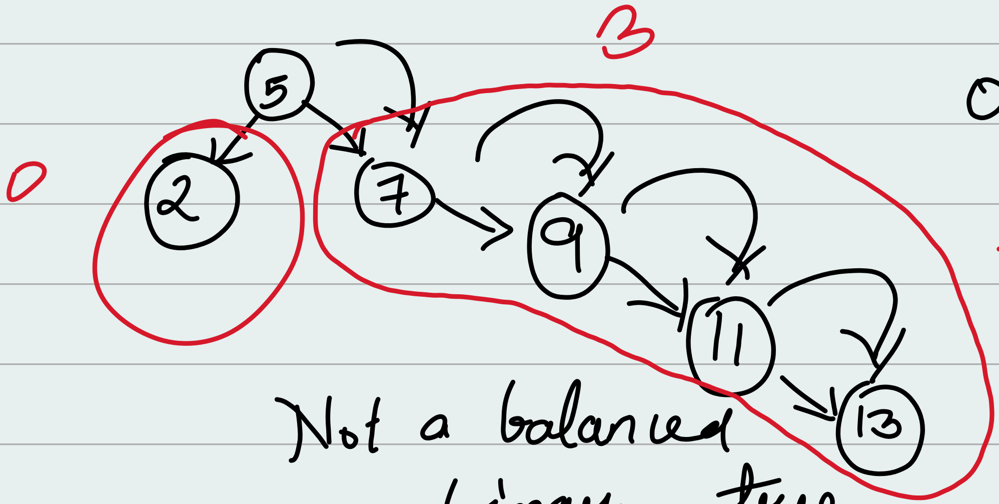
AVL Trees :~

* Not confusing

* A lot of moving parts
└→ very simple.

→ For every single node in binary Tree must be +1, -1, or 0.



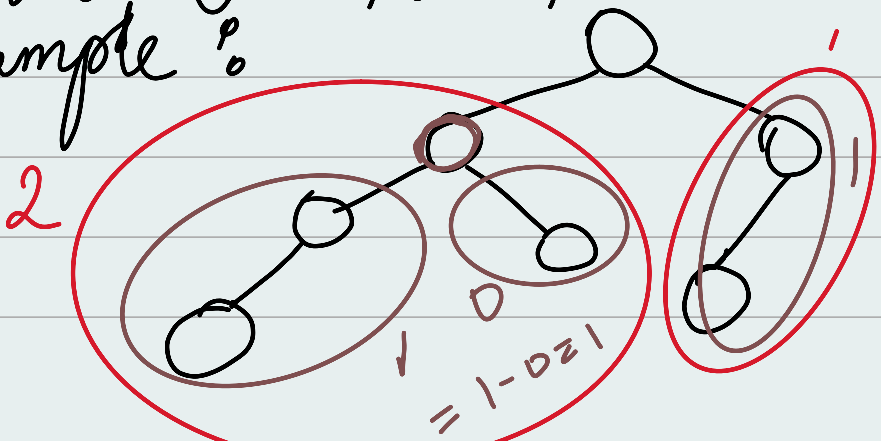
Not a balanced binary tree

O(N)

= 3-0
= 3

→ Balanced binary tree
For every single node in binary Tree must be +1, -1, or 0.

Example :



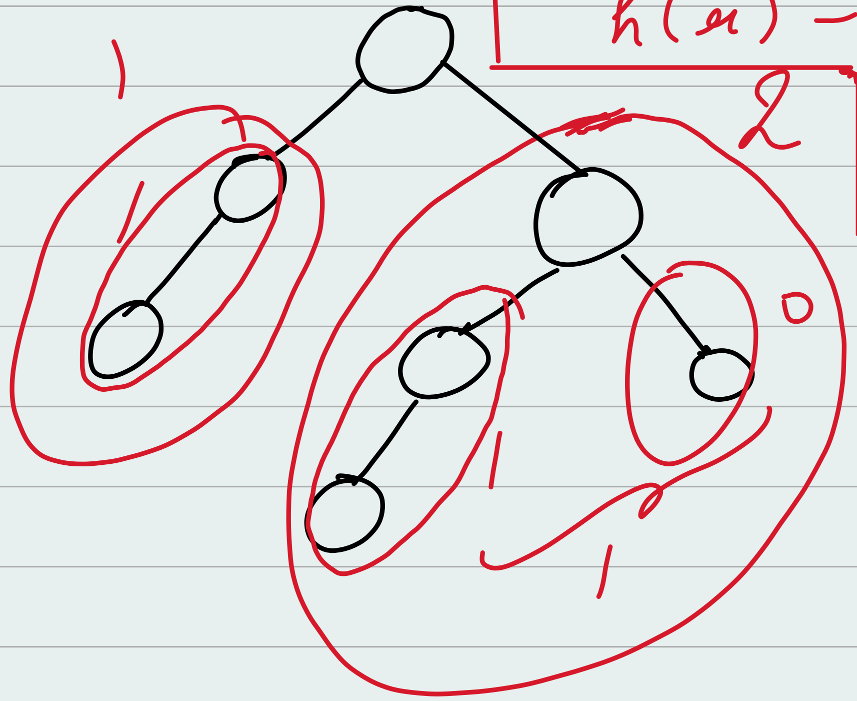
2 — 1
= 1

difference ≤ 1
+1, -1 or 0

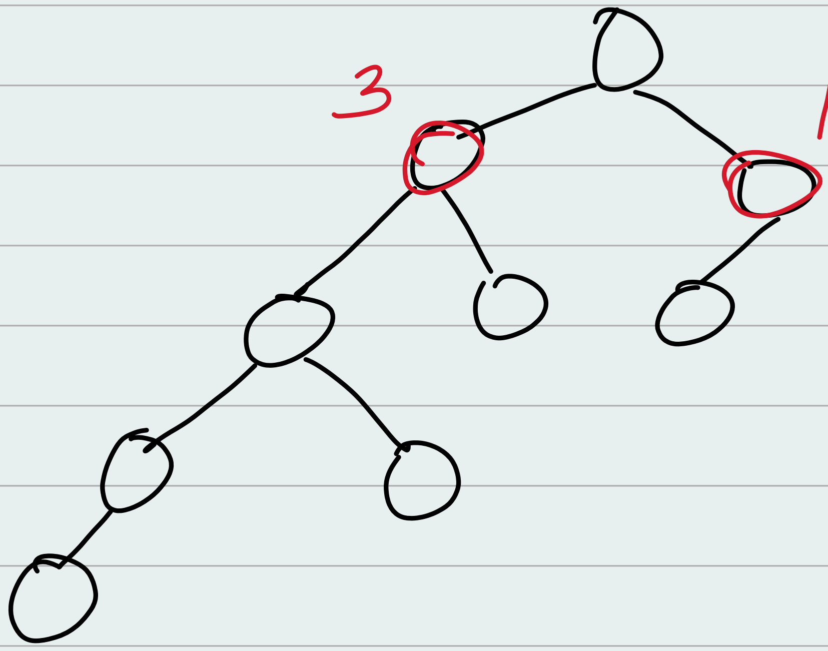
For every node, $h(l) - h(r)$
$= +1, -1, \text{or } 0$

or

$h(r) - h(l) = +1, -1, 0$



* For every Node in the tree, the difference in height of left and right subtree of that node, $\leq 1 \Rightarrow$ Balanced Tree.



