(Day - 44)

(Tree - Hard problem)

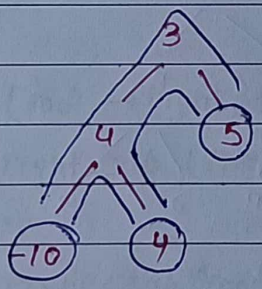
Problem 1 : Maximum path sum between 2 special Nodes

Given a binary tree
↳ each nodes element contain a number

Task : Maximum possible path sum for two special Nodes.
Special Node is a node which is connected to exactly one different Nodes.

दो Node के बीच का sum और दोनो Node सिर्फ एक Node से connect हो।

Example:



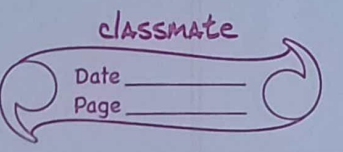
path possible :

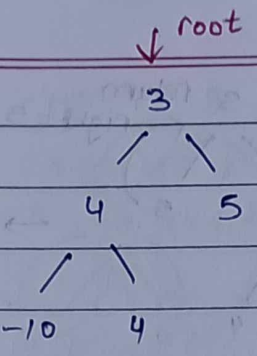
(i) -10 → 4 → 4 Sum = -2

(ii) -10 → 4 → 3 → 5 sum = 2

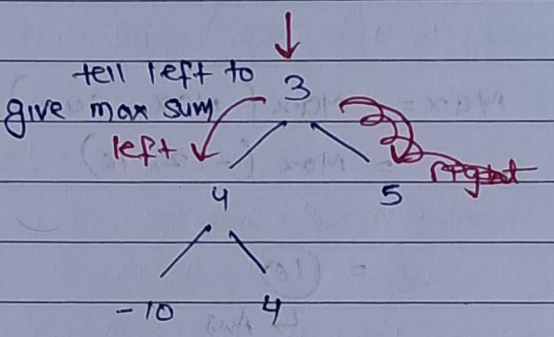
(iii) 4 → 4 → 3 → 5 sum = (16) ↓ Max

How to approach this to solve :

