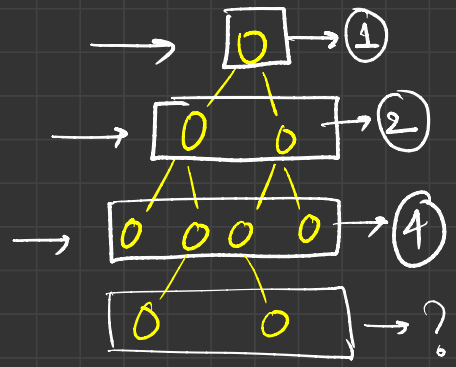
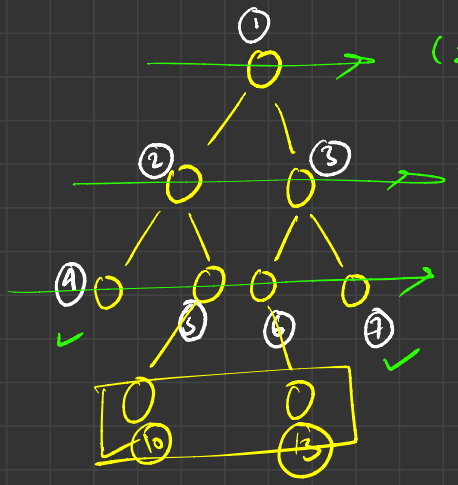
Maximum width of tree





(1 - 1 + 1) = 1

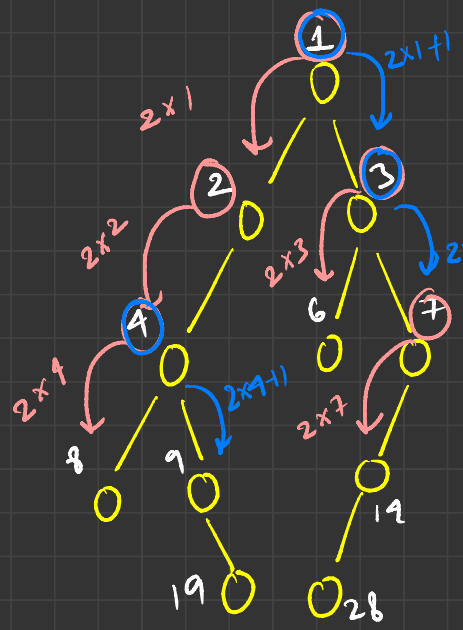
w = 2

(3 - 2 + 1)

w = 4

(7 - 4 + 1)

13 - 10 + 1 = 4



$1 - 1 + 1 = 1$

$3 - 2 + 1 = 2$

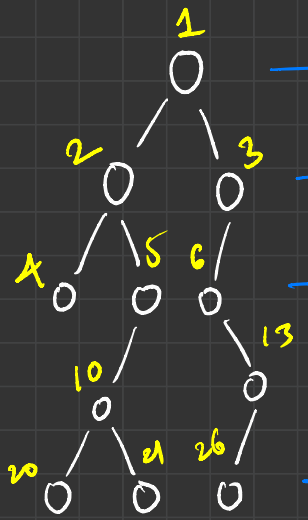
$7 - 4 + 1 = 4$

$14 - 8 + 1 = 7$

$28 - 19 + 1 = 10$

{ left Child Index = $2 \times i$

right Child Index = $2 \times i + 1$ }



$1 - 1 + 1 = 1$

$3 - 2 + 1 = 2$

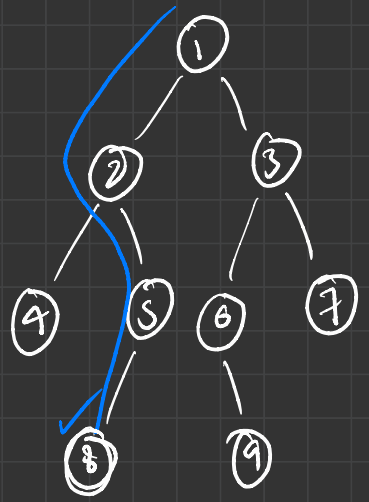
$6 - 4 + 1 = 3$

$13 - 10 + 1 = 4$

$26 - 20 + 1 = 7$

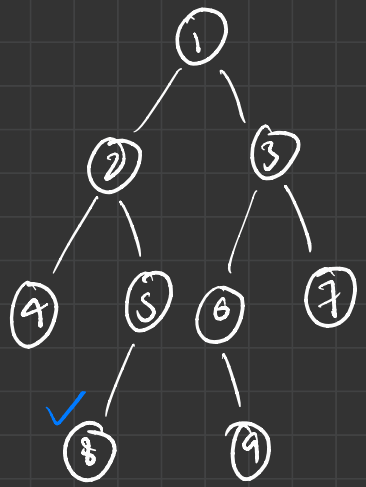
```
class Pair {
    Node;
    Index;
}
```

