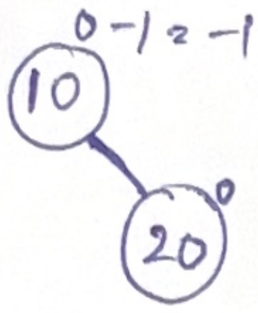
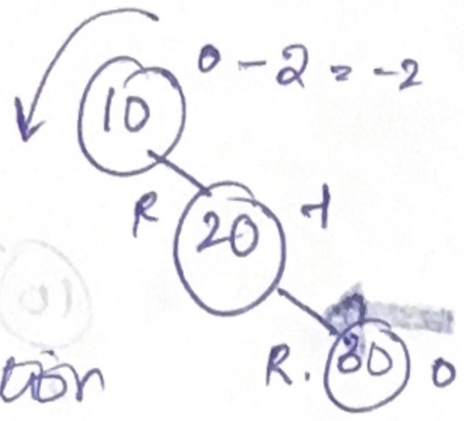
12-09-2025

AVL tree : Example :

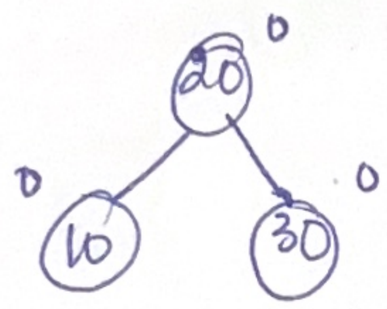
Binary search tree + (Load balance) = Left subtree height - right subtree height

Q. (10) (20) (30)

1. 

2. 

Left rotation

3. 

AVL ← not balanced :

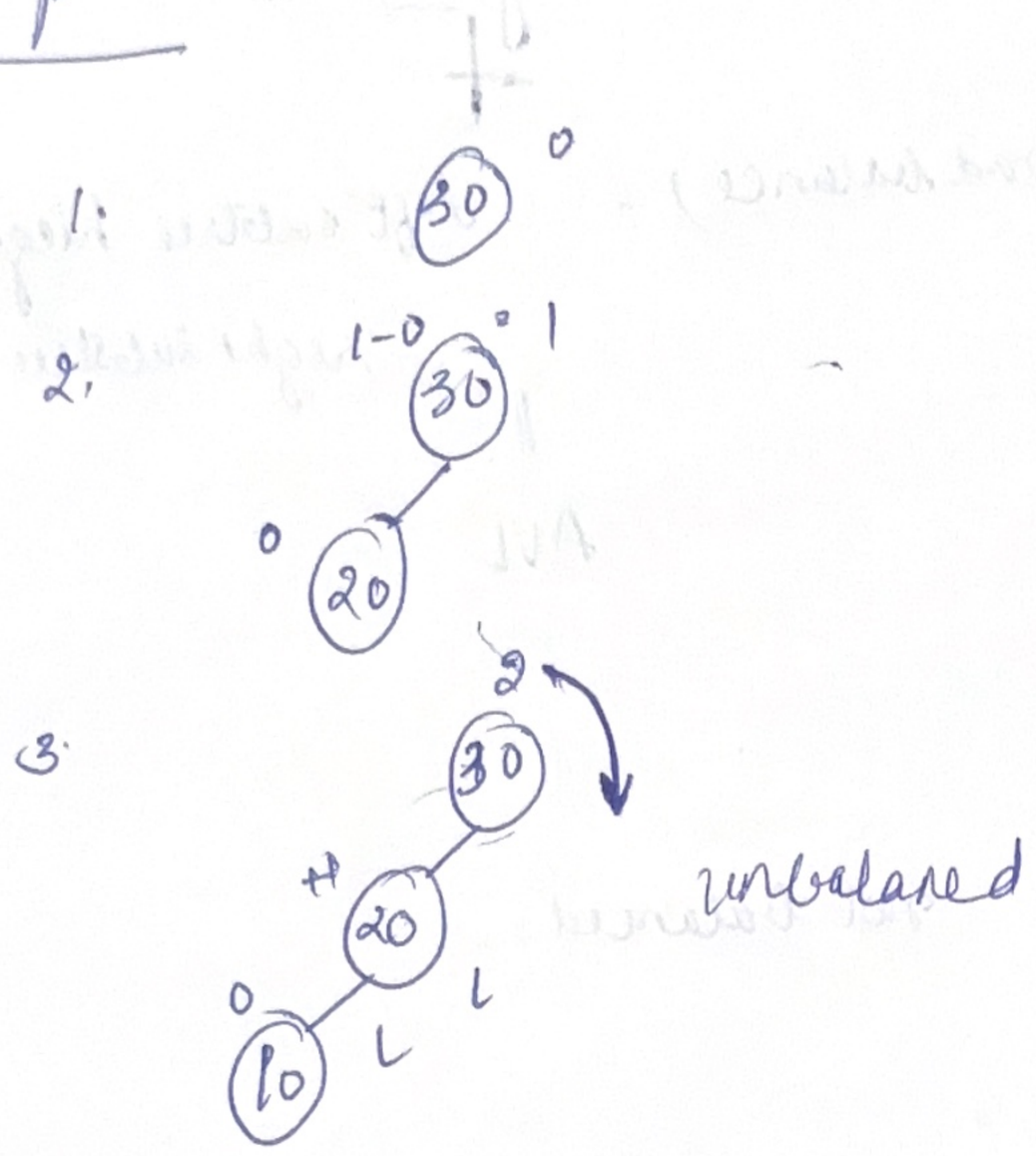
Balanced ✓

AVL tree :

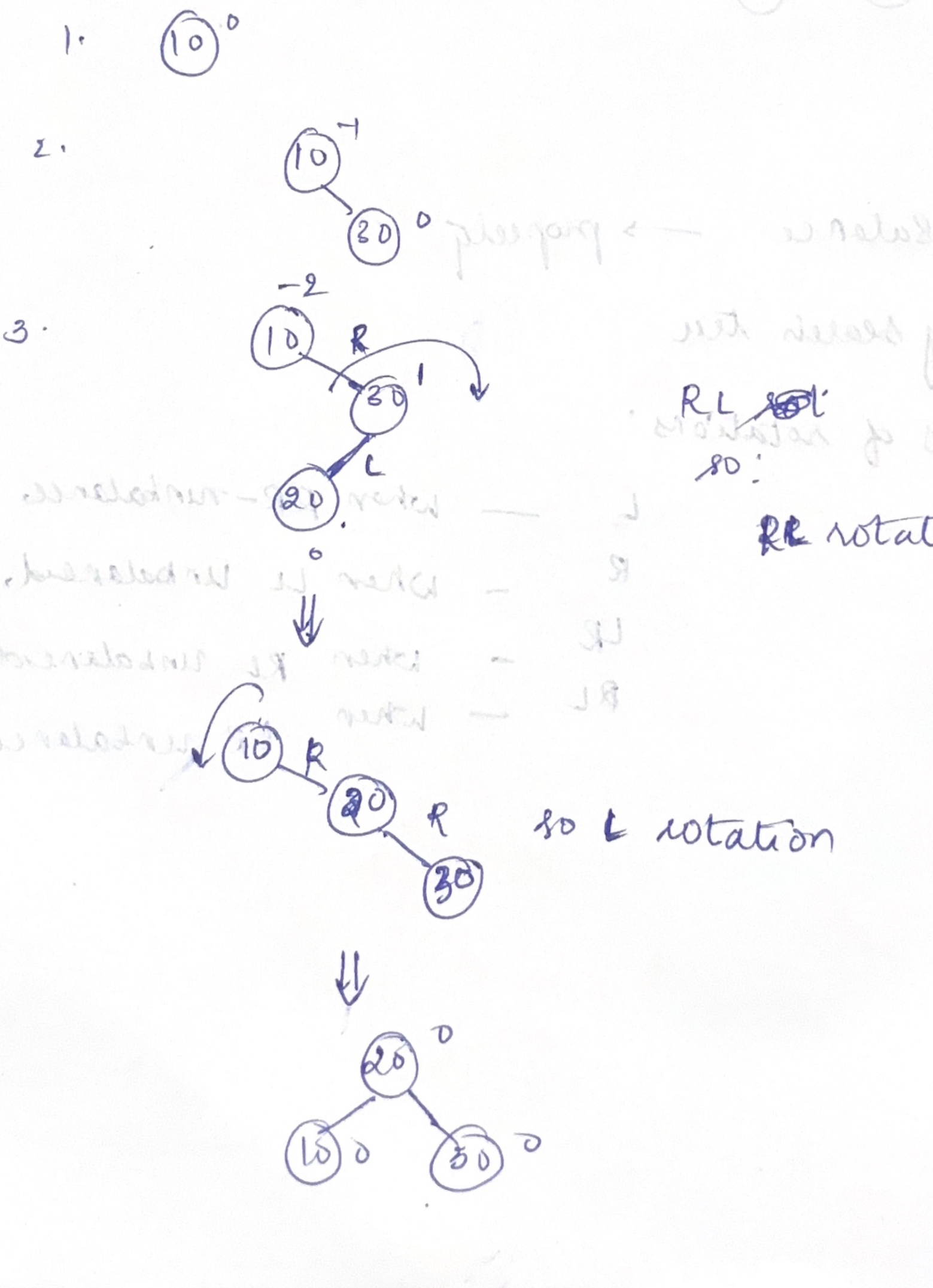
- Load Balance ⟶ property
- Binary search tree
- Types of rotations :

L — when RR-unbalance
R — when LL unbalanced.
LR — when RL unbalanced
RL — when LR unbalanced.

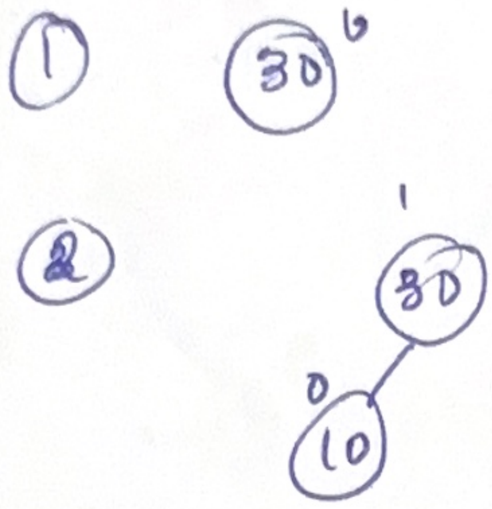
Example: 30, 20, 10



example: 10, 30, 20



example: 80, 10, 20



Red Black tree : Rules :

Insertion :

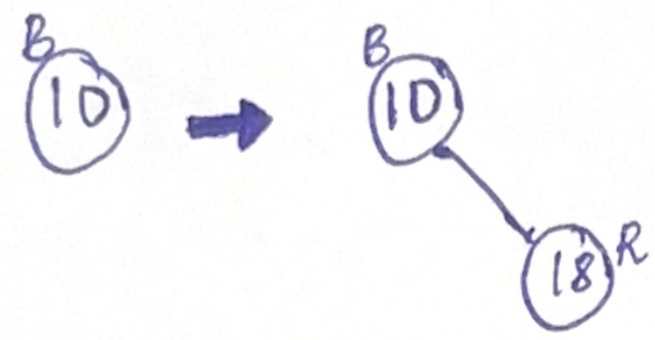
1. If tree is empty, creat new node as root node as color Black.
2. If tree is not empty, creat new new node as leaf node, with color red.
3. If parent of new node is black, the exit
4. If parent of Red is red, then check the color of parents sibiling of new node.
   (a) If color is Black or null, then do suitable rotating and recolours.
   (b) If color is red, then recolour and also check parent & sibling

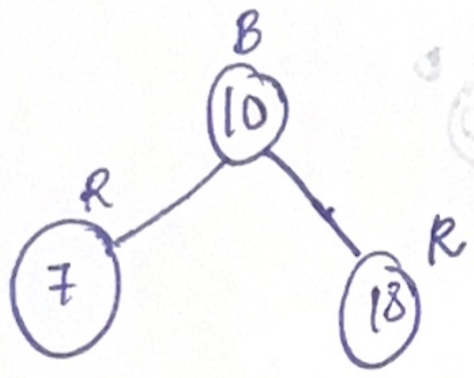
if parent's parent of new node is not root node, then recolor it & recheck.

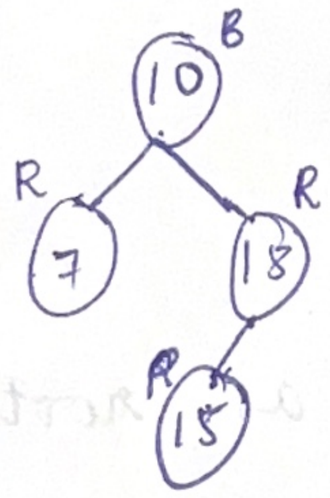
15-09-2025:

Construct RB Tree:

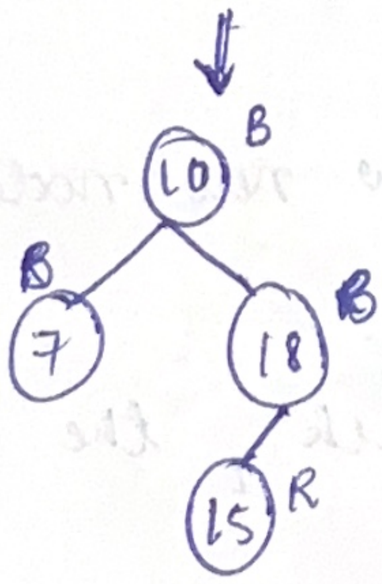
10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, 70

1. 

2. 

3. 