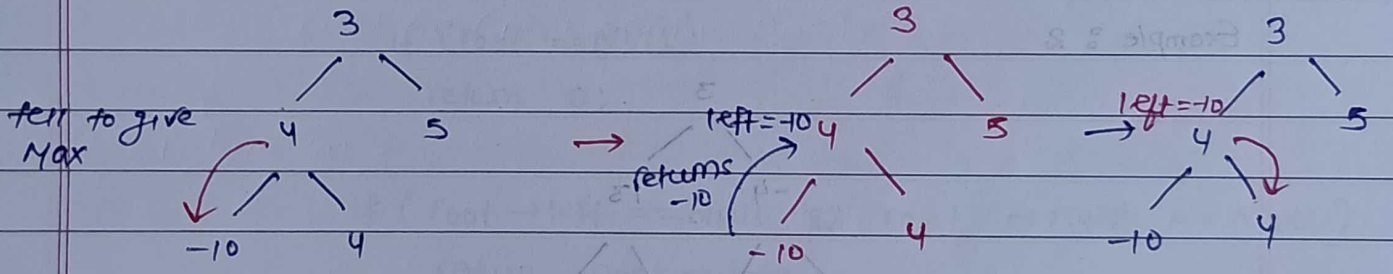


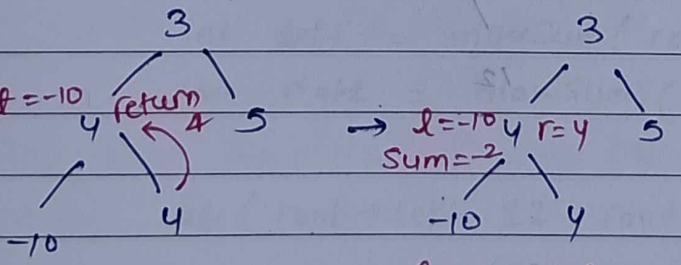


3 बोलेगा दोनो side का Max sum लाकर दो,
और left + right + root → data हो जाएगा।



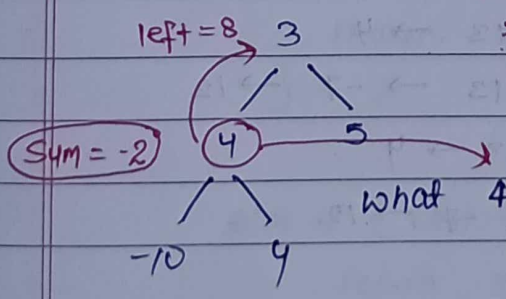
⇒ Max = INT.MIN





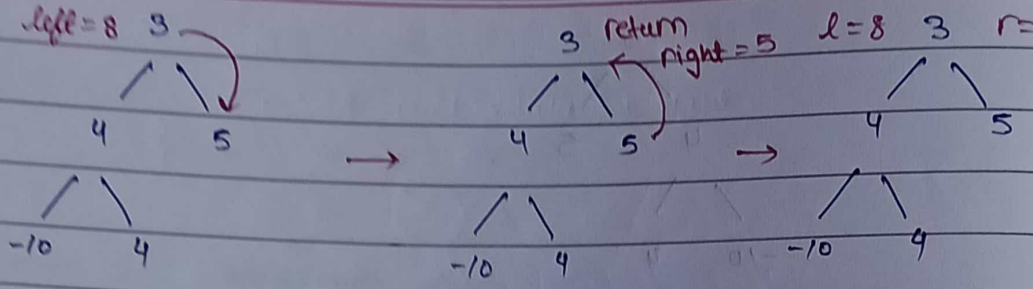
Max = ~~INT. MIN~~
-2

l + r + data
= -10 + 4 + 4
= -2



what 4 return
↳ it has 2 path,
↳ return the Max.
= root → data + max(left, right)

4 + max(-10, 4)
4 + 4
= 8



sum = root → data + left + right

= 3 + 8 + 5

= (16)

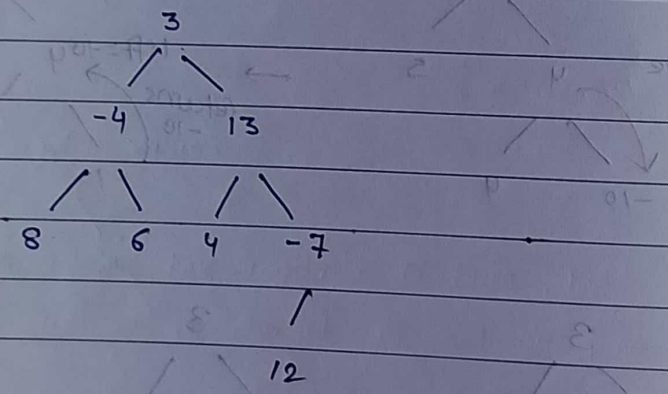
Max = Max (max, sum)

= Max (-12, 16)

= (16)

↳ Ans

Example : 2

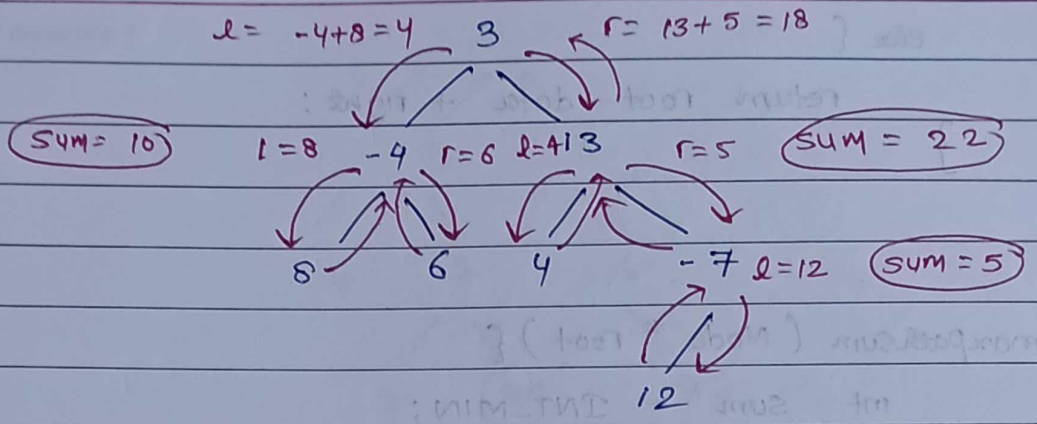


possible path :

