UNIT - III

Red Black trees :-

In the binary search tree, normally searching time is O(logn), but in skewed trees the time complexity for searching an element is O(n) in worst case.

Inorder to reduce the time complexity for searching a node we use self balancing tree called as AVL tree, which takes O(logn) for search operation.

→ But inorder to balance the tree we need to perform rotations and we might need to perform rotations for all nodes between leaf node and the root node.

So there is a need of data structure which performs better than AVL tree for performing inserting and deleting operations of a node in binary search tree. That tree is called as "Red Black tree"

A tree which follows the following properties are known as Red Black trees :-

1) The tree should be a binary search tree.
2) Every node in the tree should be either black or red.
3) The root node and all the external nodes of the tree are coloured with Black.
4) If a node is red, then its children should be Black that means no two consecutive nodes

should be in red colour.

5) Every path from a node to any of its descendent null nodes have same no. of black nodes.

Insertion in a Red black tree :-

Step-1:- If the tree is empty, then create a new node and colour it as black.

Step-2:- If tree is not empty, create a new node as leaf node and colour it as red.

a) If parent of new node is black then insertion of new node is completed.

b) If the parent of new node is red, then check the colour of parent sibling of new node.

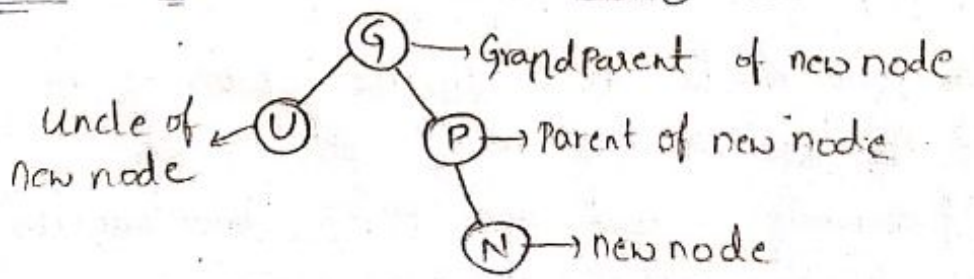
i) If the colour is red, we recolour the new node's parent, sibling & grandparent.
(If grandparent is root node, we cannot recolour it)

ii) If the colour is Black or Null, perform suitable rotation and recolour

1 → If the parent of new node, new node and grandparent of new node form a triangle, then rotate parent in opp direction of new node

2 → If the new node, parent of new node and grandparent of new node form a straight line, then rotate grandparent in opp direction of new node, recolour original parent & grand parents

Relationship of new node with existing nodes:-

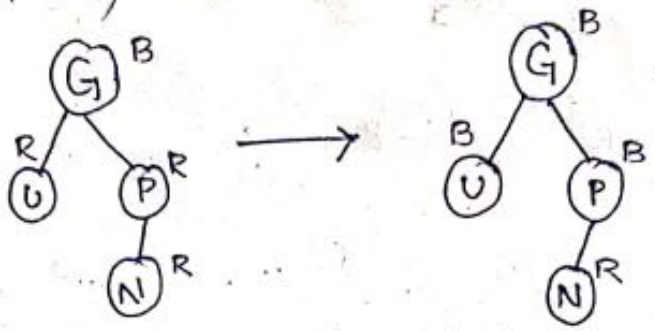


Four Cases :-

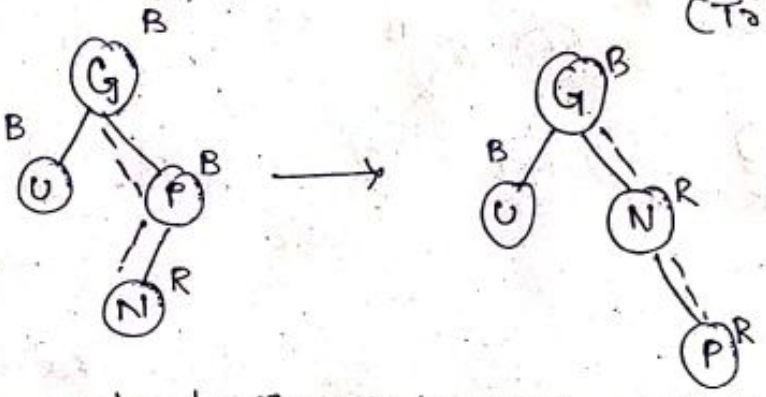
Case 0 :- Newly inserted node is root node.