Path to Given Node



{1 → 2 → 5 → 8}

find A target

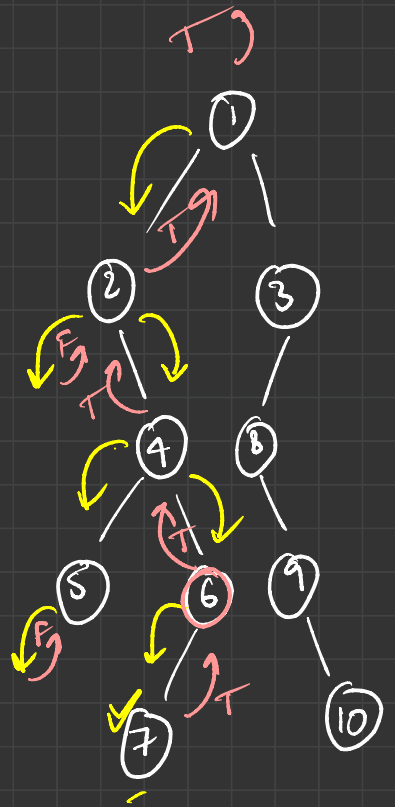


```java
public boolean find(Node root, int tar) {
    if (root == null) {
        return false;
    }

    if (root.data == tar) {
        return true;
    }

    // find in left child
    boolean filc = find(root.left, tar);
    if (filc == true) {
        return true;
    }

    // find in right child
    boolean firc = find(root.right, tar);
    if (firc == true) {
        return true;
    }

    // I was not the target, not found in left child niether in right child
    return false;
}
```

