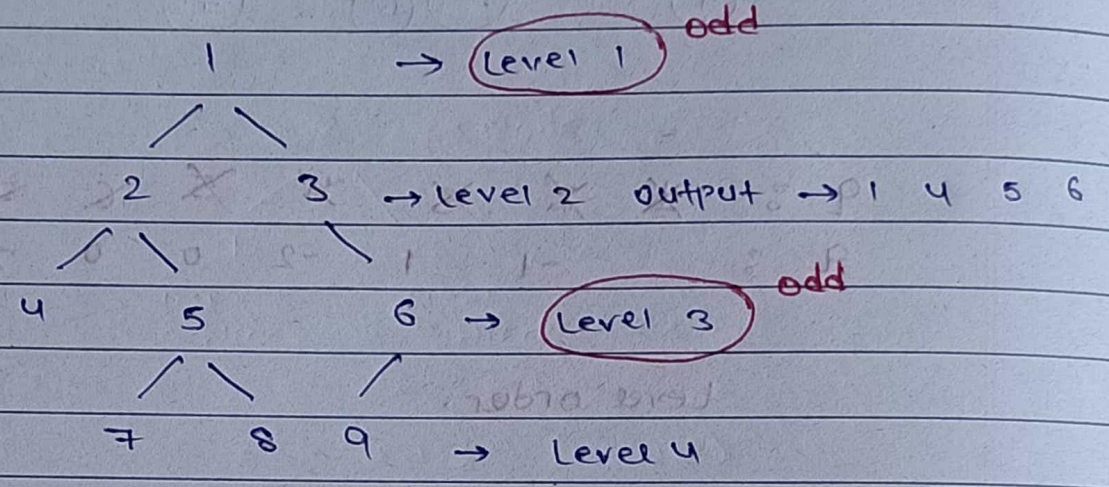
Page No.:
Date: / /

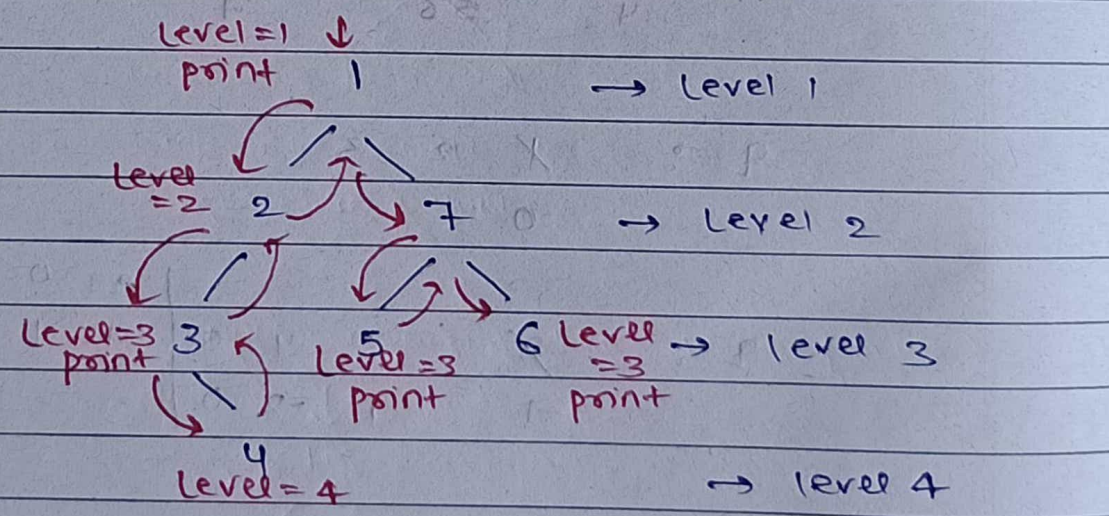
Day - 42

(Tree problem - Easy & Medium)

① Number of odd level :



on this we use ~~level~~ Pre order traversal.
Check at Every Node, that we present on which level.



Output : 1 3 5 6

Teacher's Signature.....

