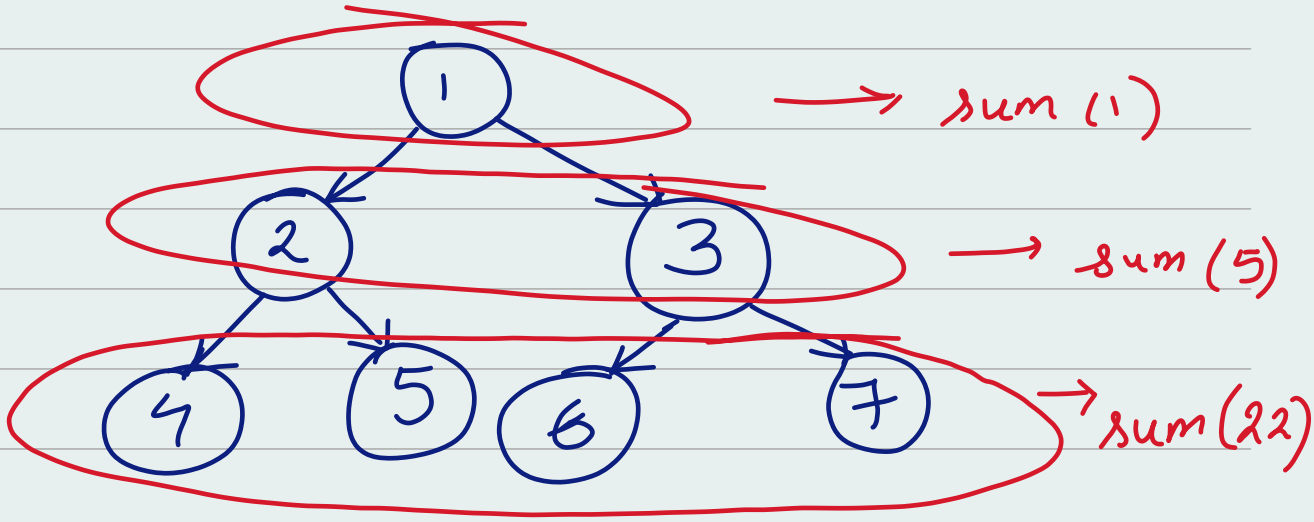
NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

Binary Tree Questions

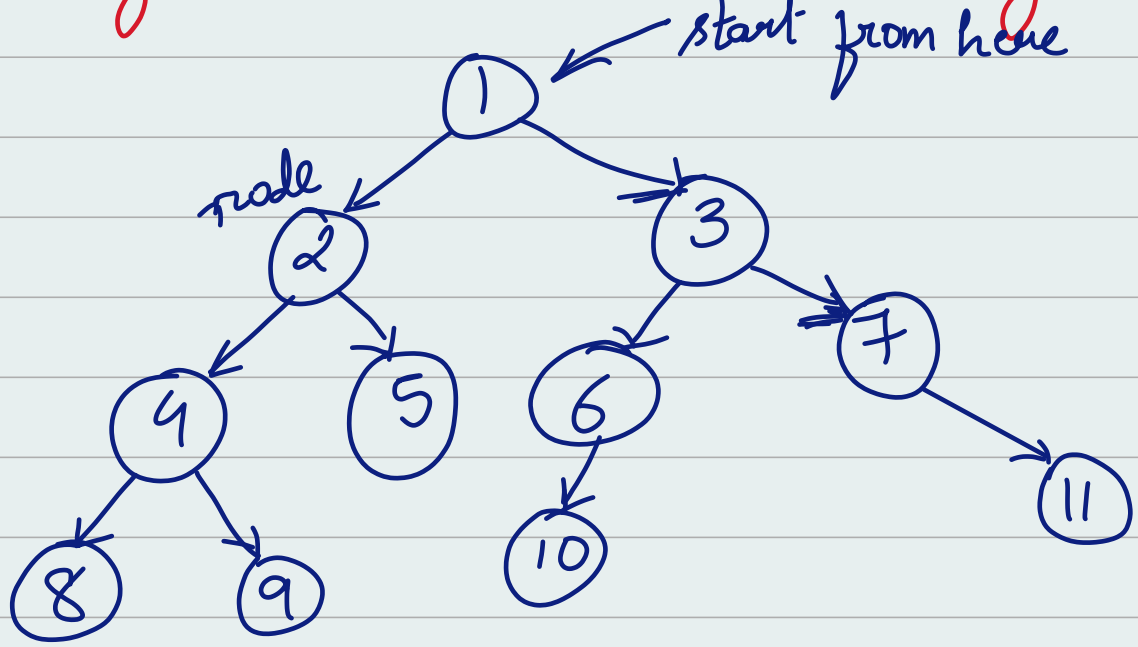
i) Breadth First Search :-



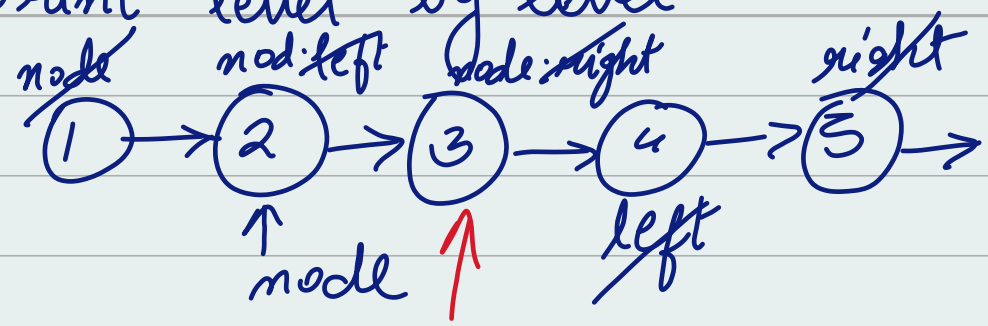
1 → 2 → 3 → 4 → 5 → 6 → 7

1) When to use this ?
→ Ans lies near the root node

2) When you are asked to search by level



→ print level by level



How to get on this node again?

FIFO → queue

store it in an array [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

every child

Print

store children of 1 in an array

similarly for 2 in an array

Ans: [[1] [2,3] [4,5,6,7] [8,9,10,11]]

Code' → result list

current level list

Class Main4

```
public List<List<Integer>>
    levelOrder(TreeNode root){
List<List<Integer>> result = new ArrayList<>();

if(root==null){
```

return result;

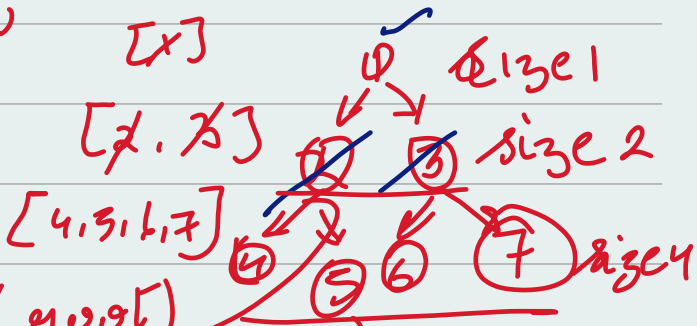
}

Queue<TreeNode> queue = new LinkedList<>();

// while we are removing items from queue we are displaying it in result.

queue.offer(root);

[1, 2, 3, 4, 5, 6, 7]



while removing 1 node (root)
node 2 & 3 got added same way for 2 & 3
after removing 2 & 3 from queue 4th, 5th, 6th & 7th node got added
size is 4 (true) → level size is also 4

while (!queue.isEmpty()){

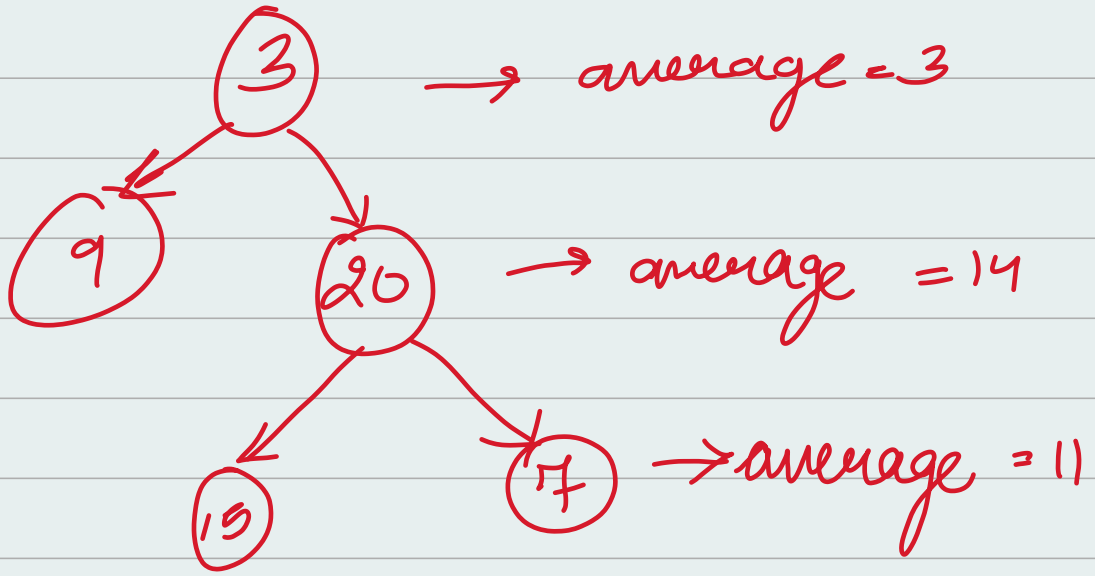
int levelSize = queue.size();
List<Integer> currentLevel = new ArrayList<>(levelSize);

i
[1 2 3

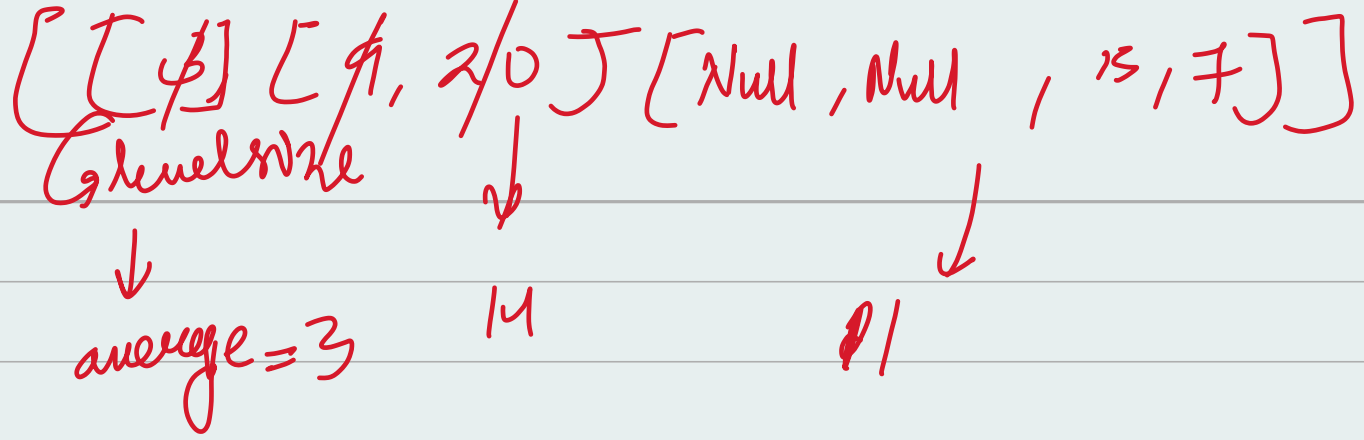
for (int i=0; i<levelSize; i++){
TreeNode currentNode = queue.poll();
// poll is used for retrieving and removing the element at the front (end of the container).

```
currentLevel.add(currentNode.val);
if(currentNode.left != null){
    queue.offer(currentNode.left);
}
if(currentNode.right != null){
    queue.offer(currentNode.right);
}
// offer inserts a new element onto the queue.
}
result.add(currentLevel);
}
return result;
}
```

Q.2) Average of levels in Binary Tree



queue size




```
public List<Double> averageOfLevels(
                    TreeNode root) {
List<Double> result = new ArrayList
                        <>();

if (root == null) {
    return result;
}

Queue<TreeNode> queue = new LinkedList
                        <>();

queue.offer(root);
while (!queue.isEmpty()) {
    int levelSize = queue.size();
    double averageLevel = 0;
for (int i = 0; i < levelSize; i++) {
    TreeNode currentNode = queue.poll();
    averageLevel = averageLevel +
                    currentNode.value;
    if (currentNode.left != null) {
        queue.offer(currentNode.left);
}
```

Google!

```
if (currentNode.right != null){
    queue.offer(currentNode.right);
}
}
result.add(averageLevel/levelSize);
}
return result;
```

[1]

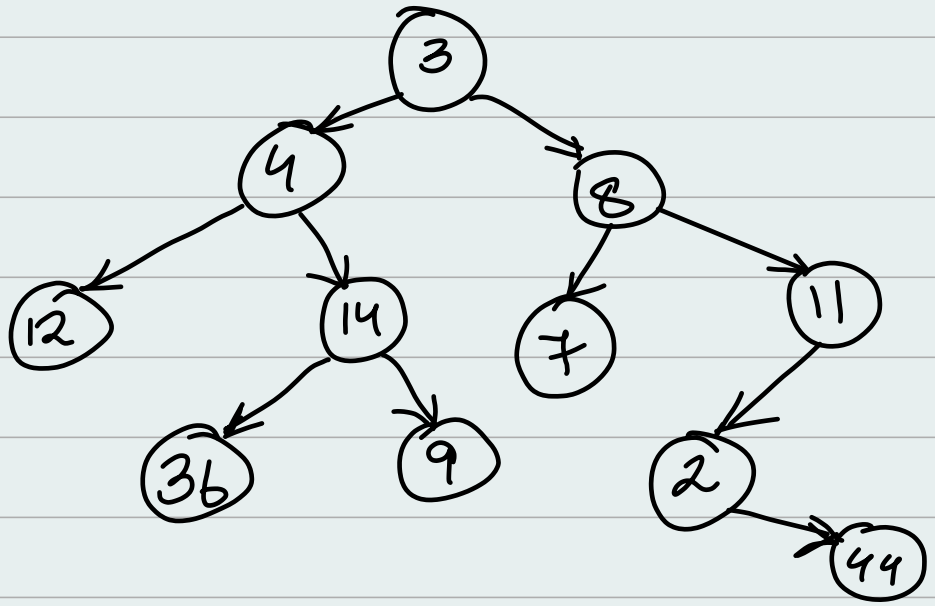
levelSize [1, 2, 3, 4, 5, 6, 7

levelSize = 2

averageLevel = 2+3 = 5 loop ends.

result = 5/2 = 2.5

Google: levelOrderSuccessor.