$= 3-1$
$= 2$

Not a balanced Tree

Solution ?

→ Self balancing binary Tree.

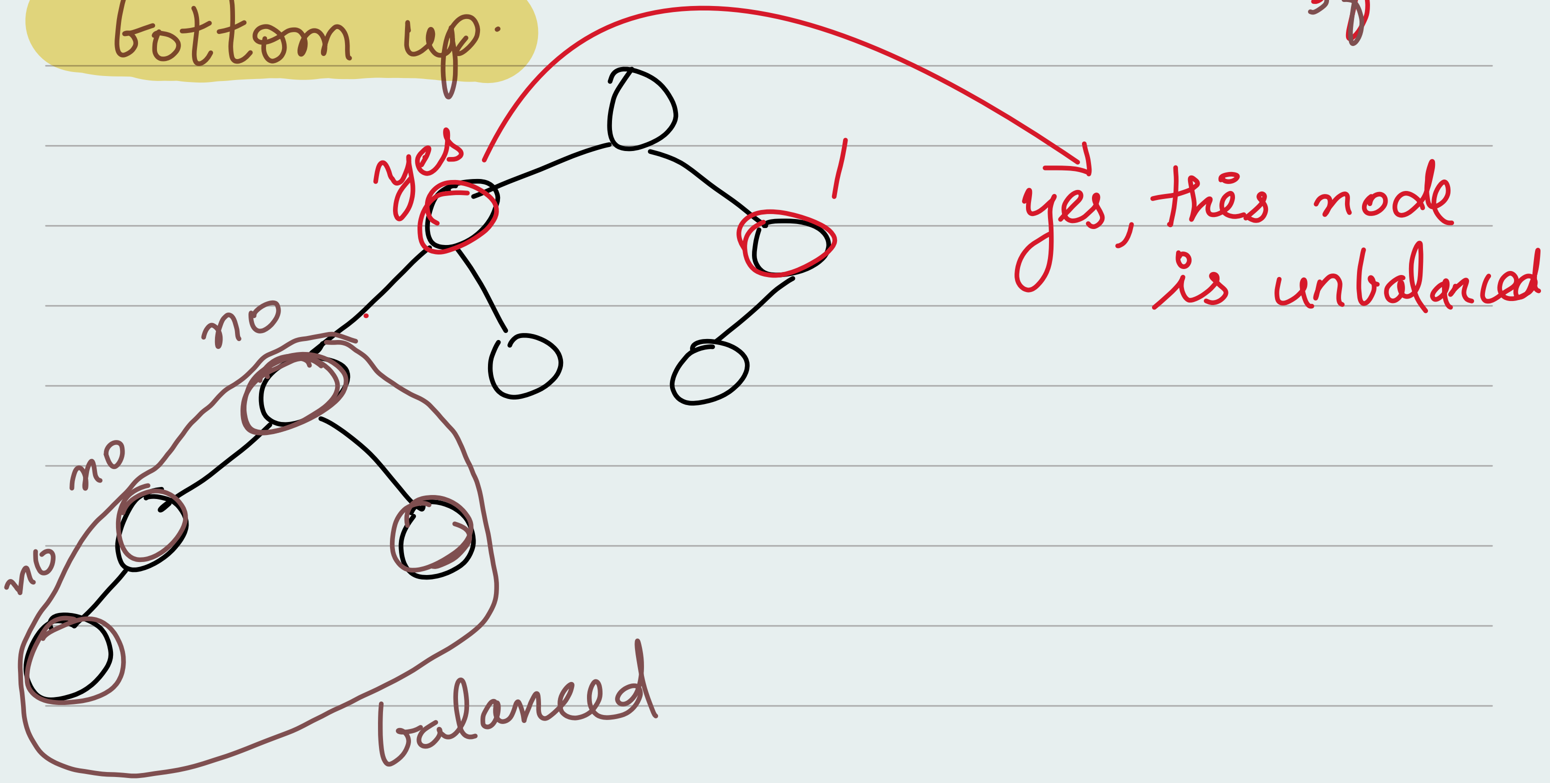
Example : AVL

(Adelson- Velski & Lendis)

Algorithm :

① Insert normally node ⓝ

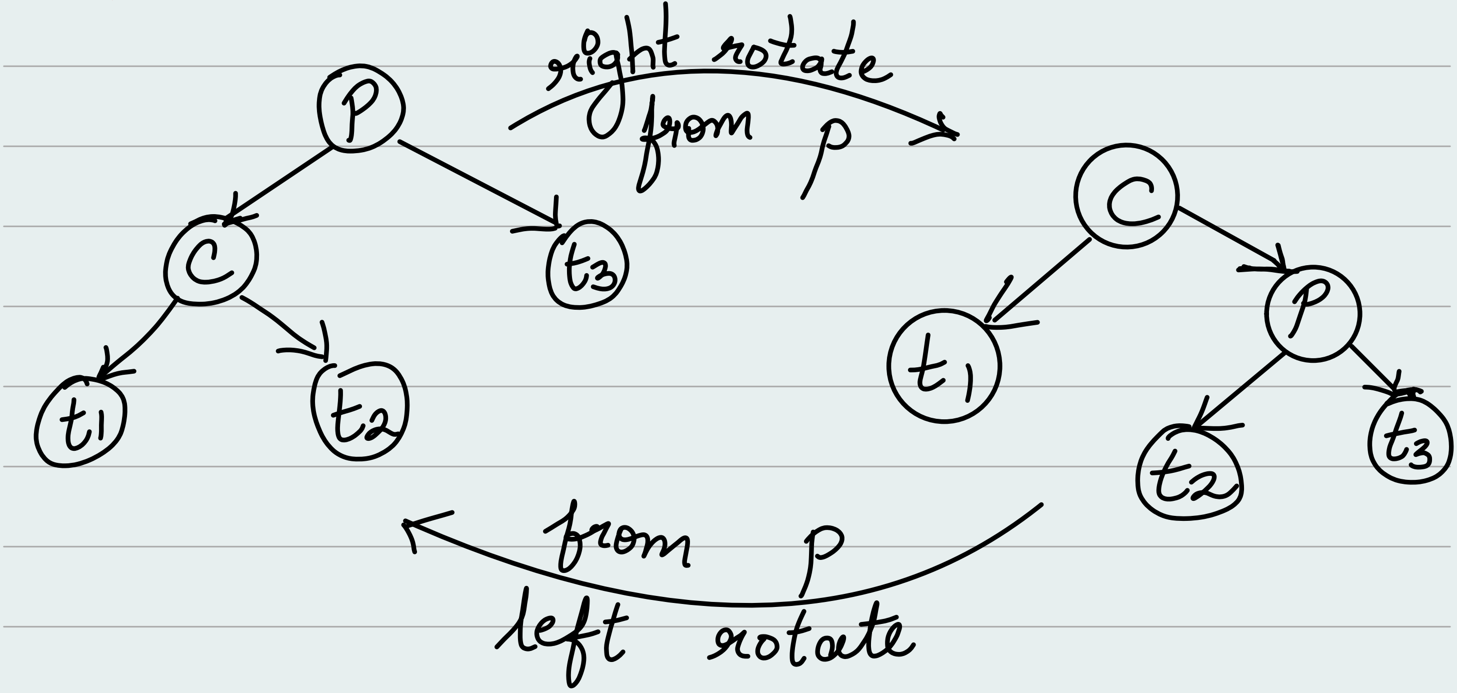
② Start from node ⓝ & find the node that makes the tree unbalanced; from bottom up.



