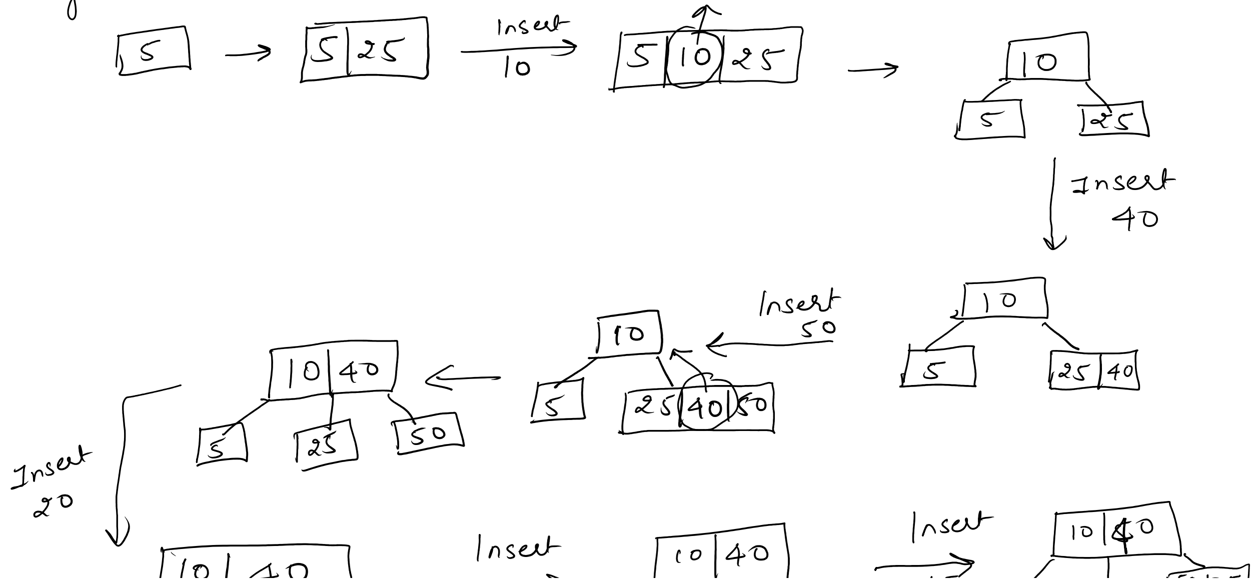


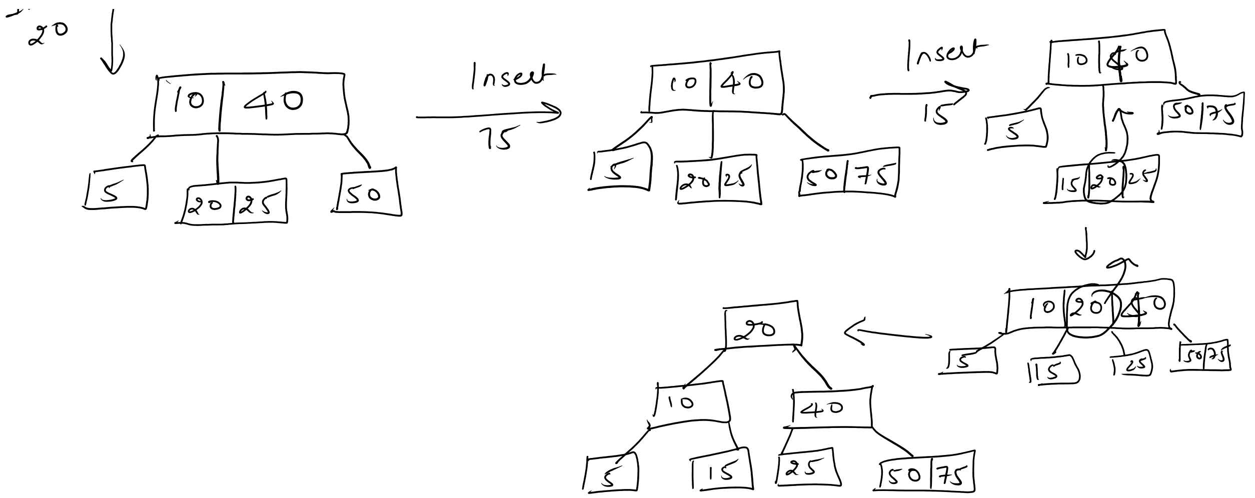


#q 50, 70, 20, 60, 40, 10



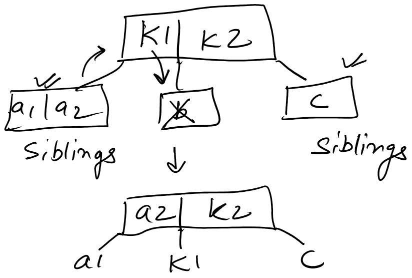
#q 5, 25, 10, 40, 50, 20, 75, 15, 45, 60



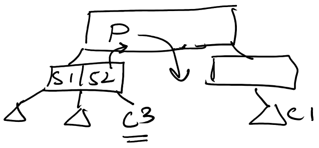


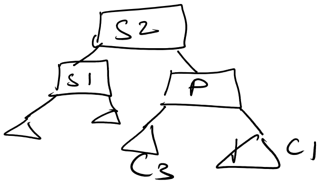
1. DELETION

1. Find the key that needs to be deleted