Page No.:
Date: / /

Take a variable label and keep record of pointer that it is at which level.

Code:

```
void preOrder(Node* root, vector<Node*>& ans, int level){
        if(root == NULL)
            return;

        if(level % 2 == 1)
        ans.push_back(root);

    preorder(root->left, ans, level+1);
    preorder(root->right, ans, level+1);
}

vector<Node*> nodesAtOddLevels(Node* root){
        vector<Node*> ans;
        int level = 1;
    preorder(root, ans, level);
    return ans;
}
```

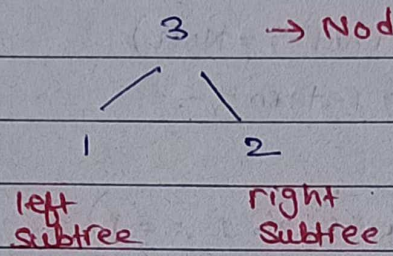
Teacher's Signature.....

Page No.:
Date: / /

Problem 2 : sum tree :

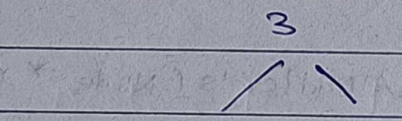
Return true, if Node = left subtree + right subtree.
Except leaf Node.

Ex :



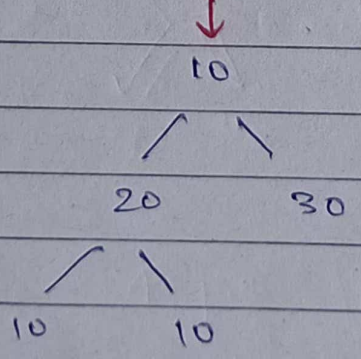
3 = 1 + 2 (True)
return.

Note : Don't care about leaf Node.
Because it always gives false.



leftSubtree = NULL
rightSubtree = NULL

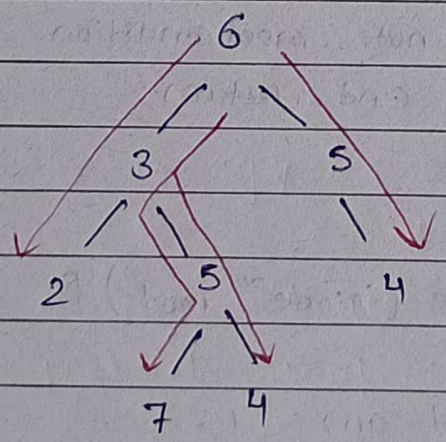
NULL + NULL != 1 (false).



Teacher's Signature.....

```
if (root == NULL)
    return 0;

if (root → left == NULL && root → right == NULL){
    return root → data;

int left = sumTree (root → left, ans);
int right = sumTree (root → right, ans);

if (left + right != root → data)
    ans = 0;

return root → data + left + right;
}
```

Problem 3: Root to leaf paths sum:



4 path from roof to leaf

632 6357 6354 654

ans = 632 + 6357 + 6354 + 654

Teacher's Signature.....

Page No.:
Date: / /

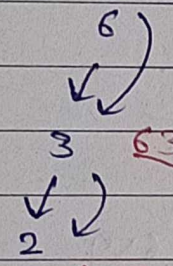
path:

6 → 3 → 2	→	632
6 → 3 → 5 → 7	→	6354
6 → 3 → 5 → 4	→	6357
6 → 5 → 4	→	654

Ans = 632 + 6354 + 6357 + 654
= 13997.