Ans ⇒ 14 → 7

11 → 36

reason

[8, 4, 6, 12, 14, 7, 11, 36, 9]

offer [8, 4, 6

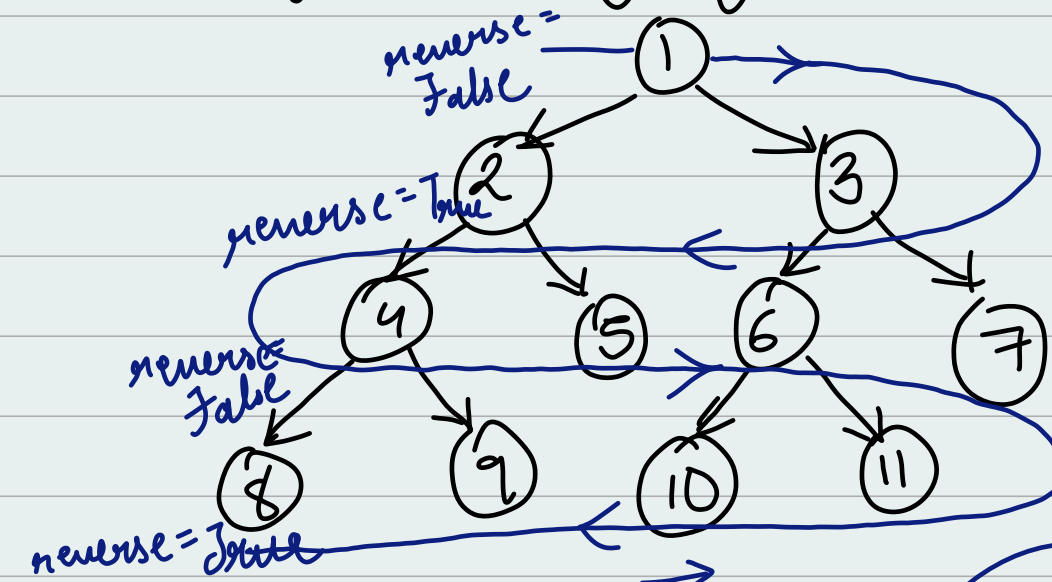
return]

Code!

```
public TreeNode findSuccessor(TreeNode root, int key){
    if(root == null){
        return null;
    }
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    while(!queue.isEmpty()){
        int levelSize = queue.size();
        TreeNode currentNode = queue.poll();
// for loop will not be used here because we are not adding the children of the node to the queue.
        if(currentNode.left != null){
            queue.offer(currentNode.left);
        }
```

```
if (currentNode.right != null){
        queue.offer(currentNode.right);
}
if(currentNode.value == key){
    break;
}
}
}
return queue.peek();
}
```

Q3) Binary Tree Zigzag level order Traversals:



Ans: 1

3, 2

4, 5, 6, 7

11, 10, 9, 8

[1, 2, 3]

[4, 5, 6, 7]

[8, 9, 10, 11]

1

3, 2

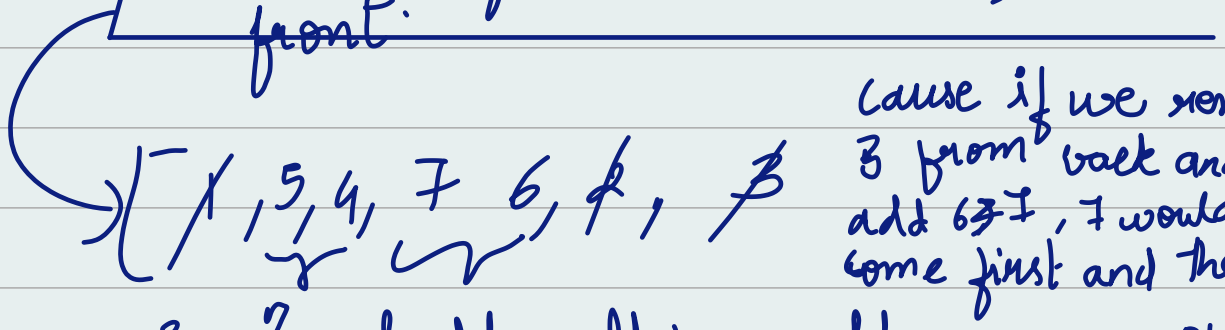
4, 5, 6, 7

11, 10, 9, 8

Deque

normal order
→ remove from front add in back.

reverse order
→ remove from back (3, 2) add in front.



cause if we remove 3 from back and add 6 & 7, 7 would come first and then 6

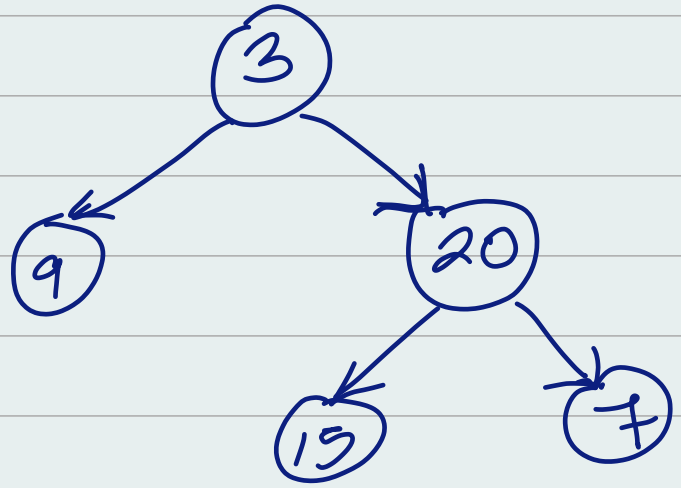
So, to tackle this problem we will take 3 right value first and then 3 left.

```
boolean reverse = false;
while (!queue.isEmpty()){

    int levelSize = queue.size();

    List<Integer> currentLevel = new
                        ArrayList<>(levelSize);

    for( int i=0, i< levelSize ; i++){
        if( !reverse ){

            TreeNode currentNode = queue
                        .pollFirst();
```

```
                currentLevel.add(currentNode.
                                    value);
                if (currentNode.left != null){
                    queue.addLast(currentNode.
                                    left);
                }
                if(currentNode.right != null){
                    queue.addLast(currentNode.
                                    right);
                }
            }
            else {
                TreeNode currentNode = queue.
                                    pollLast();
                currentLevel.add(currentNode.
                                    value);
                if (currentNode.right != null){
                    queue.addLast(currentNode.
                                    right);
                }
                if(currentNode.left != null){
                    queue.addLast(currentNode.
                                    left);
                }
            }
        }
        reverse = !reverse;
    }
```

```
return result;
}
```

04) Binary Tree level order Traversal II.



Output ⇒ [[15,7],[9,20],[3]]

normal level order Traversal

[3, 9, 20, null, null, 15, 17]

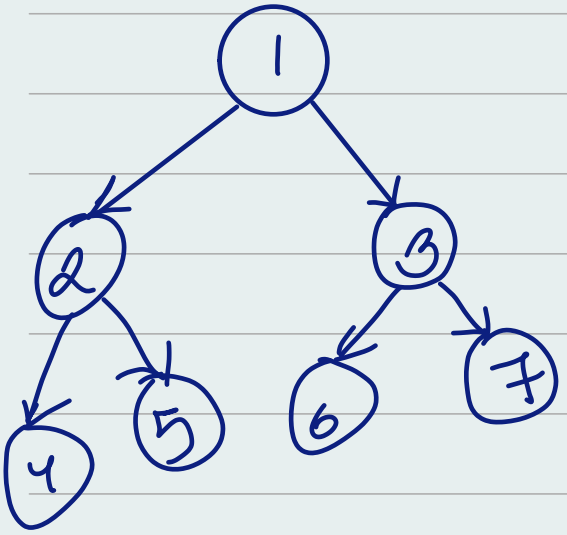
[[3], [9, 20], [15, 17]]

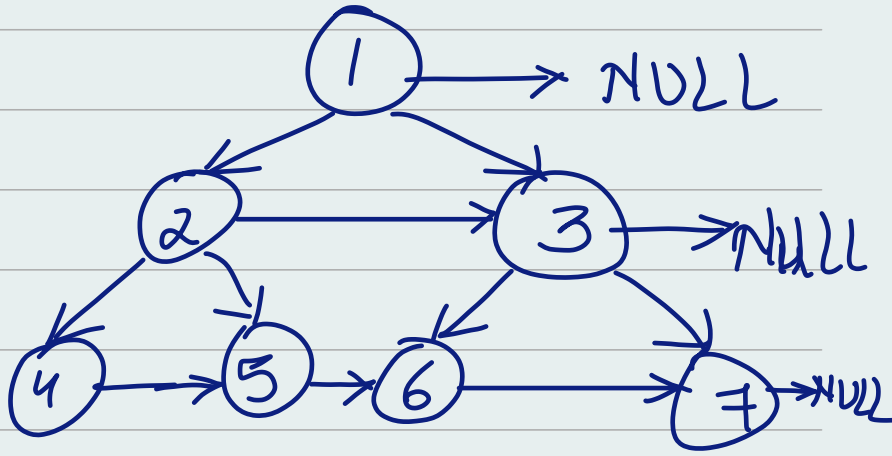
result.add(0, currentLevel)

Instead of adding currentLevel to the back of the result like queue, it inserts at the beginning. This is the key step to reverse the order.

Rest of the code is same as LevelOrder tree

05) Populating Next Right pointers in each node.

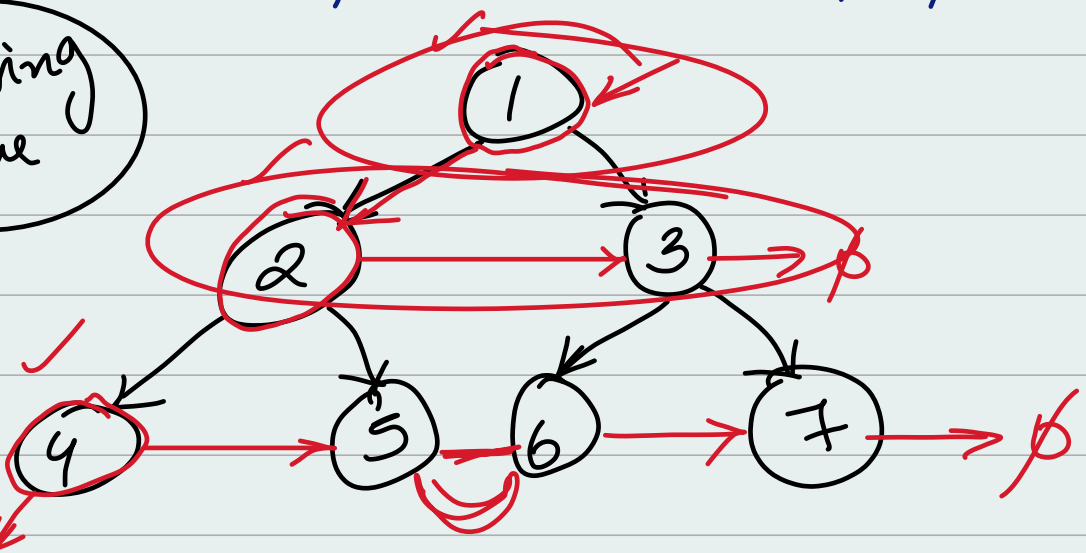




Input : root = [1,2,3,4,5,6,7]

output: [1,#,2,3,#,4,5,6,7,#]

not using queue



ϕ

leftmost

while l=null

keep moving forward

2

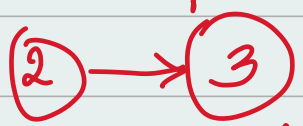
4

↓

ϕ

current

1



level 1

level 2

current.left.next = current.

1 2.next = 1.right right;

current.left.next = current.right

2. 4.next = 2.right.

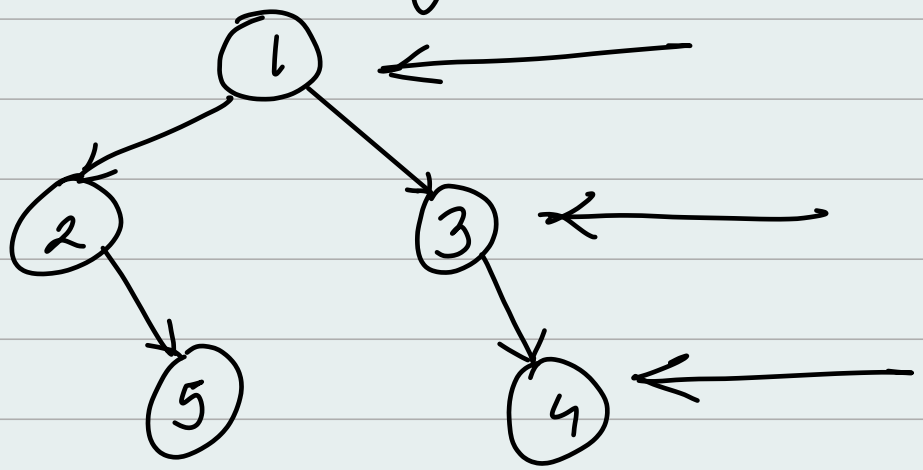
To connect 5 → 6 we have to move 2 → 3 with condition current.next != null then current.right = 5

{2 → 3 & 2 → 5}

```
class Solution{

    public Node connect (Node root){
        if (root == null){
            return null;
        }
        Node leftMost = root;

        while (leftMost.left != null){

            Node current = leftMost;

            while( current != null ){
                current.left.next = current.
                                    right;
                if (current.next != null){
```

```
            current.right.next =
                current.next.left;
        }
        current = current.next;
    }
    leftMost = leftMost.left;
}
return root;
}
}
```

86) Binary Tree Right Side View.



Input : root = [1, 2, 3, null, 5, null, 4]

Output : [1, 3, 4]

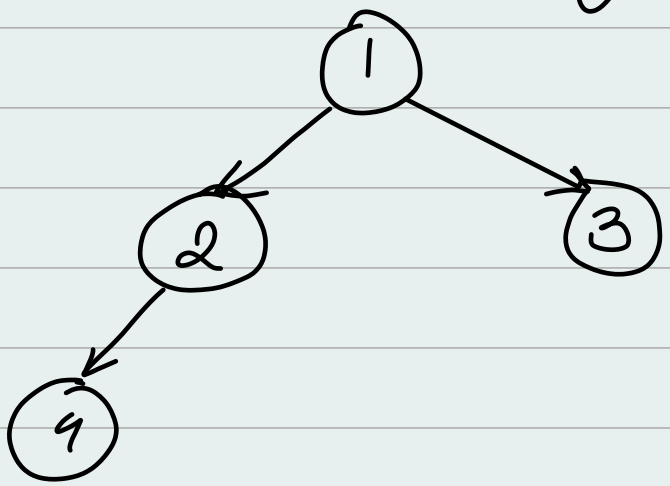
Solving this by level order traversals and only taking last element of every level.