③ Using one of the four Rules;

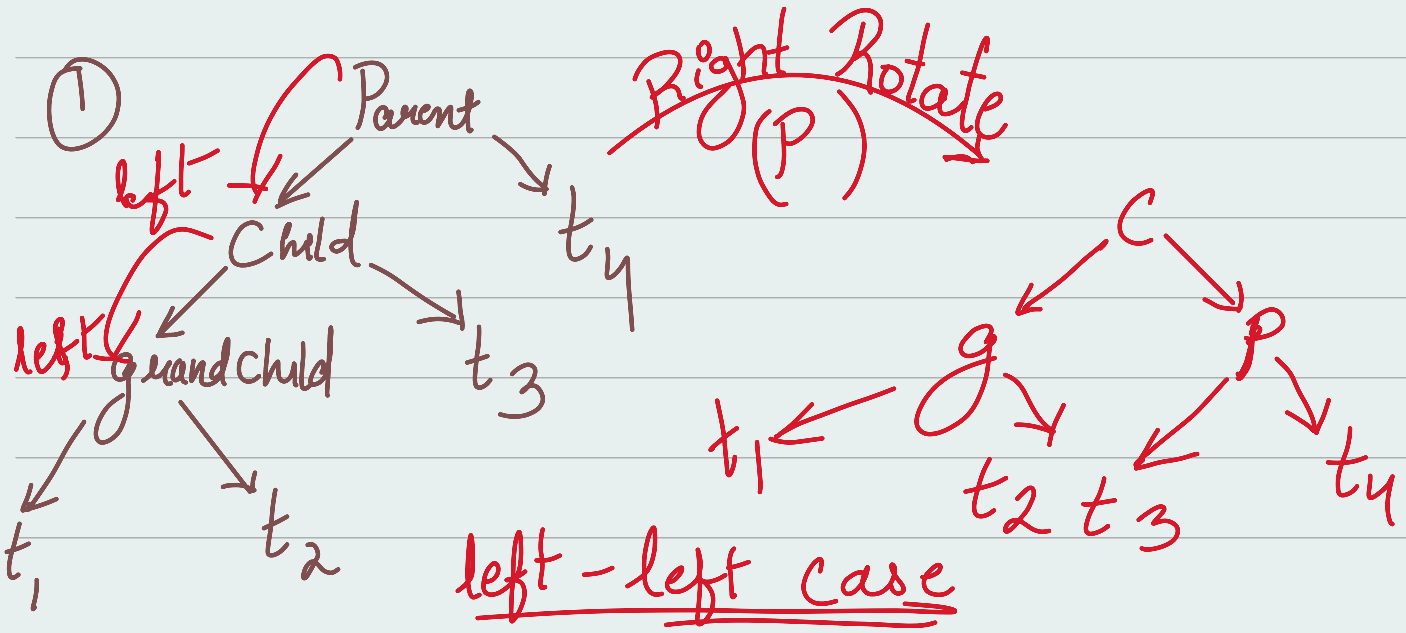
④ On rotate solve the subtree, entire tree above should be balanced in AVL trees.

We can solve the above problem either by Right rotation or left rotation



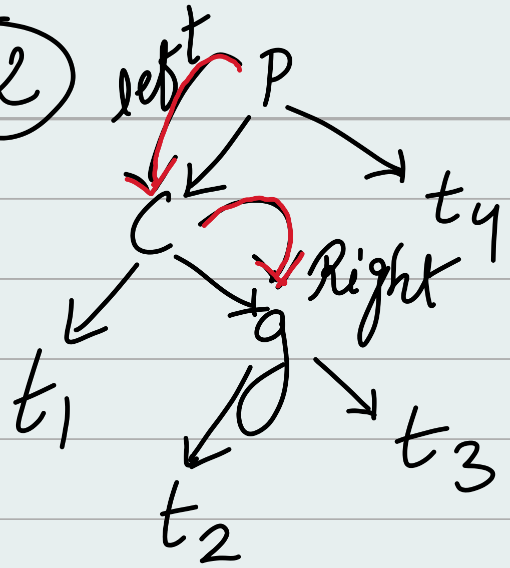
* Binary Search Tree properties are holding true for this case