How to solve?

path 1 : ↓num = 6



Sum = 0

num = root → data

num = 6

num = num × 10 + root → left

= 6 × 10 + 3

= 63

63 × 10 + 2

= 632

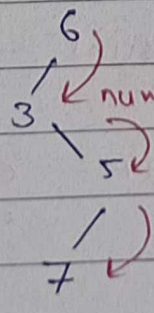
↳ Here we get 1st number from path 1.

add this to sum.

sum += num

sum = 632.

path 2 : ↓num = 6



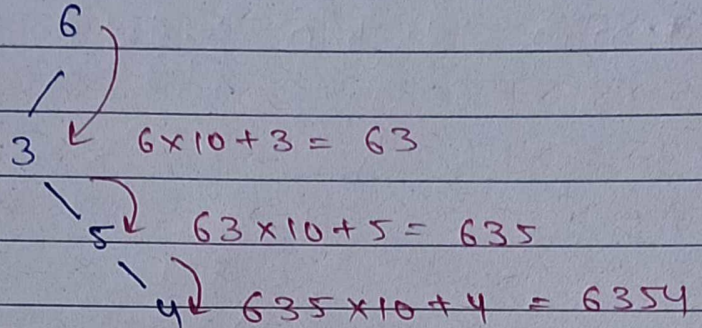
sum = sum + num

= 632 + 6357 = 6989

Teacher's Signature.....

Page No.:
Date: / /

Path 3: ↓num = 6



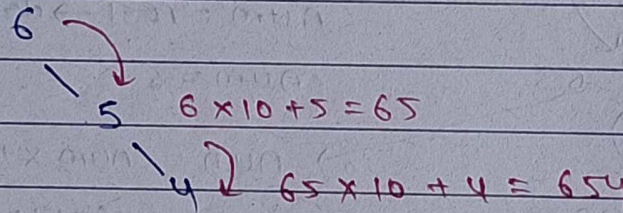
num = 6354

sum = sum + 6354

= 6989 + 6354

= 13343

Path 4: ↓num = 6



num = 654

sum = sum + num

= 13344 + 654

= 13997

We move to all path from root to leaf

then return the sum.

At every setep, we move either right or left.

Teacher's Signature.....

Code :

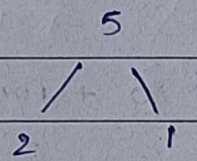
```
Void findsum (Node * root, long long int &sum,
                                long long int num) {
        if (root == NULL)
            return;

        num = num * 10 + root → data;

    if(root → left == NULL && root → right == NULL){
            sum += Num;
            return;
    }

    findsum (root → left, sum, num);
    findsum (root → right, sum, num);

}

long long int treePathsSum (Node * root) {

        long long int sum = 0;
        long long int num = 0;

    findsum ( root, sum, num);

    return sum;

}
```

Teacher's Signature.....

Page No.
Date : / /

Problem 4 : Maximum difference between node and its ancestor

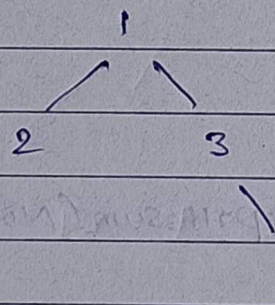
Example



Ancestor of 5 = 2, 1

5 - 2 = 3
5 - 1 = (4) → maximum.

Example



Ancestor of 1 : 2, 3, 7
Ancestor of 3 : 7
There is no ancestor of 2, 7 (leave node)

leaf Node का Acestor नहीं होता है।

1 - 2 = (-1) → maximum (return)
1 - 3 = -2
1 - 7 = -6
3 - 7 = -4

Teacher's Signature.....

Page No.:
Date: / /

Initially

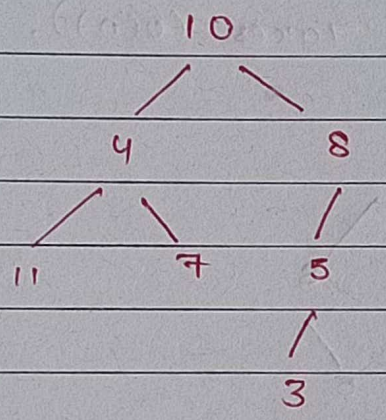
max = Integer.MIN.

