Day - 43

(problems : Tree Traversal)

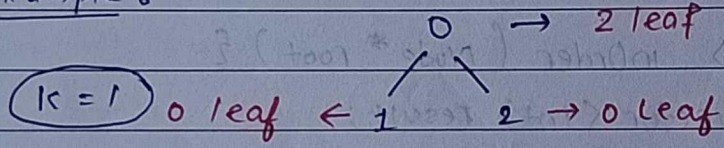
problem 1 : print Nodes having k leaves

Binary tree

integer value → k

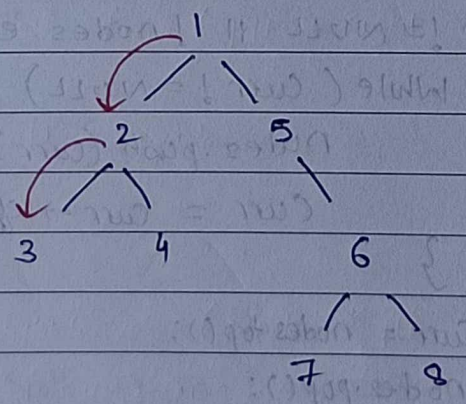
find nodes which having k number of leaves

Example :



return -1

Example :



For 1 → leaf Node = ~~2,5~~ 3, 4, 7, 8 ⇒ ~~2~~ 4

ans = {~~2~~, ~~4~~}

↑

vector

→ For 2 leaf nodes = 3, 4 → 2 Nodes

K = 2

ans = {~~2~~, 2

→ Not updated

→ For 3 leaf Nodes = NULL → (0) leaf nodes

K = 2

ans = {~~2~~, 2,

→ For 4 leaf Nodes = 0

ans = {~~2~~, 2,

→ for 5 leaf Node = ~~2~~ = ~~5~~ {7, 8}

ans = {~~2~~, 2, 5

→ For 6 leaf Node = 2 = {7, 8}

ans = {~~2~~, 2, 5, 6

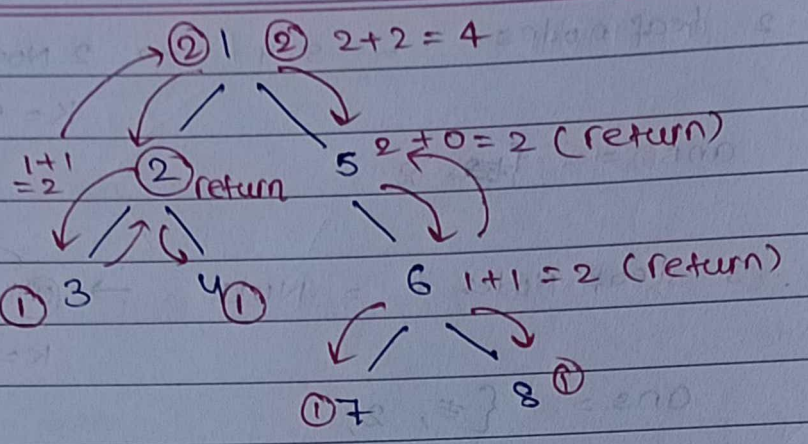
→ For 7 leaf Node = 0

ans = {~~2~~, 2, 5, 6,

→ For 8 leaf node = 0

ans = {~~2~~, 2, 5, 6}

↳ return this.



Code :

```
int find ( Node * ptr , vector <int> & ans,  int K){
            if (ptr == NULL)
                return 0;

            if (ptr → left == NULL && ptr → right == NULL)
                return 1;

        int left = find (ptr→left, ans , K);
        int right = find ( ptr → right , ans, K);

        return left + right;
    }

vector <int> btiWithkleaves (Node * ptr , int K){
            vector <int> ans;
            find (ptr, ans, K);
             if (ans. empty ())
              ans. push_back (-1);

        return ans;

    }
```

Problem 2: print Binary Tree levels in sorted order:

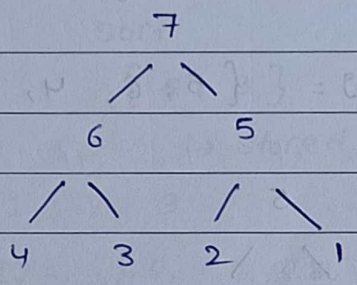
Example:1

N = 7

arr[] = {7, 6, 5, 4, 3, 2, 1}

↓

Binary Tree from this array



levels:

7

6 5

4 3 2 1

sort each levels:

7

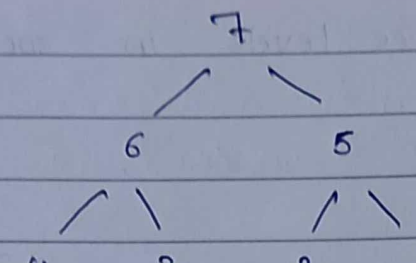
5 6

1 2 3 4

ans = {7, 5, 6, 1, 2, 3, 4}

return.