4 Rules :

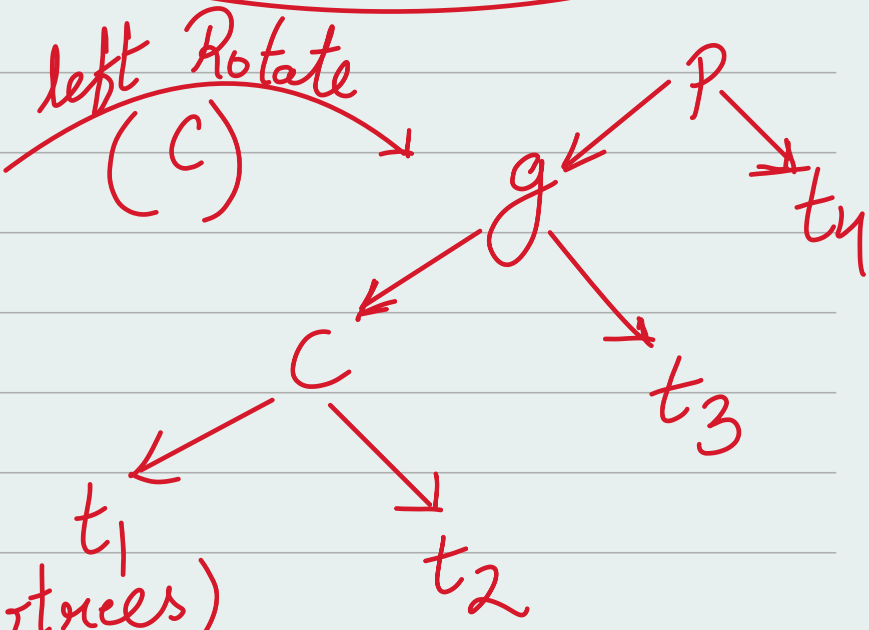


left-left case

②



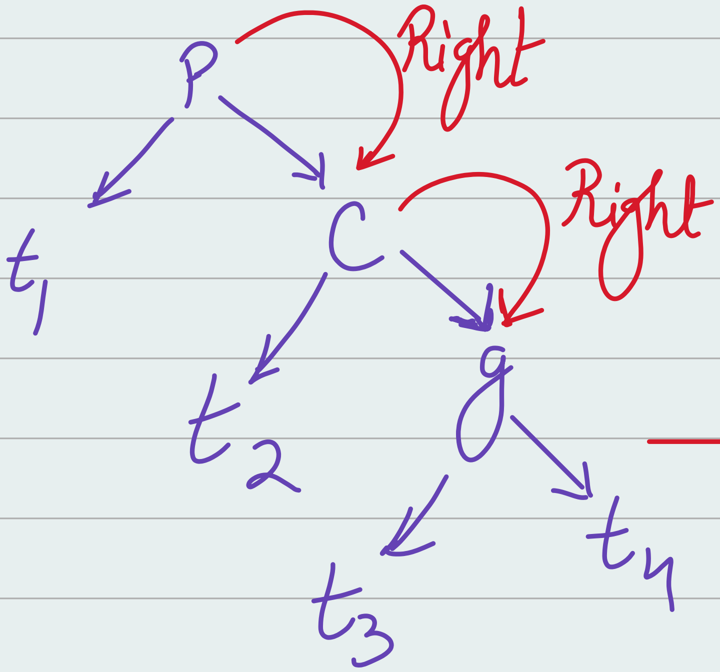
Left-Right Case



(Subtrees)
$\{t_1, t_2, t_3, t_4\}$

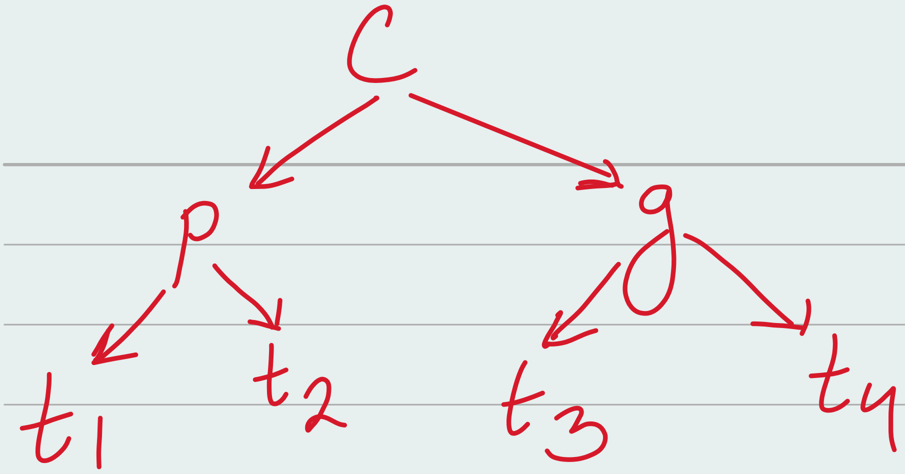
After this Right Rotate p.

③

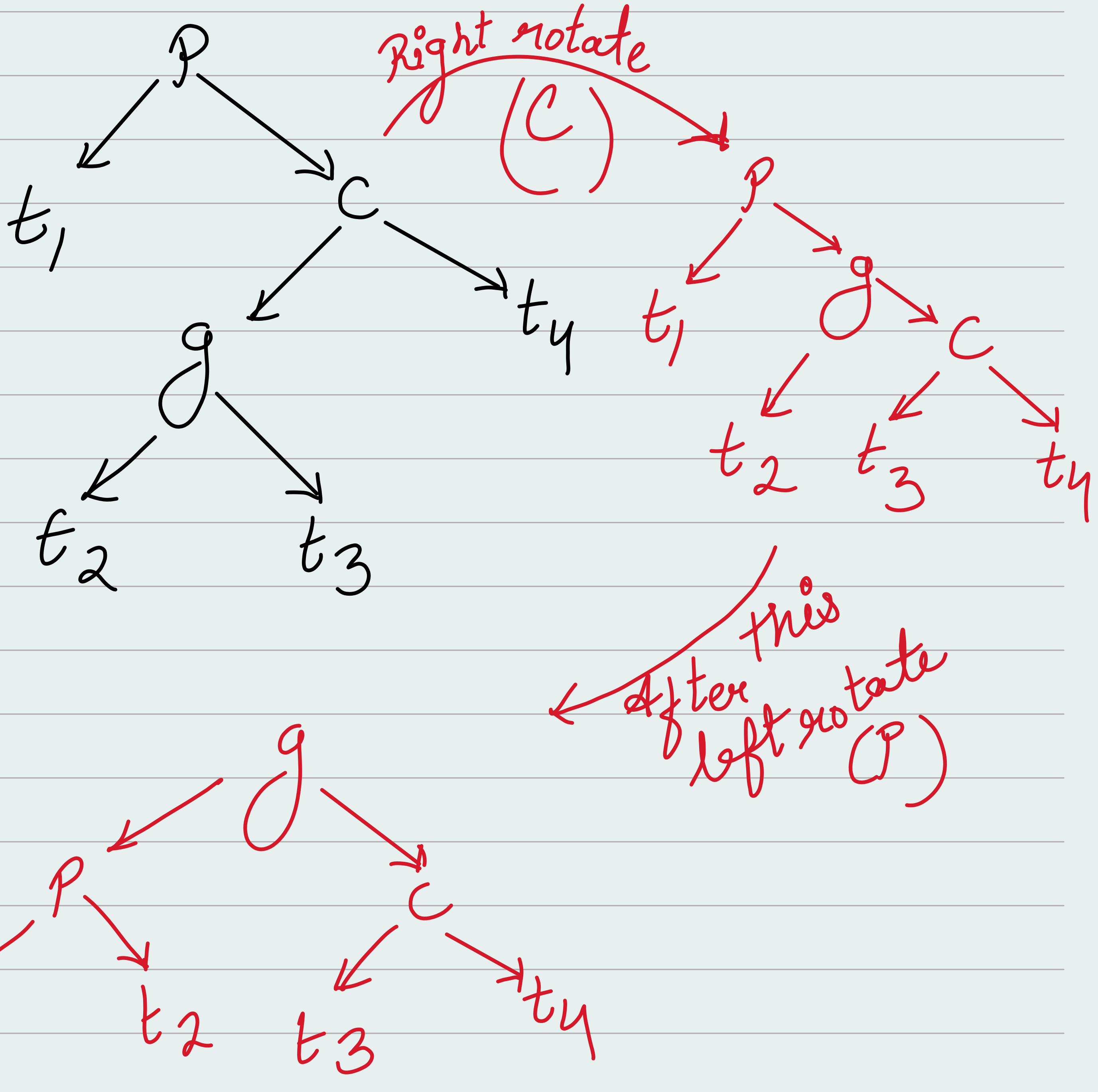


Right-Right Case

left rotate on
(P)

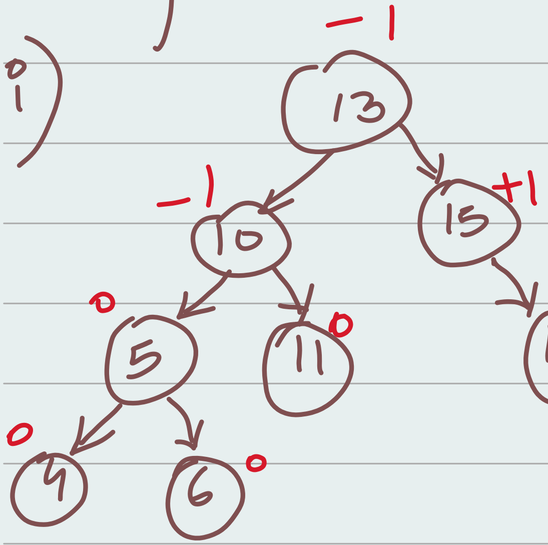


④ Right - left Case

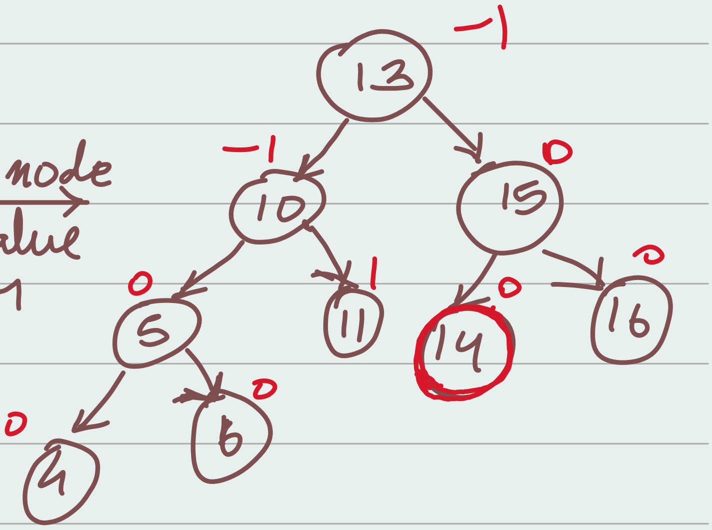


Example :

i)