(Violated properties)

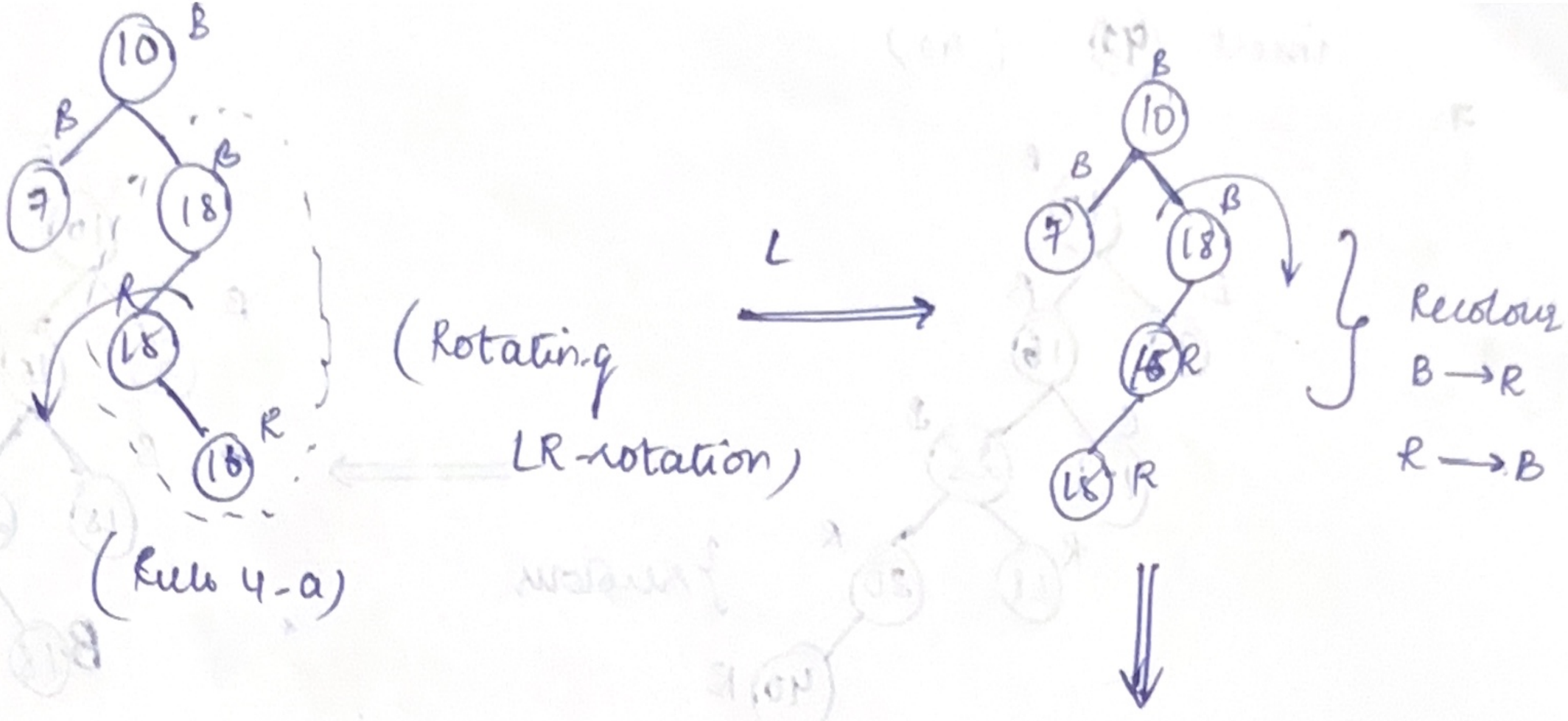


(Rule 4-a)

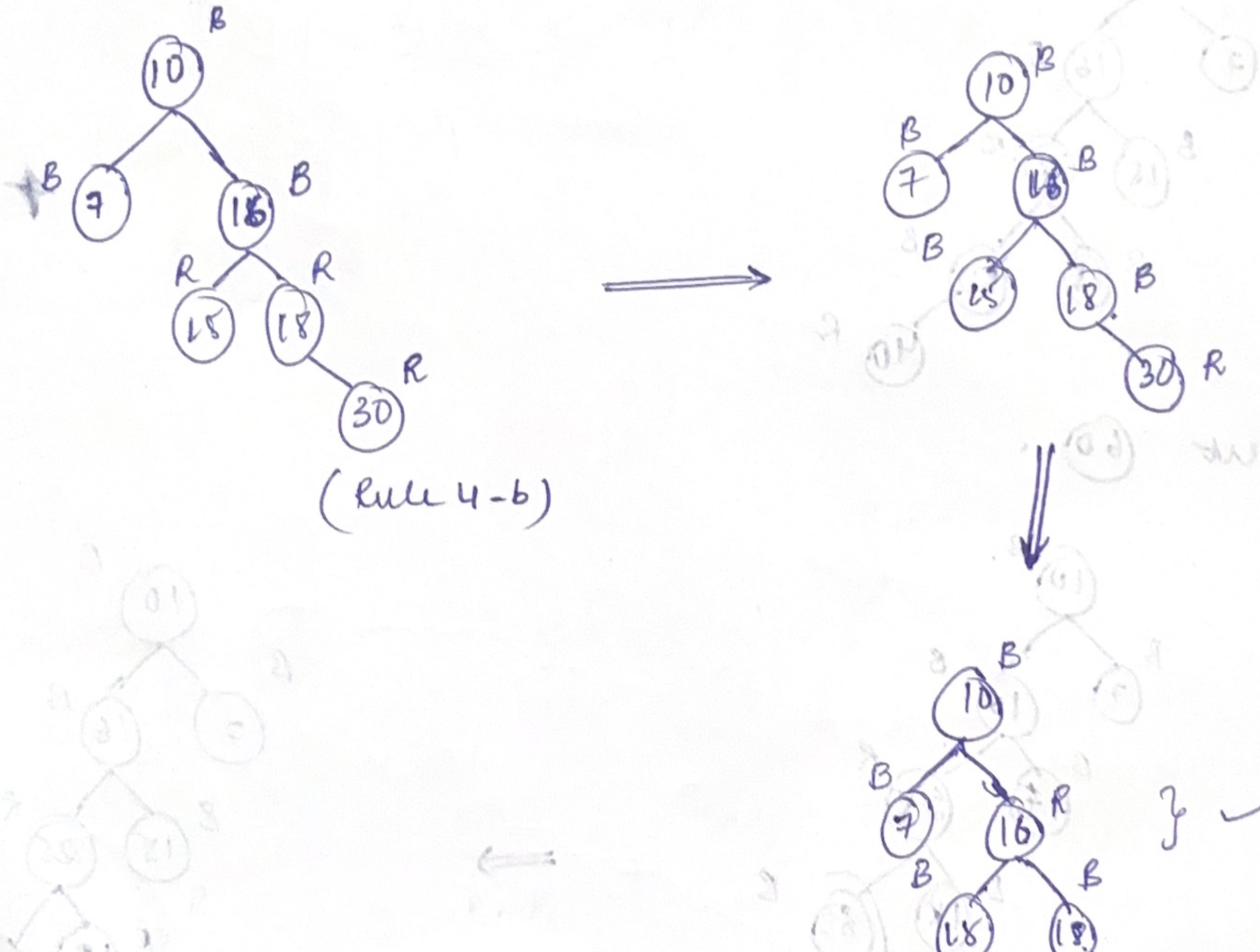
(Recolour both new nodes parent and parents sibling (uncle))

(new nodes Grandfather is Root, so exit)

4.


10 B
B
7
B
18
R
15
R
18
(Rule 4-a)
(Rotation
LR-rotation)
L
B
10
B
7
B
18
R
15
R
18
Recolour
B → R
R → B
10 B
B
7
B
18
R
15
R
18

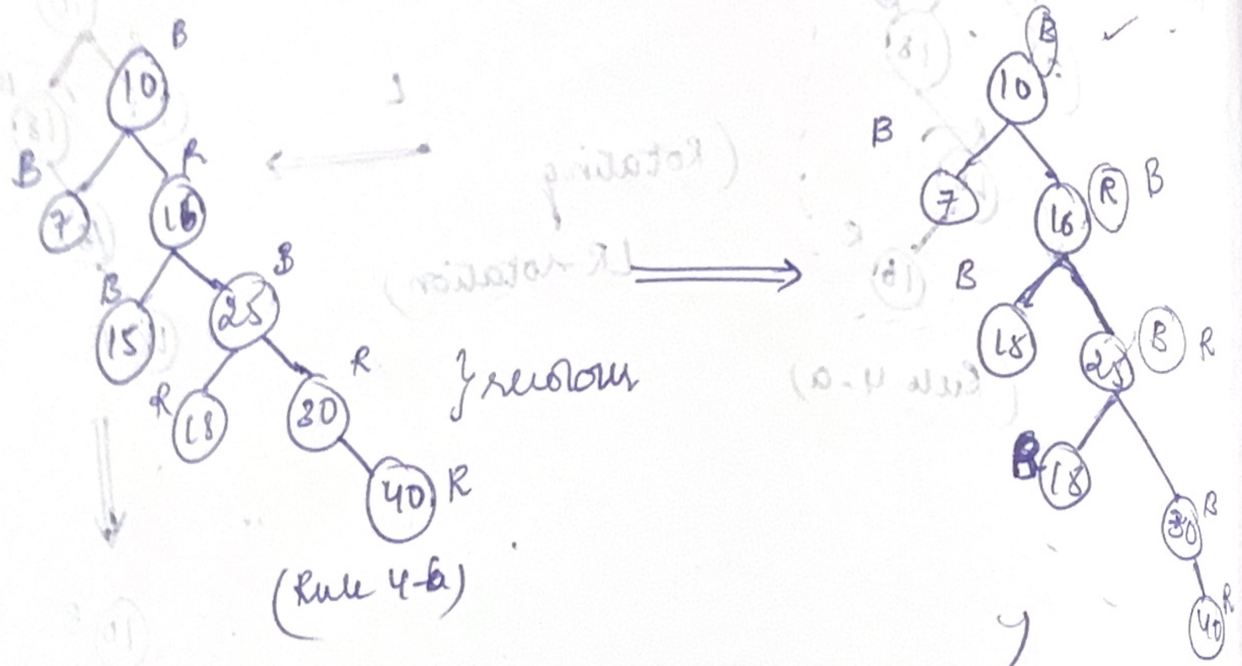
5.


10 B
B
7
B
18
R
15
R
18
30
R
(Rule 4-b)
10 B
B
7
B
18
B
15
18
B
30
R
(∵ 16 is
not
Root node)
B
10
B
7
R
16
B
B
18
18
30
R
16.

6.

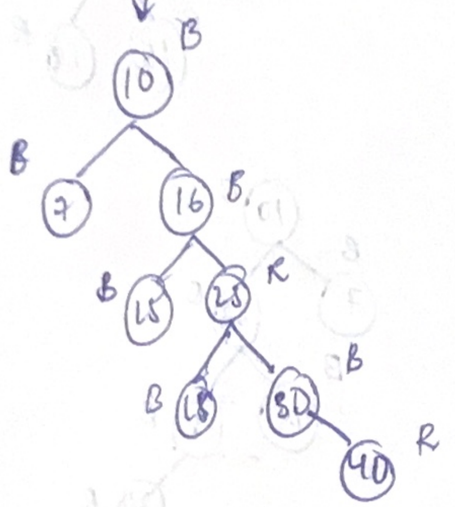
10 B
B
7
R
16
B
B
15
18
B
30
R
25
R
(Rule 4-a)
Recolour
After R
RL rotation.
10 B
B
7
R
16
B
15
R
18
25
B
30
R
10 B
B
7
R
16
B
B
15
25
R
R
18
30

insert (40) (40)

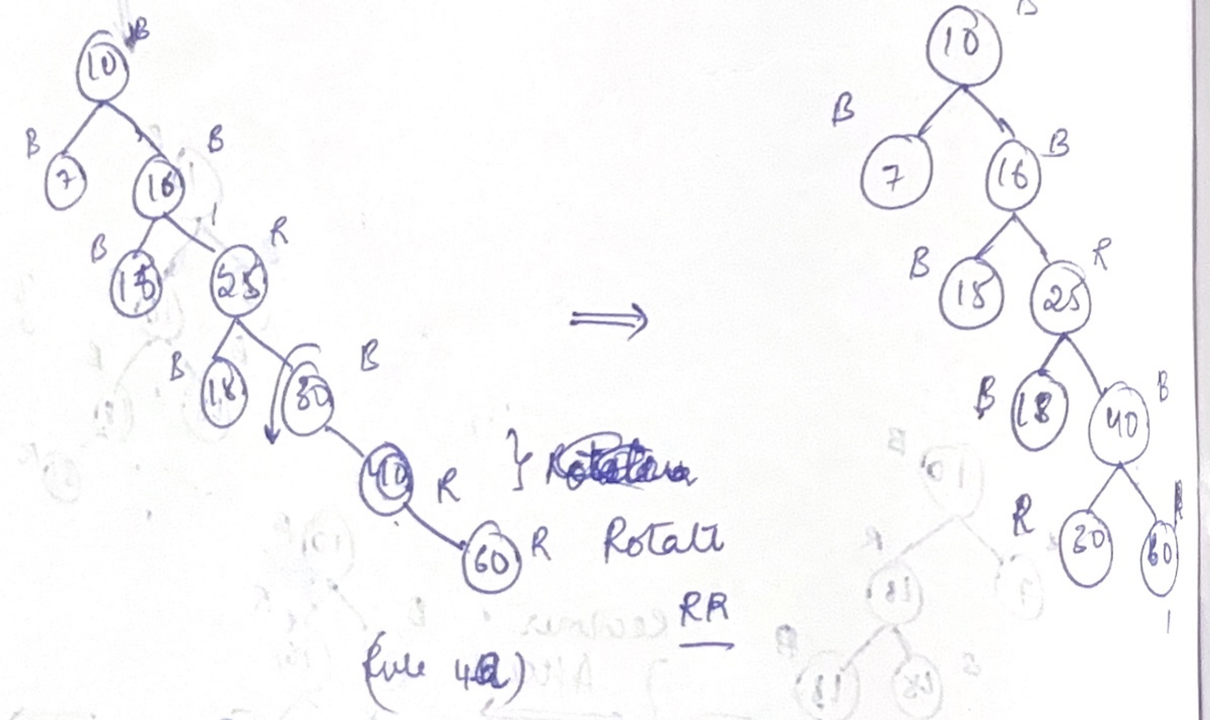


I LOVE YOU

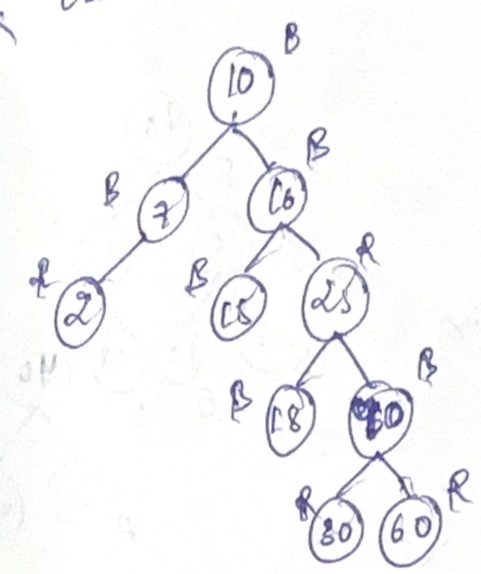
↳ POOKIE'S TUBELIGHT ✦

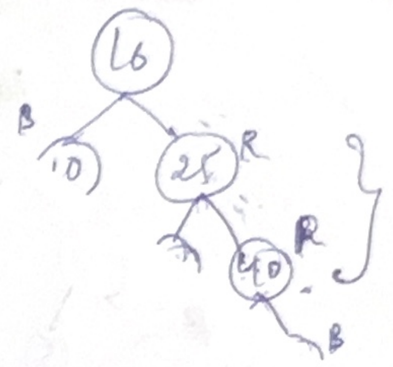


insert (60)

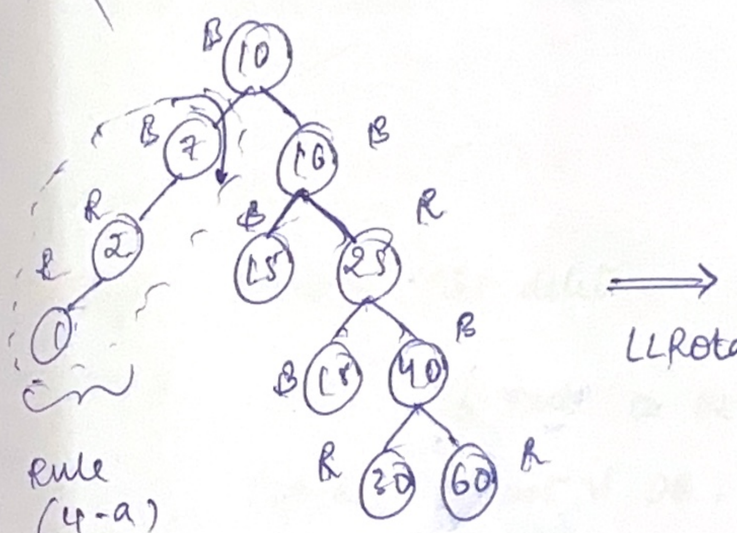


insert (2)