10	8 → -4 → 6
24	8 → -4 → 3 → 13 → 4
(25)	8 → -4 → 3 → 13 → -7 → 12
22	6 → -4 → 3 → 13 → 4
23	6 → -4 → 3 → 13 → -7 → 12

Max

Max = 18 + 4 + 3 = 25



Max = ~~MIN~~

~~10~~

~~22~~

25

Code :

```
int maxSum (Node * root, int & sum) {
        if (root == NULL)
            return 0;

        if (root -> left == NULL && root -> right == NULL)
            return root -> data;

    int left = maxSum (root -> left, sum);
    int right = maxSum (root -> right, sum);

    if (root -> left && root -> right) {
        sum = max (sum, root -> data + left + right);
        return root -> data + max (left, right);
    }

    else if (root -> left && root -> right == NULL) {
        return root -> data + left;
    }
```

```
        else {
            return root->data + right;
        }
    }
```

```
int maxpathSum (Node * root) {
        int sum = INT_MIN;
        int val = maxSum (root, sum);
        if (root -> left && root -> right)
            return sum;

        return max (val, sum);
    }
};
```

Problem 2 : Burning Tree

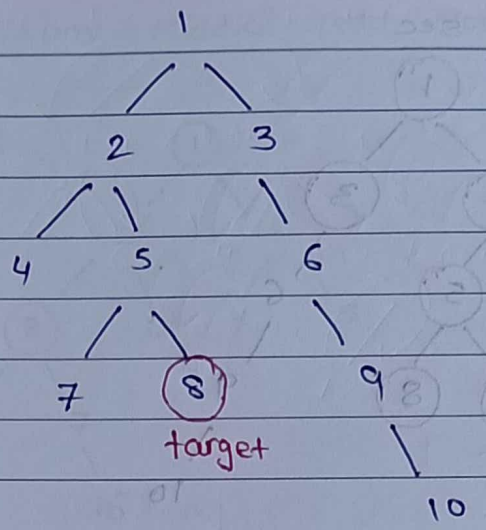
Given a Binary Tree
and a target value.

Task to find minimum time required to burn the complete binary tree, if target is set on fire.

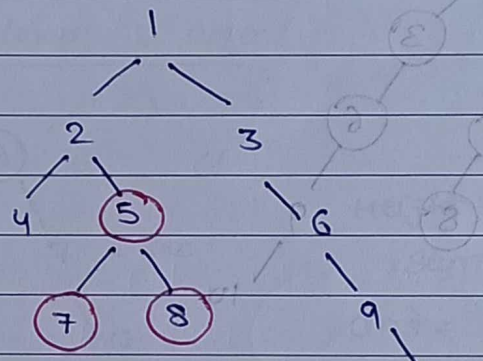
In 1 second all nodes connected to given node get burned.

Note : The tree contains unique values.

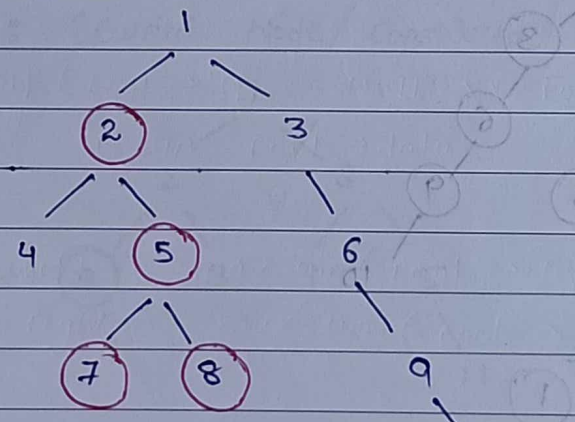
Example:



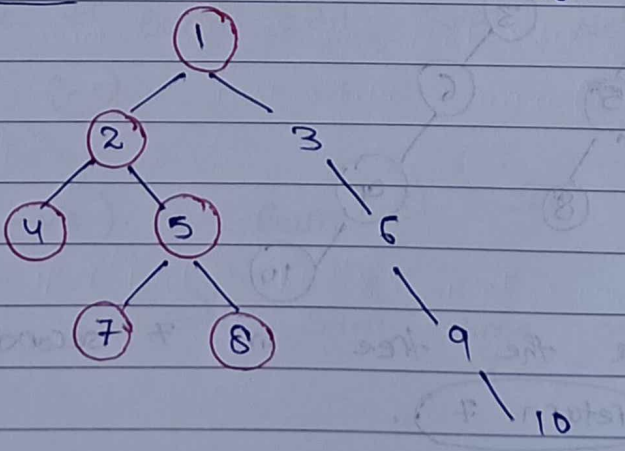
After 1 sec



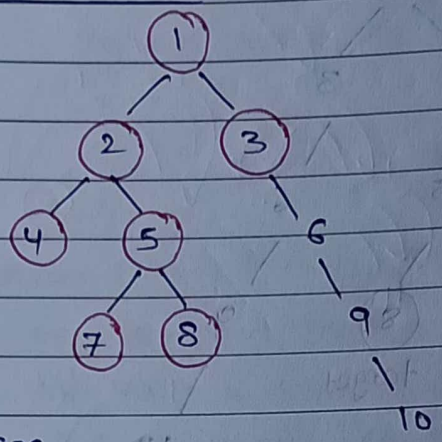
After 2 sec



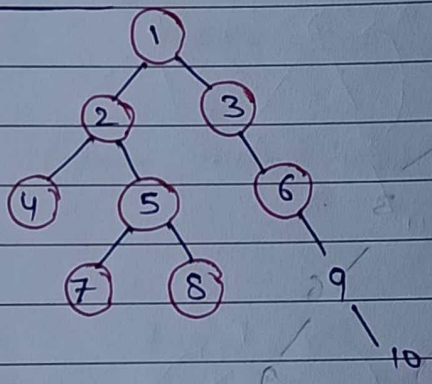
After 3 sec



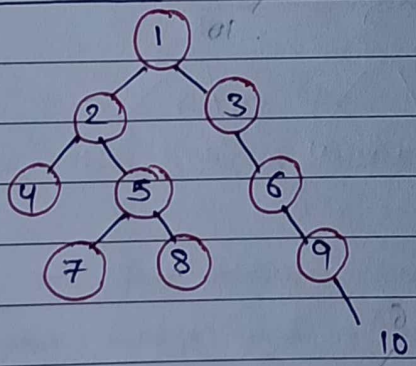
After 4 sec



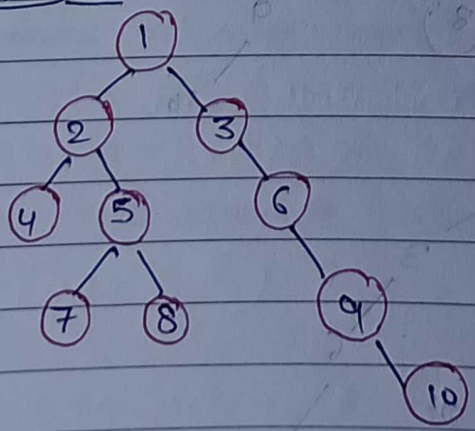
After 5 sec



After 6 sec



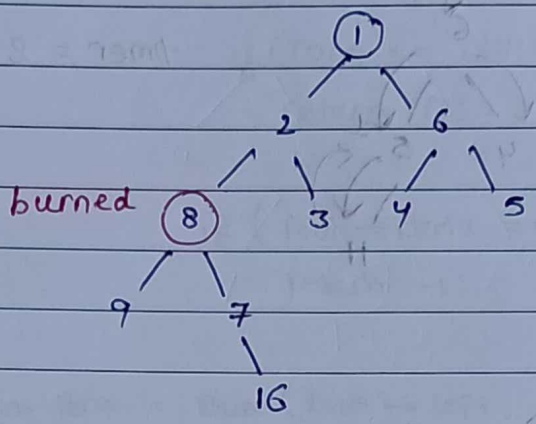
After 7 sec



It burn all the tree in 7 second.

return 7.

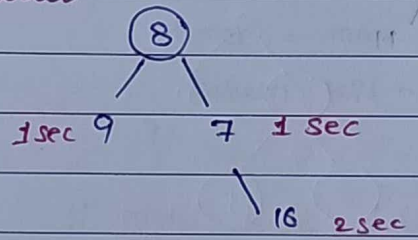