~~Case 1 :-~~

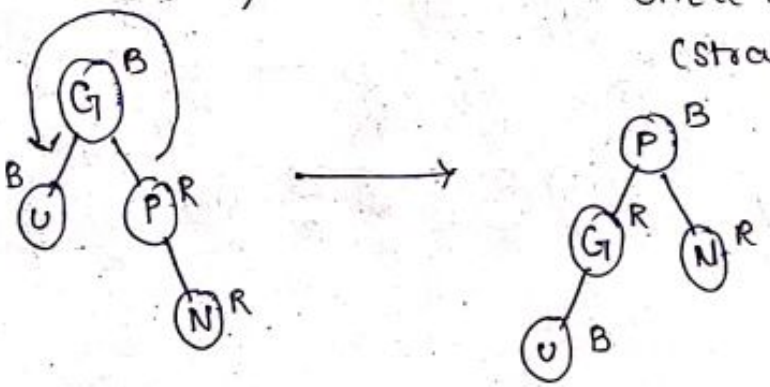
Case 1 :- Newly inserted node's uncle is red



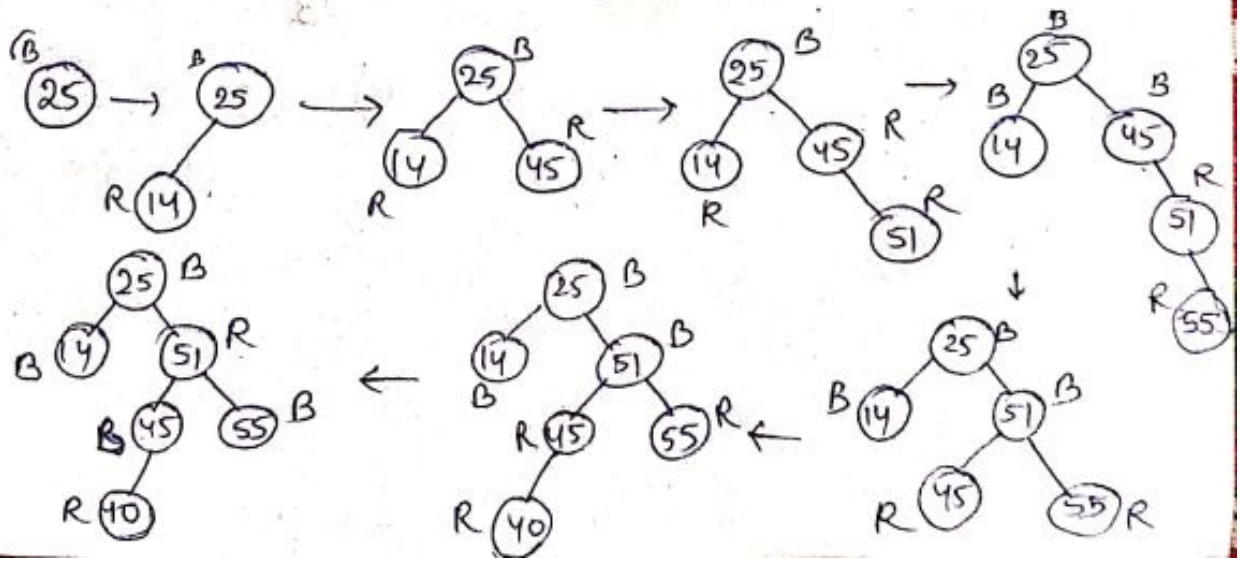
Case 2 :— Newly inserted node's uncle is black (Triangle)