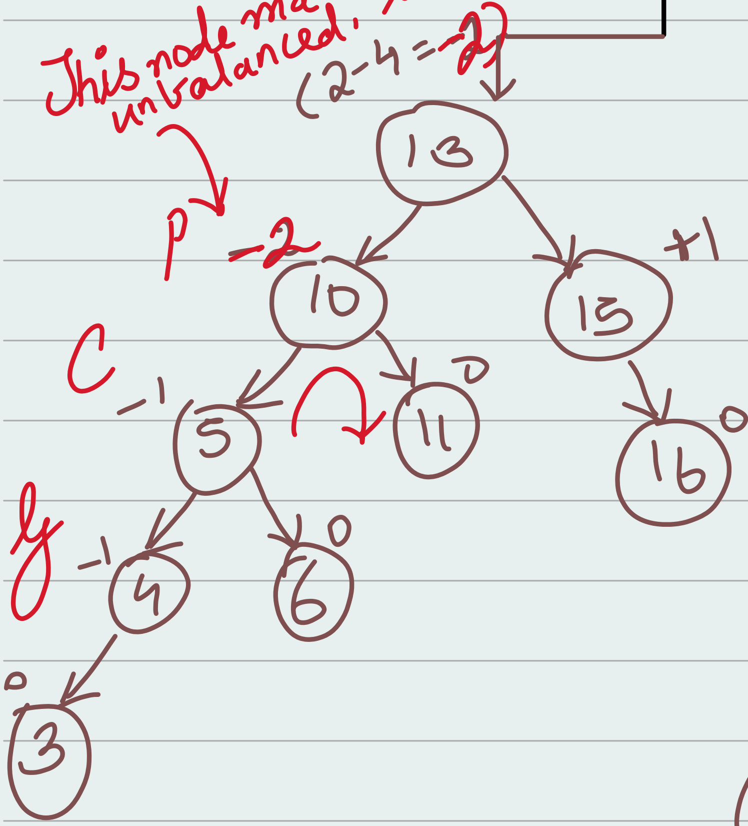


insert node with value 14

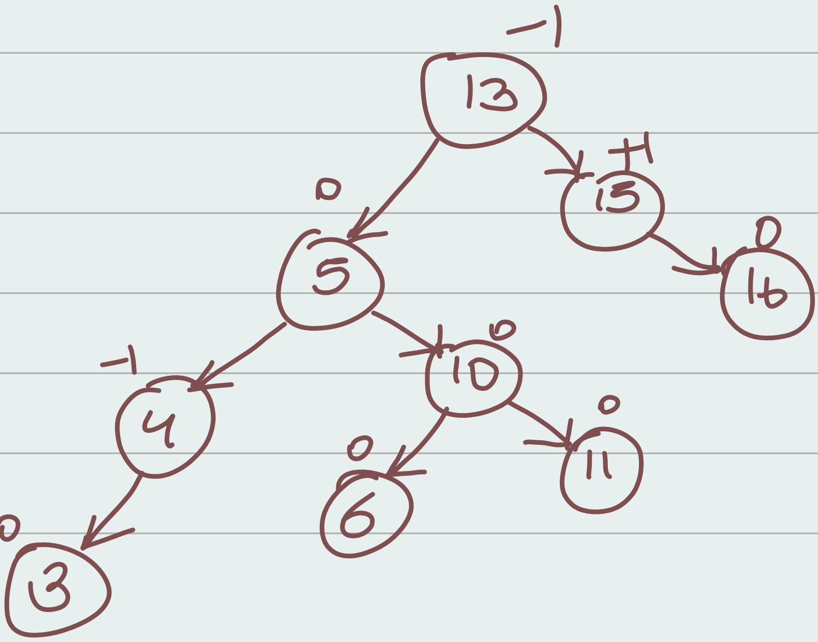


Now, insert node with value 3

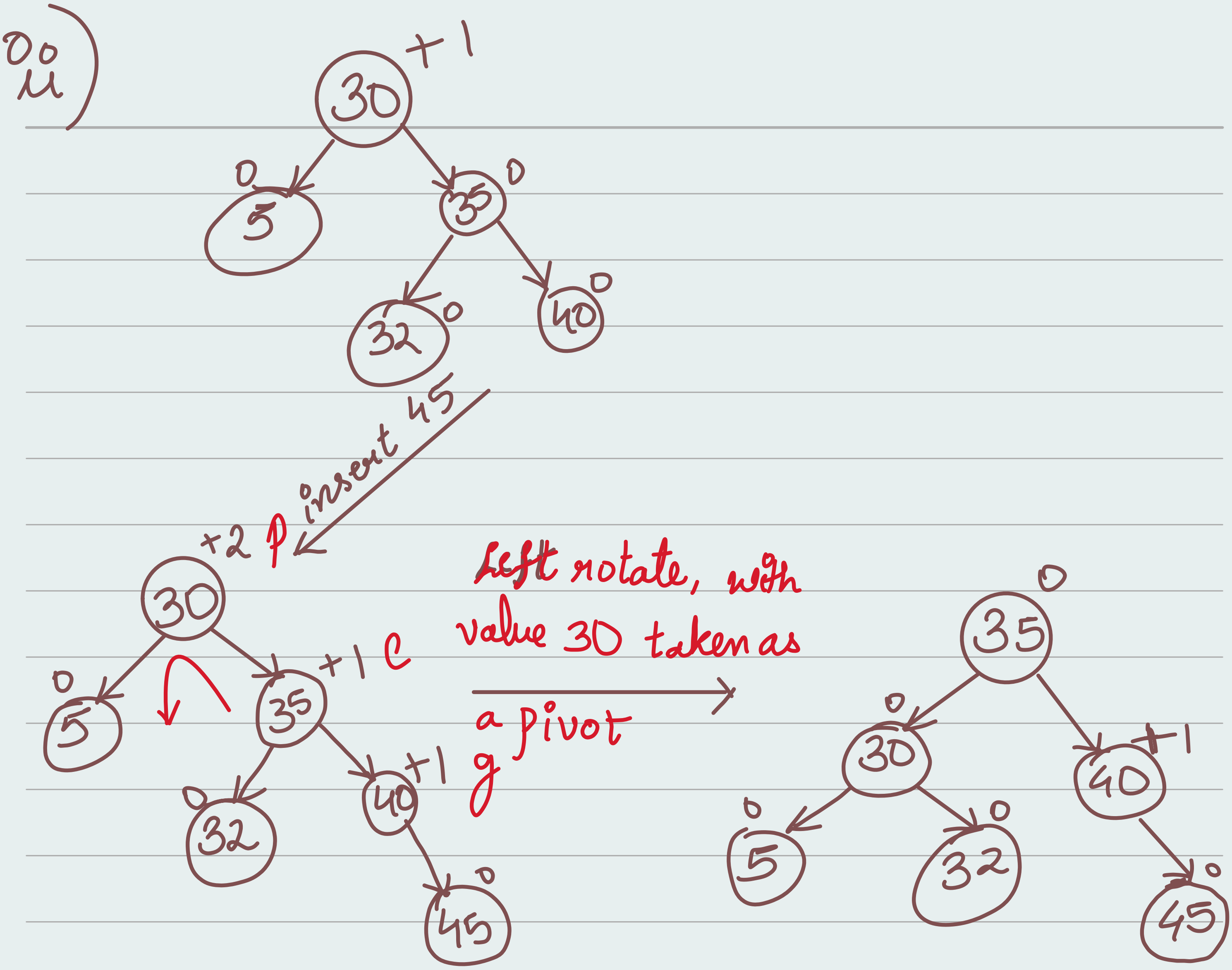
This node made it unbalanced, so AVL. (2-4=-2)



Right Rotate, node with value 10 as pivot.