A - B

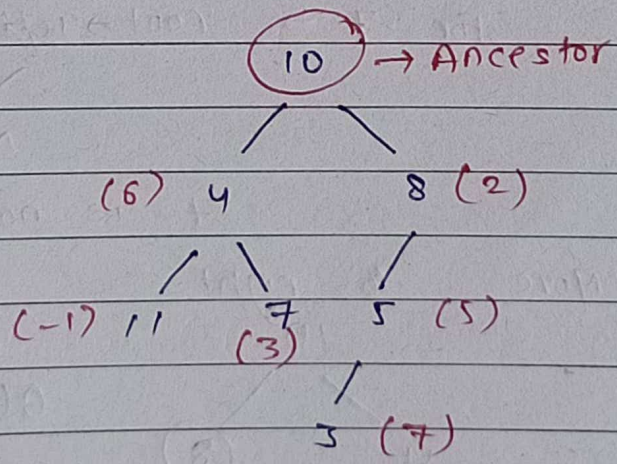
↙ ↘

Acestor Descendant.

Example :



Acestor of 10 : 4 11 7 8 5 3

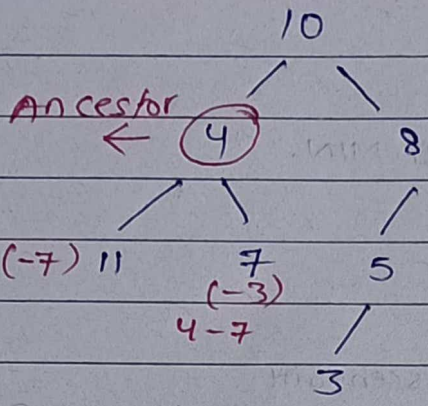


For 10 we check all possible value takes O(n) time.

But we have to check for all other.

go to left.

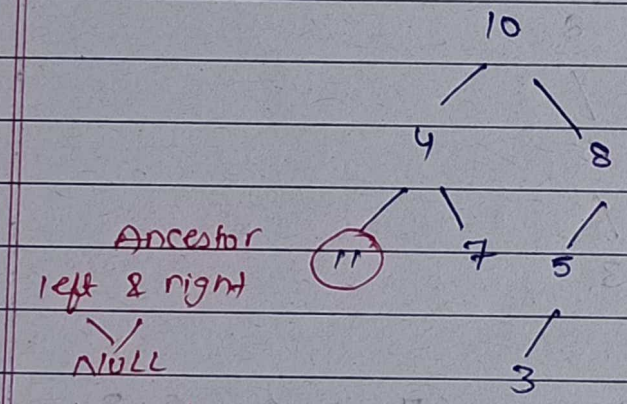
Teacher's Signature.....



Max = 7

Ancestor of 4 : 11 , 7

Takes (O(n)).



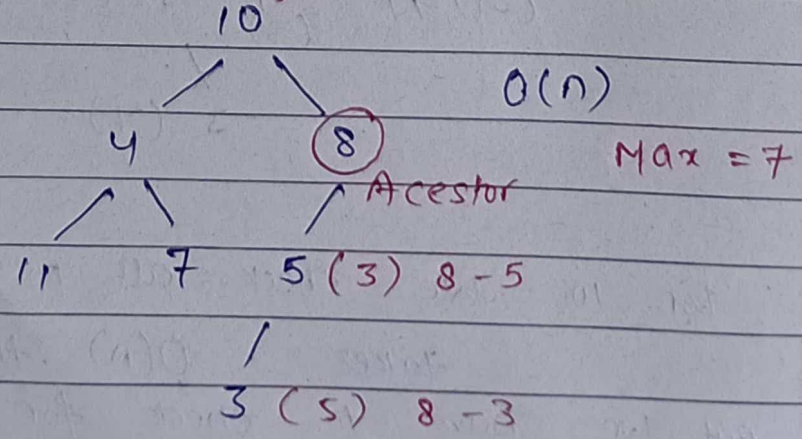
11 is not ansector

For 7 → root → left && right

NULL

7 is not ansector.

Move to right.



O(n)

Max = 7

Teacher's Signature.....