Code:

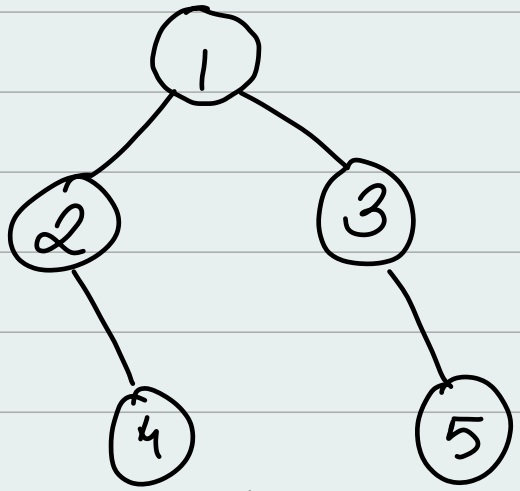
```
Queue<TreeNode> queue = new
                    LinkedList<>();

    queue.offer(root);
while(!queue.isEmpty()){
    int levelSize = queue.size();

    for(int i=0; i<levelSize; i++){
        TreeNode currentNode = queue
                                .poll();
        if(i == levelSize-1){
            result.add(currentNode.val);
        }
        if(currentNode.left != null){
            queue.offer(currentNode.left);
        }
        if(currentNode.right != null){
            queue.offer(currentNode.right);
        }
    }
}
```

```
    return result;
}
```

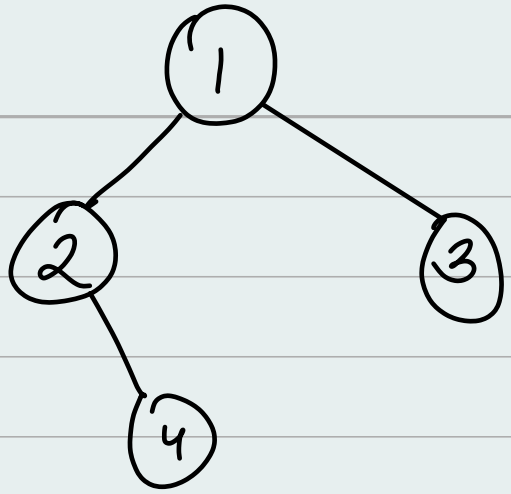
86) Cousins in Binary Tree



input = [1, 2, 3, 4] , x = 4 , y = 3
output = false



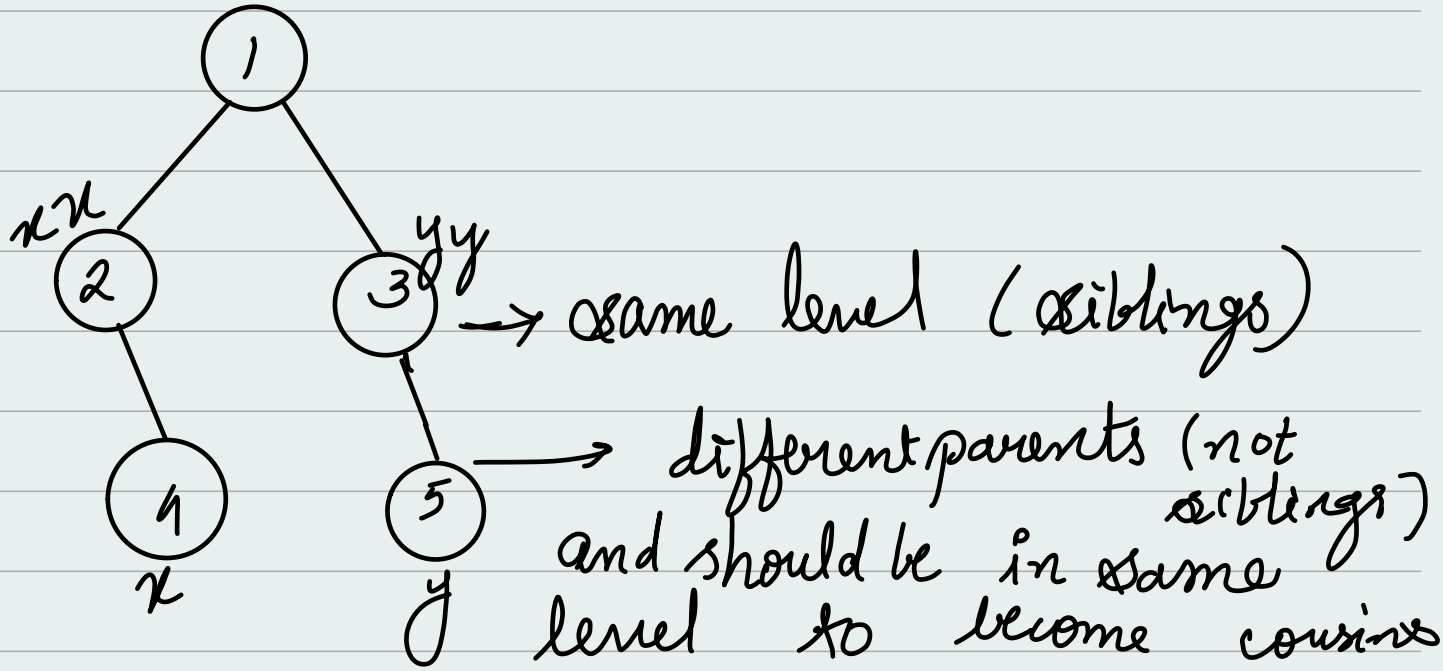
Input : root = [1, 2, 3, null, 4, null, 5]
x = 4 , y = 5
output : ~~true~~



Input : [1, 2, 3, null, 4] x = 4, y = 3
output : false.

i) Make a function that checks whether x and y are siblings or not.

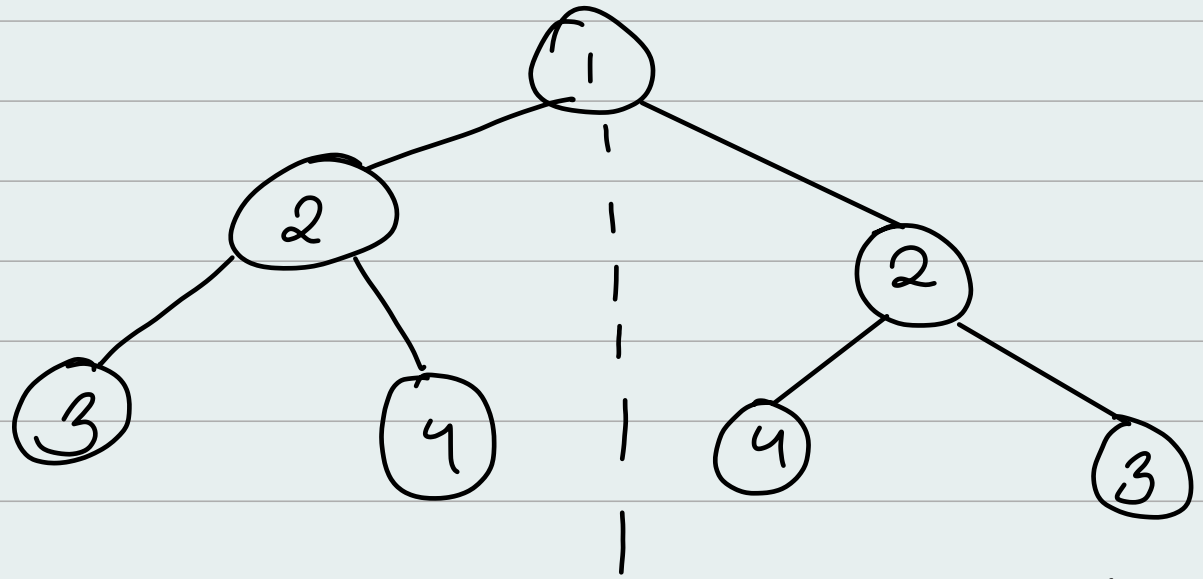
ii) Make a check that x and y should not have same parents



```
public boolean isCousins(TreeNode root,
                         int x, int y){
    TreeNode xx = findNode(root, x);
    TreeNode yy = findNode(root, y);
    return((level(root, xx, 0) == level(
        root, yy, 0)) && (!isSiblings(
            root, xx, yy)));
}
```

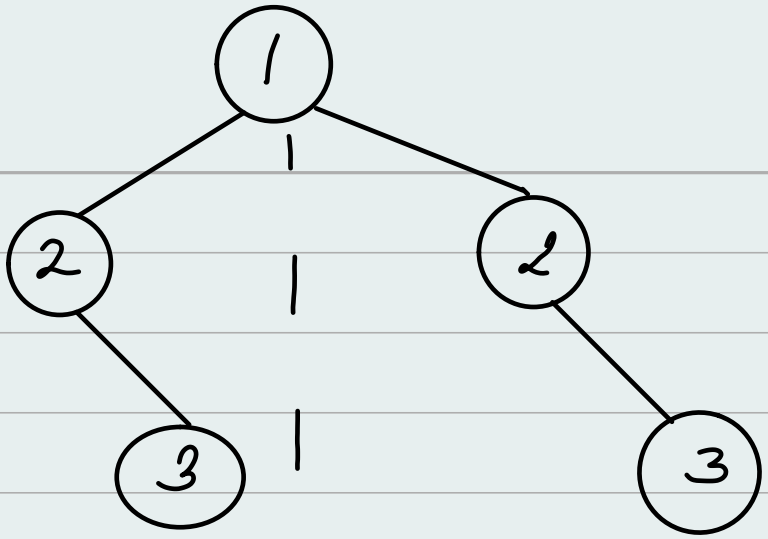
Q7) Given the root of a binary tree, check whether it is a mirror of itself (i.e symmetric around it's center)

Soln



Input root = [1, 2, 2, 3, 4, 4, 3]

output = true

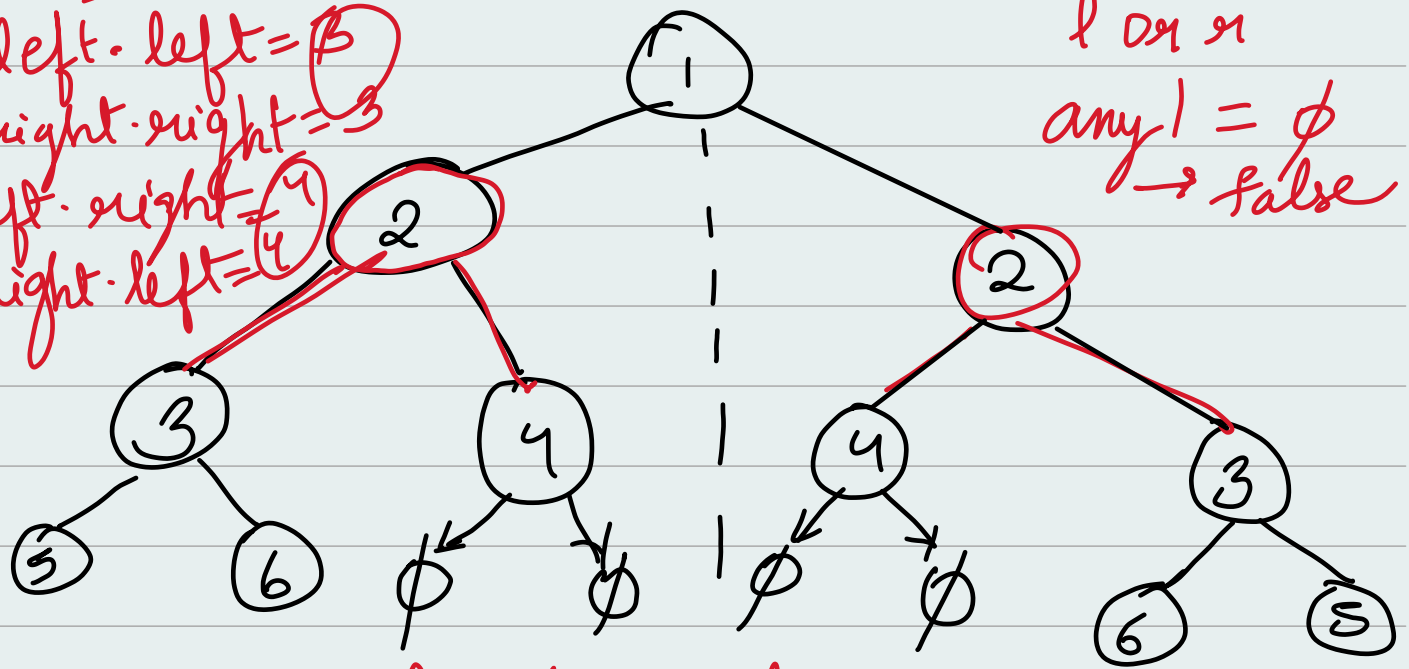


Input : root = [1, 2, 2, null, 3, null, 3]
output = false

Intuition

left.left = 3
right.right = 3
left.right = 4
right.left = 4

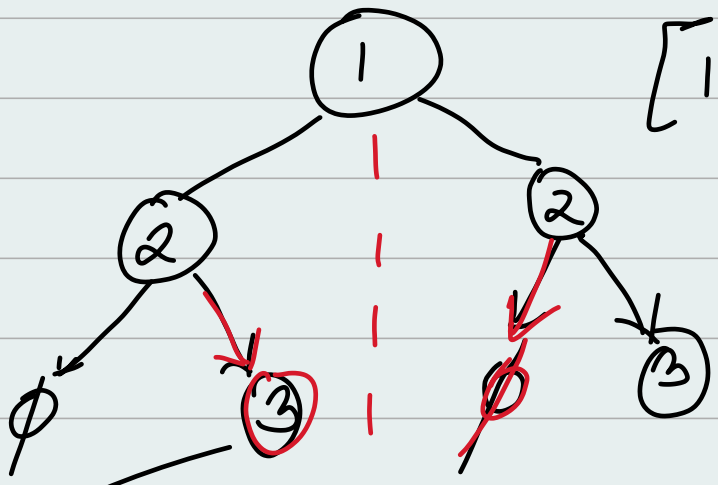
l or r
any l = ϕ
→ false



l.l r.r l.r r.l

[1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6]

[1, 2, 2, ϕ, 3, 3, ϕ]



false

once these elements are removed it has to be stored in another node.

```
public boolean isSymmetric(TreeNode root){
    if root null return true
    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root.left);
    queue.add(root.right);

    while(!queue.isEmpty()){
        TreeNode left = queue.poll();
        TreeNode right = queue.poll();
        if(left==null && right==null){
            continue;
        }
        if(left==null || right==null){
            return false;
        }
        if(left.value != right.value){
            return false;
        }
        queue.add(left.left);
        queue.add(right.right);
        queue.add(left.right);
        queue.add(right.left);
    }
```

return true;

}

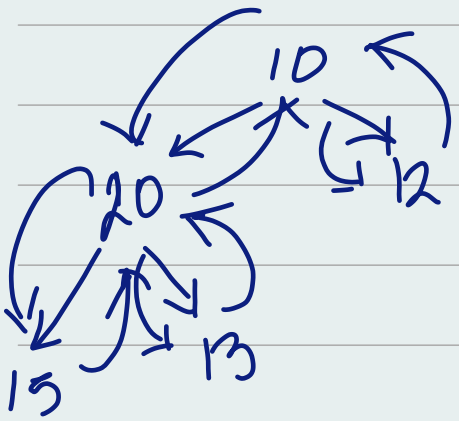
Time complexity & space complexity is O(N).