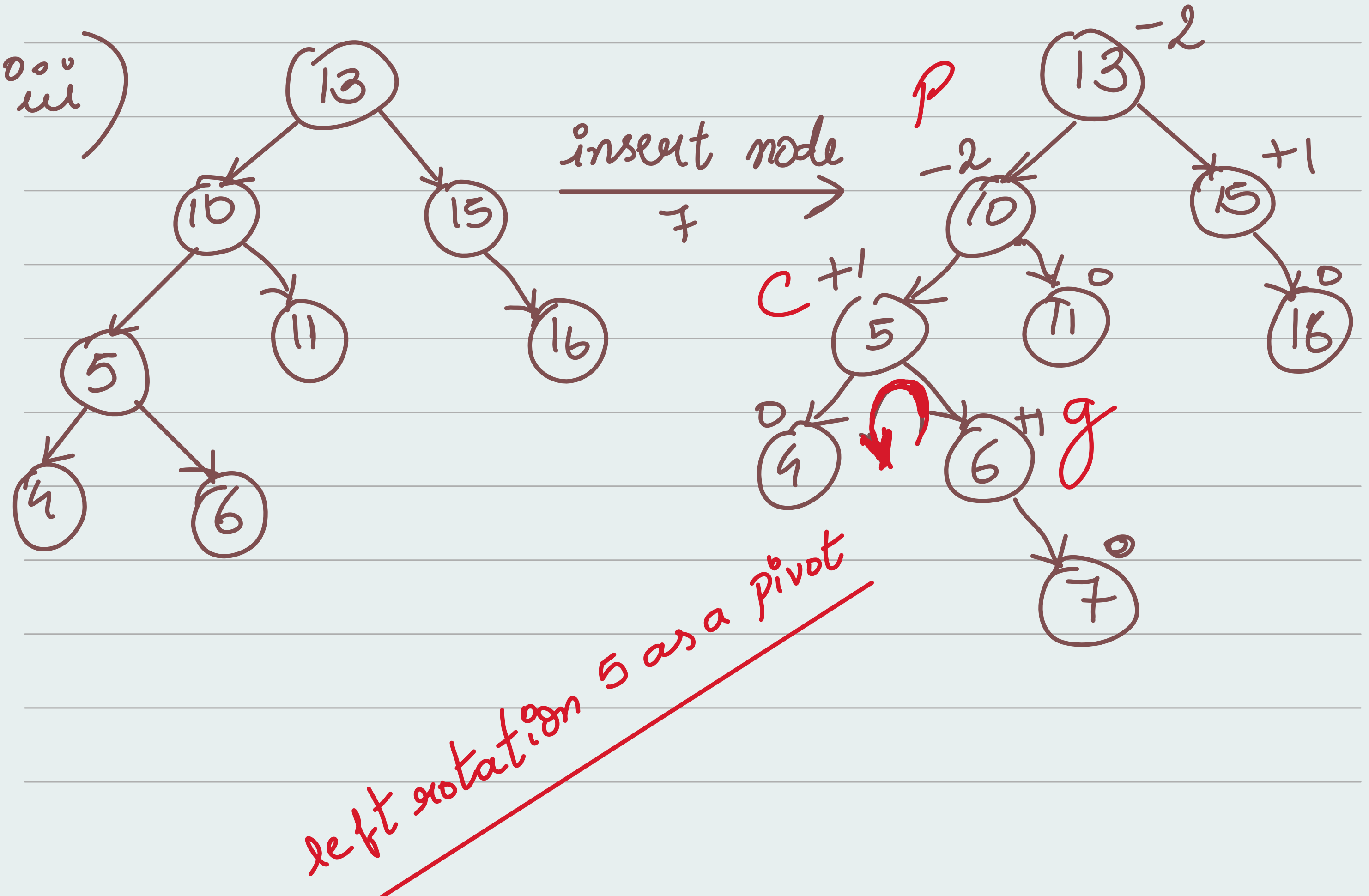


ii)

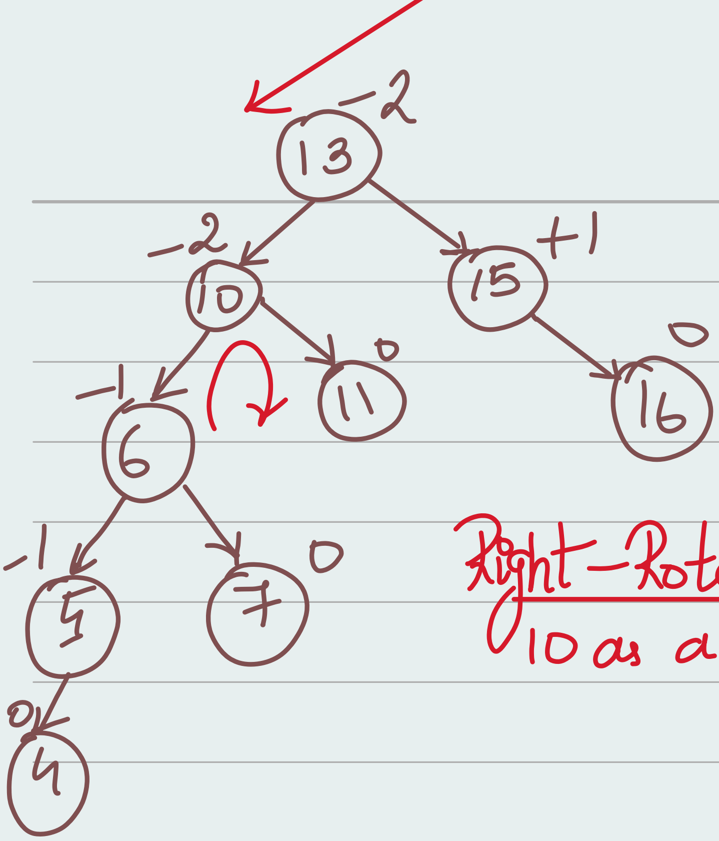


~~Left~~ Left rotate, with value 30 taken as a pivot

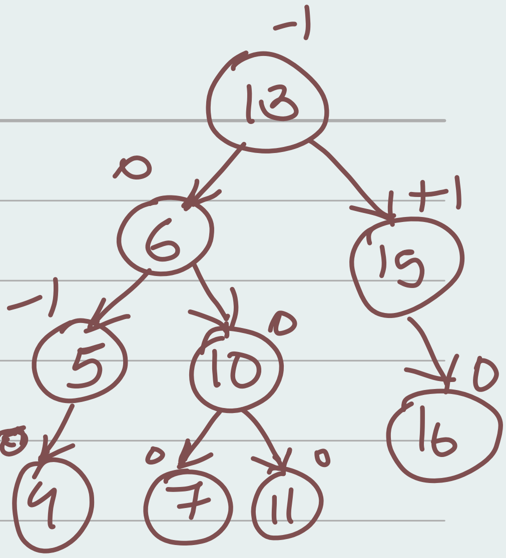
iii)