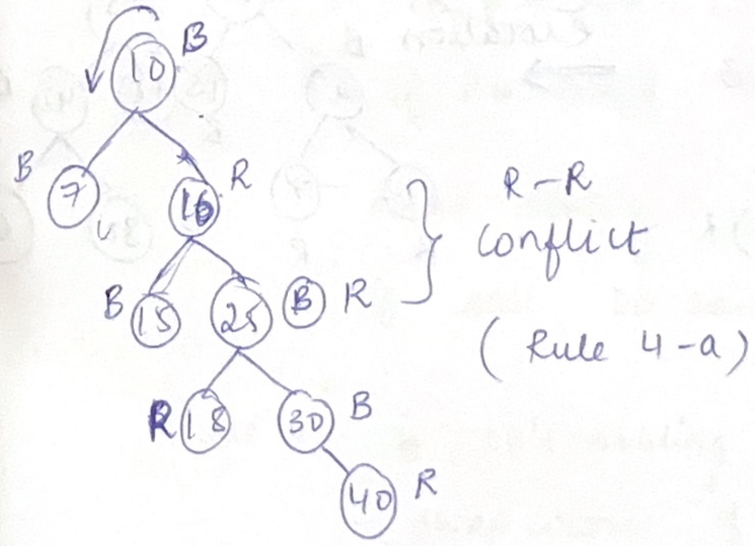




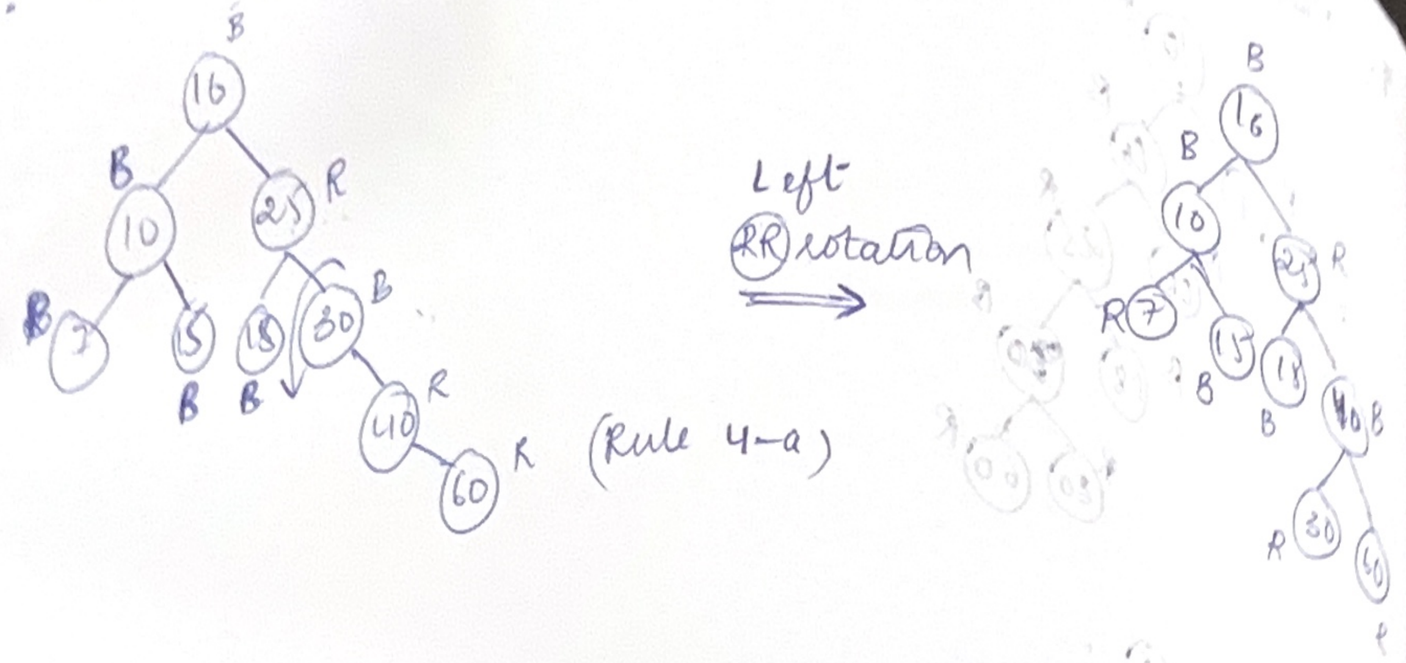
insert (1):



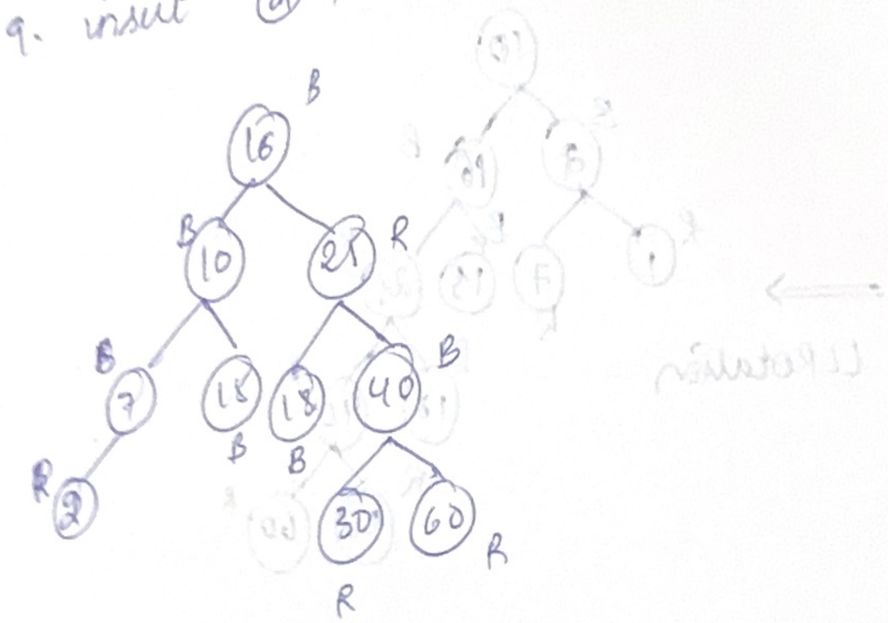
inserting (40)



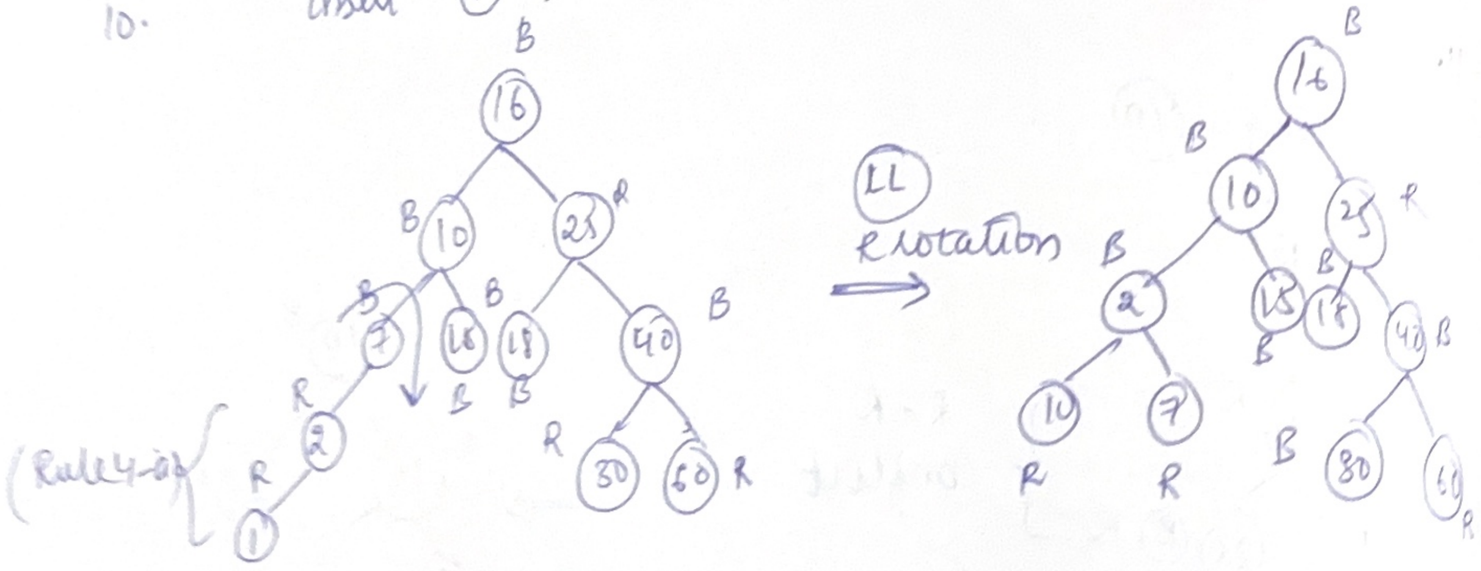
8.



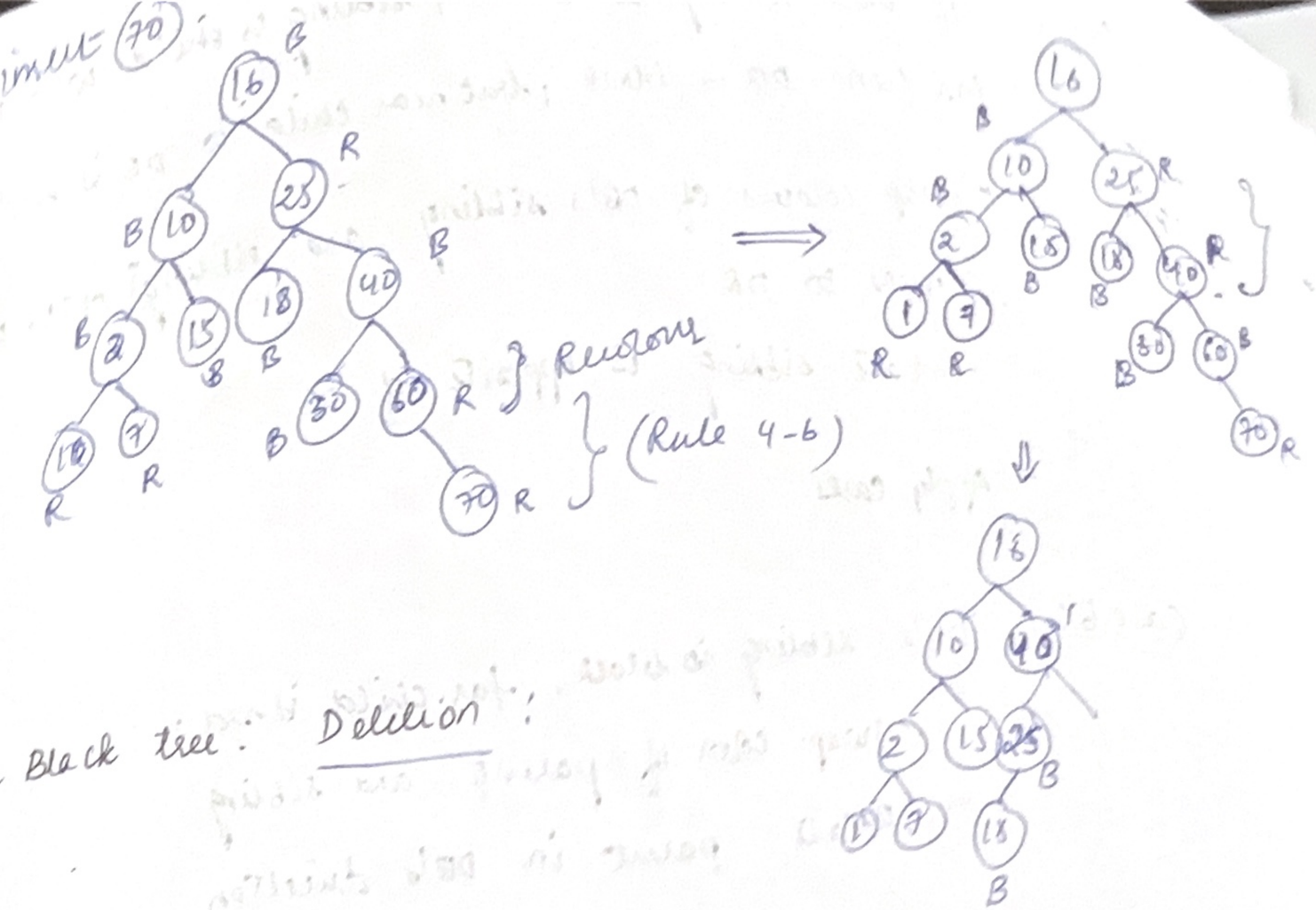
9. insert (9):



10. insert (1):



11. insert (70):



Red Black tree: Deletion:

15-09-2025:

step 1: perform BST deletion

step 2: case 1: If node to be deleted is red, delete it & exit

case 2: If root is DB, just remove DB

case 3: If DB's sibling is black, & both its children are also black.

- remove DB
- Add black to its parent (P)
  - If P is red, it becomes black
  - if P is black, it becomes DB
- make sibling red.
- if still DB exists, apply other cases.

Case 4: if DB's sibling is red

- swap colours of parent & its sibling
- Rotate parent in DB direction
- Reapply cases.

Case 5: If DB's sibling is black, sibling's child who is far from DB is black, but near child to DB is red

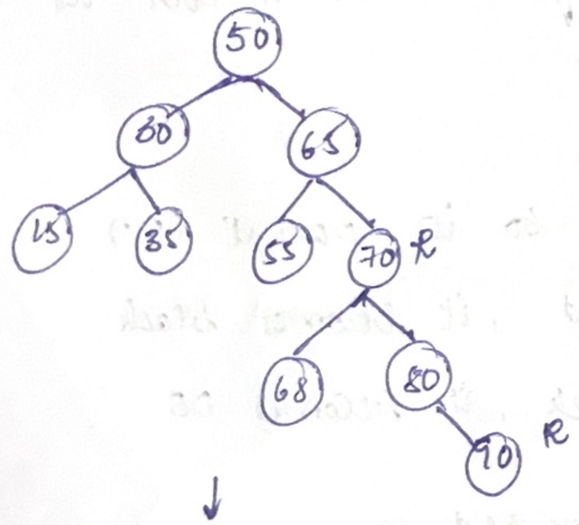
- swap colours of DB's sibling and sibling's child who is near to DB
- Rotate sibling is opposite of DB
- Apply cases

Case 6: DB's sibling is black, far child is red.

- swap color of parent and sibling
- Rotate parent in DB's direction
- Remove
- Change color of red child to Black.

19-09-2025:

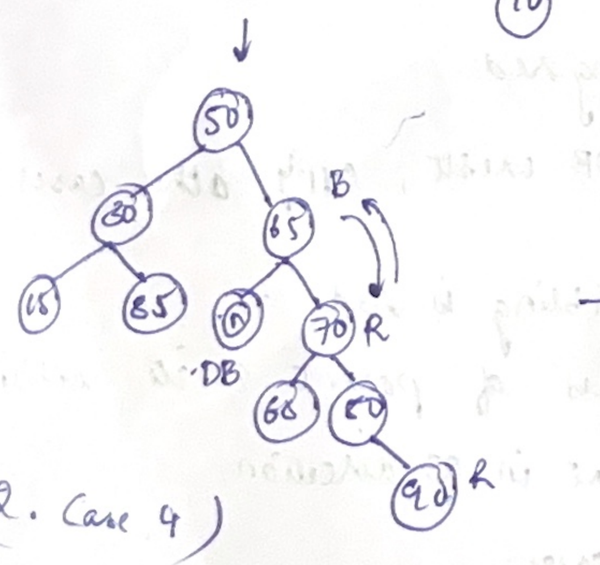
Example:



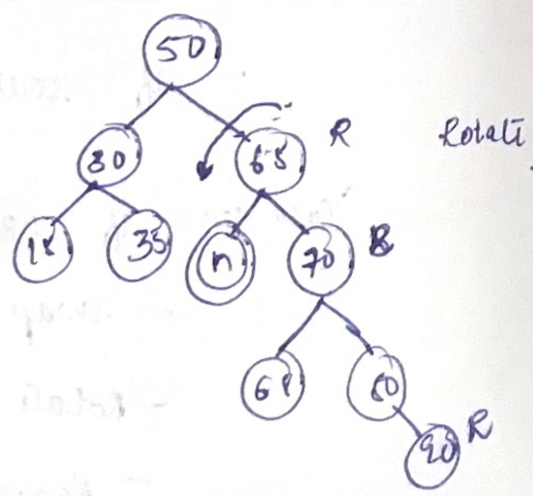
delete (55)

⇒ ?