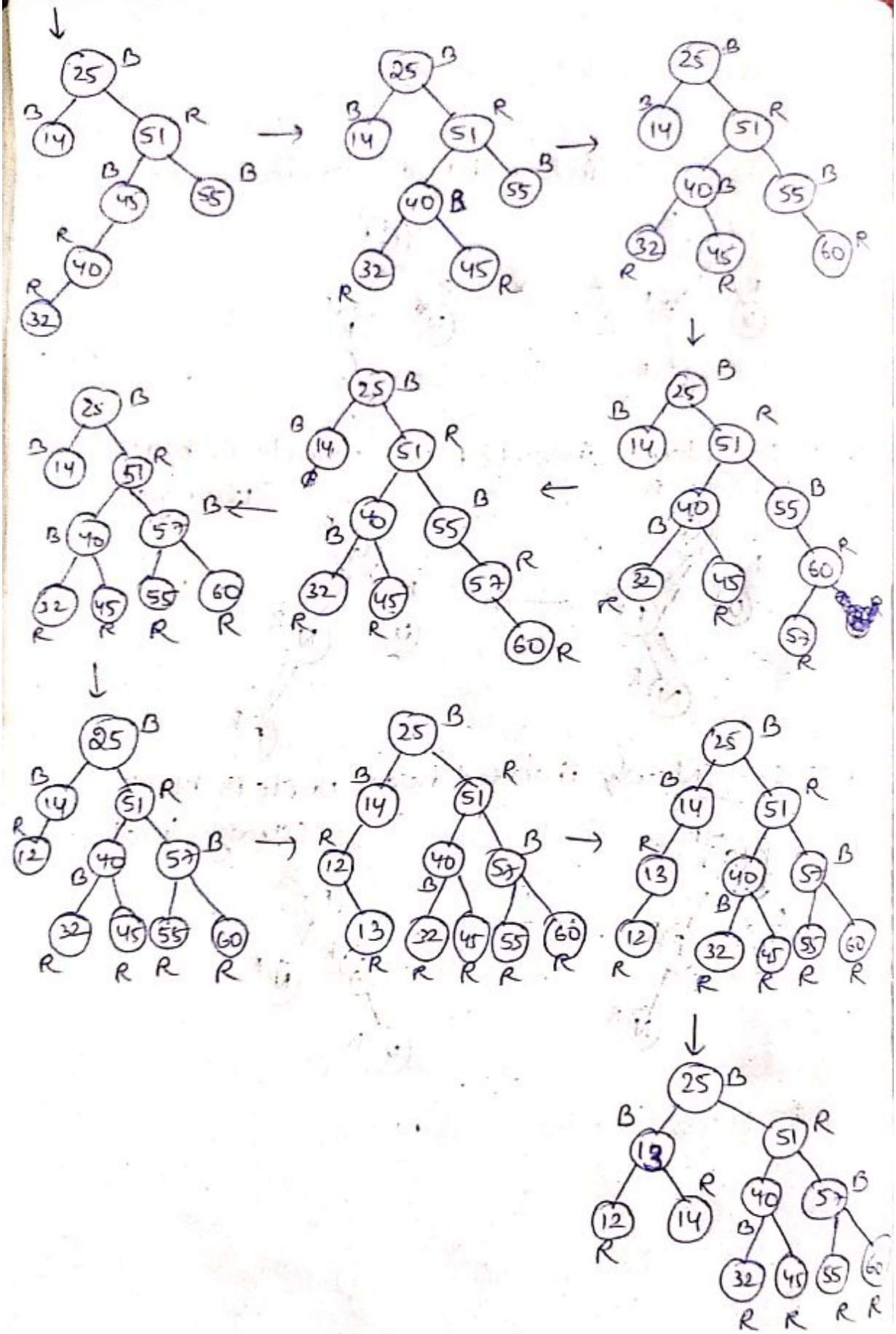


Case 3 :— Newly inserted node's uncle is black (straight line)