AL

AL

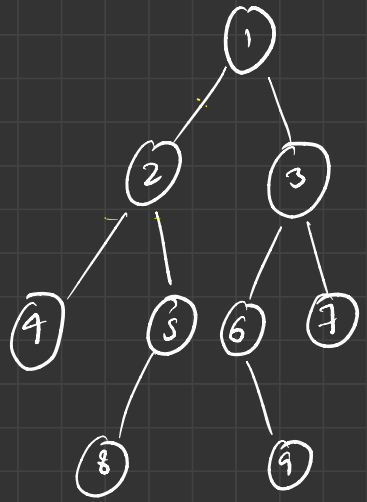
AL

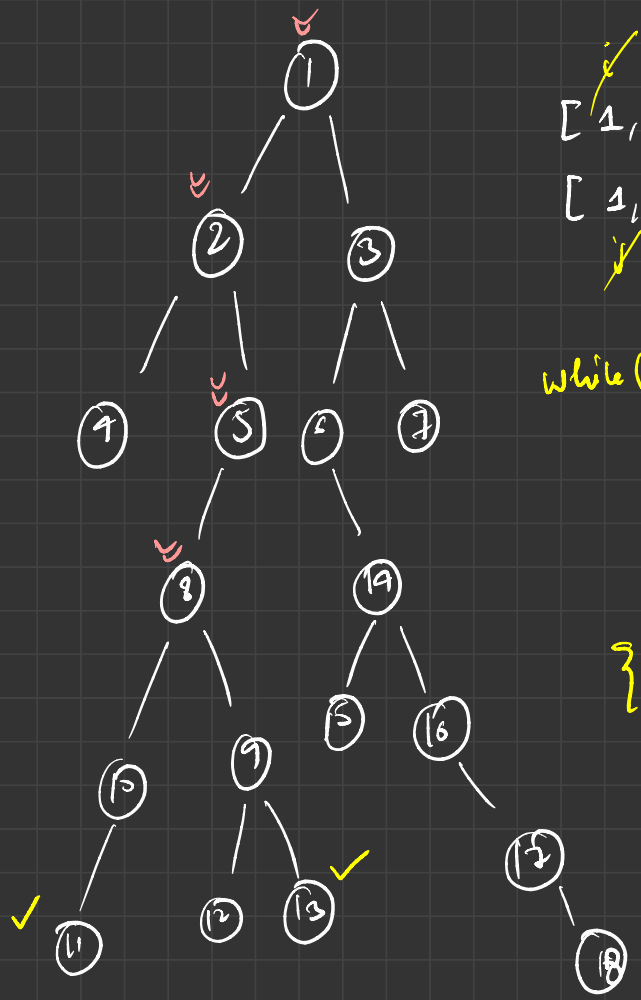


{7, 6, 4, 2, 1}

LCA

↳ lowest Common Ancestor





[1, 2, 5, 8, 9, 13]

[1, 2, 5, 8, 9, 13]

lca = 1 2 5 8

while (i < list1.size() && j < list2.size()) {

if (list1.get(i) == list2.get(i))

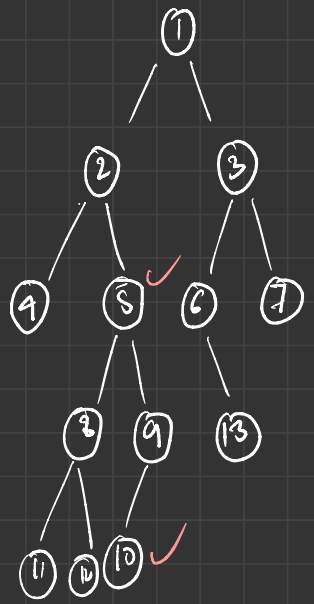
lca = list1.get(i);

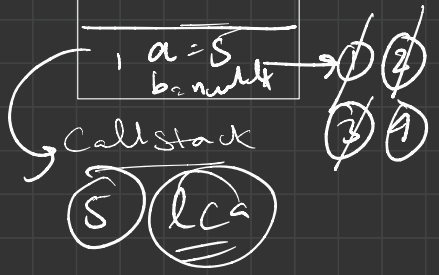
i++;

j++;

}

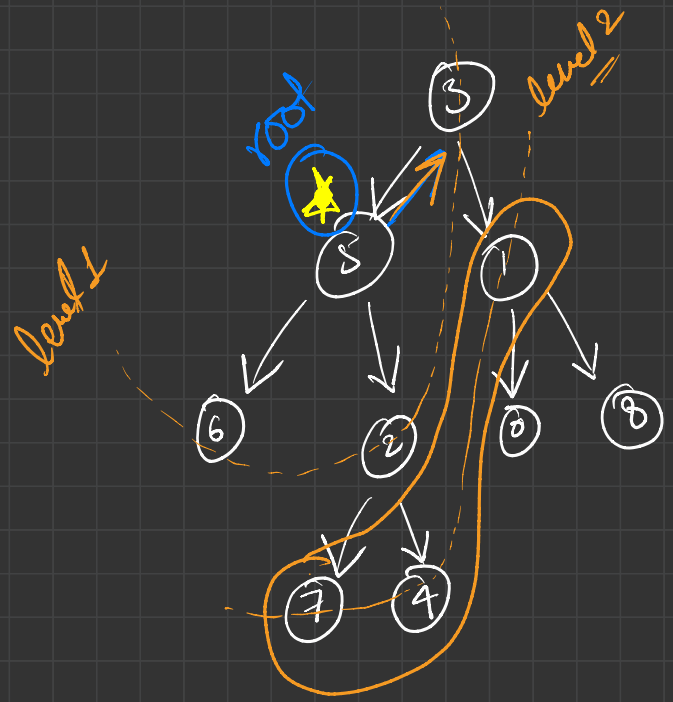
```java
public static Node findLCA(Node node, int n1, int n2) {
    // write code here
    if (node == null) {
        return null;
    }

    if (node.data == n1 || node.data == n2) {
        return node;
    }

    Node findOneInLeft = findLCA(node.left, n1, n2);
    Node findOneInRight = findLCA(node.right, n1, n2);

    if (findOneInLeft != null && findOneInRight != null) {
        return node;
    }

    if (findOneInLeft != null) {
        return findOneInLeft;
    } else if (findOneInRight != null) {
        return findOneInRight;
    } else {
        return null;
    }
}
```