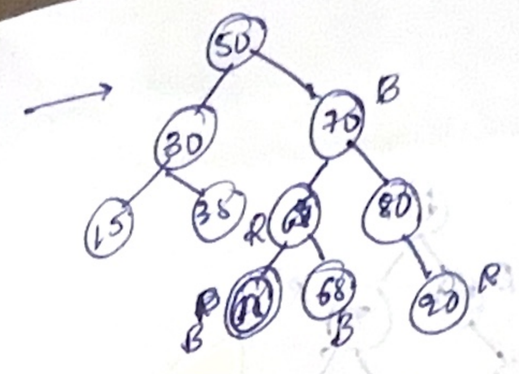
DB → double black



(2. Case 4)

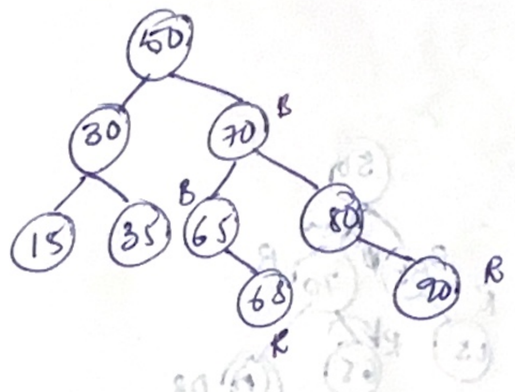


Rotate L

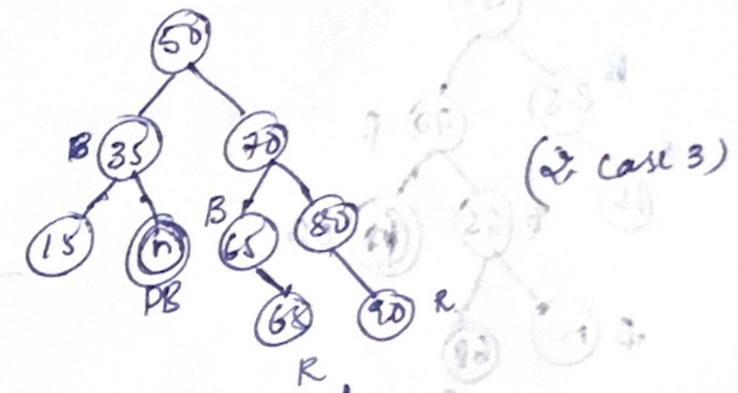


(2. case 3)

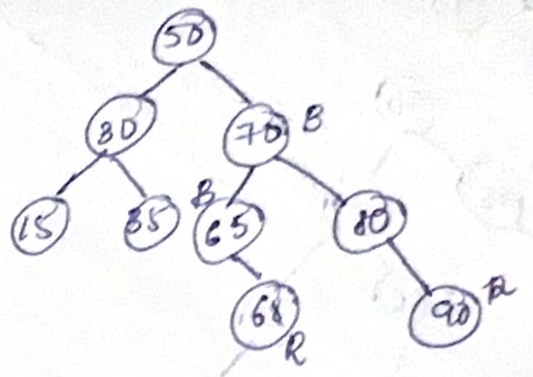
Delete (30):

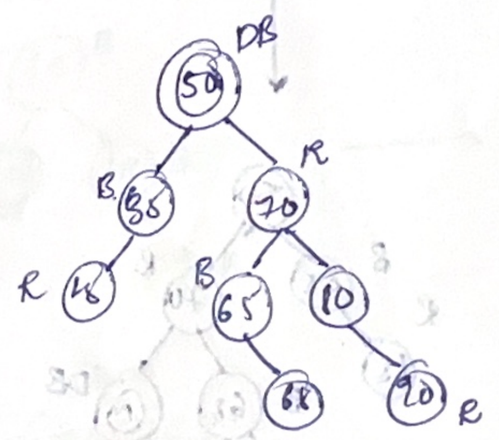


BST deletion

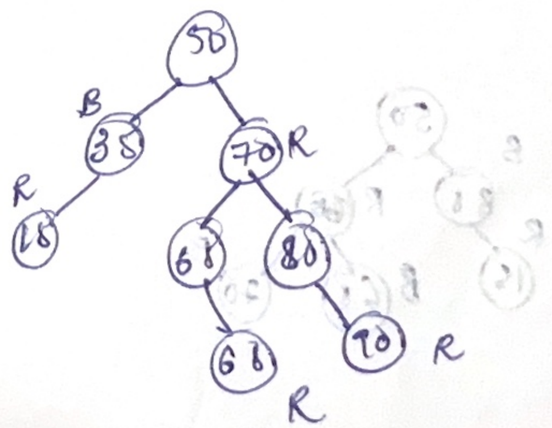


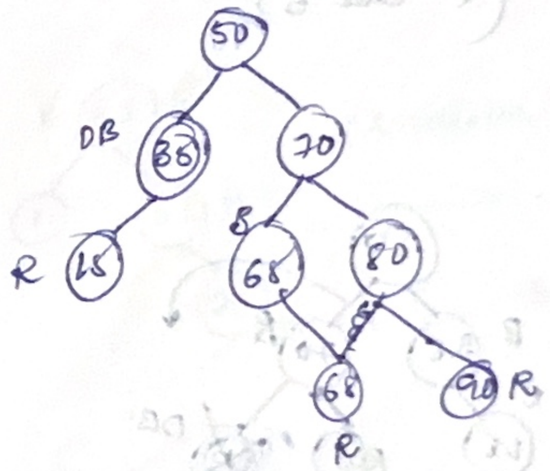
(2. case 3)



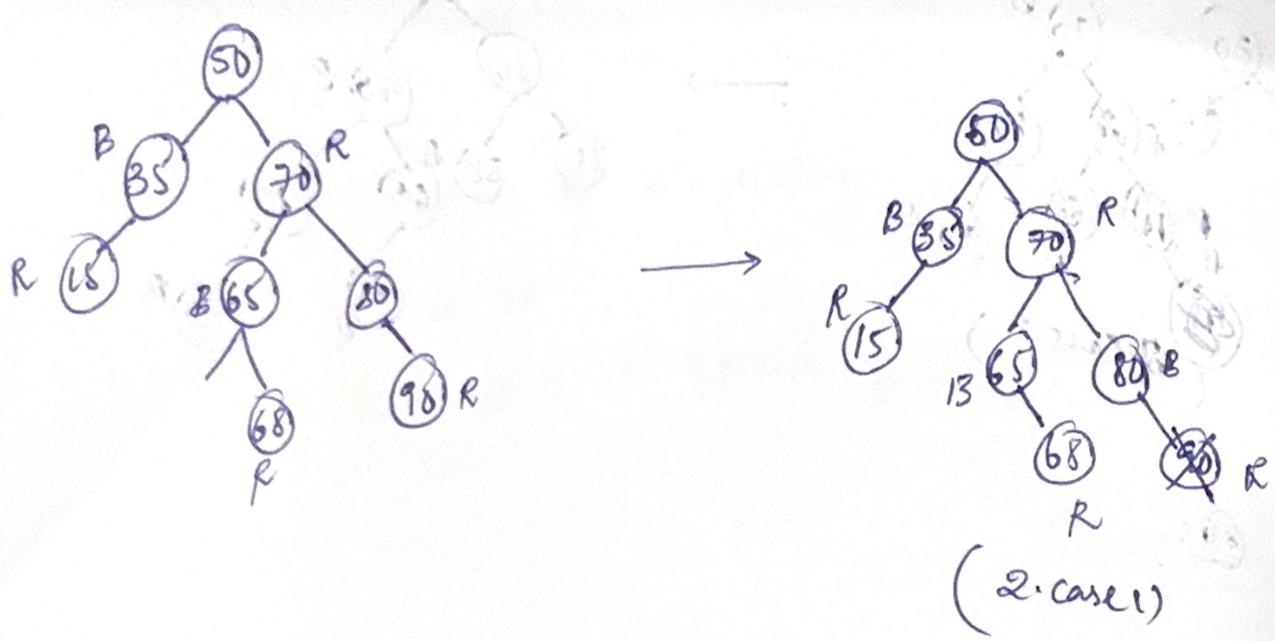


(2. case 2)



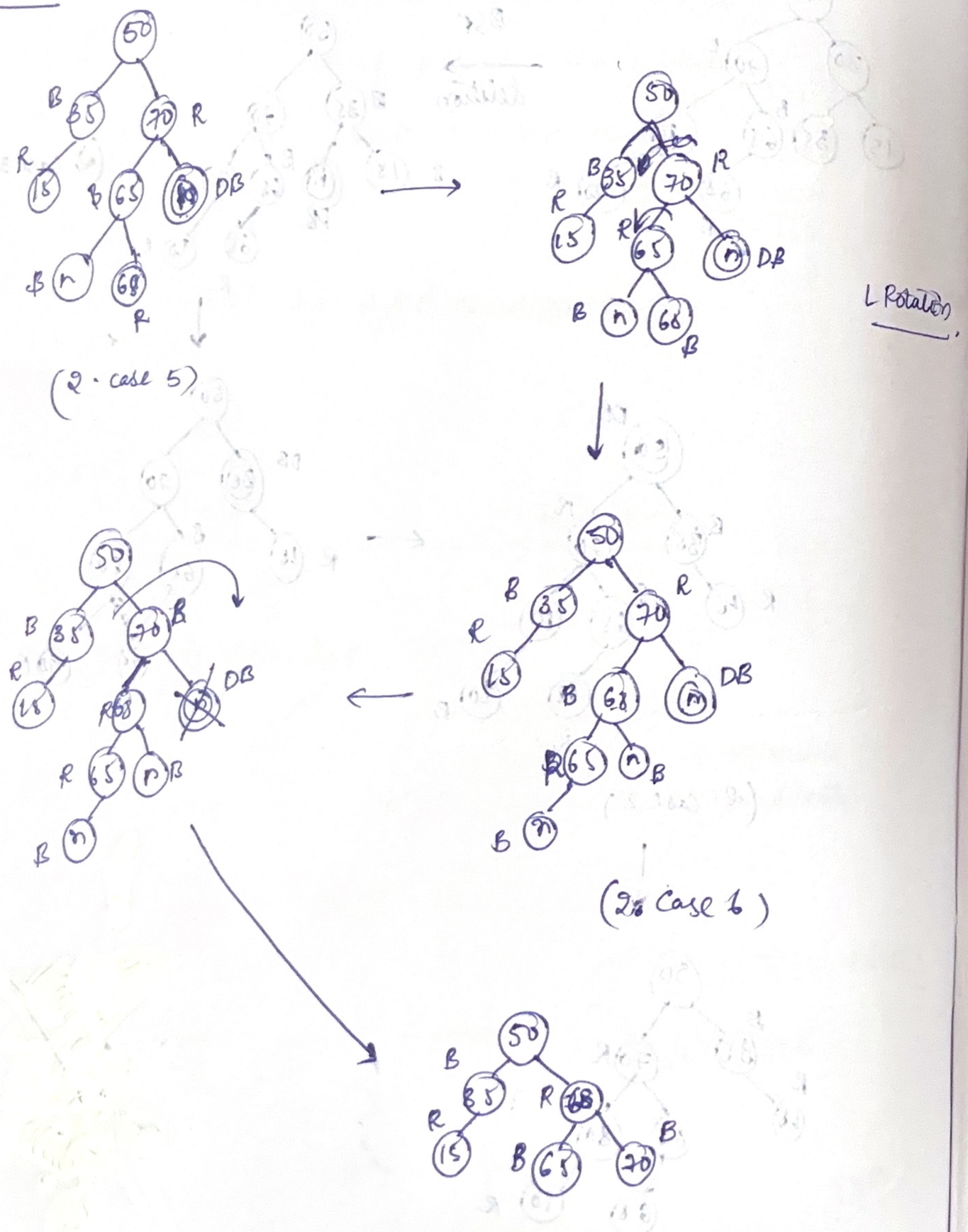


Delete (90) :

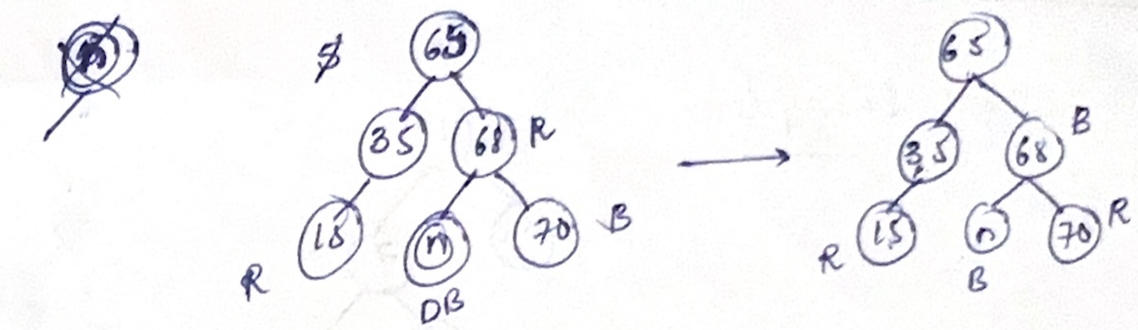


(2. case 1)

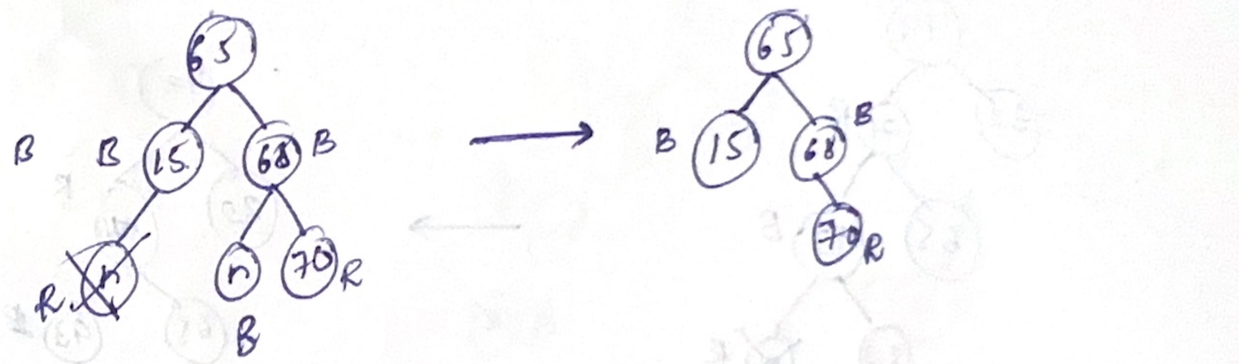
Delete (80)