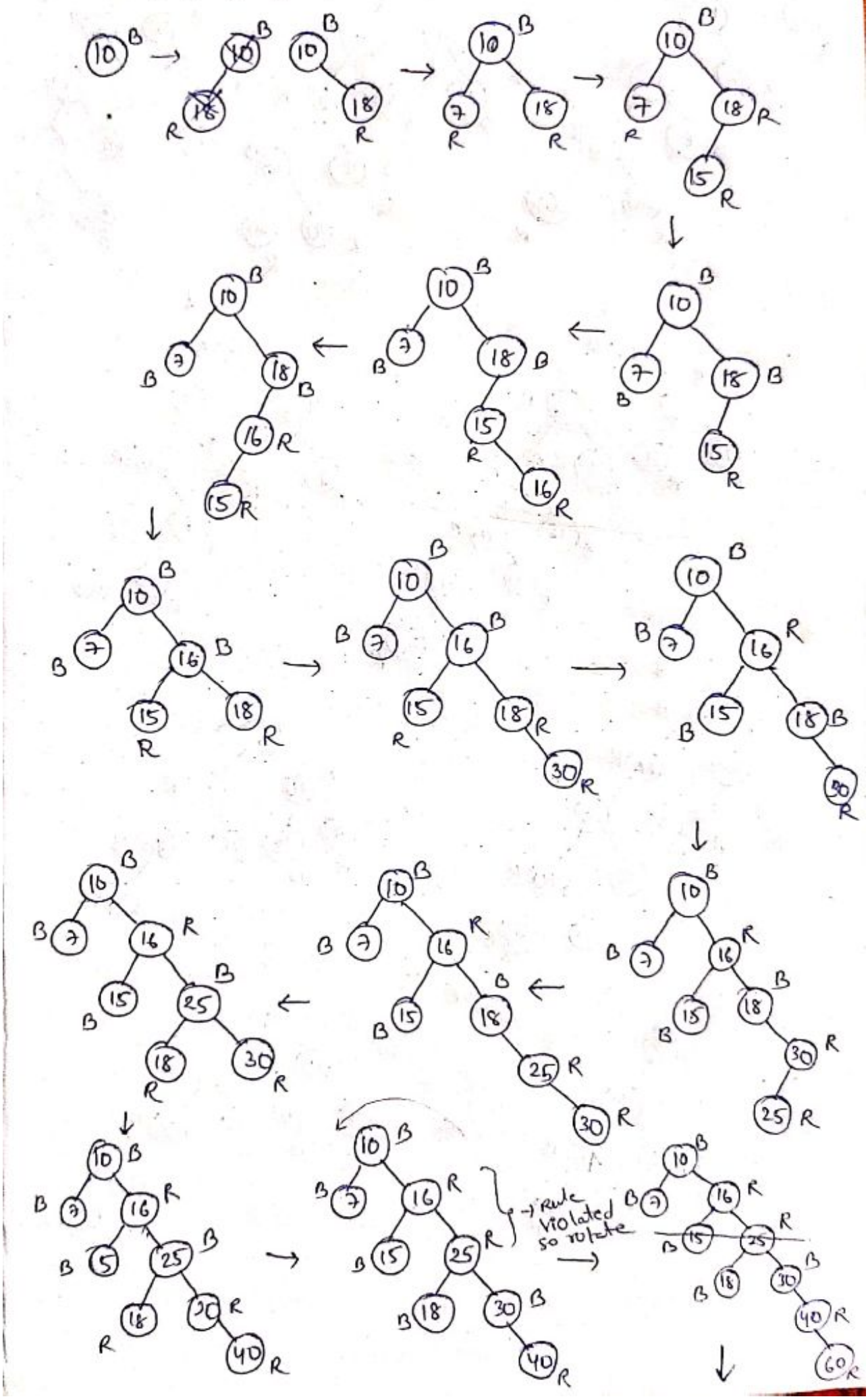


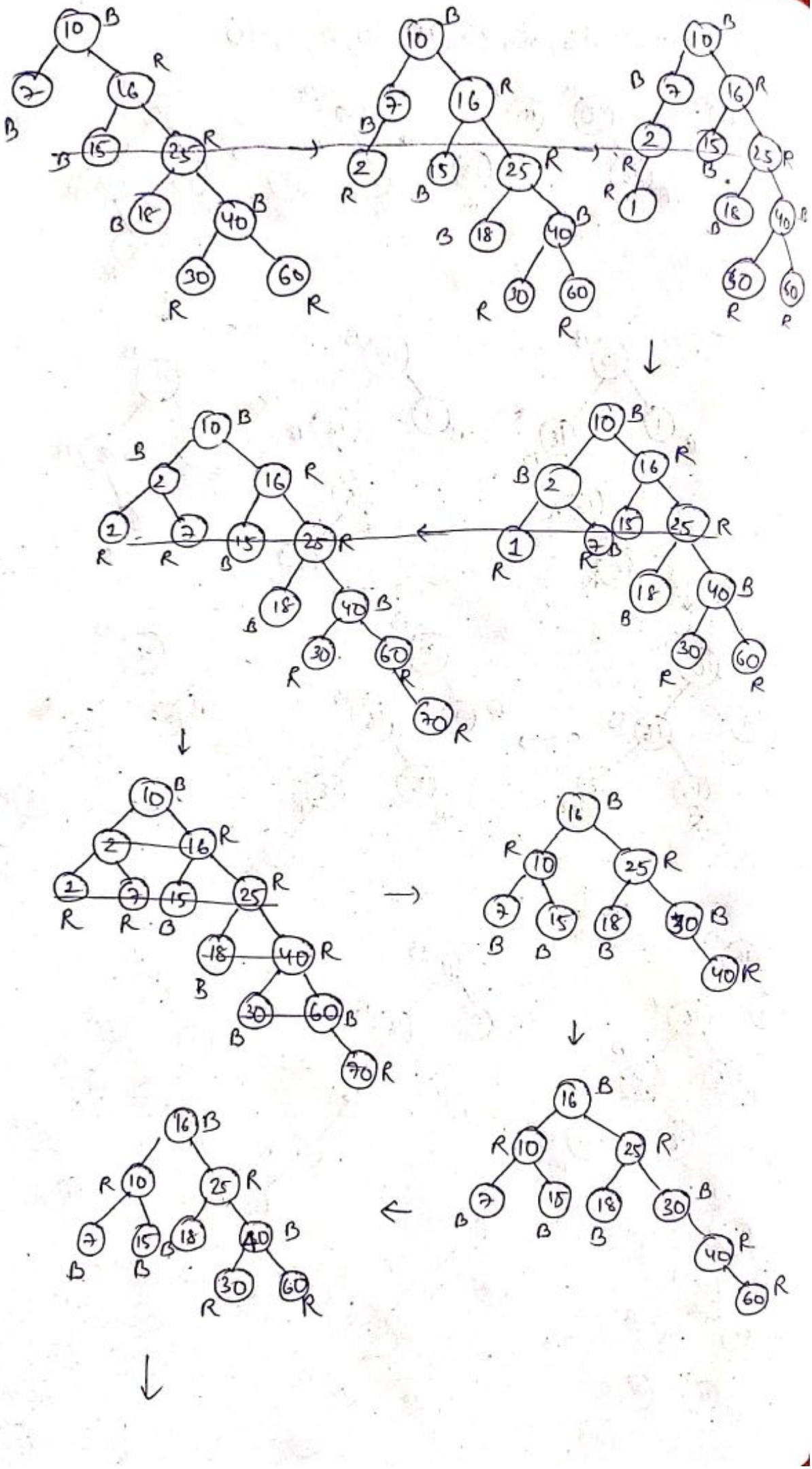
Ex :- 25, 14, 45, 51, 55, 40, 32, 60, 57, 12, 13