2. If non-leaf node
   - Replace with the predecessor
   - Replace with the succesor, if no predecessor
3. If leaf node
   i) left/right sibling has enough keys (>1)
   - donote by rotating with the parent



⚡ may need to take over sibling's child





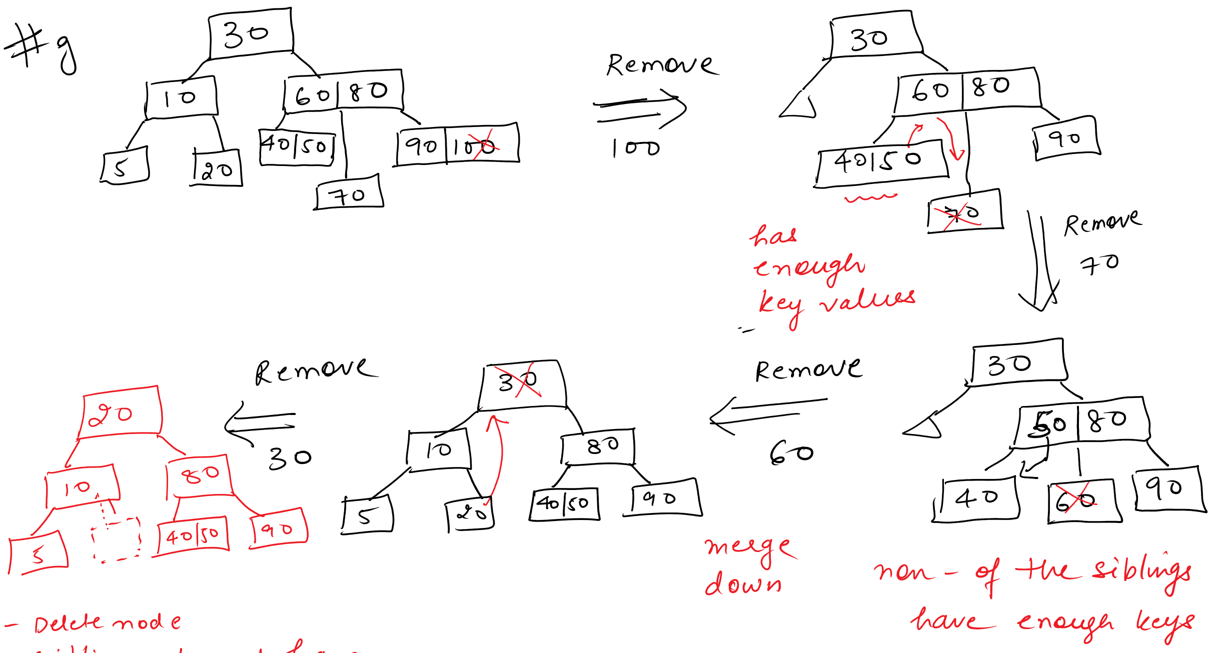
ii) Else

→ merge down (pull the parent down)