Example 2: If any Middle level Node is Burned.



problem divided in two parts:

lower & upper

For lower:



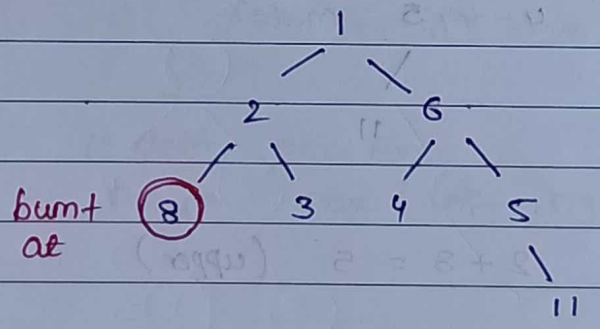
Height = 3

Burned in 2 sec

ans = (Height - 1)

↳ for lower part.

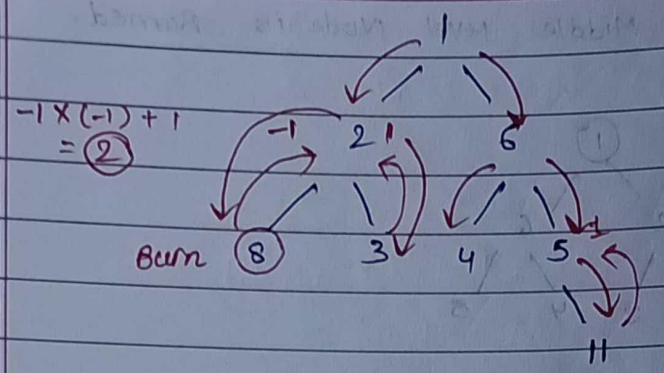
For upper: (Burned Node Considered as leaf Node)

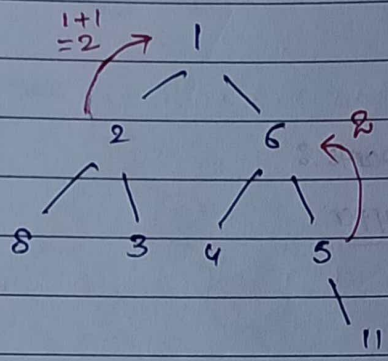


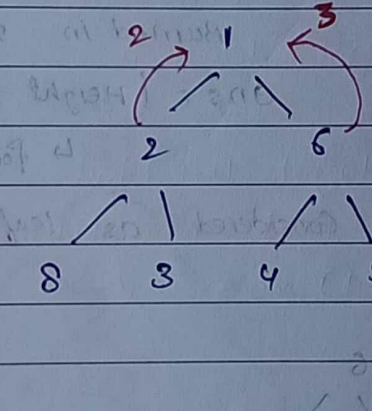
जिस side से Burnt हुआ उसको Negative से denote कर देंगे। (-1)