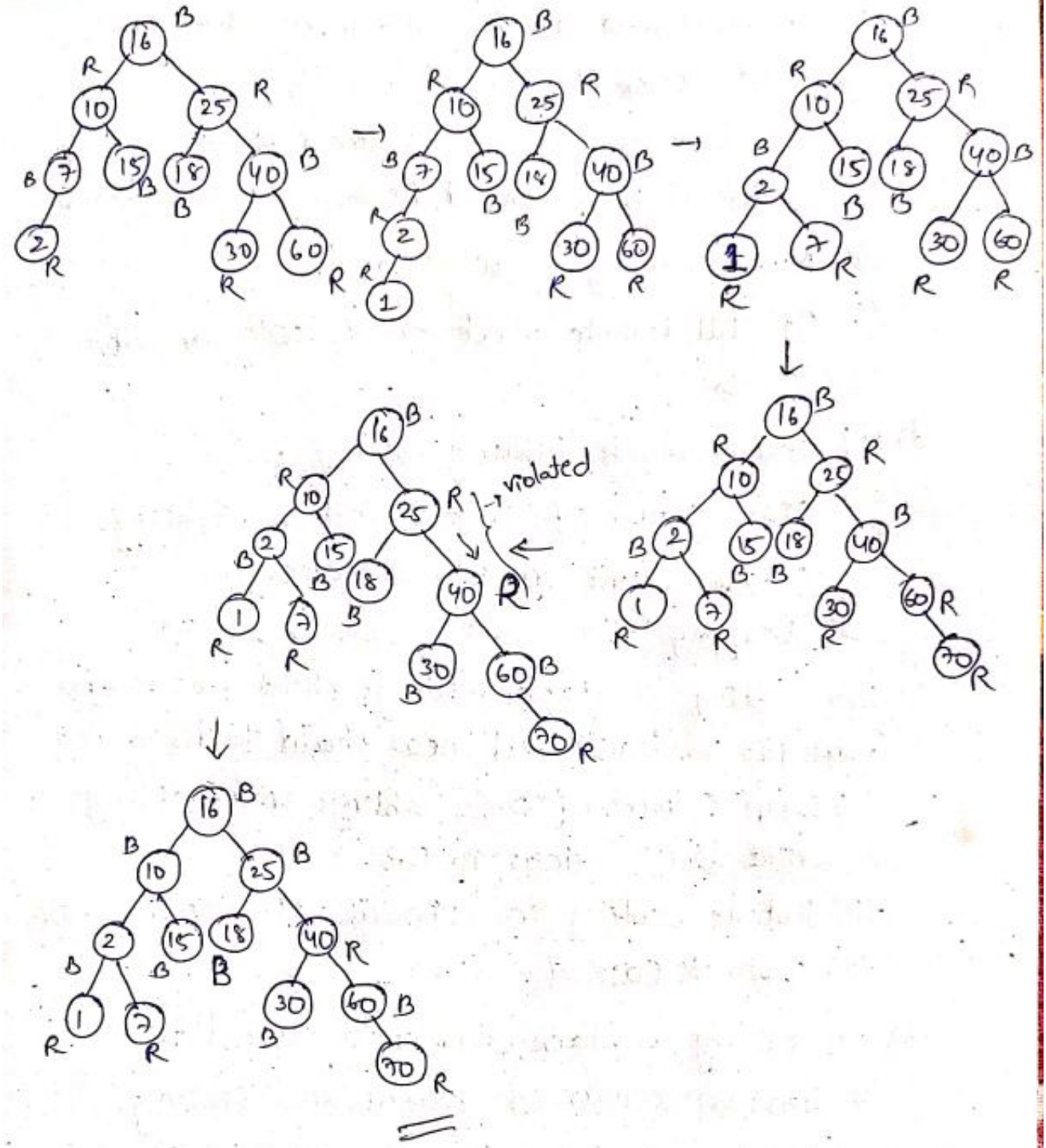




→ 10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, 70







Deletion in a red black tree :-

1. Deletions will be of type binary search tree

2.
   (a) If a node to be deleted is red, just delete it.
   (b) If root is double black (DB) just remove DB.
   c) If DB's sibling is black and both of its children are black then

(i) Remove double black (DB) from that node

(ii) Add Black to its Parent P.

→ If P is red, it becomes black.

→ If P is black, it becomes doubleblack.

(iii) Make sibling Color as red.

(iv) If still double black exists look for other Cases.

(d) If double black sibling is red :-

(i) Swap colour of Parent and its sibling.

(ii) Rotate parent in DB direction.

(iii) Reapply other Cases, if still DB exists

e) DB sibling is black, siblings child who is far from DB is black but near child to db is red

(i) Swap Colours of DB's siblings and siblings Child who is near to DB.

(ii) Rotate sibling in opposite direction to DB.

(iii) Apply Case f

f) DB sibling is black, far child is red :-

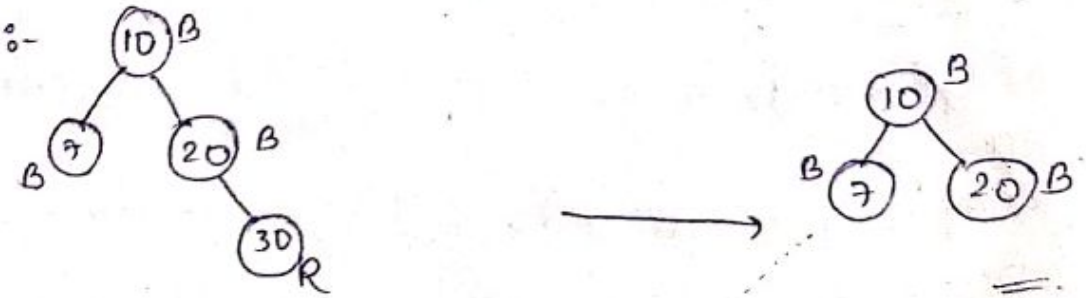
(i) Swap colour of Parent and sibling.

(ii) Rotate Parent in the direction of DB.

(iii) Remove Double Black.

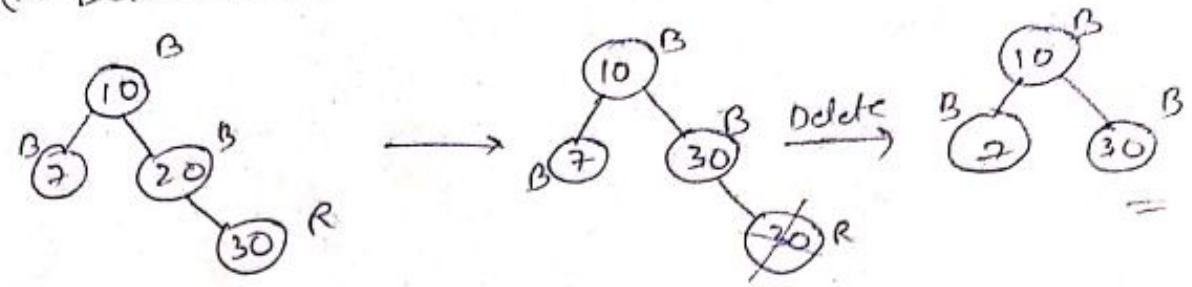
(iv) Change Colour of red child to black.