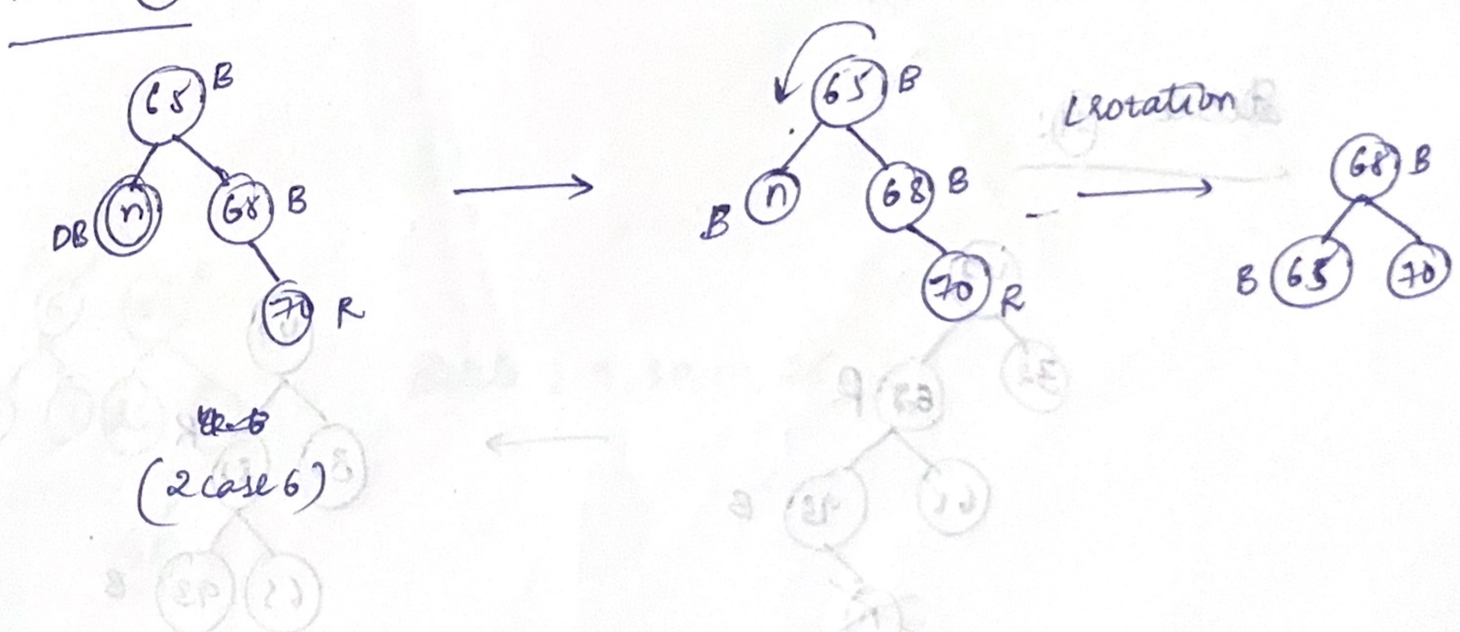
delete (50) :



delete (65) :

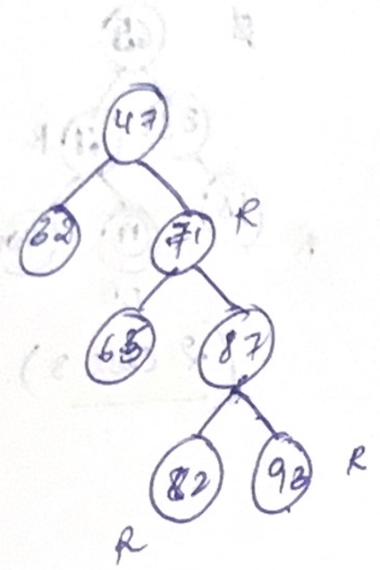


delete (15) :

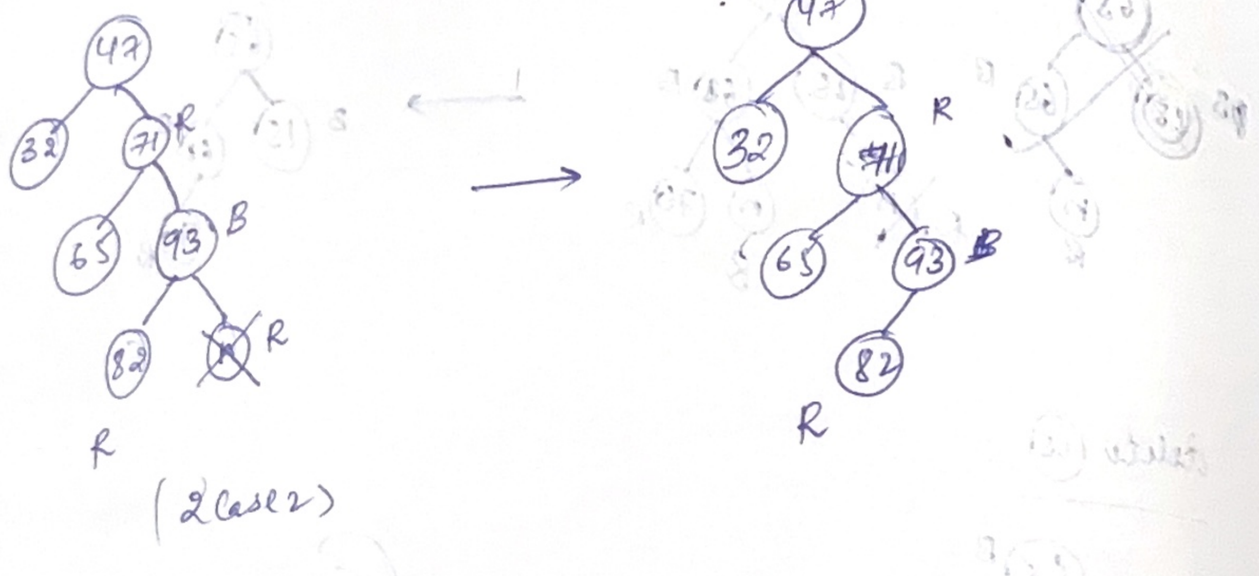


20-09-2025 :

Example:

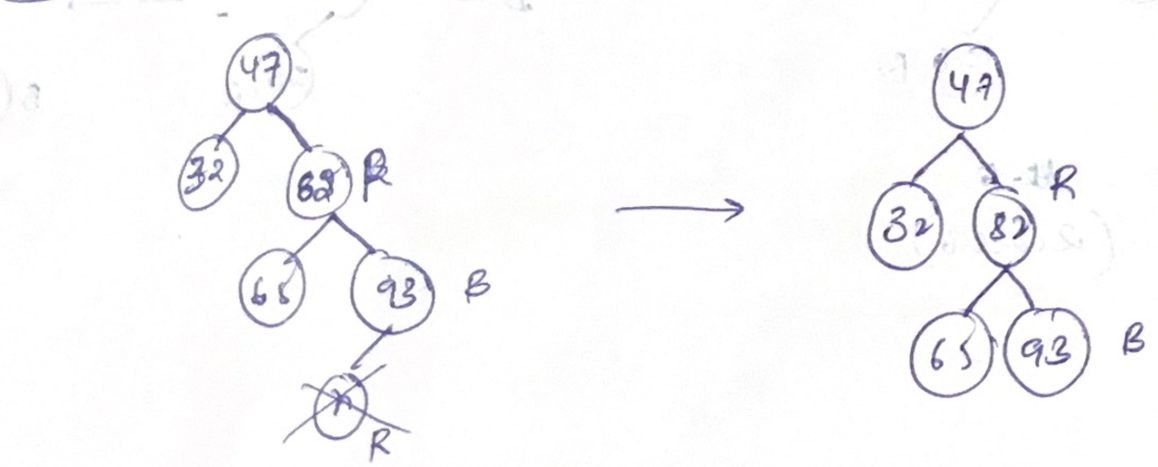


Delete (87):



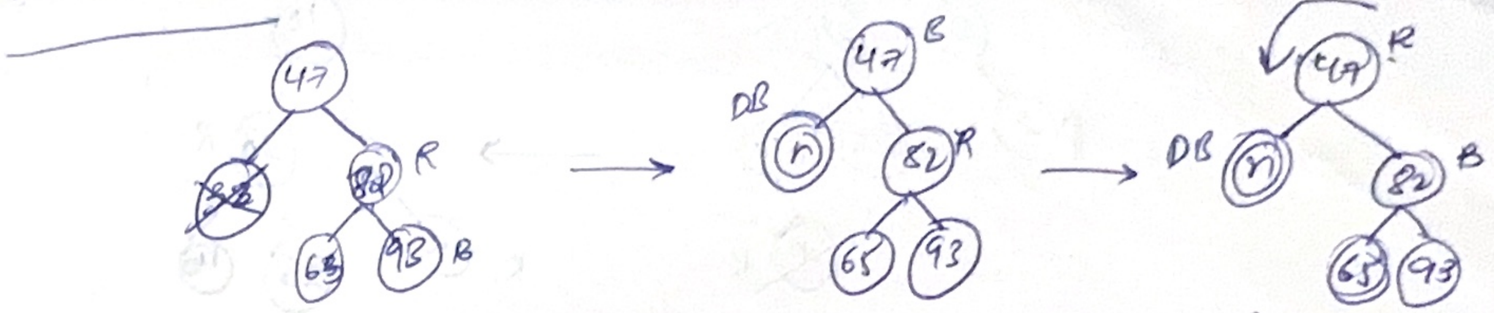
(2 Case 2)

Delete (71):

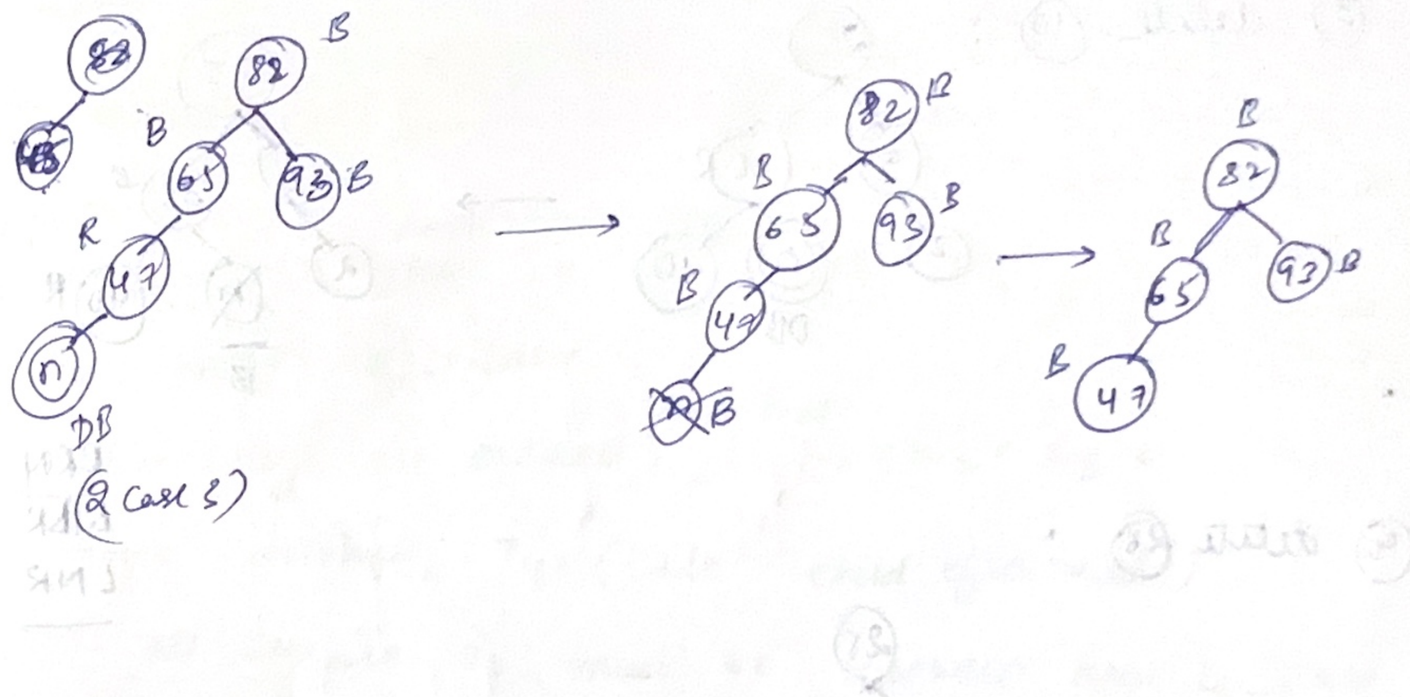


(2 Case 2

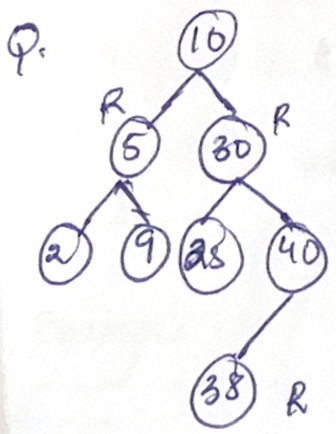
Delete (32):



(2 Case 4)

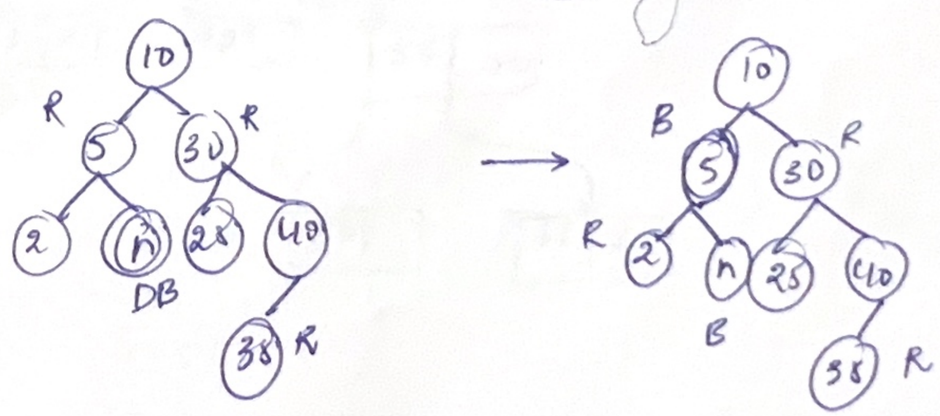


Q.

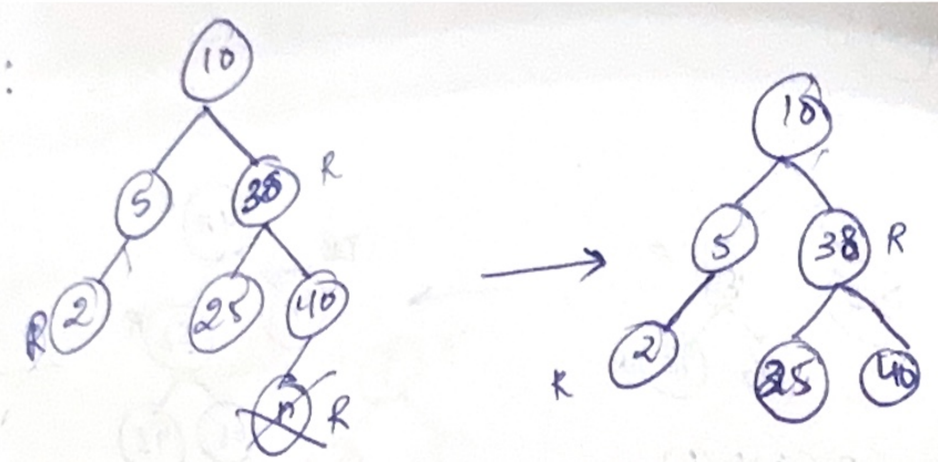


Delete : 9, 30, 10, 25

(1) Delete (9) :

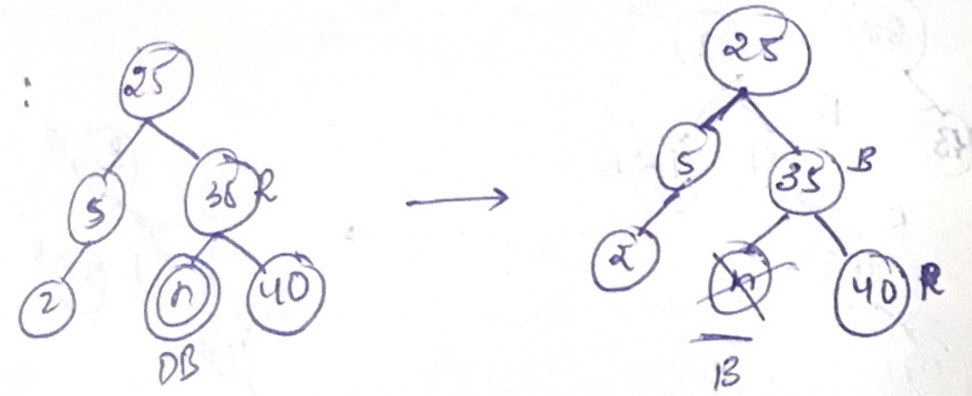


(2) delete (30) :



(2 case 2)