Depth First Search.

* Node
L R

Pre Order.

N → L → R



In-Order

L → N → R

post order

L → R → N

Pre-Order
10 → 20 → 15 → 13 → 12
root - left - right

In Order ⇒ 15 → 20 → 13 → 10 → 12
Leftmost first. L N R

In a B.S.T when you want lowest values

Post Order Traversal

$$\underbrace{15 \to 13 \to 20}_{L} \to \underbrace{12}_{R} \to \underbrace{10}_{N/root}$$

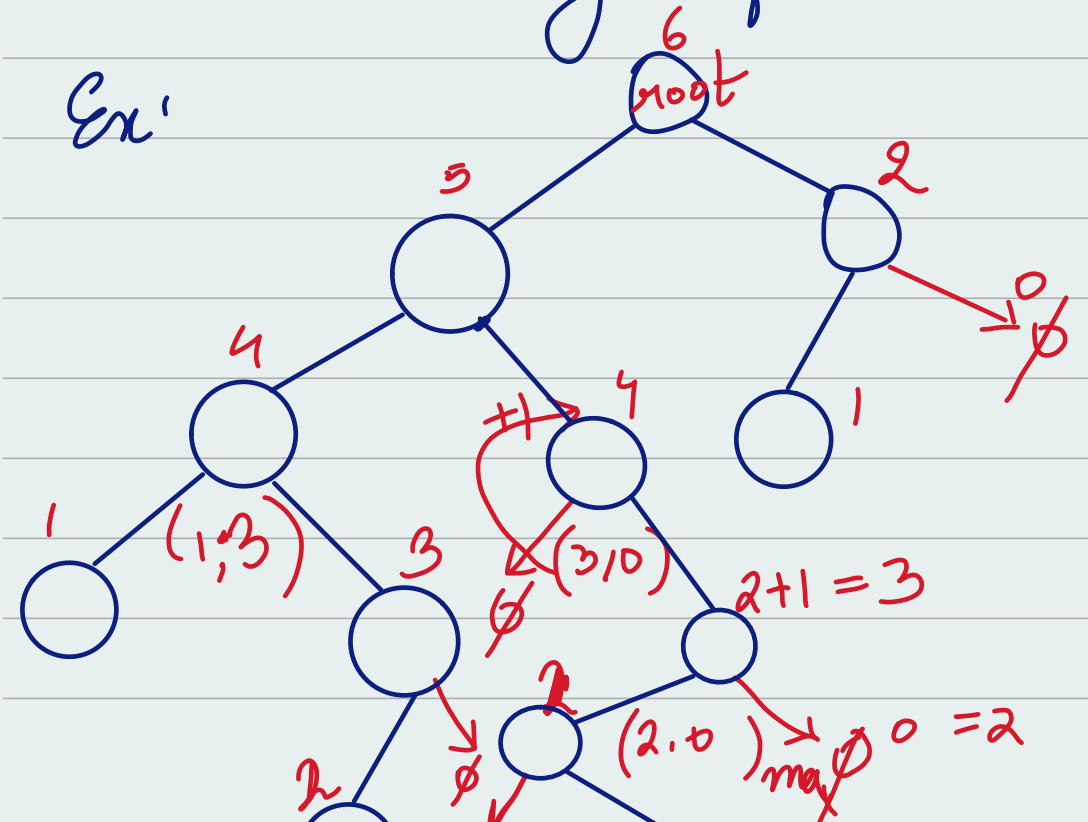
When we want to delete a tree we use post Order traversals

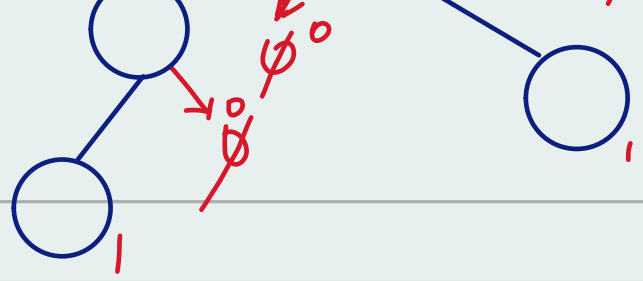
Q8) Diameter of a Binary Tree.

Google, Facebook, Amazon

return the length of the diameter of the tree

Ex:



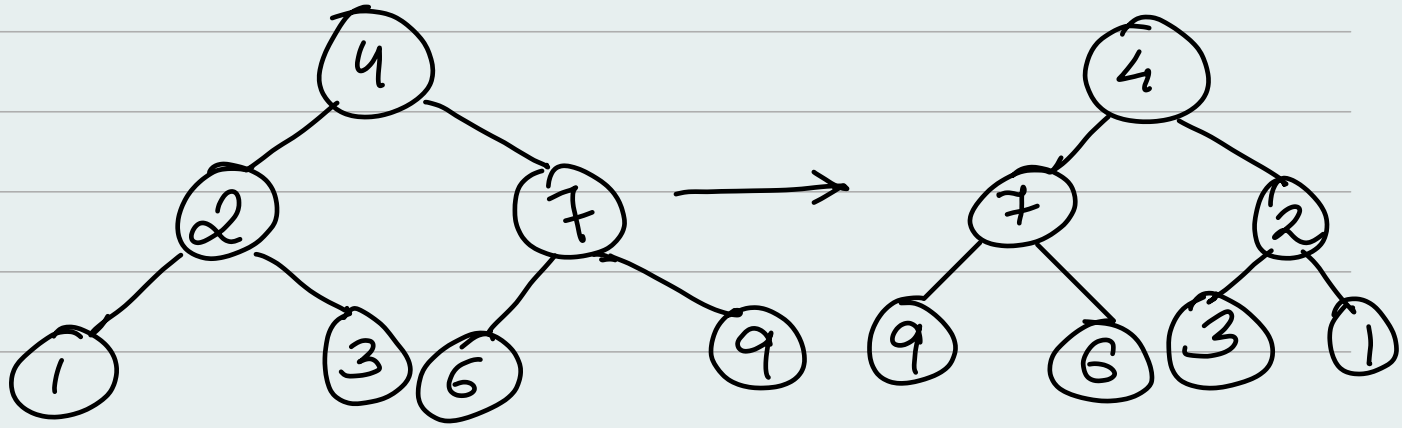


Code: Post Order Traversal

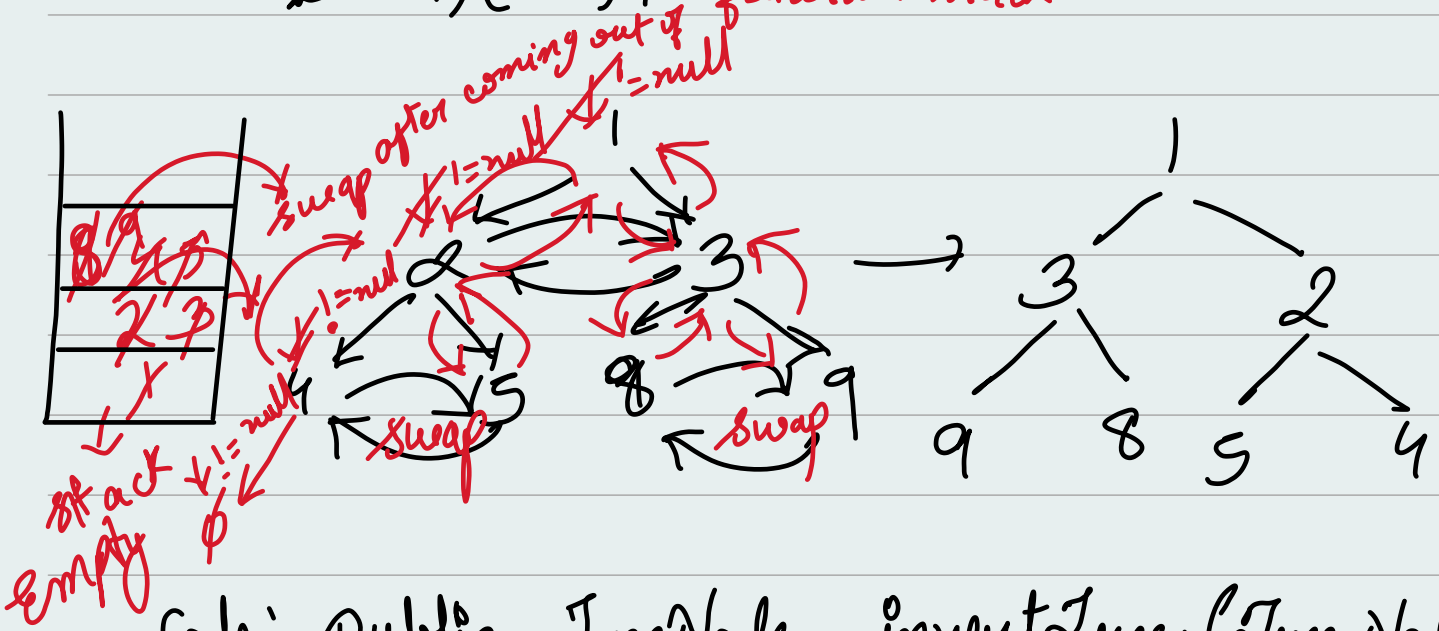
```
class DFS {

    int diameter = 0;
    public int diameterOfBinaryTree(
                    TreeNode root) {
        height(root);
        return diameter - 1;
    }

    int height(TreeNode root) {
        if (node == null) {
            return 0;
        }

        int leftheight = height(node.left);
        int rightheight = height(node.right);

        int dia = leftheight + rightheight + 1;

        diameter = Math.max(diameter, dia);
        return Math.max(leftheight, rightheight)
                                    + 1;
    }
}
```

Time Complexity O(N)

89) Invert a Binary Tree



Post order Traversal :-
L - R - N



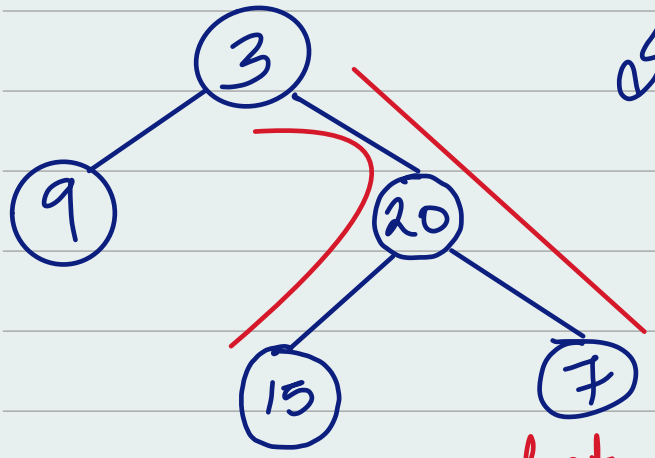
Code: public TreeNode invertTree (TreeNode root){

```
if (root == null){
    return null;
}
TreeNode left = invertTree(root.left);
TreeNode right = invertTree(root.right);
// swap the nodes:
```

```
root.left = right;
root.right = left;

return root;
}
}
```

Time Complexity → O(N)
Space complexity → O(log(N))
O(h)

Q10) Maximum depth of a binary Tree.



Input: root = [3,9,20, null, null, 15, 7]

output = 3

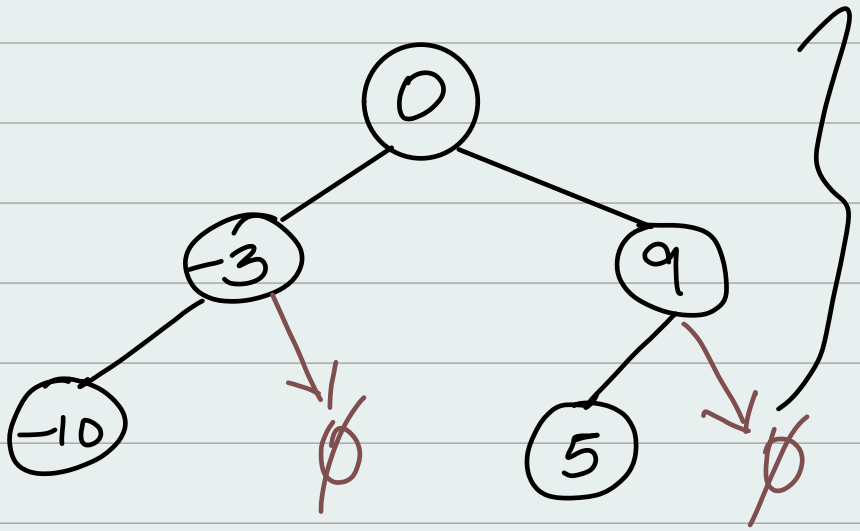
Get the depth from Left then Right +1. Post Order

Code:

```
public int maxDepth(TreeNode root){
    if(root == null){
        return 0;
    }
    int left = maxDepth(root.left),
```

```
int right = maxDepth(root.right);
int depth = Math.max(left, right)
                + 1;

return depth;
}
```

B, 11) Convert Sorted Array to Binary Search



height balanced binary search.