Ex :-

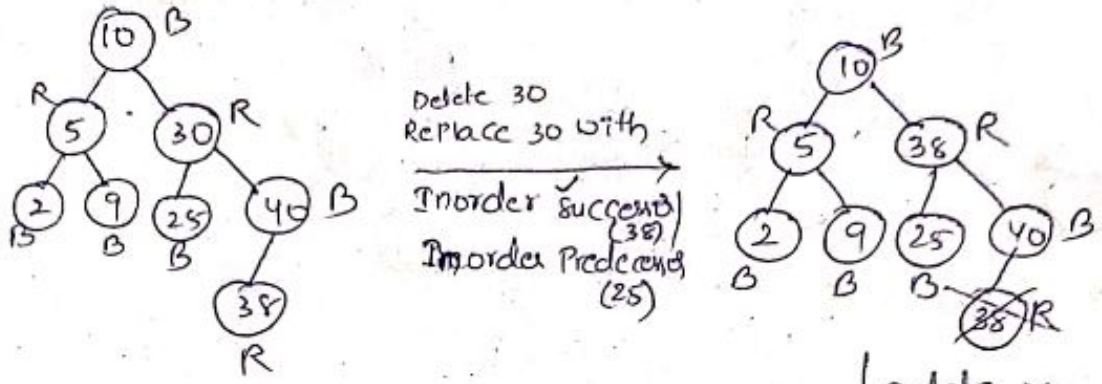


(i) Delete the node '30'

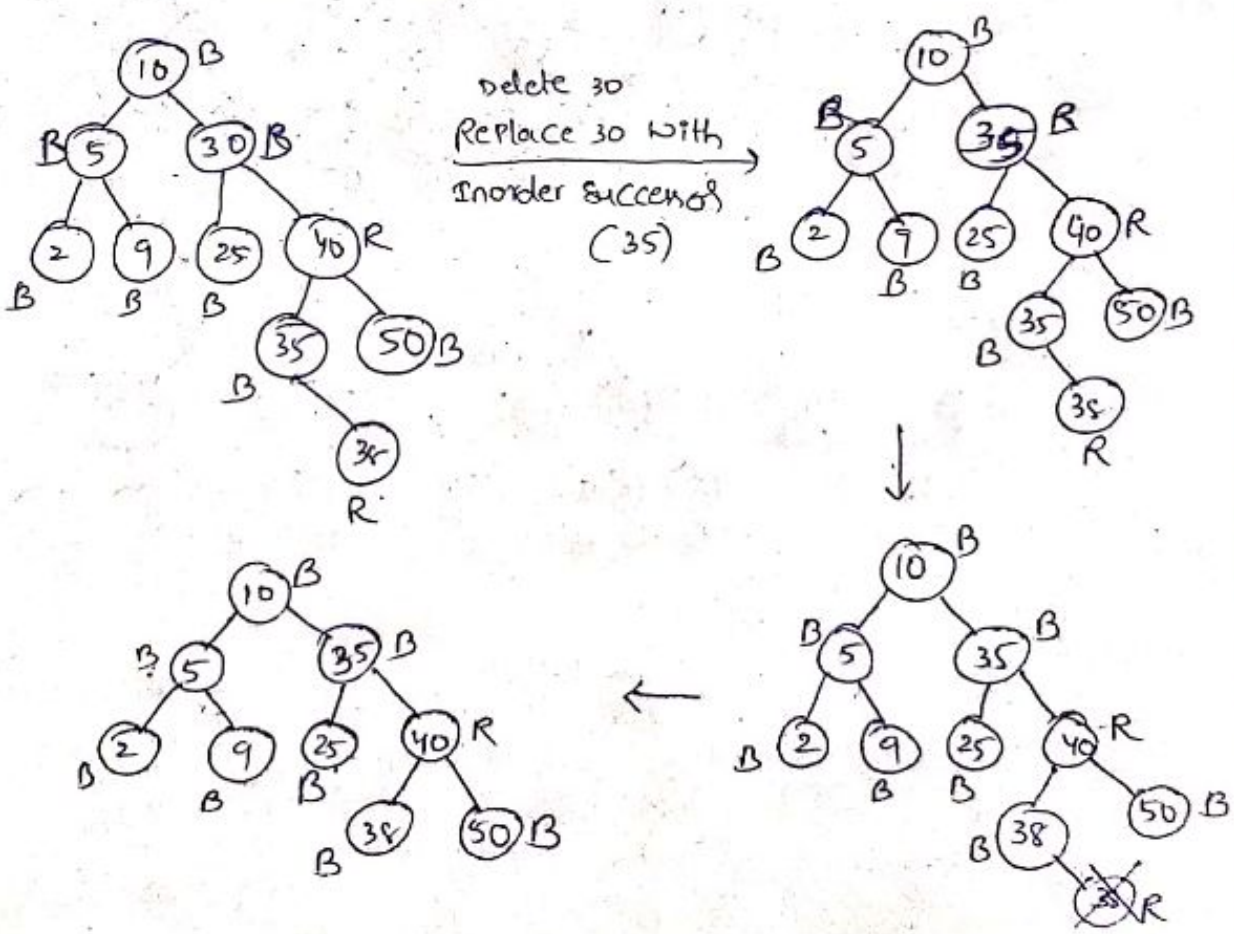
(ii) Delete the node '20'