B for 8 = 5, 3

We check for 5 & 3 and Maximum we get:

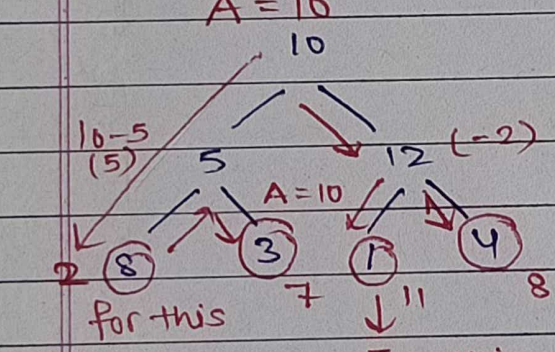
max = 7.

$n * (n-1)$

Time Complexity : $O(n^2)$

↳ We check for every node and for every node we subtract every possible.

* How we solve this in O(N)?



A - B ⇒ Max
↳ When A is big number.

diff = 5 7 11

Max (A, root → data)

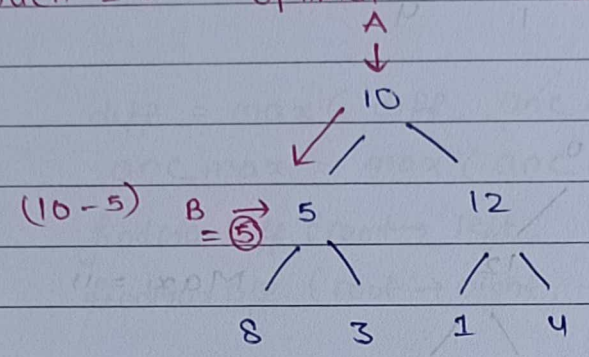
10 - 8 = 2 → Alway take max.
5 - 8 = -3

By this we can solve in O(N)

Teacher's Signature.....

Problem 4 : Maximum difference between node and its ancestor :

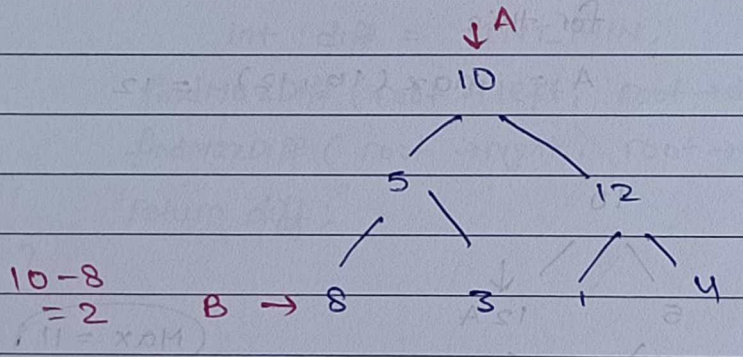
Approach 2 : Optimal O(N)



A - B
↓
Ancestor.

Max = ~~Min~~ 5

We use preorder traversal here.



Max = 5

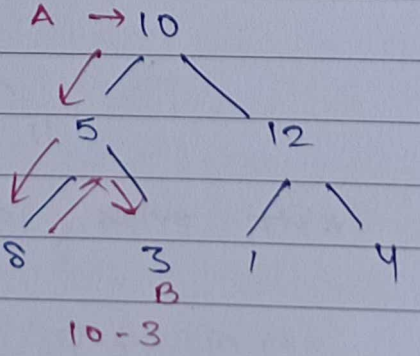
for 8 → which is Ancestor who give maximum value.

10 or 5

Bigger Number gives Max.

So, we take max(A, root→data)