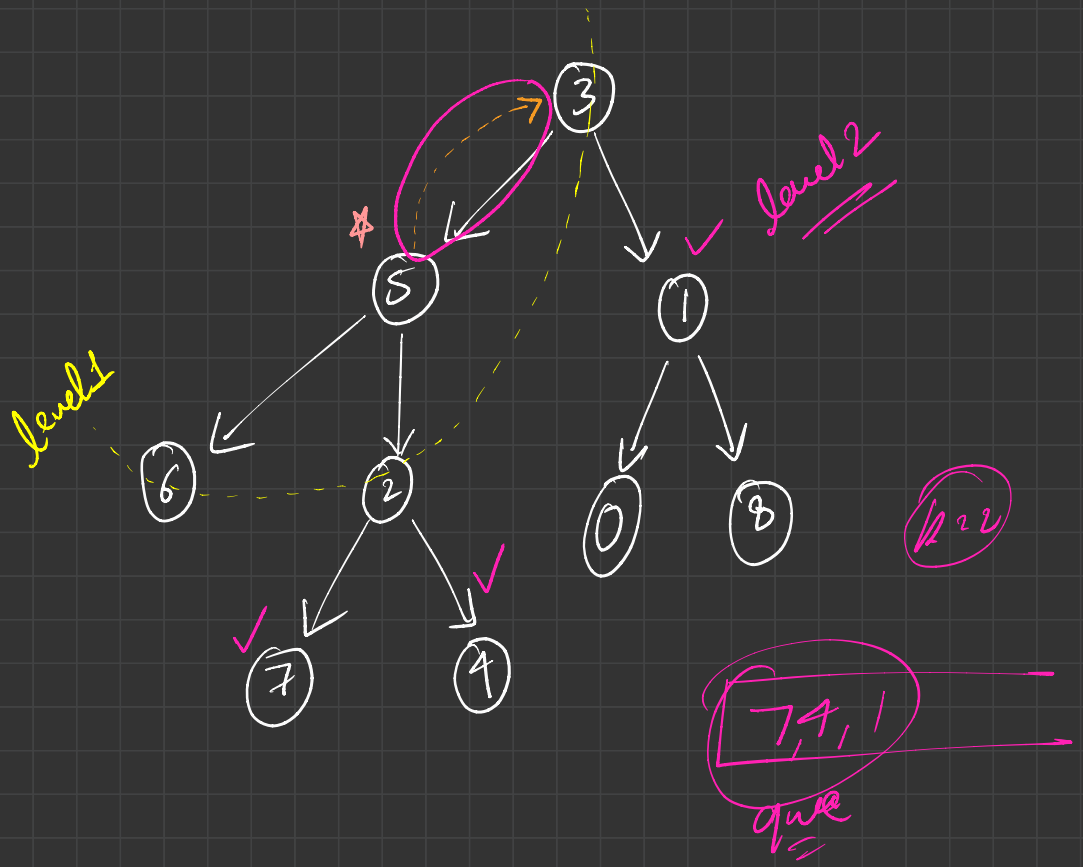
TC: O(N)
SC: O(1)