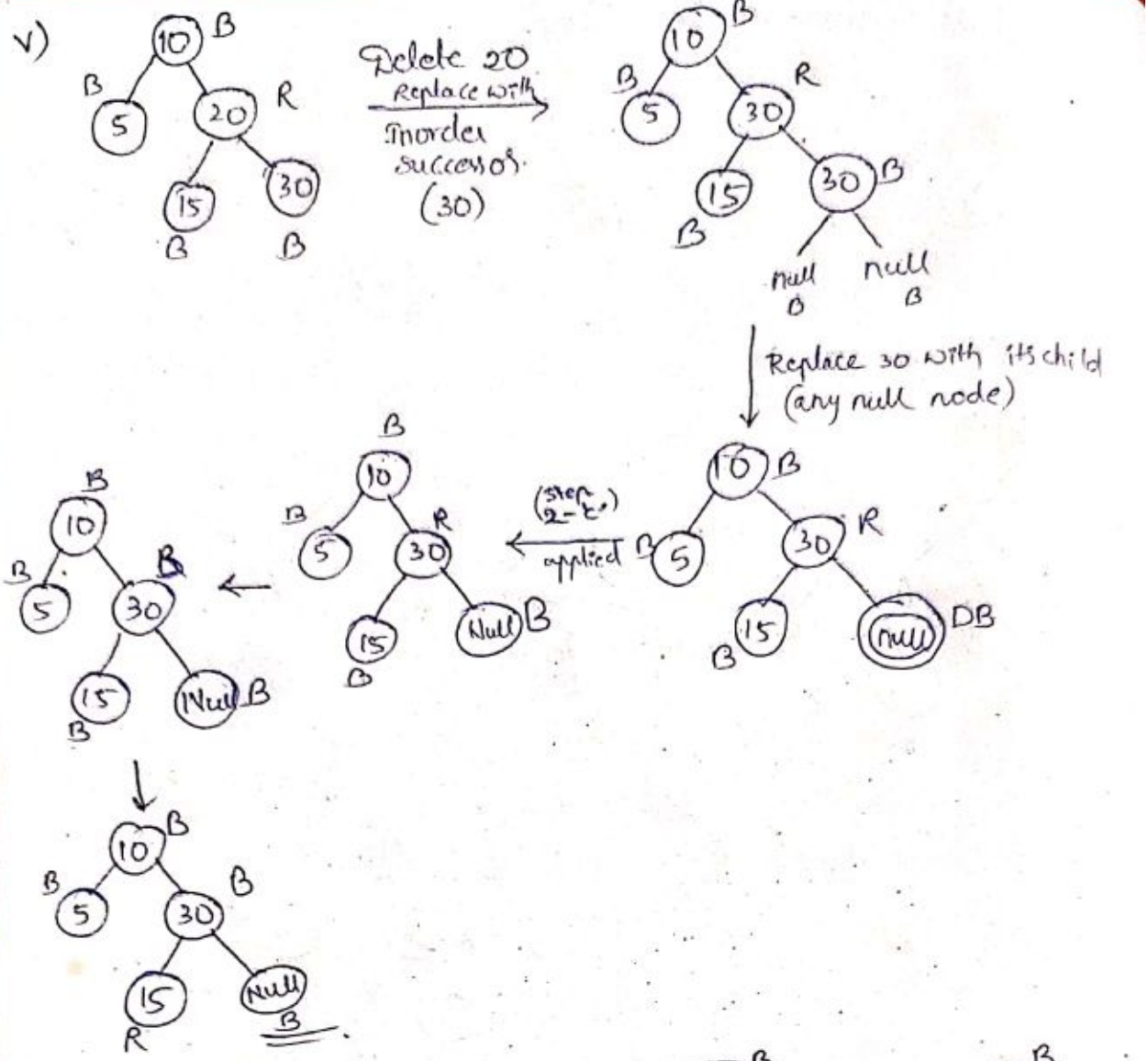
iii) Delete the node '30'



iv) Delete the node '30'



v)



vi)