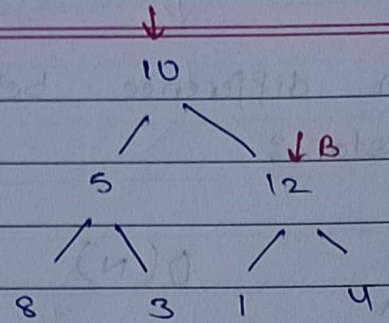
Max = A



Max = ~~5~~ 7

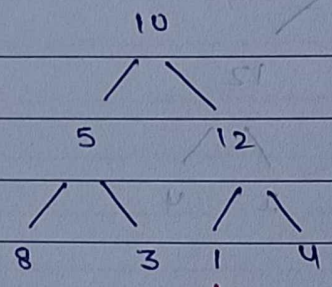
10 - 3

= 7



10 - 12 = -2

Max = 7

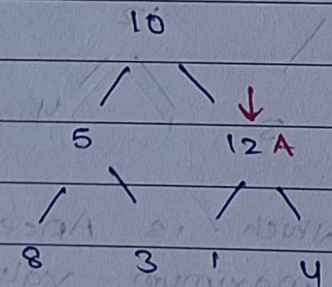


Max = 11

12 - 1 = 11

for this

A = max(10, 12) = 12



Max = 11

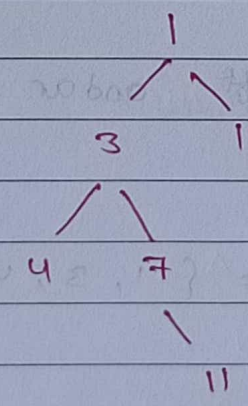
12 - 4 = 8

By this approach, We can solve in O(n) time.

Code:

```
void findMaxDiff ( Node * root, int anc_max, int &diff ) {
        if (root == NULL)
            return;

        diff = max( diff, anc_max - root→data );
        anc_max = max ( anc_max, root→data );
        findMaxDiff (root→ left, anc_max, diff );
        findMaxDiff (root→right, anc_max, diff );
}

int maxDiff (Node * root) {
        int diff = INT_MIN;
        findMaxDiff (root→left, root→data, diff);
        findMaxDiff ( root→right, root→data, diff);
        return diff;
}
```

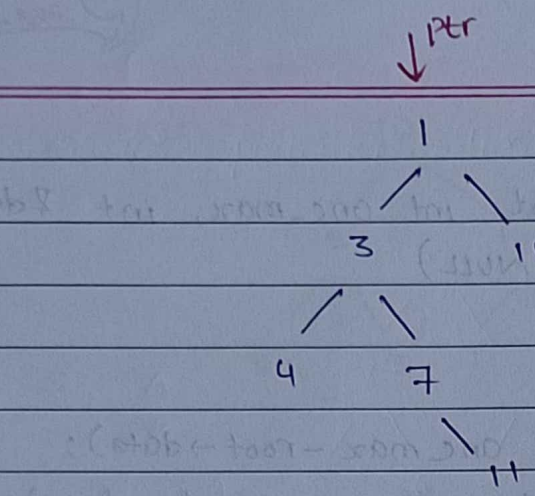
Problem 5: preorder traversal (iterative)



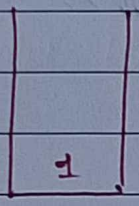
1, 3, 4, 7, 11, 10

We have to solve this without recurion.

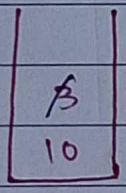
so, we use stack in it.

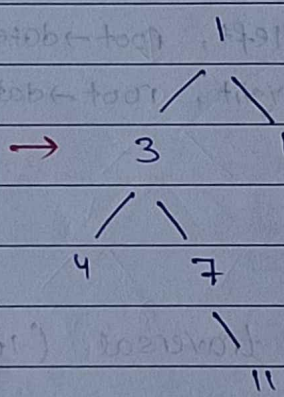


NLR



→ print 1 ans = {1

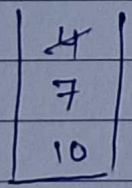




Now we put right ~~first then left~~

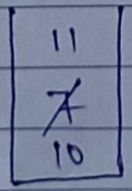
ans = {1, 3

3 का right left andar

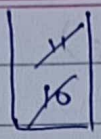


ans = {1, 3, 4

4 का right & left andar



ans = {1, 3, 4, 7



ans = {1, 3, 4, 7, 11, 10}

(1) root ko daal do stack me

(2) Jab tak stack khali nahi hota.