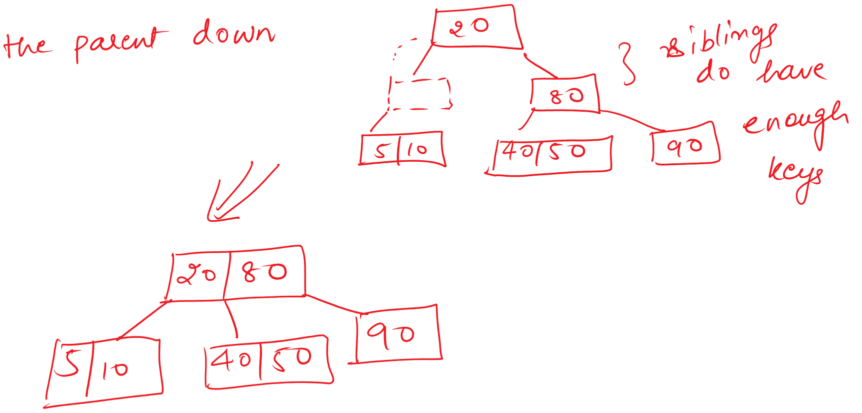
→ If parent is underful after the merge

repeat step 3, recursively.

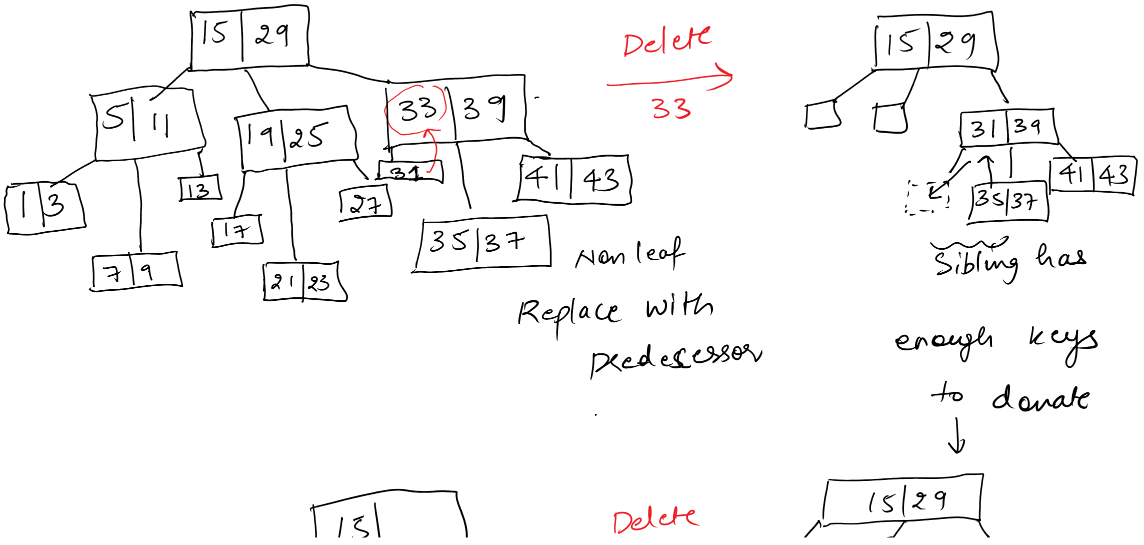
#g

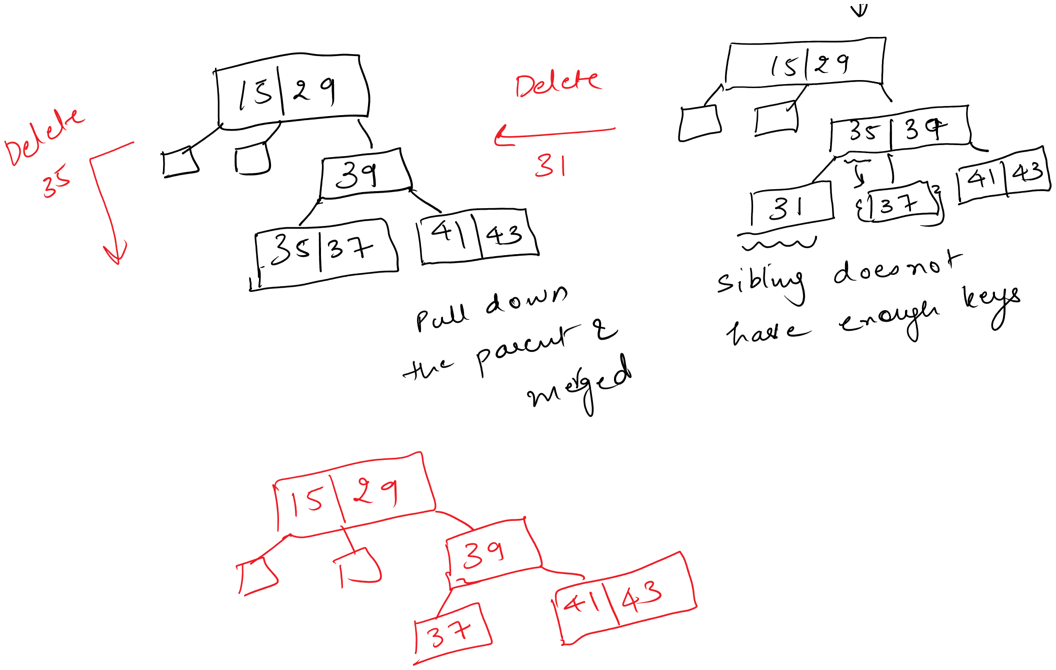


- Delete node
- Siblings do not have enough values
- merge down, pull the parent down