Input: nums = [-10, -3, 0, 5, 9]

output: [0, -3, 9, -10, null, 5, null]

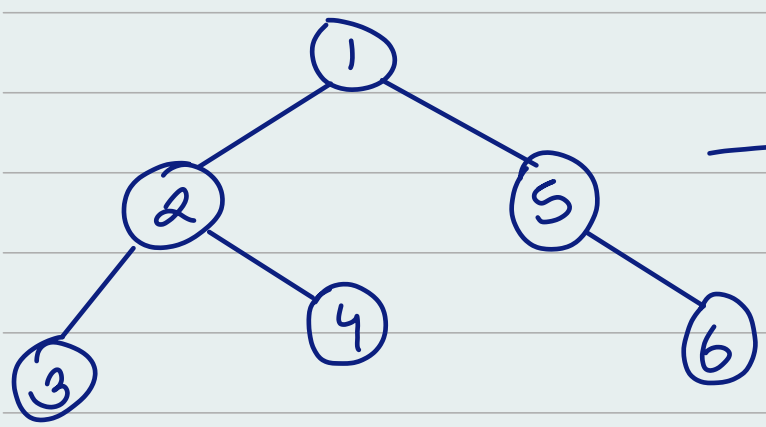
Code:

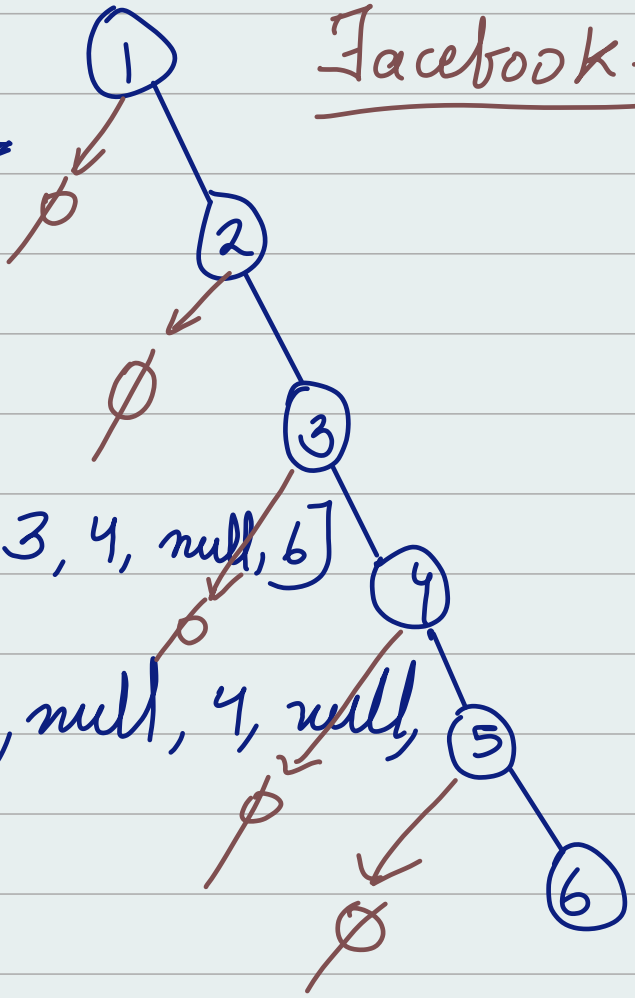
```
public TreeNode sortedArrayToBST(
    int[] nums) {
```

```
return sortedArrayToBST(nums,
                        0, nums.length);

TreeNode sortedArrayToBST(int[] nums,
                          int start, int end){
    if (start >= end){
        return null;
    }
    int mid = (start + end)/2;
    TreeNode node = new TreeNode(
        this.Insert(nums[mid]);

    node.left = sortedArrayToBST(
                    nums, start, mid);
    node.right = sortedArrayToBST(nums,
                    mid+1, end)

    return node;
}
}
```

Q12) Flatten Binary Tree to Linked List.

Facebook.

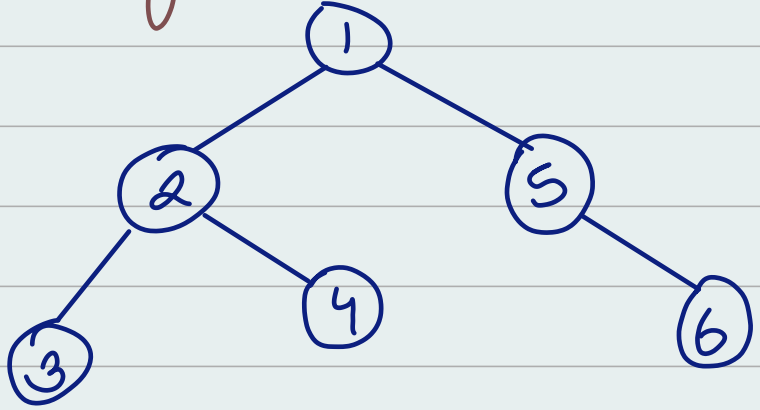




Input : root = [1, 2, 5, 3, 4, null, 6]

output : [1, null, 2, null, 3, null, 4, null, 5, null, 6]

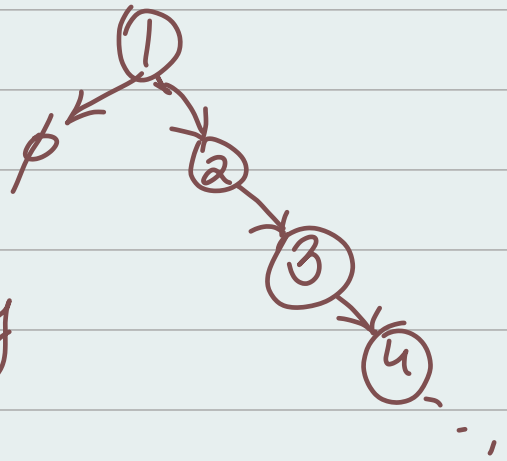
→ The linked list should be in the same order as a pre-order traversal of the binary tree.



① Normal preOrder traversal
② As you traverse, store nodes in queue
③ In the end remove from queue and make a linked list

Node → L → R

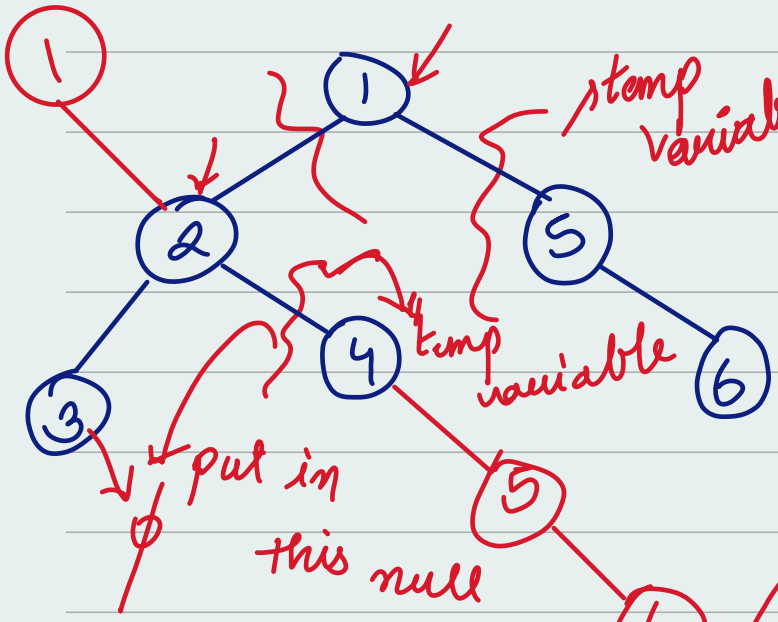
1, 2, 3, 4, 5, 6



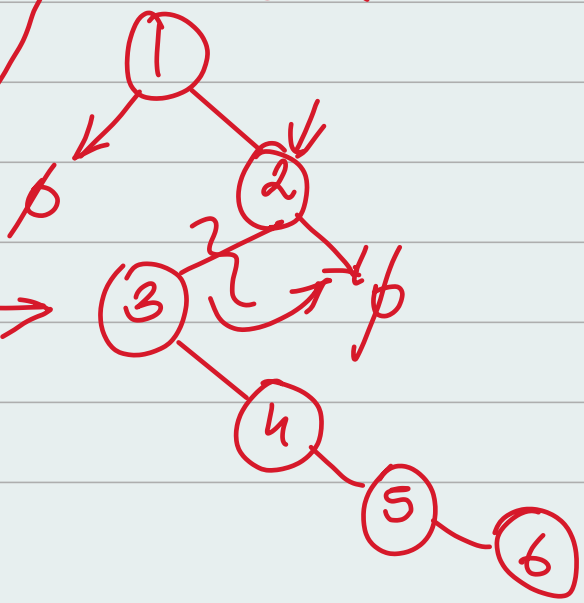
space & Time complexity both O(N)

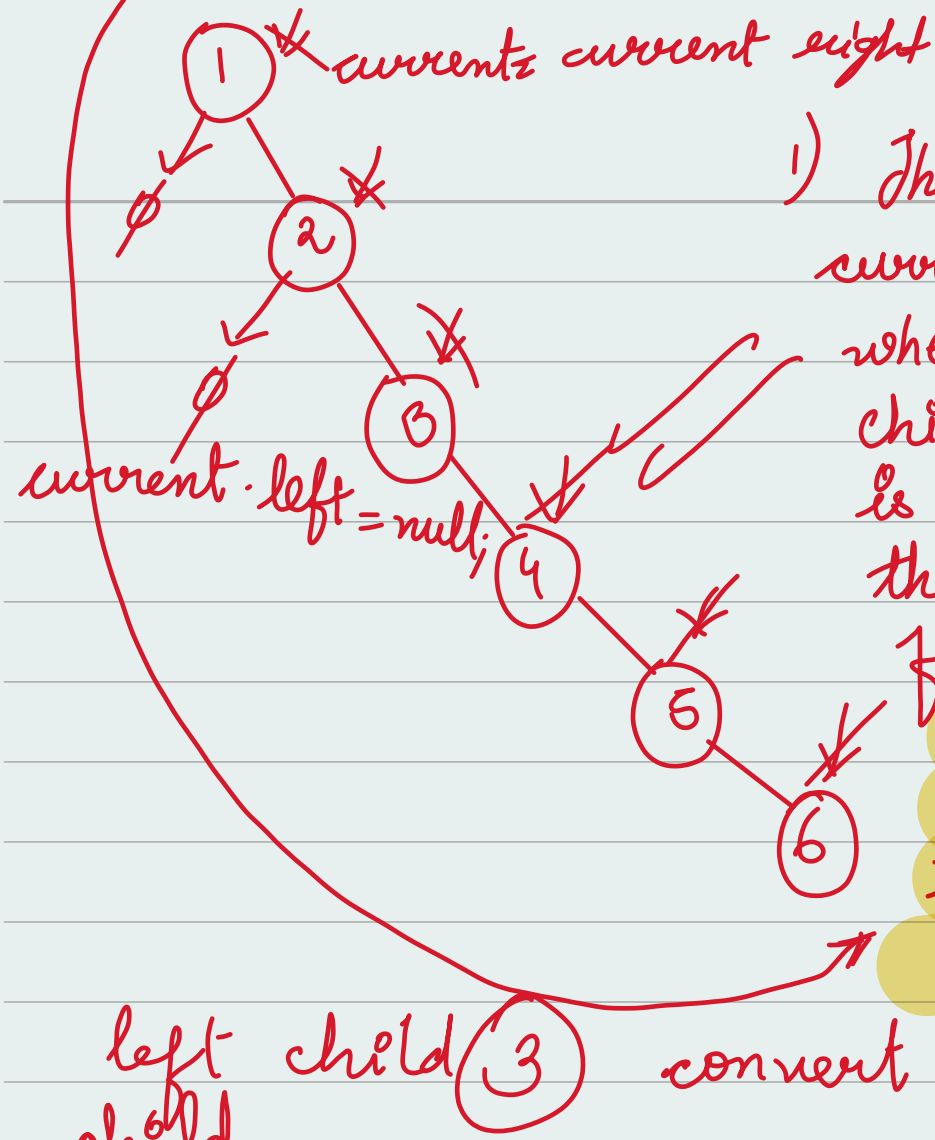
Can it be done in O(1) space ?

N - L - R (order)



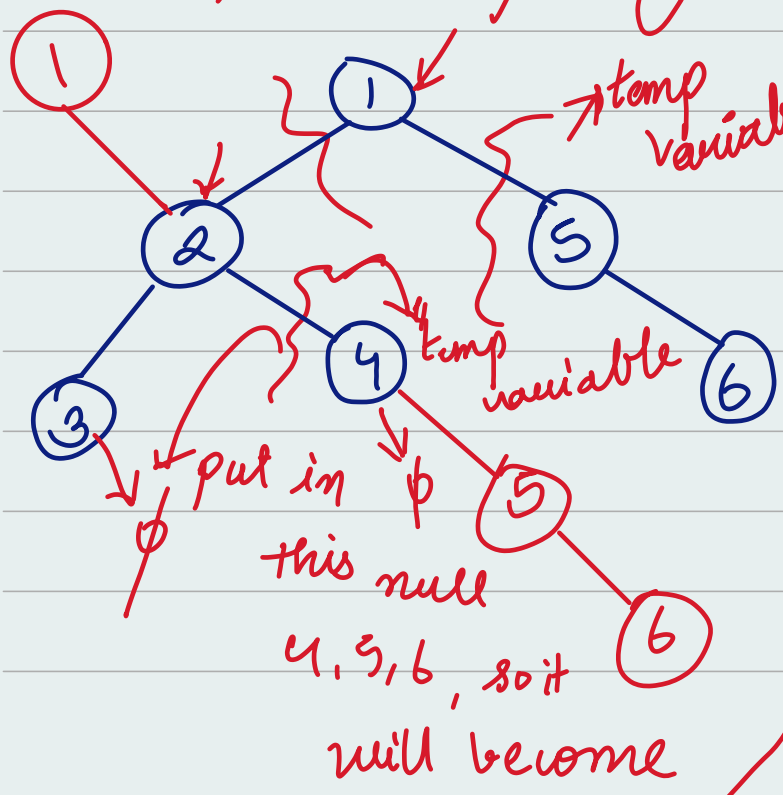
i) 5 → 6 fix it to left of subtree to rightmost empty node i.e. 4





1) The node that I am currently at check whether it has left child and if there is a left child find the rightmost element for it. Then do rightMost element (5→6) is equal to current node.right Then

Here time complexity is O(N) & space complexity is constant



Code:

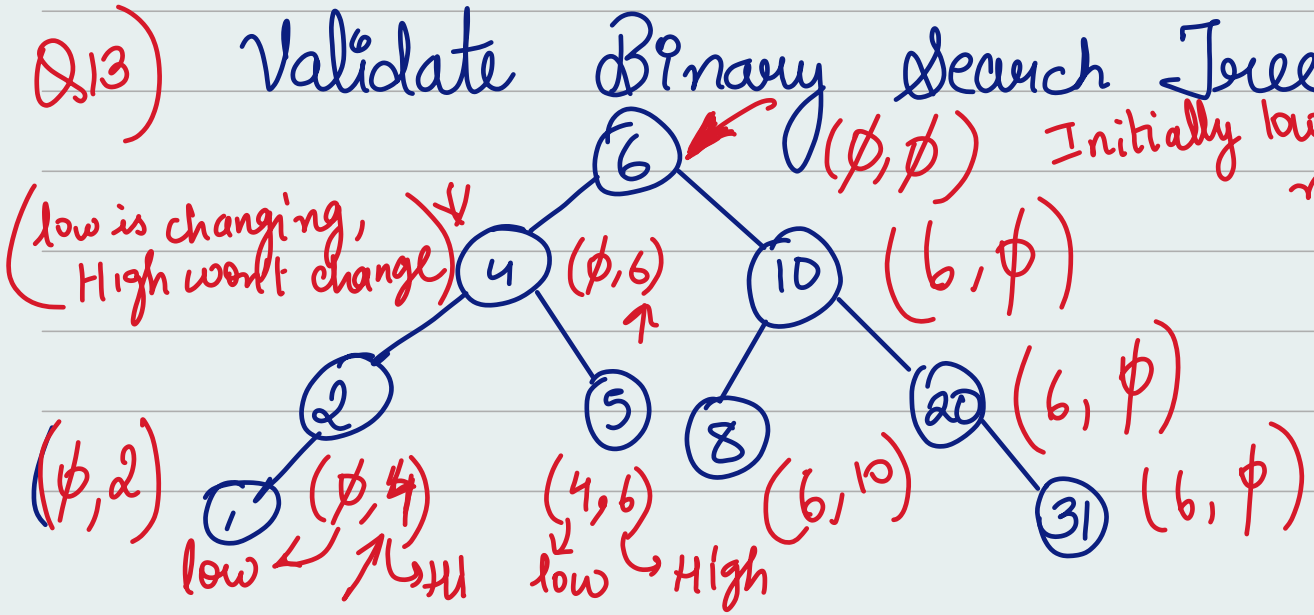
```
public void flatten
        (TreeNode root){
if (root == null){
return;
}
TreeNode current =
        root;
```

Don't change structure of the root

```
while (current != null){
// check whether it has left subtree for node 1
if (current.left != null){
TreeNode temp = current.left;
while (temp.right != null){
temp = temp.right;
}
temp.right = current.right;
current.right = current.left;
current.left = null;
}
current = current.right;
```

// just keep going right until it becomes null
i.e. right = null

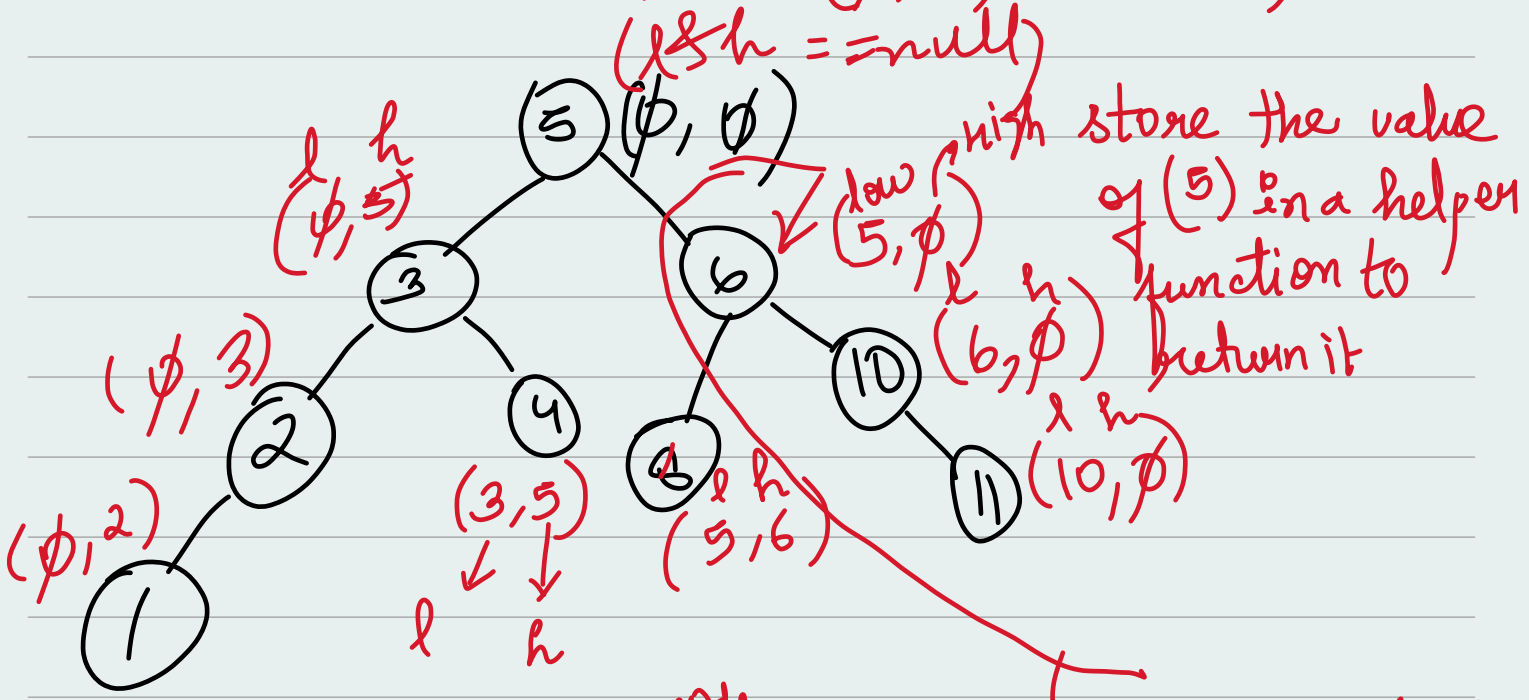
Q13) Validate Binary Search Tree.

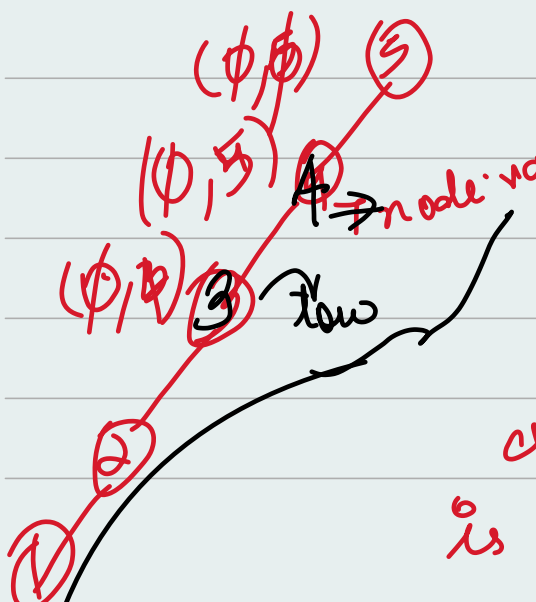


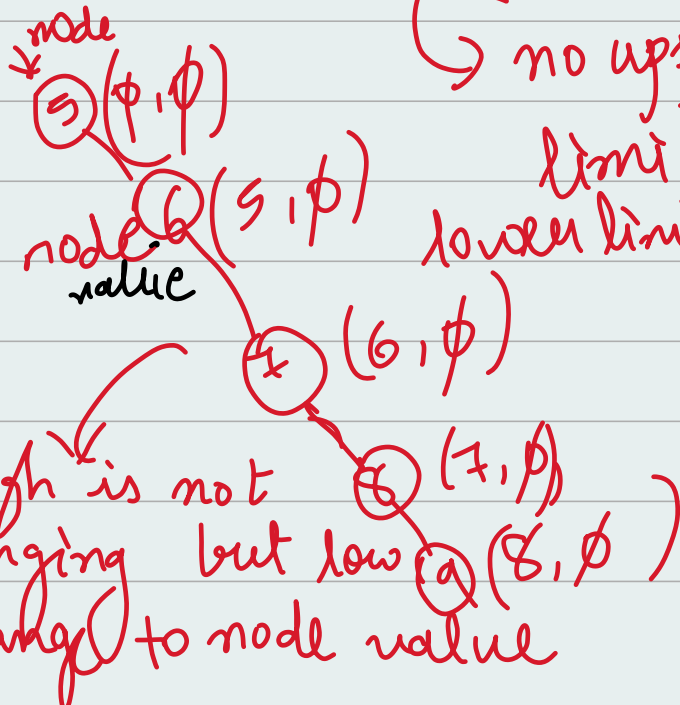
If we go on the left subtree low is not changing it can go low as much as it want but high is changing to node.value → (∅,∅), (∅,6), (∅,4)

But as we go to Right subtree low is changing to node.value but high is not changing. (6 did not change)

→ (∅,∅), (∅,6), (4,6)





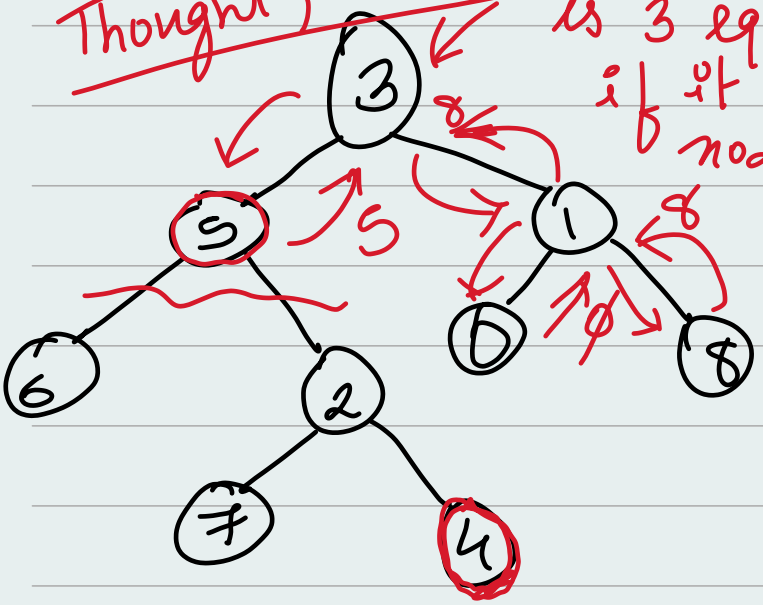


Code: Pre-Order

```
public boolean isValidBST(TreeNode root){
    return helper(root, null, null);

public boolean helper(TreeNode node,
                      Integer low, Integer high){
    if(node == null){
        return true;
    }
    if(low != null && node.val <= low)
        return false;
    }
    if(high != null && node.val >= high){
        return false;
    }

    boolean leftTree = helper(node.left,
                              low, node.val);

    boolean rightTree = helper(node.right,
                               node.val, high);

    return leftTree && rightTree;
}
```

Q14) Lowest common Ancestor

Thought process



is 3 equal to either 5 or 8 if it is then the (5, 8) node is ans, no it's not recursion

We have found the ans (5,8) through recursion so 3 is the ans

```
if (node = p org)
    return node;
else go,
    left  → return 5   (5,4)   } (3)
    right → return null (5,4)  }
if (left & right != null)
    return node you are at i.e. (3)
```

But if the node is (5, 4) (p, q)

4 may not lie in the right subtree of 3 so return null

It only means that 4 lie in the right subtree of 5.

if left = null then right = ans
else left = ans

If both p & q found from top of the node that node is the ans & if any one of them is found like 5 in (5,4) then (5) as a node is ans in lowest common ancestor.

Code:

```
public TreeNode lowestCommonAncestor
        (TreeNode root, Treenode p,
         TreeNode q){
    if (root == null){
    return null;
    }
    if (root == p || root == q){
    return root;
    } // No need to go down tree just
    return from there otherwise go down
    TreeNode left = lowestCommonAncestor
                    (root.left, p, q);
    TreeNode right = lowestCommonAncestor
                     (root.right, p, q);
    if (left != null && right != null)
        return root;
}
```

```
if (left == null){
    return right;
}
return left;
```

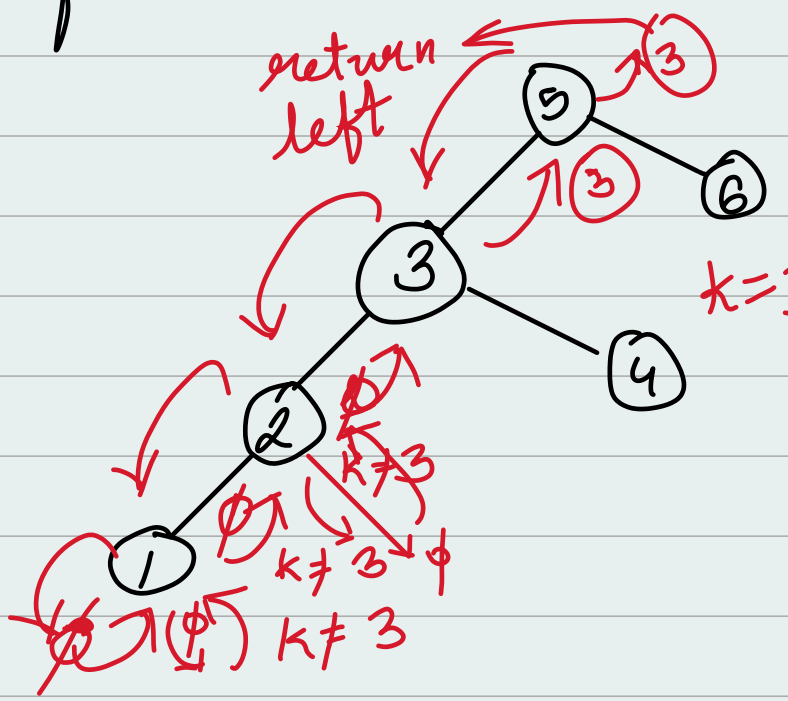
```
return
left == null ?
right : left;
```

Q15) K^{th} smallest element in a BST

Google, Facebook, Amazon

In Order Traversals & when trying to find smallest items.