(3) delete (10) :



RRN
NLR
LNR

(4) delete (28) :

22-09-2025 :

2-3-4 Trees: (01) 2-4 Trees

- These are m-ary trees (not Binary in particular)

2-3 trees:

- These trees can be / have not more than 2-3 children nodes

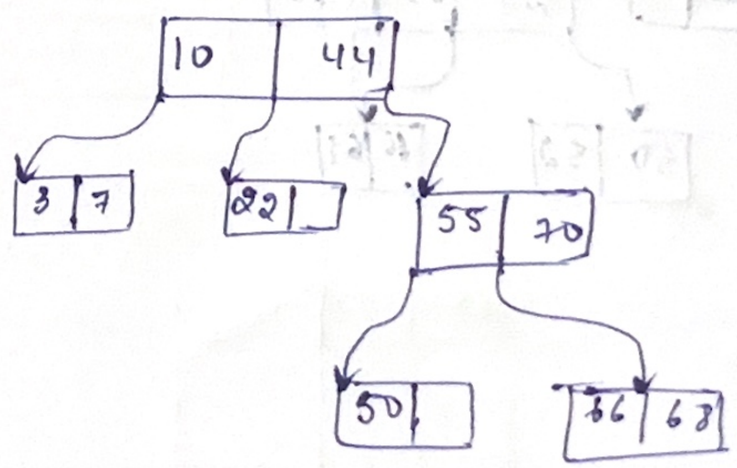
→ m-ary trees: ## corresponds to maximum no: of nodes too for a parent node,

properties:

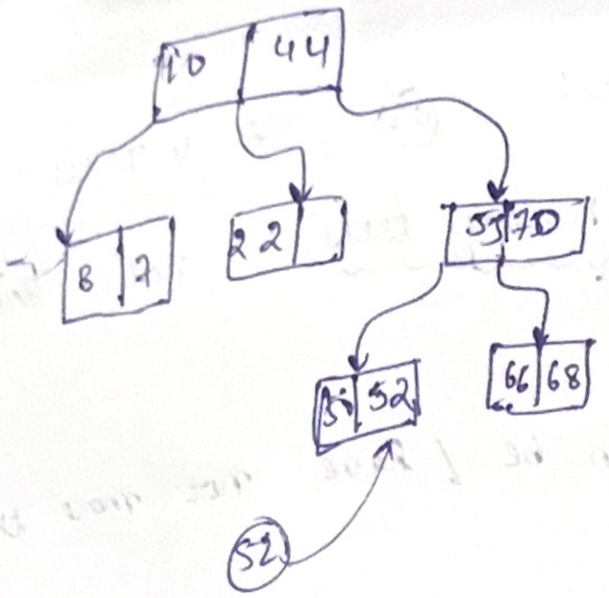
1. 1 to (m-1) values (data values / keys) in each (week) node
2. At most (m) children
3. These keys are ordered such that $k_1 < k_2 < k_3 < \cdots k_{m-1}$
4. For a subtree T_i (i^{th} child of a node)
   all keys in T_i must be between keys k_{i-1} and k_i
5. All key in first subtree T_i, keys$(T_1) < k_1$
   All keys in last subtree T_m, keys$(T_m) > k_{m-1}$

Example:

2-3 TREE

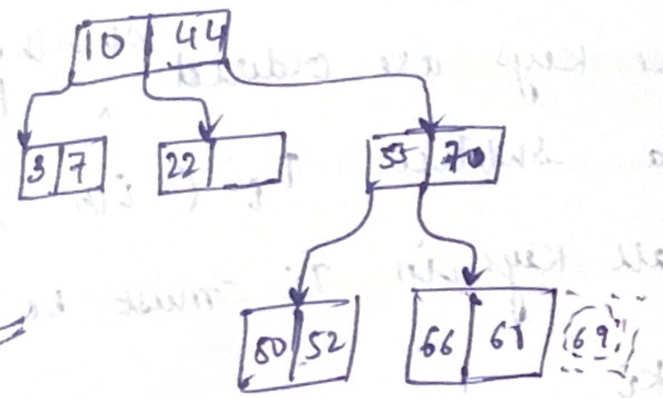


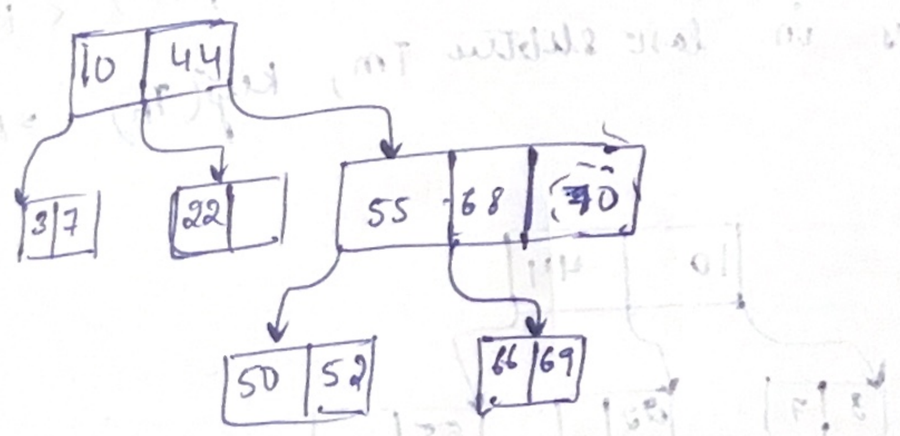
: Add (52)



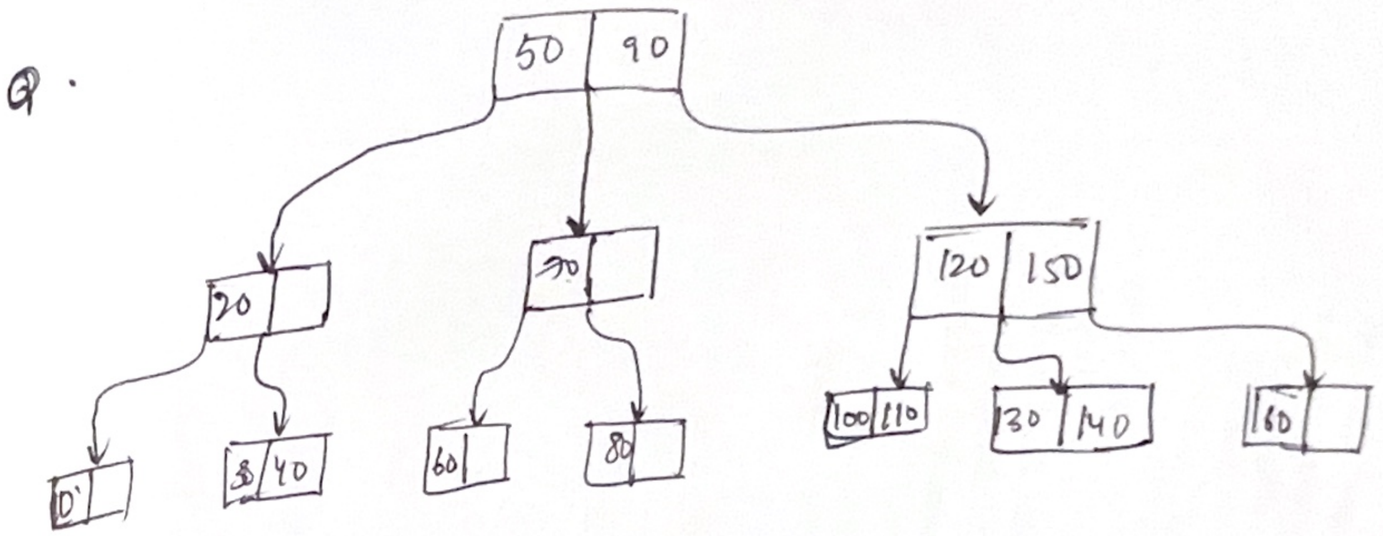
- Insertion should be done like binary tree and keep all leaf nodes.

Add (69)



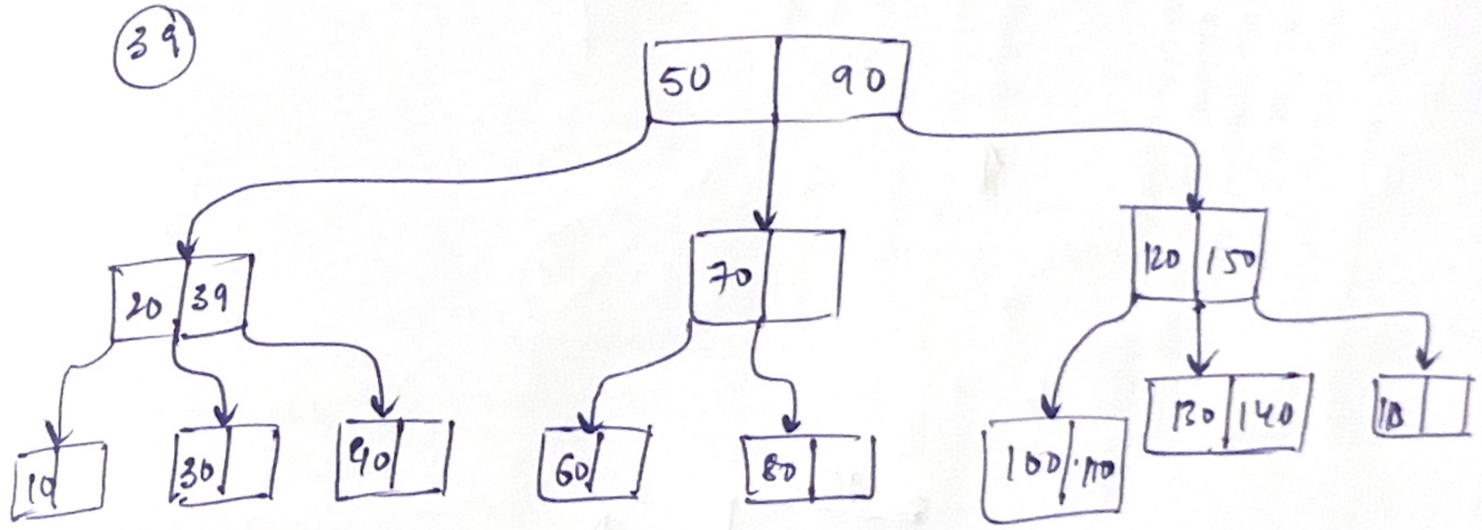


Q.

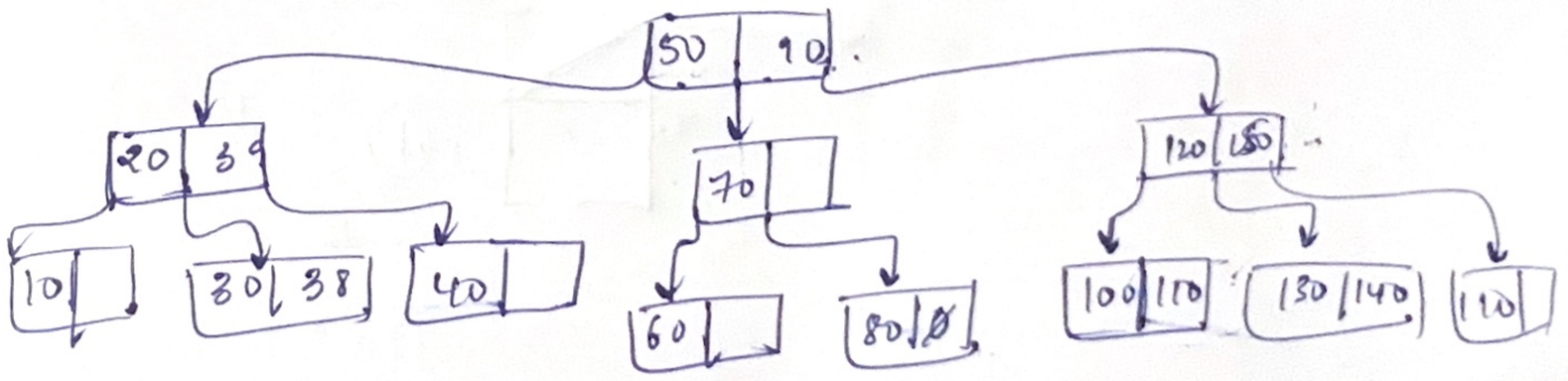


Insert : 39, 38, 135, 125

(39)



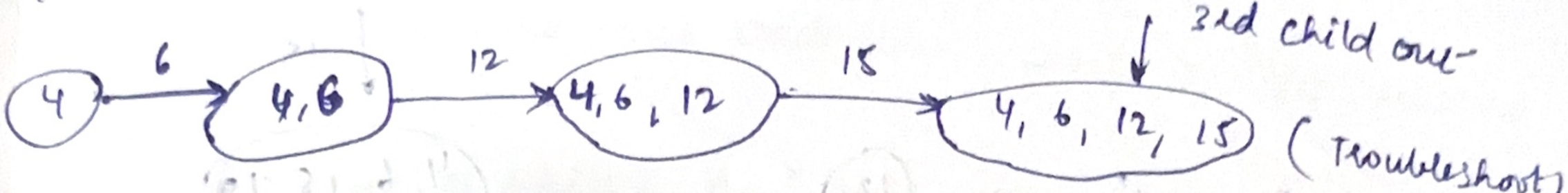
(38)