(-ve) → Burn हो गया

और उतना time लगा







Time = 2 + 3 = 5 (upper)

ans = max (lower_timer, upper_timer)
= max (2 , 5)

ans = 5

Code:

```
int Burn (Node * root, int target, int &timer) {
        if(root == NULL)
            return 0;

        if ( root -> data == target)
            return -1;

    int left = Burn (root -> left, target, timer);
    int right = Burn (root -> right, target, timer);
        // left side burnout
        if(left < 0){
            timer = max (timer, abs(left) + right);
            return left - 1;
        }
        // right side burnout
        if ( right < 0) {
            timer = max( timer, abs(right) + left);
            return right - 1;
        }
        // Both positive
        return max (left, right) + 1;
}

Void find (Node* root, int target, Node * &temp){
        if (root == NULL)
            return;

        if (root -> data == target){
            temp = root;
            return;
        }
```

```
        find (root->left, target, temp);
        find (root->right, target, temp);
    }

int height (Node * root){
        if (root == NULL)
            return 0;

    return 1 + max (height (root->left), height (root->right));
}

int minTime (Node * root, int target){

        int timer = 0;
        Burn (root, target, timer);
        Node *temp;
        find (root, target, temp); // find address of burn node
        int num = height(temp) - 1; // find height of below
        return max (timer, num);    part of the tree.

}
```

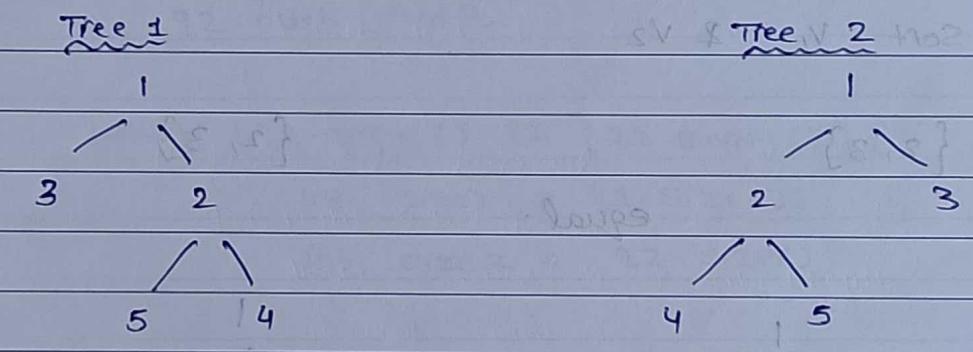
problem 3: Check if all level of two trees are anagram or not

Given two binary trees
↳ having same number of nodes

Task: check at each level that the levels are anagram or not

We use level order traversal here
use Queue for storing the Nodes of each level.

Example:



q1 [X]

q2 [X]

size1 = 1

size2 = 1

same

V1 [1]

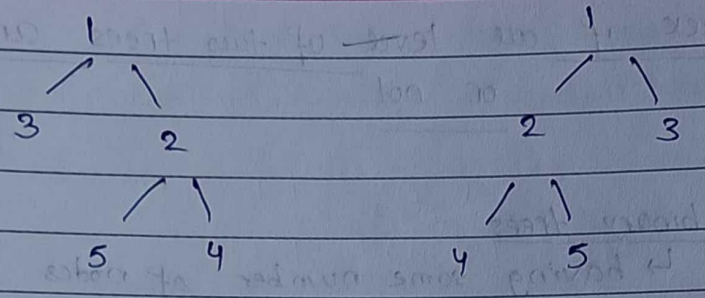
V2 [1]

Sort vector

{1}

{1}

both equal proceed



q_1 | ~~3~~, ~~2~~ | q_2 | ~~2~~, 3 |

size1 = 2 size2 = 2

$V_1 = \{3, 2\}$ $V_2 = \{2, 3\}$

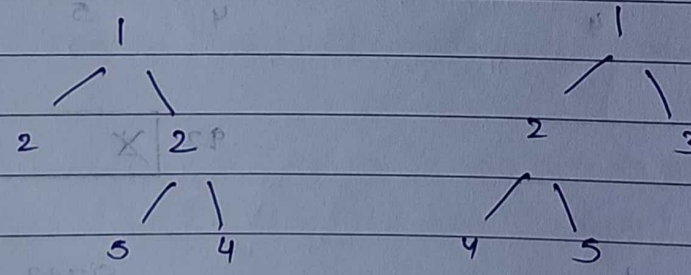
size1--

~~size1 = 0~~

Sort V_1 & V_2

$\{2, 3\}$ $\{2, 3\}$

equal.