L - N - R



k≠0 k≠1, k≠2

k=3 | count = 0; k=3

1 2 3

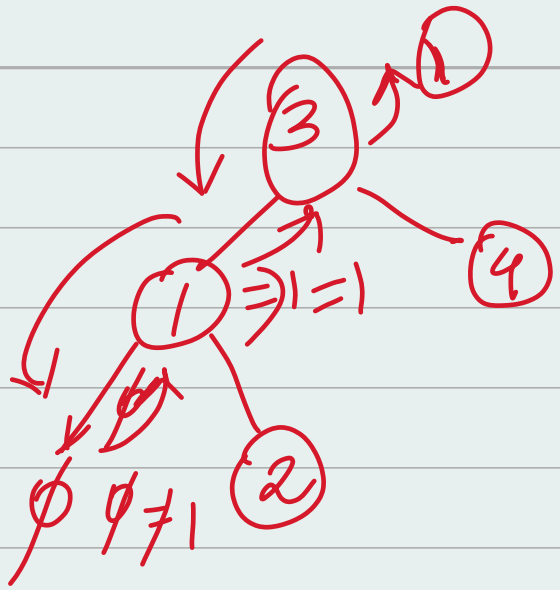
1 is the first smallest item we found.
Take a global counter. Initial count is 0 Then second smallest and then 3rd smallest

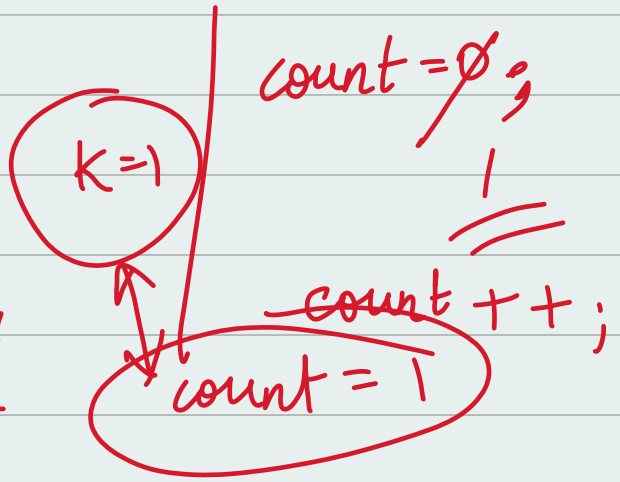
Call left first and then current level Node

In-Order {L-N-R

and then right

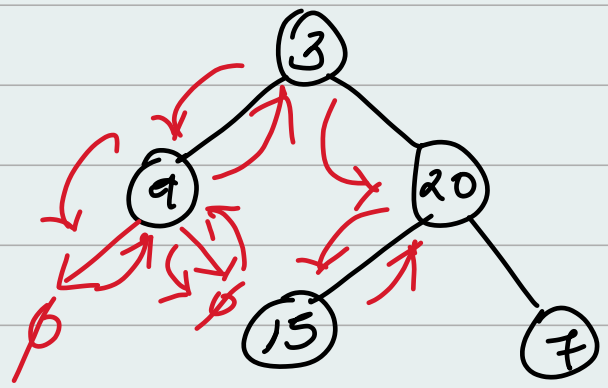
If (left != ∅)
    return left;
count ++;
if (count = k)
    return node
else right node
go to





In Order traversal give Sorted Order of a Binary Search Tree.

Q16) Construct a Binary Tree from Pre Order and In Order Traversals



Pre Order : N-L-R (root)

3 → 9 → 20 → 15 → 7

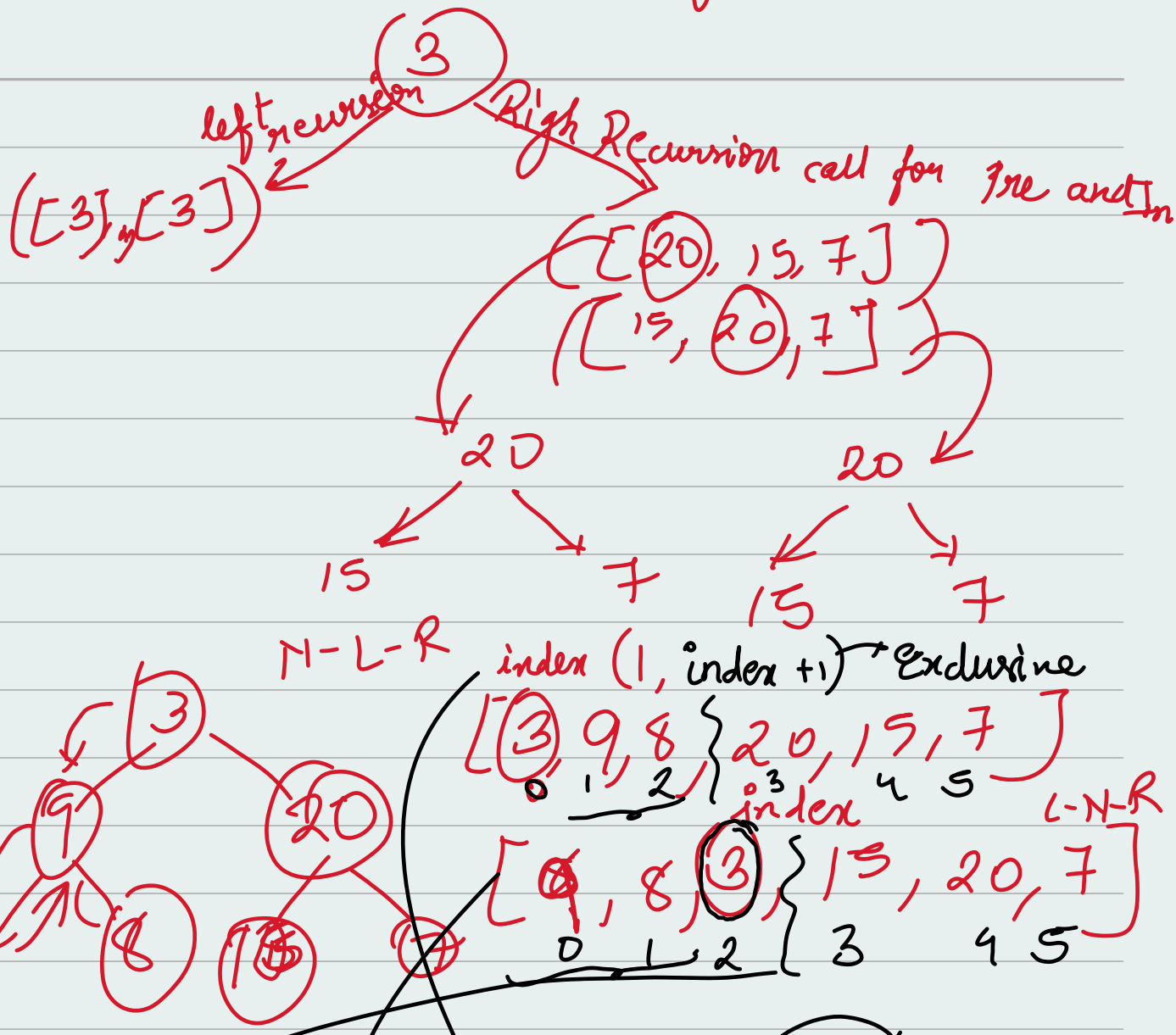
3 — Root node; 9 — left; 20 — R Recursion

In Order to L-N-R

9 → 3 → 15 → 20 → 7

9 — left; 3 — root; 20 — R Recursion

Recursion call for Pre & In Order



Pre Order ⇒ l = (1, index+1)

r = (index+1, end) ↳ preorder

and for inOrder ⇒ l = (0, index) & r = (index+1, end) — length

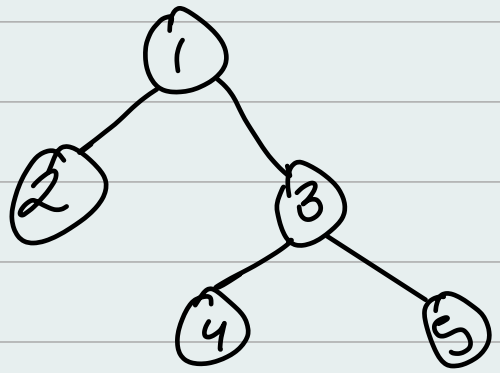
Check whether the tree is null or not — inOrder length

& then set the root node for both true using loop built a new node of root and then do Recursive call for both left and right to build up the tree (Pre Order & InOrder)

Code:

public TreeNode

Q17) Serialize and Deserialize Binary Tree



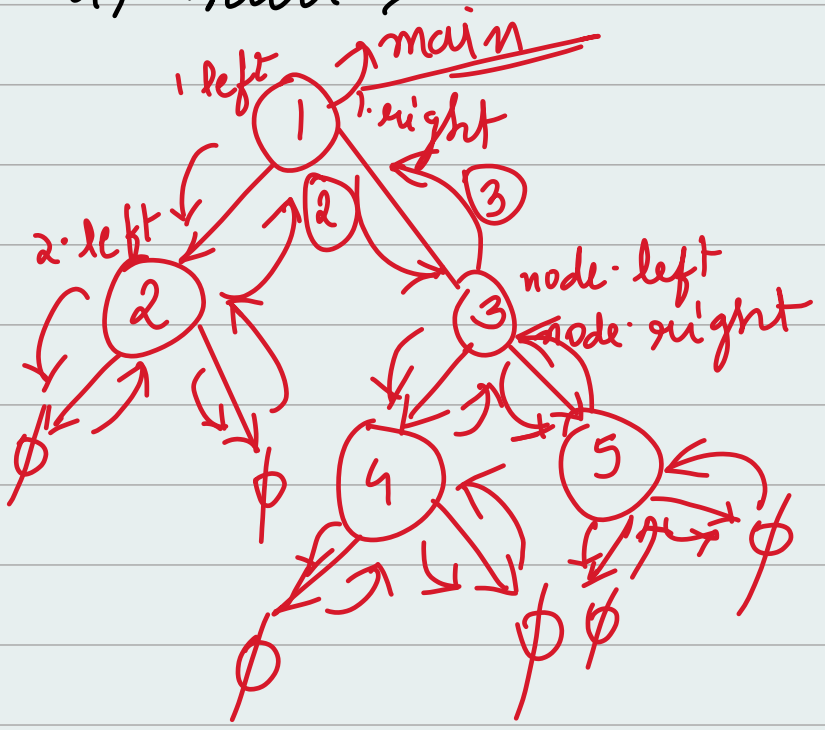
In Order
L - N - R
2 → 1 → 4 → 3 → 5
[2, 1, 4, 3, 5]

Can we built a tree simply by having an InOrder, or preOrder or Post Order. Answer is No. cause if any one order is given it will be difficult to know which is is left node which is root node and which is right node Therefore it is necessary to have any 2 orders to know where is nodes are placed and there null values

(1) Have Pre and In Ordered stored.
(2) Store preOrder with null

Pre Order Traversal :-

<1, 2, null, null, 3, 4, null, null, 5, null, null>



If remove from front 1, 2, .. we have to shift every element to left by 1 until it reaches last element. Efficient way to do it is just by reversing the array and removing from last in this way time complexity will get reduced

Code :-

```
class serializeDeserialize {

    public List<String> serialize(Node
                                   node ){
```

```
List<String> list = new
                    ArrayList<>();
helper (node, list);
return list;
}

void helper (Node node, List<String>
                    list) {
if (node == null) {
    list.add ("null");
    return;
}
  list.add (String.valueOf(
                    node.val));

helper( node.left, list);
helper (node.right, list);
}

Node deserialize (List<String> list){

Collections.reverse (list);
```

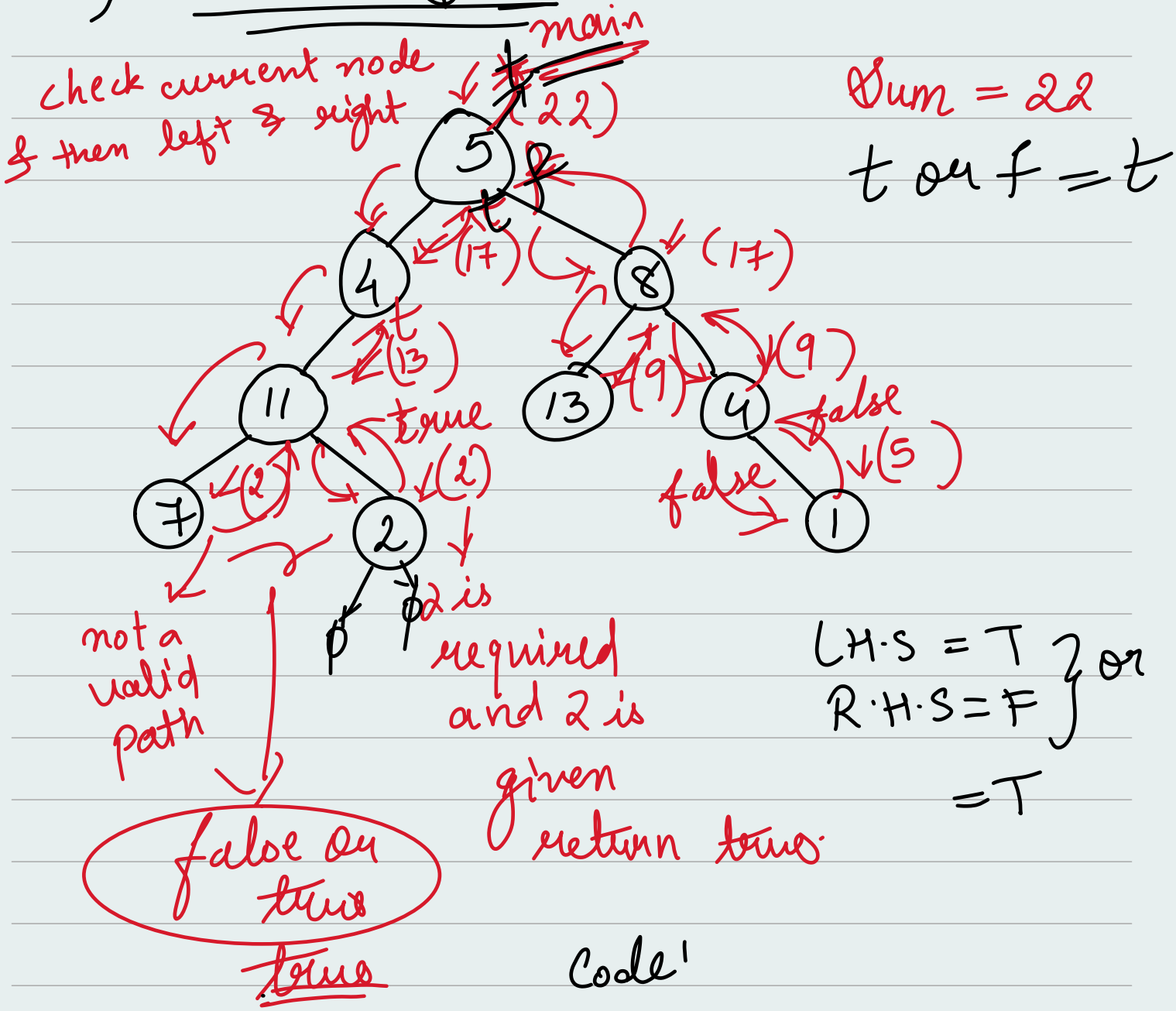
```
        Node node = helper2(list);
        return node;
    }

    Node helper2(List<String> list){
        String val = list.remove(list.size()-1);
        // create the node from the removed string val And also check for null values at 0th index i.e. first element
        <null, null, 5, null, null, 4, 3, null, null, 2, 1>
        if(val.charAt(0) == 'n'){
            return null;
        }
        convert it into integer cause it's a string
        Node node = new Node(Integer.parseInt(val));
        node.left = helper2(list);
        node.right = helper2(list);

        return node;

    }
```

Study Again

Convert it into string

Amazon

Q.18) Path Sum.