Approach:



Level 0 → $2^0 = 1$

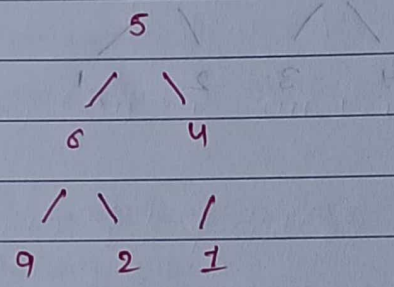
Level 1 = $2^1 = 2$

Level 2 = $2^2 = 4$

At last level, it may be gives less Nodes

Because

arr[] = {5, 6, 4, 9, 2, 1}



At last level only 3 nodes.

We will sort here level wise.

So, we use the Concept of level order traversal.

{ 7 6 5 4 3 2 1 }

At 1st level → only 1 element / Nodes

↳ No need to sort

We will start sorting with level 2.

Approach :

arr = { 1 | 3 2 | 6 7 8 9 | 4 9 }

1 2 3 4

Level 1 → put in ans array. (K = 1)

ans = { 1,

Level 2 → temp[] = { 3, 2 } K = (K * 2) = 2

sort

{ 2, 3 }

↳ stored in ans.

ans = { 1, 2, 3

Level 3 → temp[] = { 6, 7, 8, 9 } K = (K * 2) = 4

sort

{ 6, 7, 8, 9 }

↳ stored in ans.

ans = { 1, 2, 3, 6, 7, 8, 9 K = (K * 2) = 8

Level 4 : temp = { 4, 9 }

sort

{ 4, 9 }

↳ stored in ans.

If not 8, then जितना है उतना निकाल लो।

ans = { 1, 2, 3, 6, 7, 8, 9, 4, 9 }

We have to stored in 2D vector → level wise :

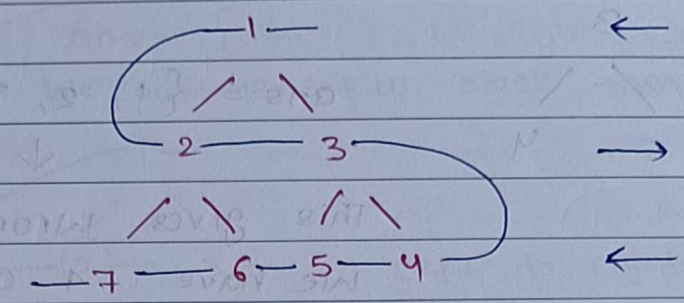
ans = { { 1 }, { 2 3 }, { 6 7 8 9 }, { 4 9 } }

↳ return.

Code :

```
vector<vector<int>> binTreeSortedLevels (int arr[], int n) {
vector <vector <int>> ans ;

  int K = 1;        // for level 1 : No. of Nodes
  int i = 0;

  while (i < n) {

     int numNodes  = min (K, n-i)
                  // No. of nodes at current level.

    vector <int> temp (arr + i, arr + i + numNodes);

    sort (temp.begin(), temp.end());

     ans.push_back (temp);

      i = i + numNodes;
       K *= 2;
  }
  return ans;
}
```

problem 3: level order Traversal in spiral form



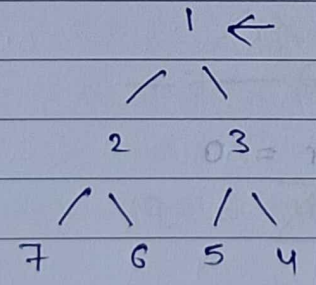
ans = { 1, 2, 3, 4, 5, 6, 7 }

1st level = Right to left
2nd level = Left to Right
3rd level = right to left

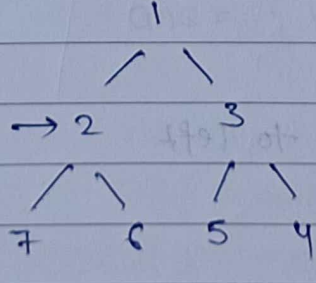
for odd level = Right to left
for even level = left to right.

We use Queue to solve this problem:

Queue: | ~~1~~ ~~2~~ 3 |

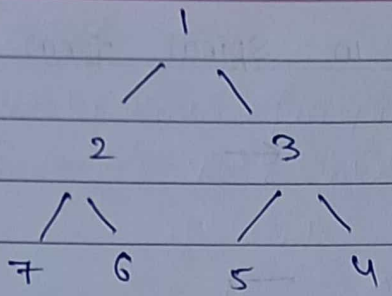


ans = { 1



q | ~~3~~ 7 6 5 6 |

ans = { 1, 2, 3



q: | ~~7~~ 6 5 4

ans = {1, 2, 3 7

↓

This gives wrong answer
We have 4 after 3
but we get 7.