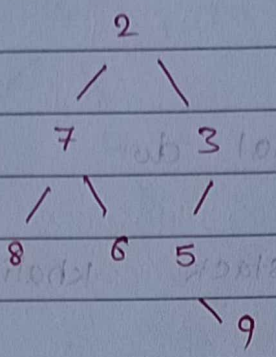
→ top ko print kro
→ Pop kr do top ko !
→ top ke right ko stack Me dalo

→ top ke left ko stack Me dalo.

Code :

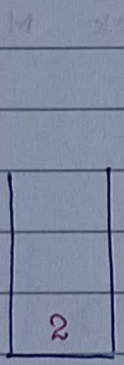
```
Vector <int> preorder (Node * root) {
        vector <int> ans;
        stack <Node*> s;
        s.push (root);
        Node * temp;

        While (! s.empty ()) {
            temp = s.top ();
            s.pop ();
            if (temp → right)
                s.push (temp → right) ;
            if (temp → left)
                s.push (temp → left);
            ans.push_back (temp → data);
        }
        return ans;
}
```

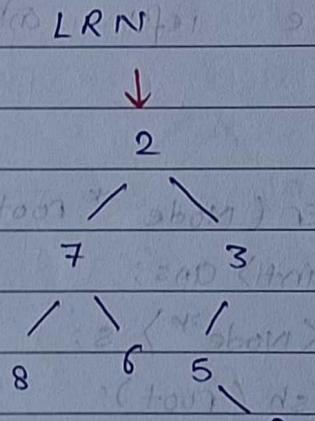
problem 6: postorder Traversal (iterative)



Not by recursion

solve by stack.





NRL

2 3 5 9 7 6 8

reverse

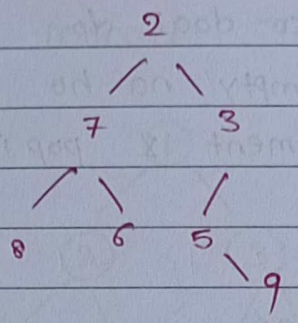
8 6 7 9 5 3 2

Ans

So, here we solve by NRL.

opposite.

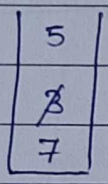
We want N first.

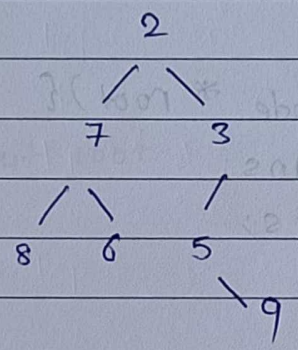


NRL

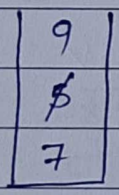
So, in this we put left part first in stack then we put right part.

ans = { 2

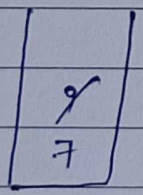




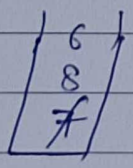
ans = { 2, 3



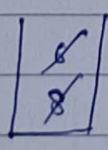
ans = { 2, 3, 5,



ans = { 2, 3, 5, 9



ans = { 2, 3, 5, 9, 7



ans = { 2, 3, 5, 9, 7, 6, 8 }

↓

reverse

8 6 7 9 5 3 2

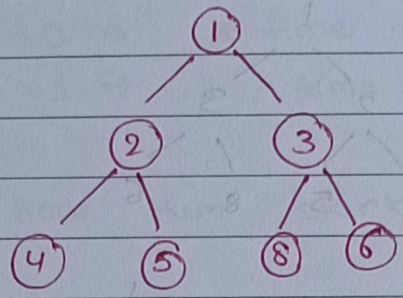
↳ required answer.

1. Stack → root ko daal do
2. Jab tak stack empty na ho
   - → print top element & pop it
   - → Left push
   - → Right push.
3. Reverse the answer.