#g





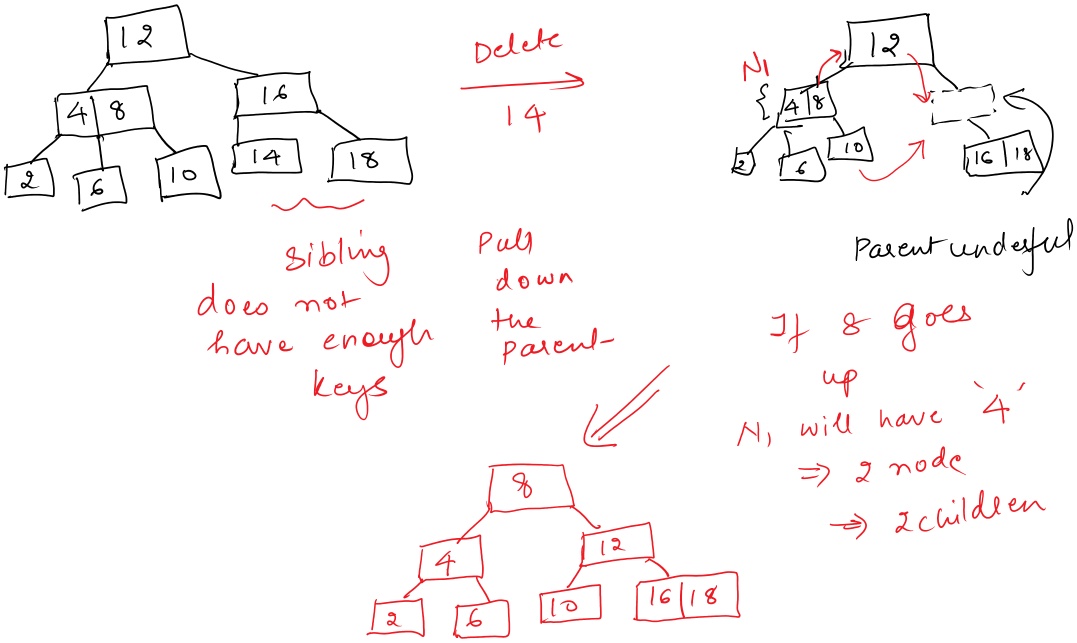
Delete 31

sibling does not have enough keys

Pull down the parent & merged

Delete 35

#9



Delete 14

sibling does not have enough keys

Pull down the parent

Parent underful

If 8 goes up N1 will have '4'
⇒ 2 node
⇒ 2children