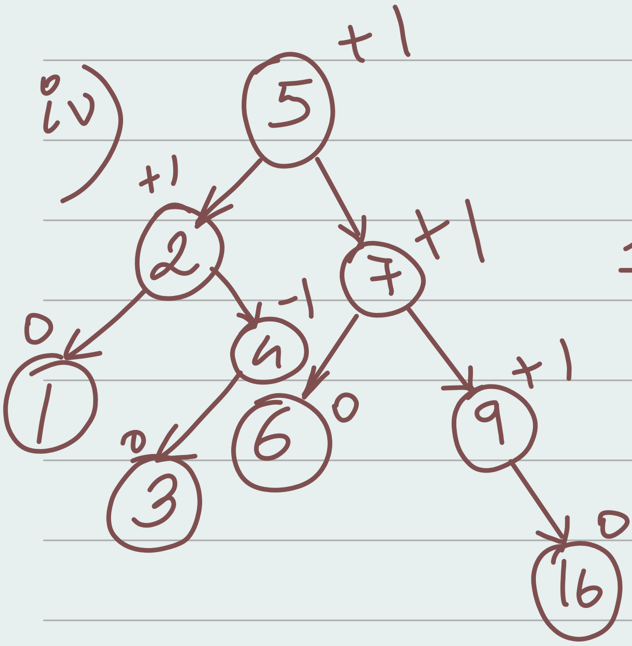
left rotation 5 as a pivot



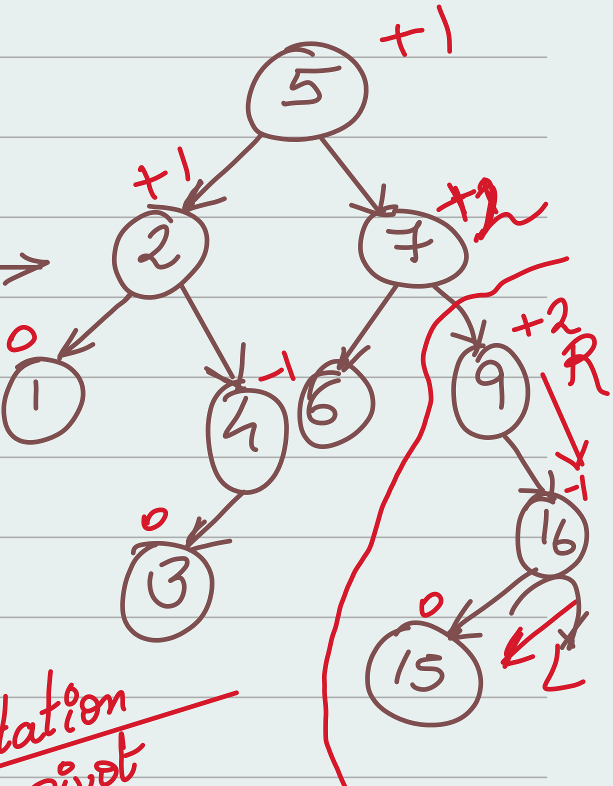
Right-Rotation taking 10 as a pivot



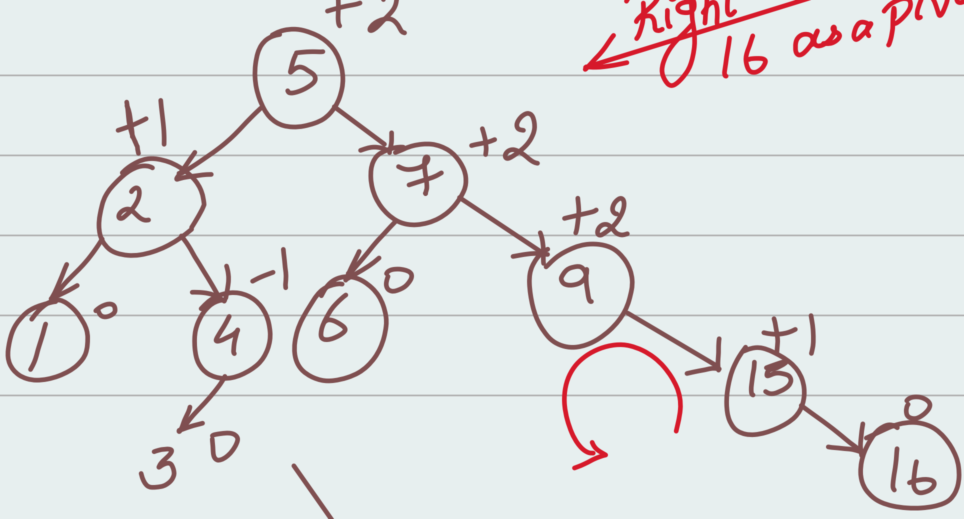
iv)



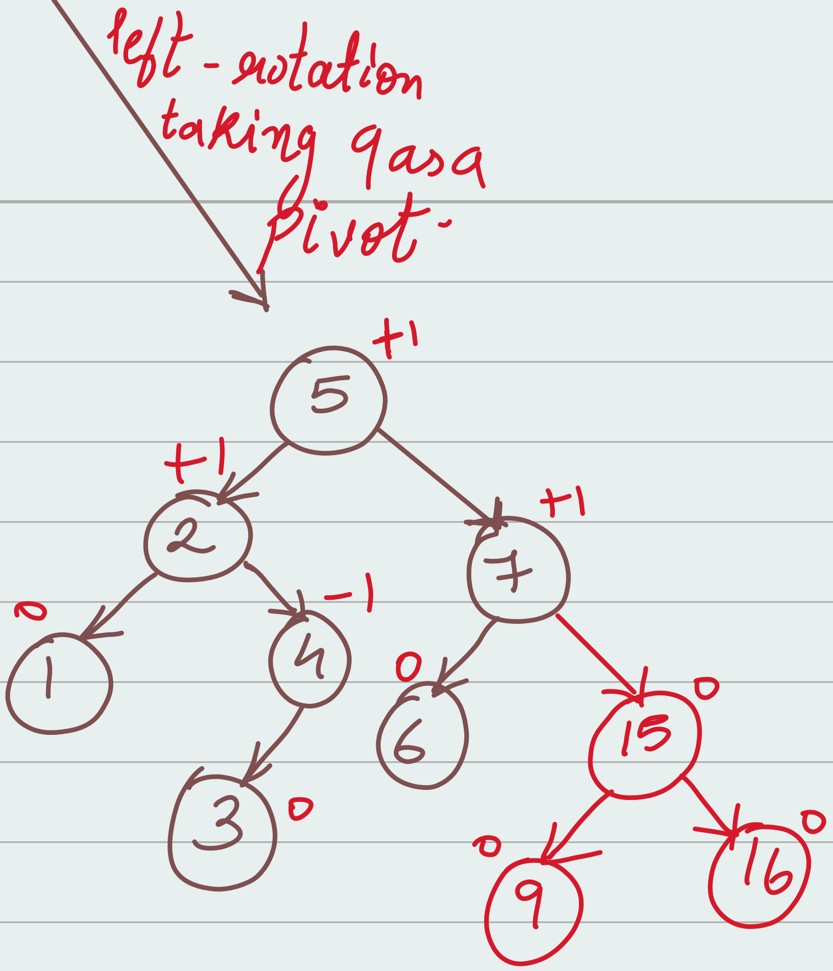
Insert 15



Right-Rotation 16 as a pivot



Solve this Then whole tree will be balanced



→ Time Complexity for adding a node.