Code :

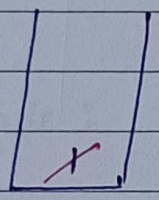
```
vector <int> postorder (Node * root){
        vector <int> ans;
        stack <Node*> s;
        s.push(root);
        Node * temp;

        while (!s.empty()){
            temp = s.top();
            s.pop();

            if (temp → left)
             s.push (temp → left);

            if (temp → right)
             spush (temp → right);

            ans.push_back (temp → data);
        }
    reverse (ans.begin(), ans.end());
    return ans;
}
```

problem 7 : Inorder Traversal (Iterative)

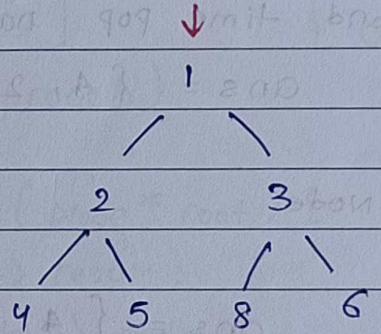


4 2 5 1 8 3 6

Approach

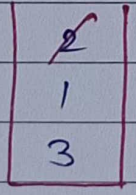


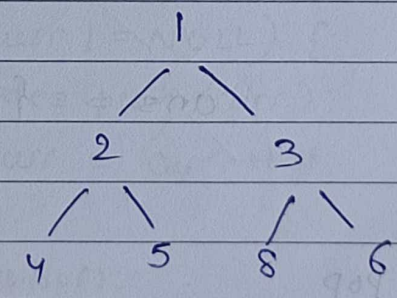
put root



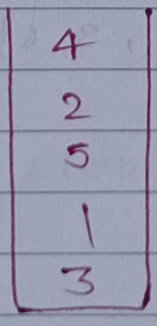
1 को निकालो

उसका Right
तो दूसरा
उसका left
} → stack Me dalo





top को निकालो
Right, तो दूसरा, left } → stack में डालो।

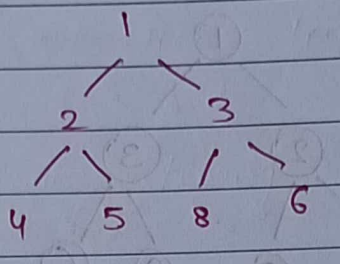


अगर stack का top leaf Nod होगा तो उसको point कर देंगे।

S.top() → right == NULL && S.top() → left == NULL

ans = { 4

~~2~~
5
1
3



अगर stack का top पहले pop हो कर फिर से push हो चुका है तो उसको print कर के pop कर देंगे।

Second time pop nahi karna hai.

ans = { 4, 2

5 → leaf Node

~~5~~
1
3

ans = { 4, 2, 5

1 → पहले हो चुका है।

~~1~~
3

ans = { 4, 2, 5, 1,

3 pop

(right, self, left) → push

~~8~~
~~3~~
~~6~~

ans = { 4, 2, 5, 1, 8, 3, 6}

8 → leaf

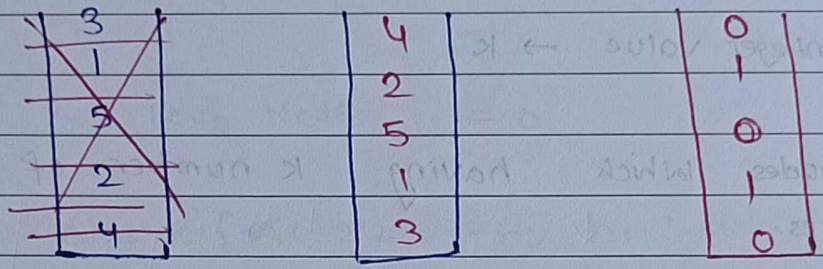
3 → पहले हो चुका है

6 → leaf

ans = {4, 2, 5, 1, 8, 3, 6}

0 → 0 time popped
1 → 1 time popped.

We made here two stack



Code: (By Another Approach)

```
vector<int> inOrder (Node* root) {
        vector<int> result;
        stack<Node*> nodes;

        Node* curr = root;
    while (curr != NULL || !nodes.empty()) {
            while (curr != NULL) {
                nodes.push(curr);
                curr = curr->left;
            }
            curr = nodes.top();
            nodes.pop();

        result.push_back(curr->data);
            curr = curr->right;
    }
    return result;
}
```