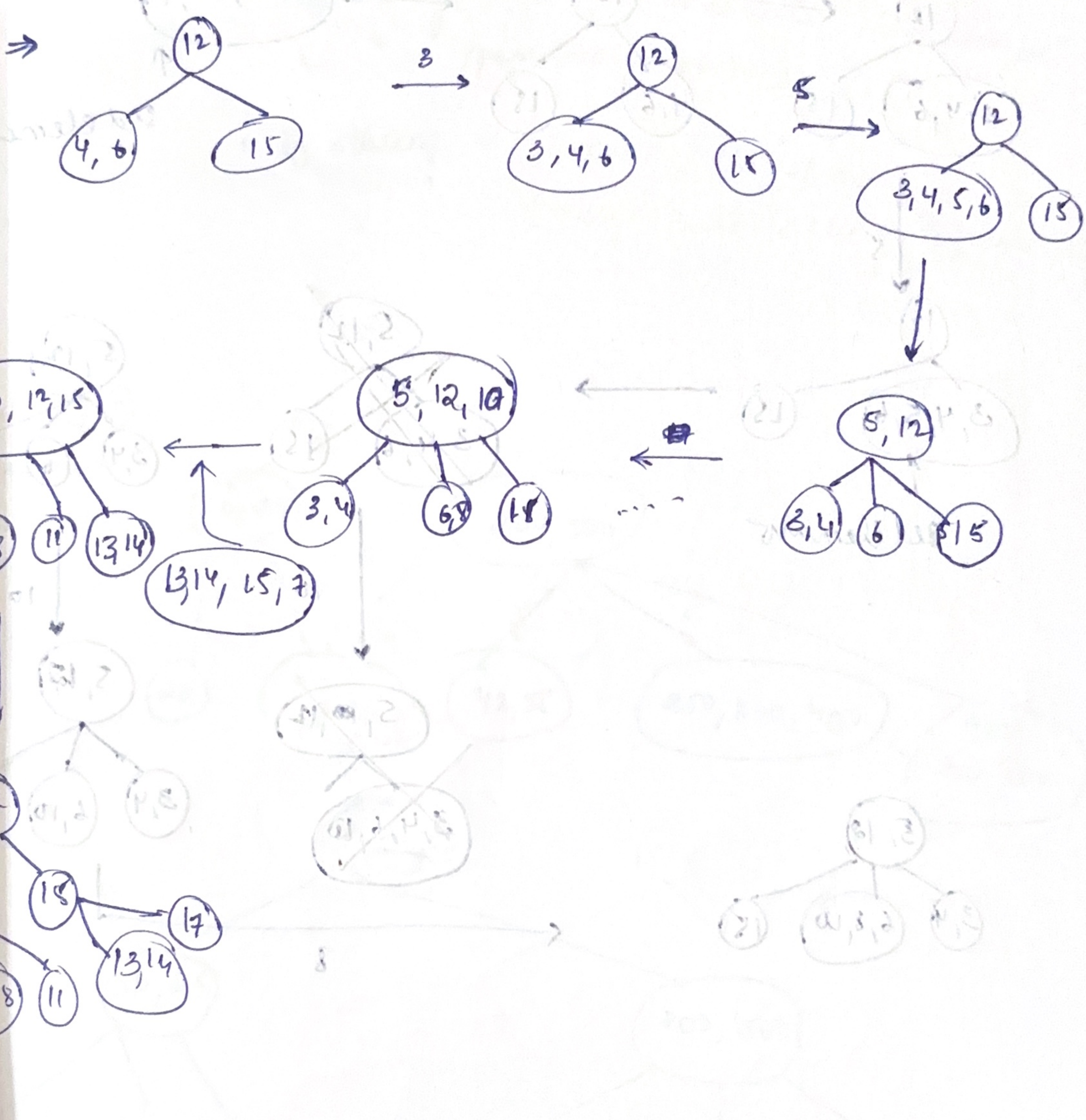


24-09-2025

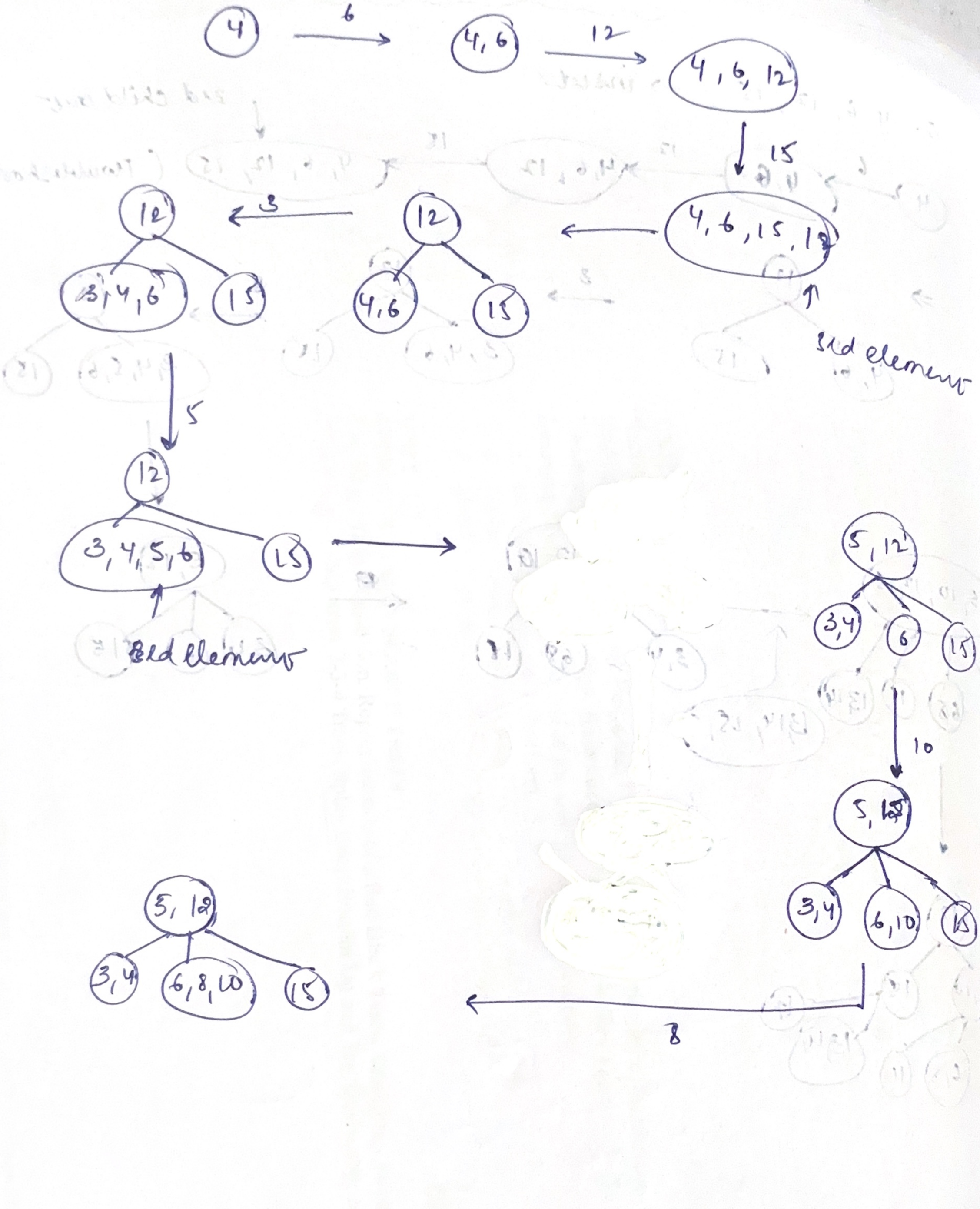
2-3-4 (Insertion)

Q. 4, 6, 12, 15 → insert:





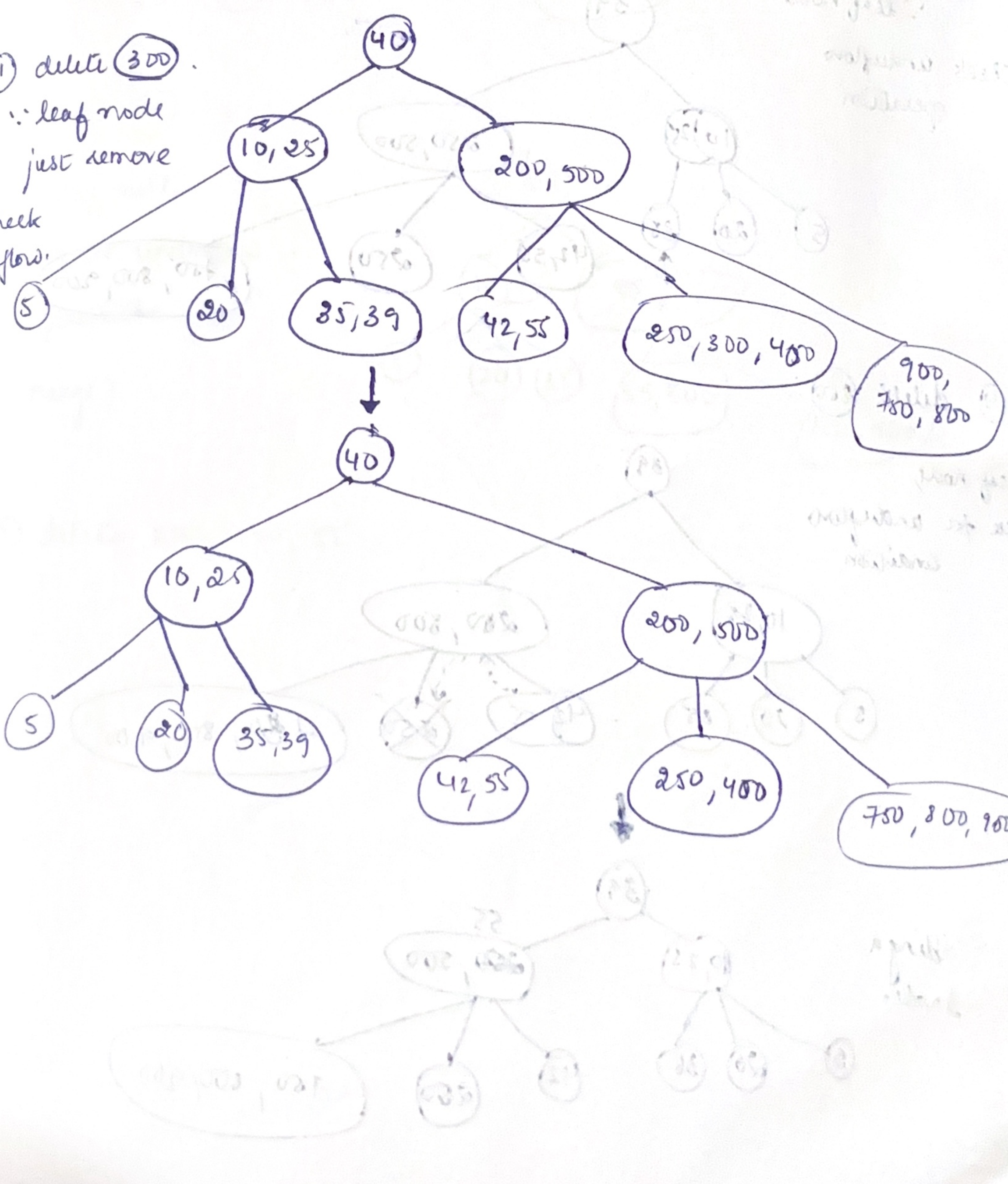
Q. insert : 4, 6, 12, 15, 3, 5, 10, 8



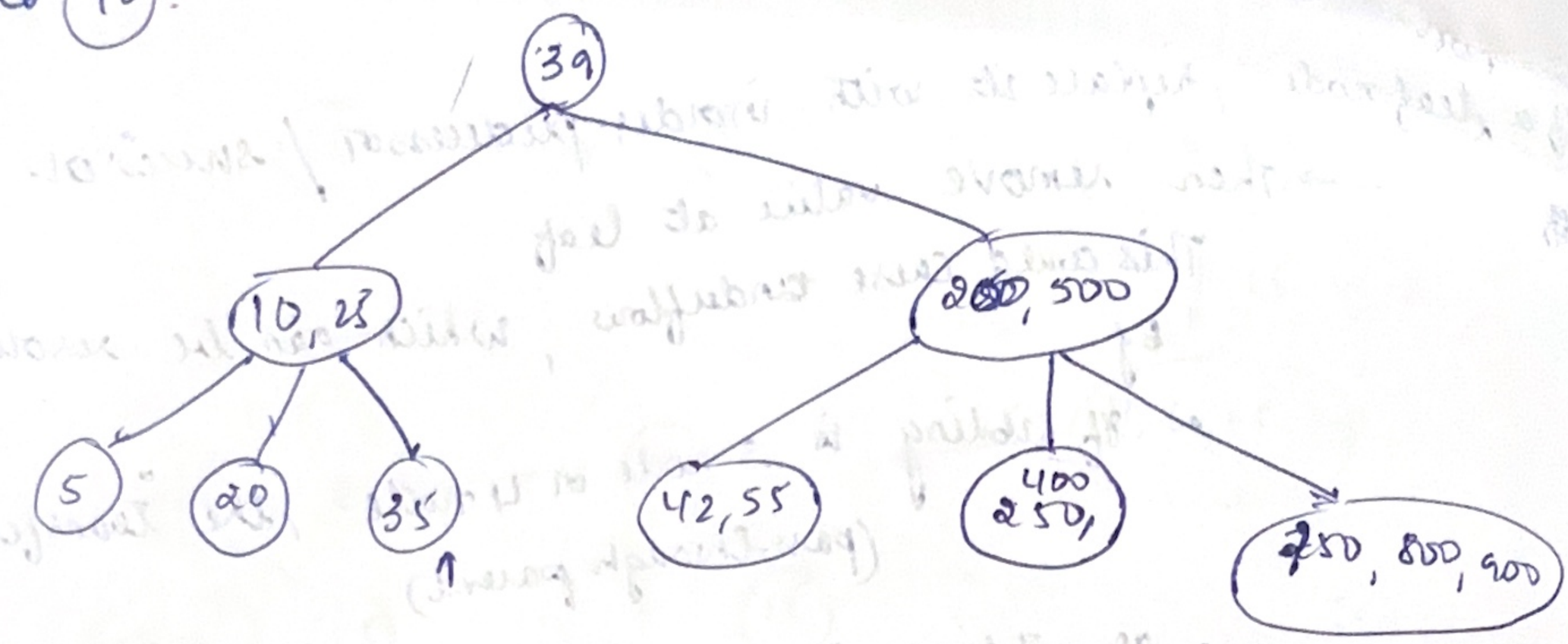
2-3-4 Deletion:

- If a non leaf node, replace it with inorder predecessor / successor.
  - → Then remove value at leaf
    This could cause underflow, which can be resolved by
    - If sibling is 3 node or 4 node, the "transfer" (pass through parent)
    - If sibling is 2-node, the "fusion" (merge with sibling & parent key)

① delete 300.
∵ leaf node
just remove
and check
underflow.



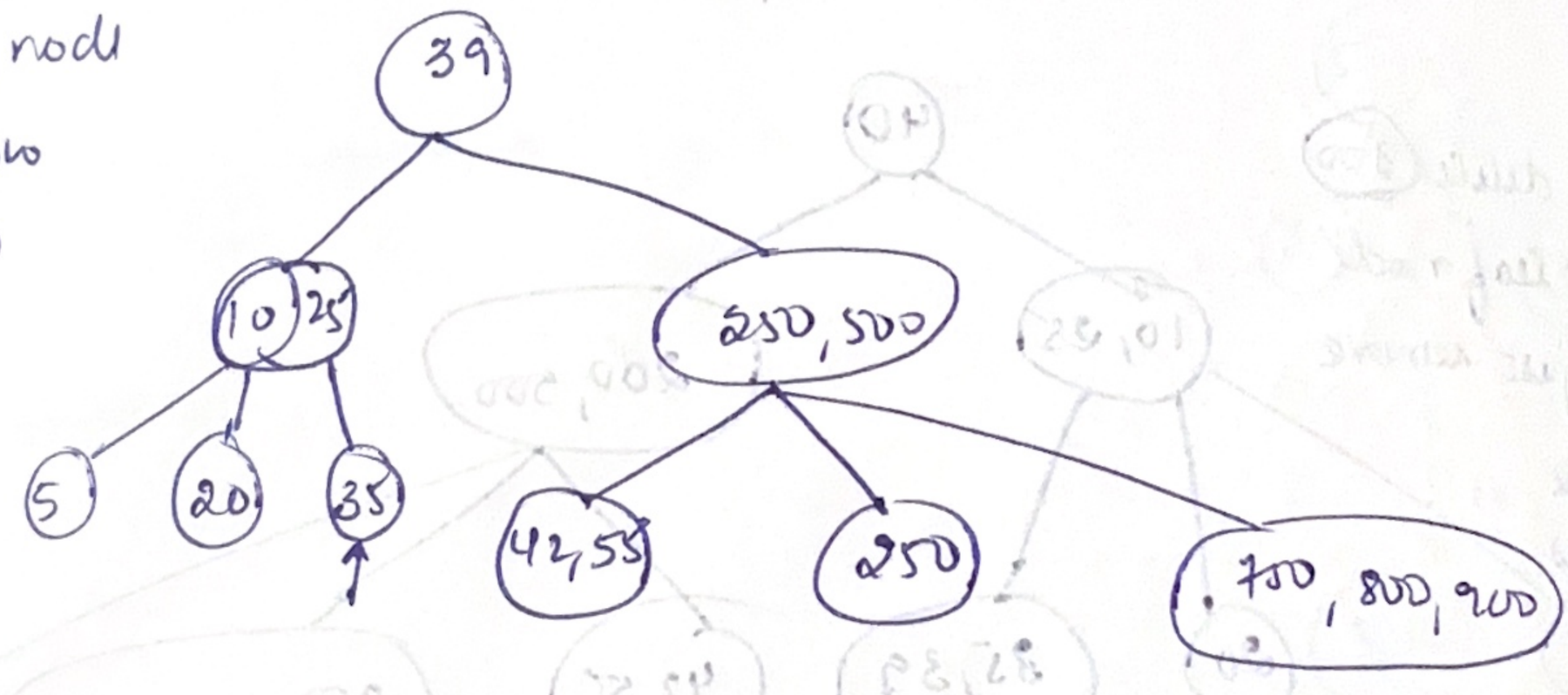
② Delete (40): (Exchange with inorder predecessor)