q_1 | ~~5~~ ~~4~~ | q_2 | ~~4~~ ~~5~~ |

$V_1 = \{5, 4\}$ $V_2 = \{4, 5\}$

sort

$\{4, 5\}$ $\{4, 5\}$

equal → return 1

size = 0

4 loop bread

Code:

```
Bool areAnagrams (Node * root1, Node * root2) {
    if (root1 == NULL && root2 == NULL) {
        return true;
    }

    if (root1 == NULL || root2 == NULL) {
        return false;
    }

    queue<Node *> q1;
    queue <Node *> q2;
    q1.push (root1);
    q2.push (root2);

    while (!q1.empty() && !q2.empty()) {
        int size1 = q1.size();
        int size2 = q2.size();

        if (size1 != size2) {
            return false;
        }

        Vector <int> V1;
        Vector <int> V2;

        while (size1 > 0) {
            Node* node1 = q1.front();
            Node* node2 = q2.front();
            q1.pop();
            q2.pop();
```

```
        v1.push_back(node1->data);
        v2.push_back(node2->data);

        if(node1->left)
            q1.push(node1->left);

        if(node1->right)
            q1.push(node1->right);

        if(node2->left)
            q2.push(node2->left);

        if(node2->right)
            q2.push(node2->right);

        size--;
    }

    sort(v1.begin(), v1.end());
    sort(v2.begin(), v2.end());

    if(v1 != v2)
        return false;
}

return true;
}
```

Problem 4: Construct tree from Inorder & preorder

Given 2 Array: Inorder & preorder

Tree contain duplicate elements.

Task: Construct a tree and print the postorder Traversal.

Norder: 3 1 4 0 5 2 (LNR)

Preorder: 0 1 3 4 2 5 (NLR)

preorder: 1st Node is always root Node.

root Node.

Tree = { 0

check in Inorder

3 1 4 0 5 2

left side right side.

preorder: 0 1 3 4 2 5

1 Comes left side.

0 Tree = { 0, 1

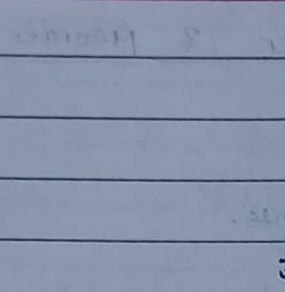
1

3 4

left side right side.

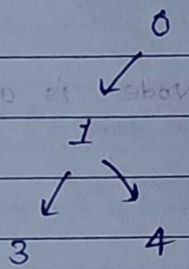
0 1 3 4 2 5

3 Comes left side of 1



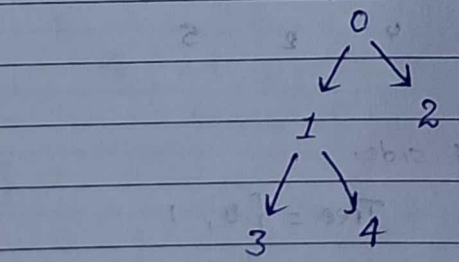
0 1 3 4 2 5

4 comes right of 1.



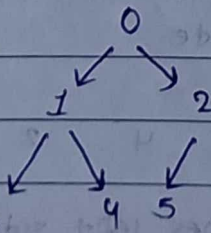
0 1 3 4 2 5

2 comes right of 0



0 1 3 4 2 5

5 comes left of 2.



Tree

preorder : एक - एक कर के traverse कर देंगे।

Inorder : direction बता देगा।

problem 5 : Construct tree from postorder & inorder

In this problem

Inorder direction बता देगा।

postorder : एक एक कर के traverse कर देंगे।

(पीछे से).