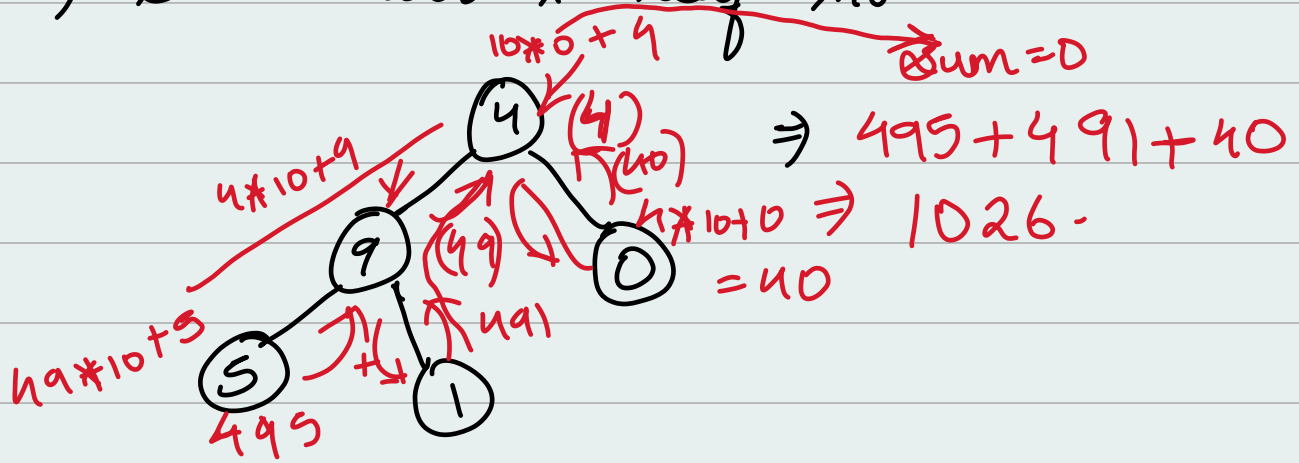
Code!

```
public boolean hasPathSum(TreeNode root,
        int targetSum) {
    if(root == null){
        return false;
    }
    if(root.val == targetSum && root.left
        == null && root.right == null){
        return true;
```

```
}
return hasPathSum(root.left, targetSum
                  - root.val) ||
    hasPathSum(root.right, targetSu
                  -root.val);
}
```

}

8) 19) Sum root to leaf nodes



⇒ 495+491+40

⇒ 10*0+4 = 4

⇒ 10*4+9 = 49

⇒ 10*49+5 = 495

Passing 4 or 49 down pass it in the argument recursion.

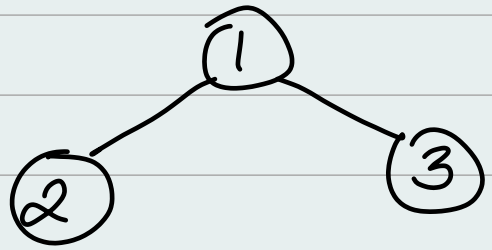
Code:

```
public int sumNumbers(TreeNode root){
    return helper(root, 0);
}
```

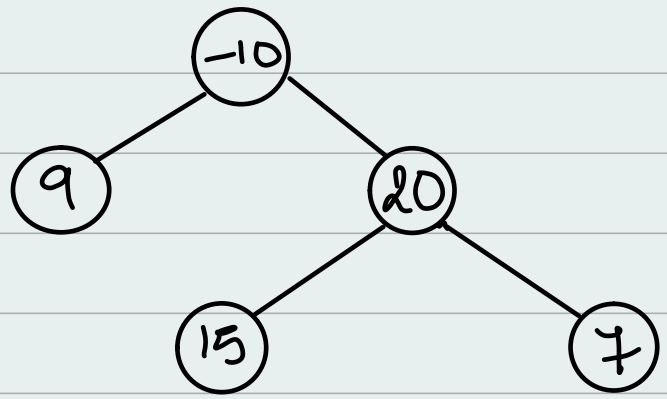
```
int helper (TreeNode node, int sum){
    if (node == null){
        return 0;
    }
    sum = sum * 10 + node.val;
    if( node.left == null && node.right==null){
        return sum;
    }
    return helper(node.left, sum) + helper
                (node.right, sum);
```

820) Binary Tree Maximum Path Sum.

Facebook

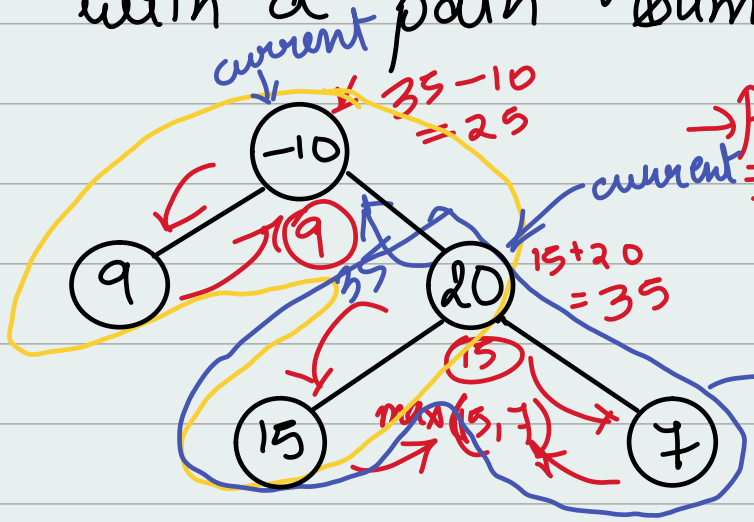


Input: root = [1, 2, 3]
output: 6



root = [-10, 9, 20, null, null, 15, 7]
output = 42

Explaination: The optimal path is 15 → 20 → 7 with a path sum of 15 + 20 + 7 = 42.



→ path with max sum
current ⇒ maxSum(left, right) + current

Compute maximum pathSum in L.H.S and compute maximum pathSum in R.H.S

→ left Recursion & Right Recursion.

path sum = current + left + right

update global

global pathSum
∅
9
42

Code:

```
int ans = Integer.Min_Value;

public int maxPathSum (TreeNode root){
    helper (root);
    return ans;
}
```

```
int helper(TreeNode node){
    if(node == null){
        return 0;
    }

    int left = helper(node.left);
    int right = helper(node.right);
    left = Math.max(0, left);
    right = Math.max(0, right);
    int pathSum = left + right + node.val;

    int ans = Math.max(ans, pathSum);

    // path with maxSum:

    return Math.max(left, right) + node.val;
```

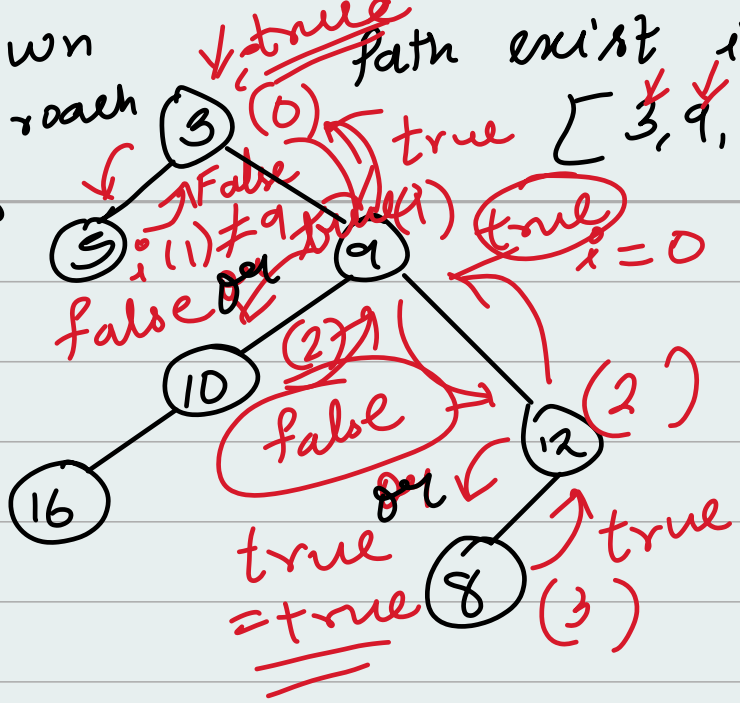
But what if there is a path in left or right that is negative just ignore it. Just make it 0

0,21) Path exists in Binary Tree from Root to leaf.

check leaf node or not.

Top_down approach → DFS

Path exist in tree or not [3, 9, 12, 8]



Code:

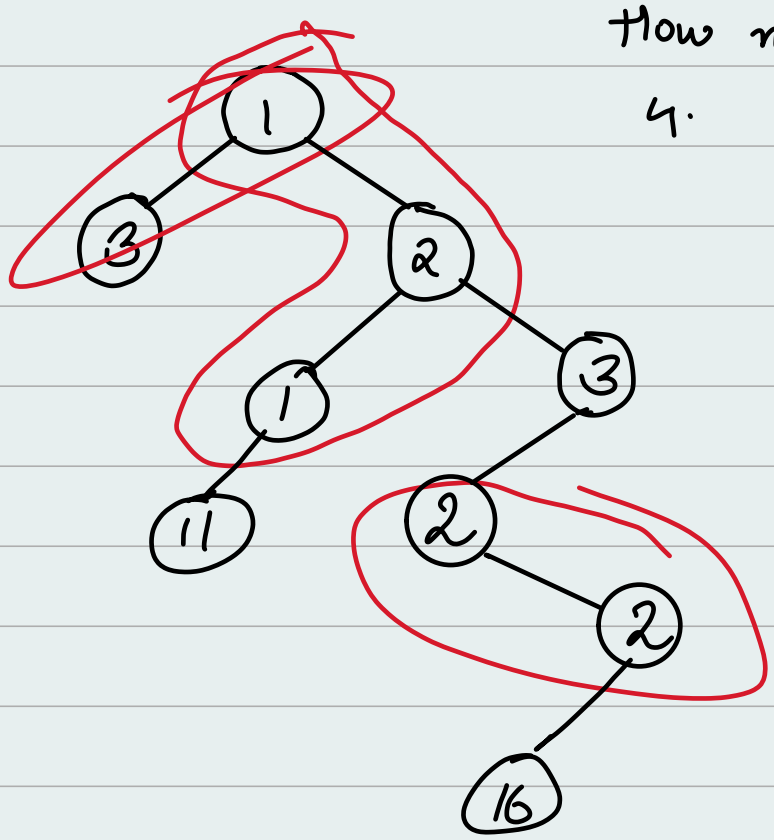
```
boolean findPath (Node node, int[] arr){
    if (node == null){
        return arr length = 0;
    }
    return helper( node , arr, 0 );

boolean helper ( Node node, int[] arr
                                int index ){
    if (node == null){
        return false;
    }
    if (index >= arr.length || node.val != arr[index])
                                                {
        return false;
    }
```

out of bound

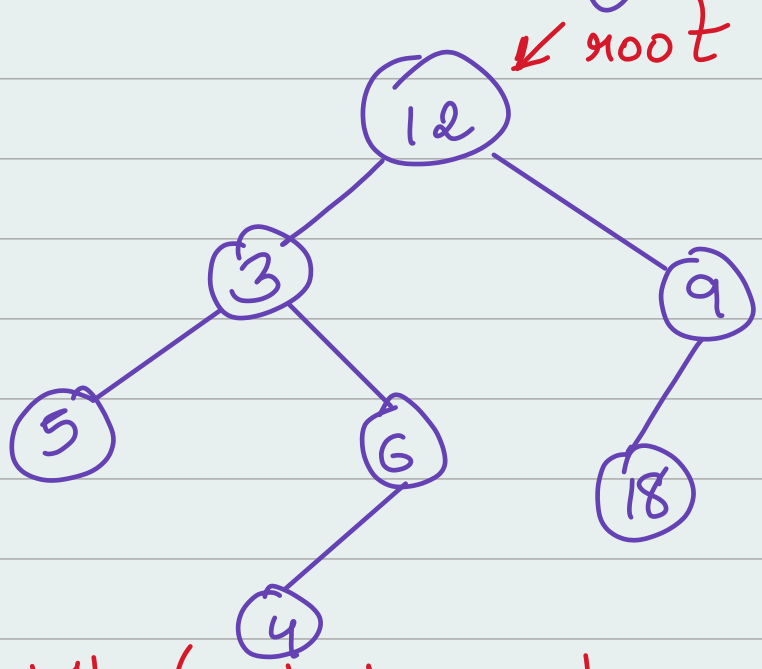
```
if (node.left == null && node.right
        == null && index = arr.length-1)
{
    return true;
}
return helper(node.left, arr, index+1)
    || helper(node.right, arr, index+1);
}
```

Q.22) Path exists in Binary Tree at any nodes

How many path sum of 4.



DFS Using Stack



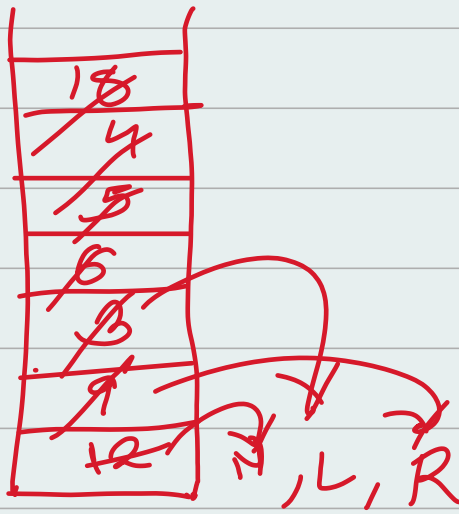
while (stack is not empty)
remove 12 and add 12
left and right child.

But now 9 will be removed. We want 3 to be removed first.

We will use Pre Order Traversal
N-L-R

Time Complexity
O(N)

Space complexity
height.



N, L, R
12, 3, 5, 6, 4, 9, 18

Code'

```
void dfsStack (Node node){
    if (node == null){
        return;
    }
Stack <Node> stack = new stack<>
                        ();
 stack.push (node);

 while (! stack.isEmpty()){

  Node removed = stack.pop();
  System.out.print(removed.val + " ");

  if (removed.right != null ){
     stack.push (removed.right);
  }

  if (removed.left != null ){
  } stack.push (removed.left );
}
```