③ delete (40):

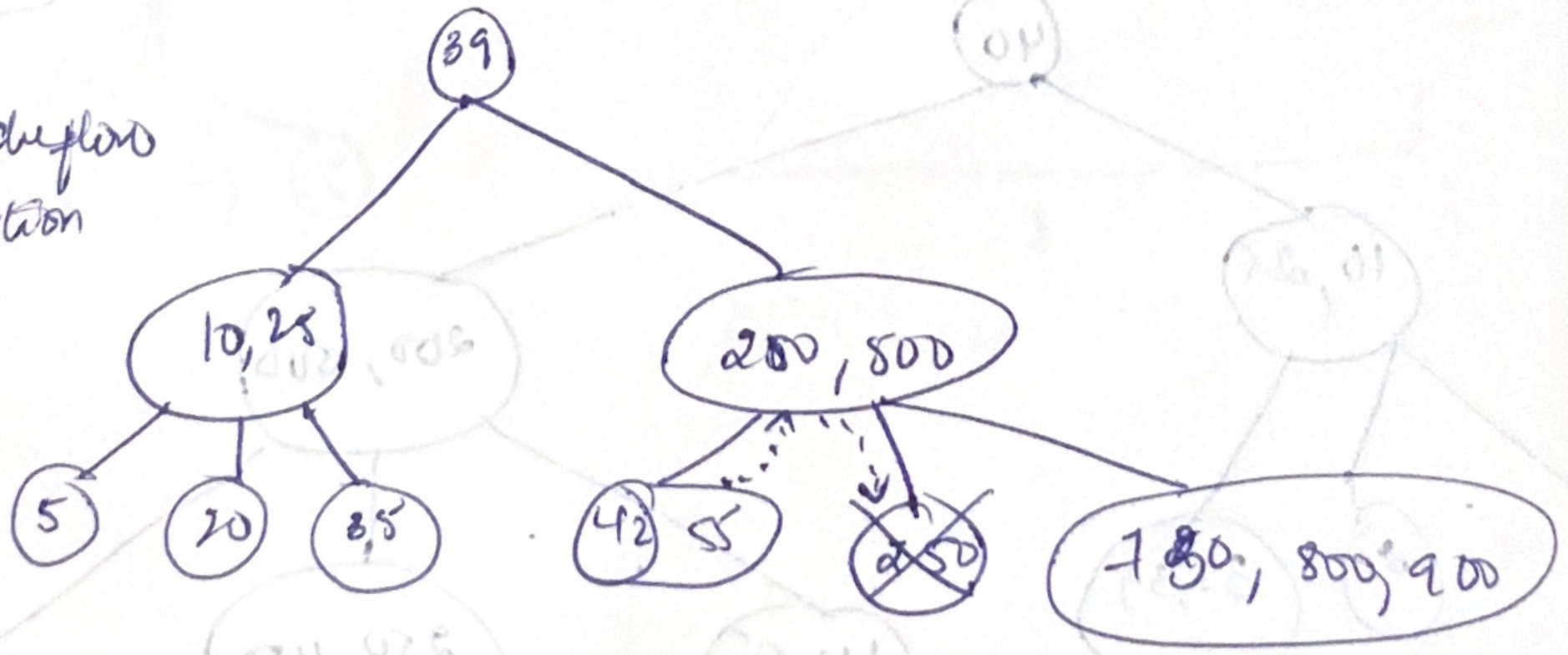
∵ leaf node

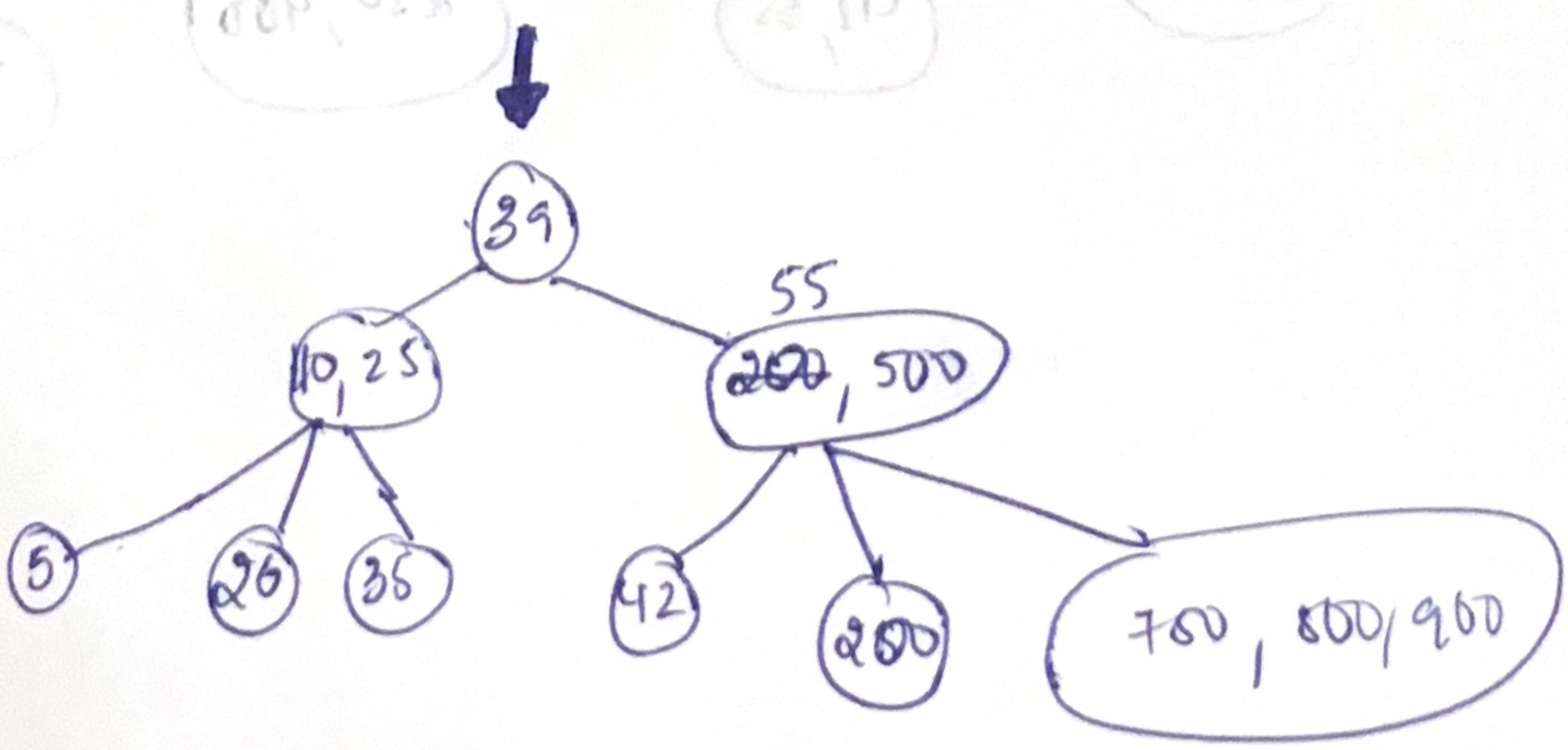
Check underflow operation



④ delete (200):

∵ leaf node,
Check for underflow condition



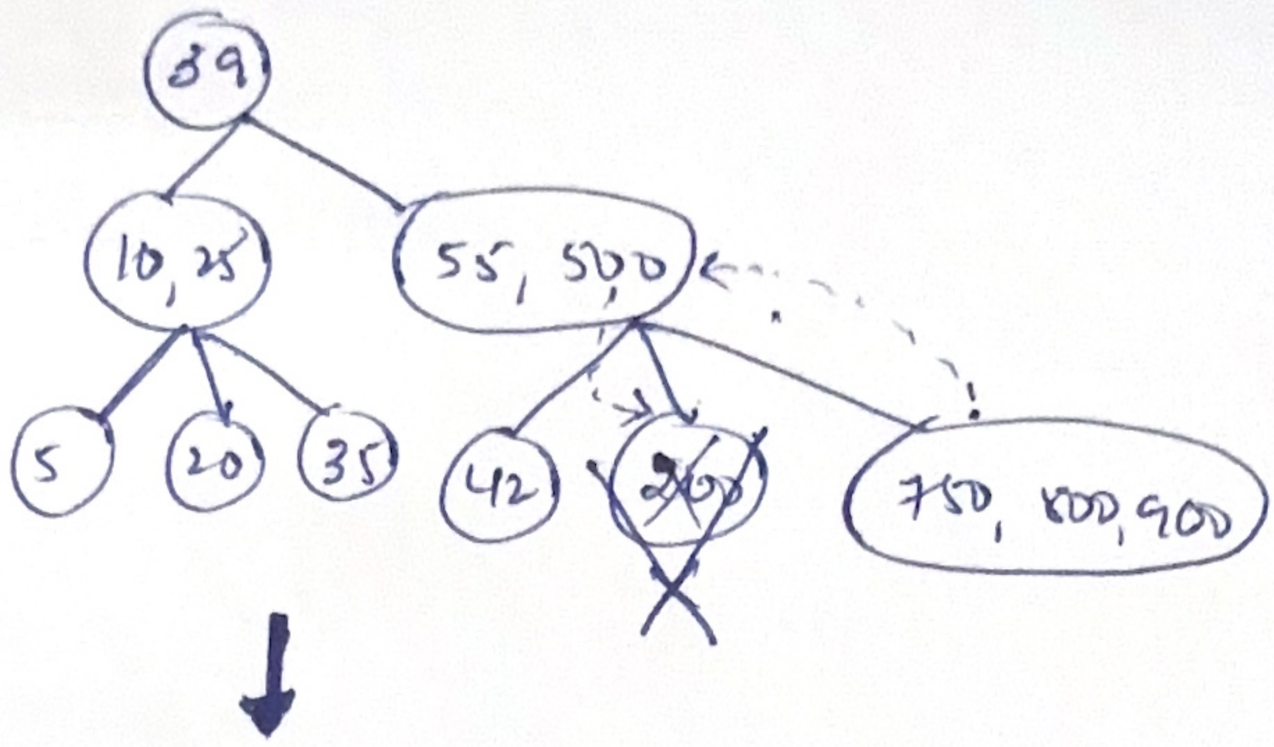


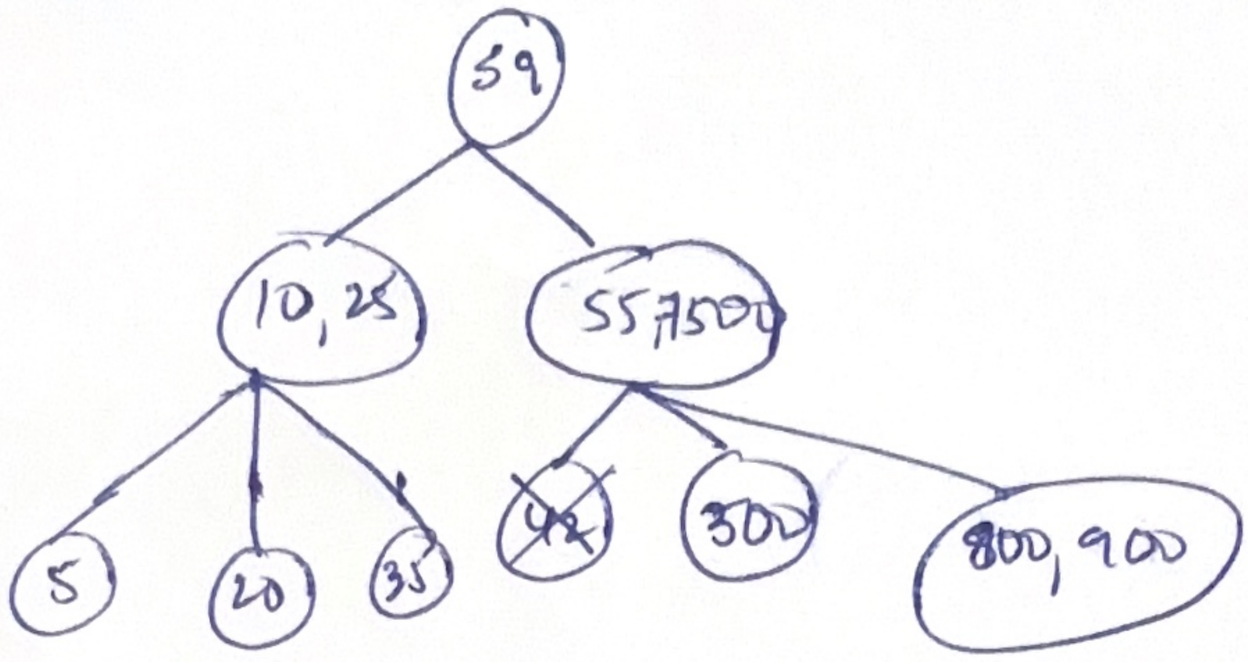
∵ sibling is 3 node

(5) delete 200 :

∵ leaf node

Check for underflow condition.





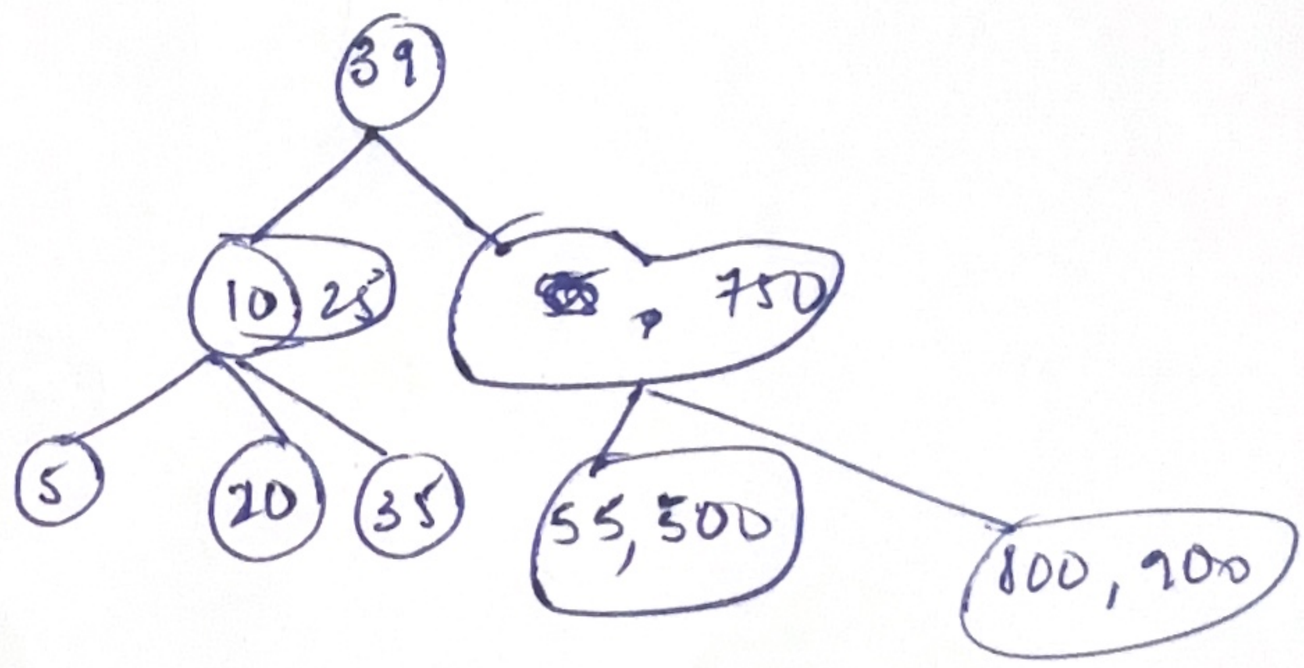
(Transfer)

(6) delete (42) :

∵ leaf node

Check for underflow condition

(Merge)



(7) delete 800, 500, 55