→ log (N) as the tree is balanced.

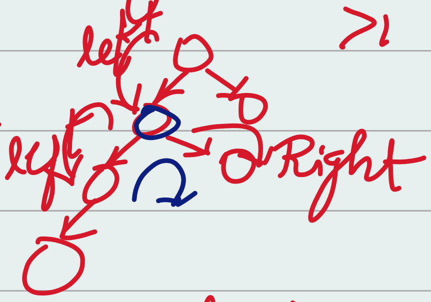
→ Rotation Time complexity is constant because we only fixing a sub tree and cause of this whole tree will be solved automatically. O (1).

$$= \log (N) + O(1)$$

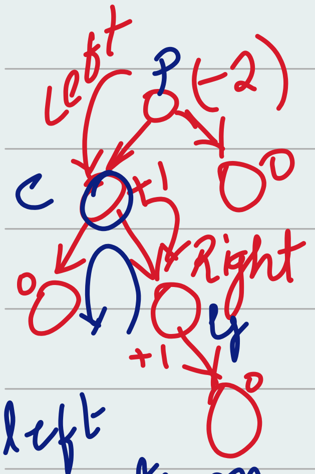
Ans = $O(\log N)$

Code: for left-left case

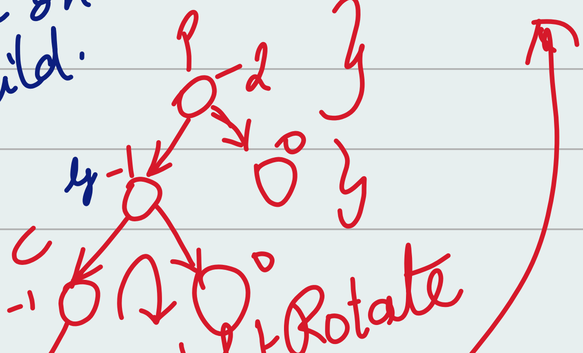
```
node height = Math max(
        height(node left), height(
         node. right)) +1;
  return rotate(node);

private Node rotate(Node node){
    if (height(node.left) - height(node.right)
                                   >1 ){
      // left heavy case
```



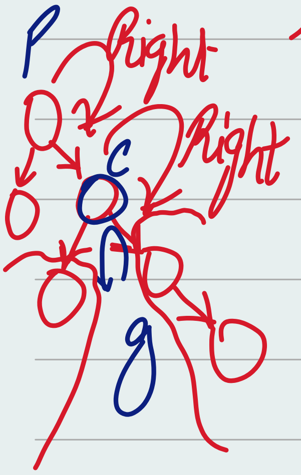
```
          if(height (node left. left) -
              height(node. left.right)>0){
              // left-left case
              return rightRotate(node)
          }
          if (height (node left.left) - height
               (node. left. right) < 0){
          node left = leftRotate(node left);
          return rightRotate(node)
          }
      }
```

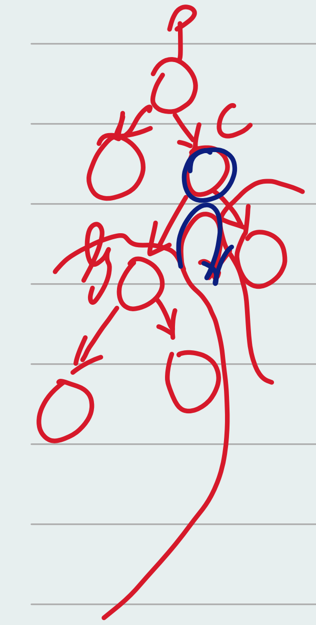


left rotate on child.

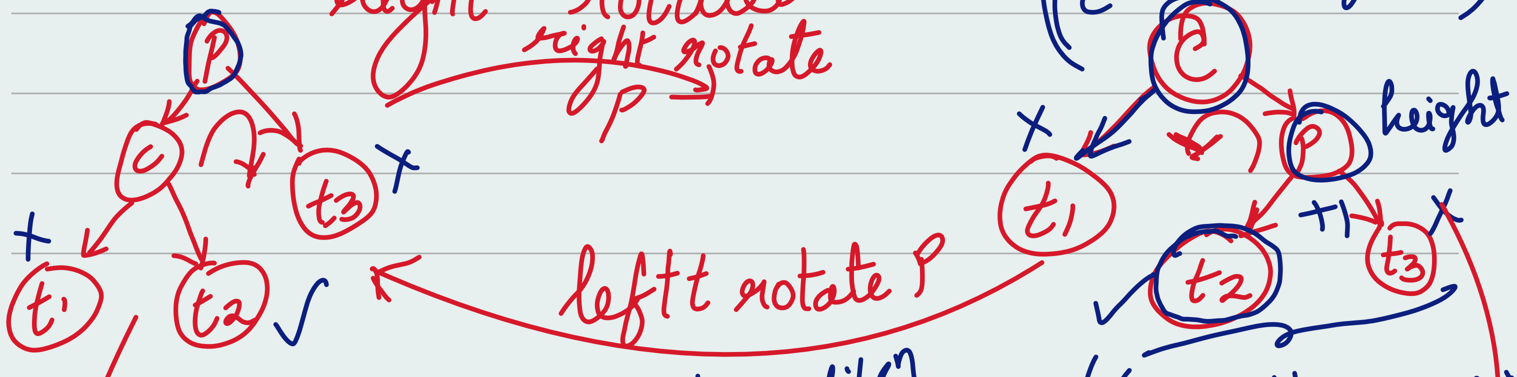


```
return node;

if (height(node.left) - height
        (node.right) < -1) {
    // right heavy
    if (height(node.right.left) -
        height(node.right.right)
            < 0) {
        // right - right case
        return leftRotate(node);
    }
    if (height(node.right.left) -
        height(node.right.right) > 0){
    node.right = rightRotate(node.right);
        return leftRotate(node);
    }
}
return node,
```





// Now create left rotate or right rotate.



t_2 will change to diff. node ((p left ; P right +1)

```
public Node rightRotate (Node p ){
    Node c = p. left;
    Node t = c. right;

    c. right = p;
    p. left = t;

    // Now update the height.

    p. height = Math. max (height (p. left),
                height (p. right) +1);

    c. height = Math. max (height (c. left),
                height (c. right) +1);
    return c;
}
public Node leftRotate (Node c){
    Node p = c. right;
    Node t = p. left;

    p. left = c;
    c. right = t;

    // Now update the height
    c. height = Math. max (height(c. left),
                height(c. right) +1)
    p. height = Math max (height (p. left),
              height(P.right) +1);
           return P;
      }

class Main {

     public static void main(string [] args){
        AVL tree = new AVL();

        for( int i=0 , i<1000;i++){
              tree. insert (i);
          }
      System. out. println(tree.height()),
      }
   }
```