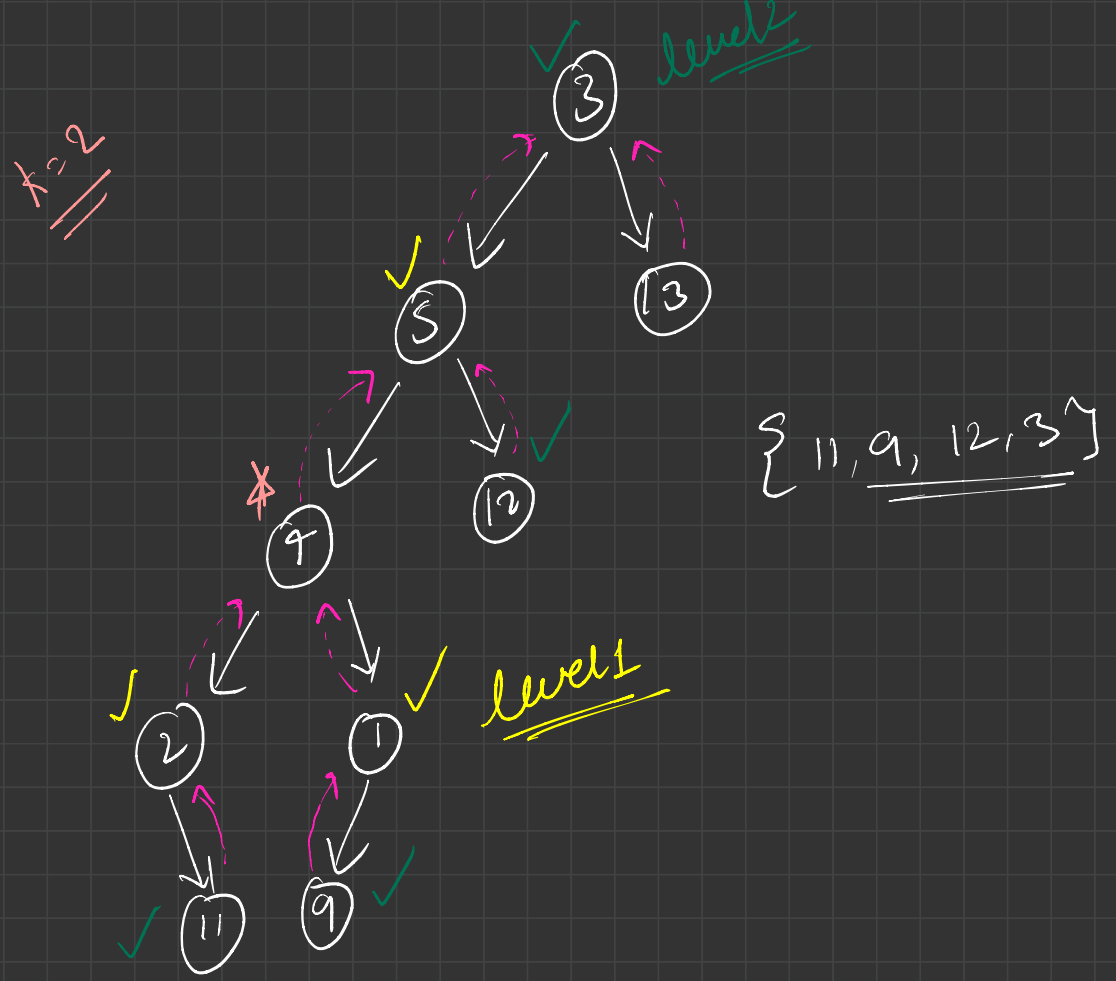
All Nodes Distance K in a Binary Tree

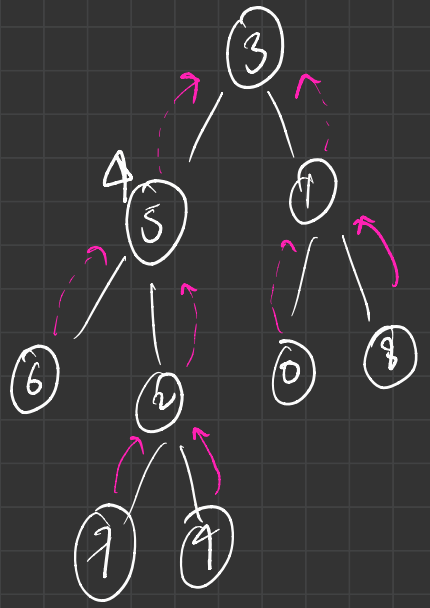


K = 2

{7, 4, 1}

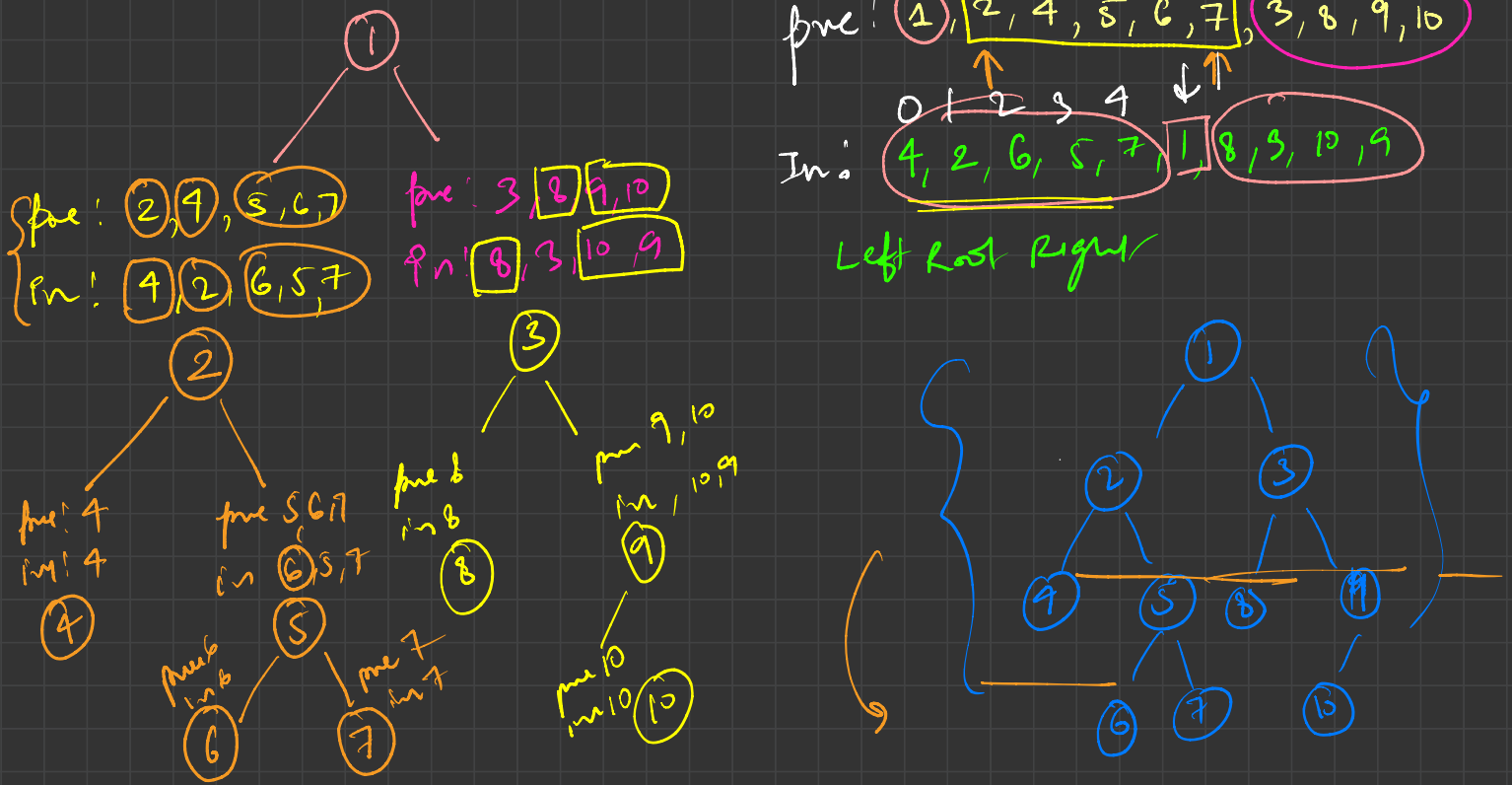

3
5
1
6
2
0
8
7
4
level 1
level 2
7, 4, 1
queue


k=2
level2
3
5
13
7
12
level1
2
1
11
9
{11, 9, 12, 3}



task1 : get parent pointer for every node!

Construct A Binary tree from Preorder & Inorder traversal



```java
class Solution {
    TreeNode constructTree (int[] preorder, int psi, int pei, int[] inorder, int isi, int iei) {
        if (psi > pei) {
            return null;
        }

        if (isi > iei) {
            return null;
        }

        TreeNode root = new TreeNode(preorder[psi]);

        int val = preorder[psi];
        int itr = isi;
        int cntElements = 0;
        while (inorder[itr] != val) {
            cntElements++;
            itr++;
        }

        root.left = constructTree(preorder, psi + 1, psi + cntElements, inorder, isi, itr - 1);
        root.right = constructTree(preorder, psi + cntElements + 1, pei, inorder, itr + 1, iei);

        return root;
    }

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return constructTree(preorder, 0, preorder.length - 1, inorder, 0, inorder.length - 1);
    }
}
```