What we have to change:

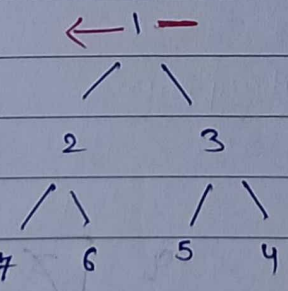
(1) When we move from Right to left
i.e. odd levels.

→ then we put Right Node first in queue then we put left Node.

(2) When we move from left to Right

→ then we put left Node first in queue the we put right node.

Example:



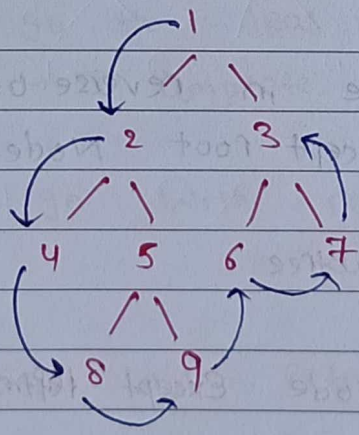
Step 1: Level 1 → Right to left

Code:

```
Vector <int> findSpiral (Node * root) {
    Vector <int> ans;
    if (root == NULL)
      return ans;
    queue <Node *> q;
    stack <Node *> s;

    q.push(root);
    bool dir = 0;

  while ( !q.empty()){
     if (dir == 0){
        int size = q.size();
        while (!q.empty()) {
           Node * temp = q.front();
           q.pop();

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

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

           ans.push_back (temp → data);
     }
     dir = 1;
  }
  else {
        int size = q.size();
        while ( !q.empty()){
           Node * temp = q.front();
```

```
            q.pop();
            if (temp -> left)
            s.push(temp -> left);

            If (temp -> right)
            s.push (temp -> right);

        ans.push_back (temp->data);
    }
    dir = 0;
}
    while (!s.empty()){
        q.push(s.top());
        s.pop();
    }
}
return ans;
}
```

problem 4: Boundary Traversal of binary Tree



ans = 1 2 4 8 9 6 7 3

The problem divides in 3 parts:

→ Left boundary Nodes:

Root se start karna hai
left side traverse karke print karte jana hai.
When you find leaf Node, first print it and the 2nd part comes.

→ Leaf Nodes:

jahaan par left bounday khatm hua uske baad saara leaf Node Traverse karenge aur print karenge.
jo Node left ya ~~root~~ right boundary me aata hai usko chor denge.

→ Reverse right most boundries:

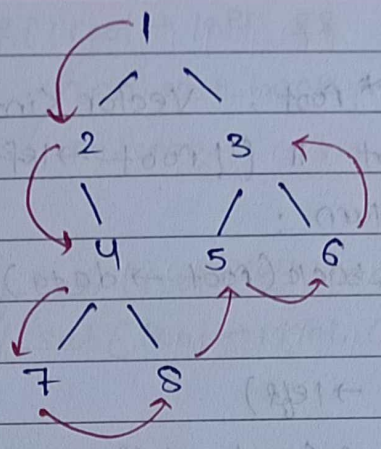
Start from the right most node of root and it is leaf nodes

You can traverse in reverse order and print all except root Node.

① left subtree

② leaf Node Except leftmost & rightmost

③ right subtree in reverse.



left : {1, 2, 4, 7}
↳ leaf Node then stop

leaf : {*, 8, 5, *}
↓ ↳ right most
left most

right : {6, 3, *}
↳ root

Here we first call the left subtree.
↳ It complete the work.

then, we go to leaf Node,
↳ It complete the work.

then, we go with reverse right subtree.

then at last return the ans.

Code :

```
Void left_sub (Node * root, vector <int> & ans ){
        if ( !root || (!root->left && !root->right))
            return ;
        ans.push_back (root->data);

        if (root->left)
            left_sub (root->left, ans);
        else
            left_sub (root->right, ans);

    return;
}

Void right_sub (Node * root , vector <int> & ans ){
        if (!root || (!root->left && !root->right))
            return ;

        if (root->right)
            right_sub (root->right, ans);
        else
            right_sub (root->left, ans);

        ans.push_back (root->data);
}

Void leaf_sub (Node * root, vector <int> & ans){
        if ( root == NULL ){
            return;
        }
```

```
    if(!root->left && !root->right){
        ans.push_back(root->data);
        return;
    }
    leaf_sub(root->left, ans);
    leaf_sub(root->right, ans);
}

vector<int> boundary(Node *root){

    vector<int> ans;
    ans.push_back(root->data);
    left_sub(root->left, ans);
    if(root->left || root->right)
     leaf_sub(root, ans);
    right_sub(root->right, ans);

  return ans;
}
```