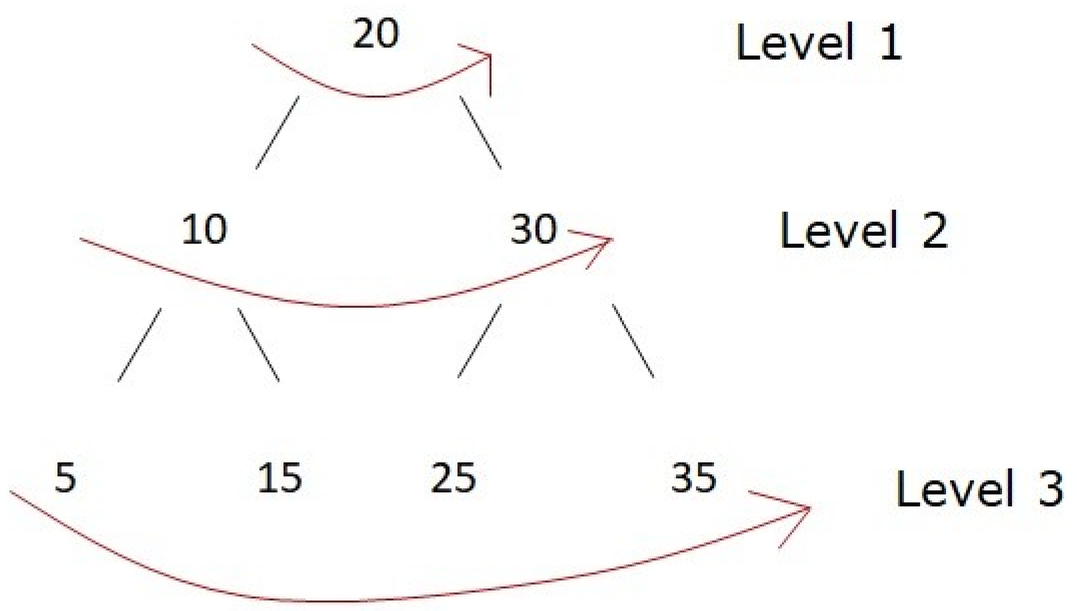
Level Order Traversal of a Binary Tree

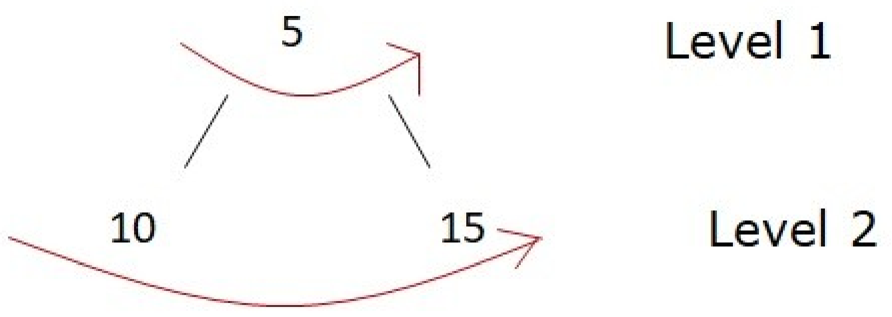
Problem Statement: Level order traversal of a binary tree. Given the root node of the tree and you have to print the value of the level of the node by level.
Example 1:



Output:

```
20 10 30 5 15 25 35
```

We will print the nodes of the first level (20), then we will print nodes of second level(10,30) and at last we will print nodes of the last level(5,15,25,35)
Example 2:

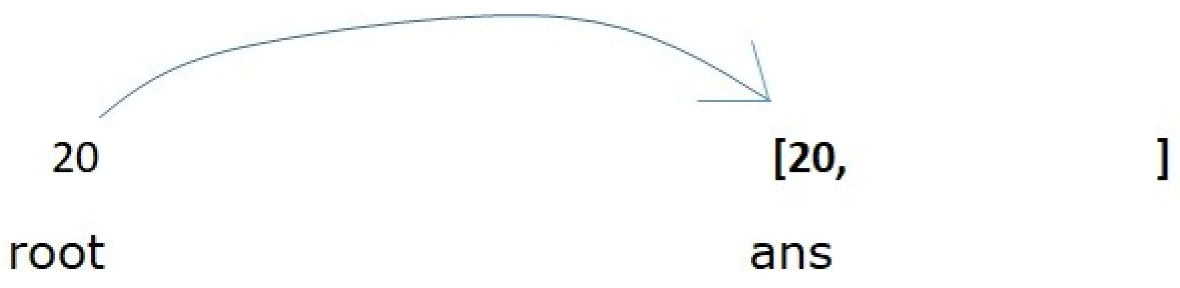


```
5 10 15
```

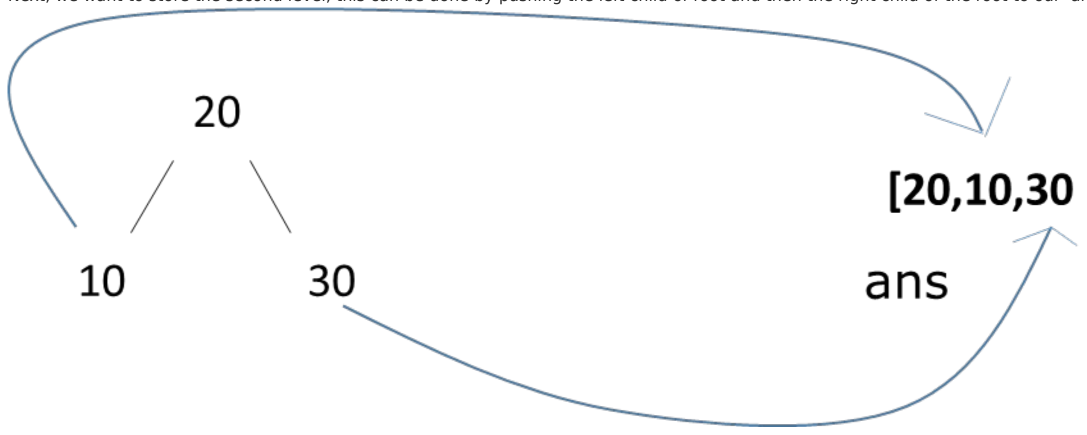
Solution:

Disclaimer*: Don't jump directly to the solution, try it out yourself first.*

Let us consider the example 1 given above.
We need to print a level by level traversal. Let us say, we store all the levels in a data structure named 'ans'. We can use a vector<int> or List<int> to store our answer.
Now we need to store the first level to 'ans'. In any given binary tree, the first level will always be the root node, so we can easily store it.



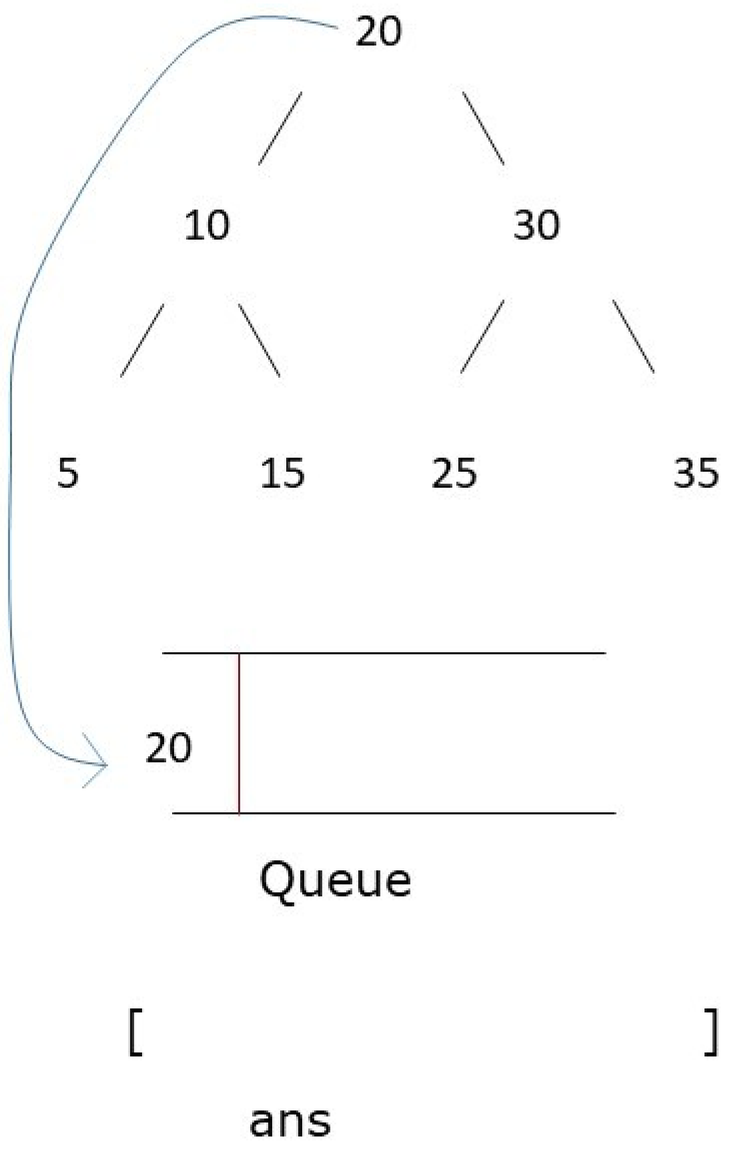
Next, we want to store the second level, this can be done by pushing the left child of root and then the right child of the root to our ‘ans’ data structure.



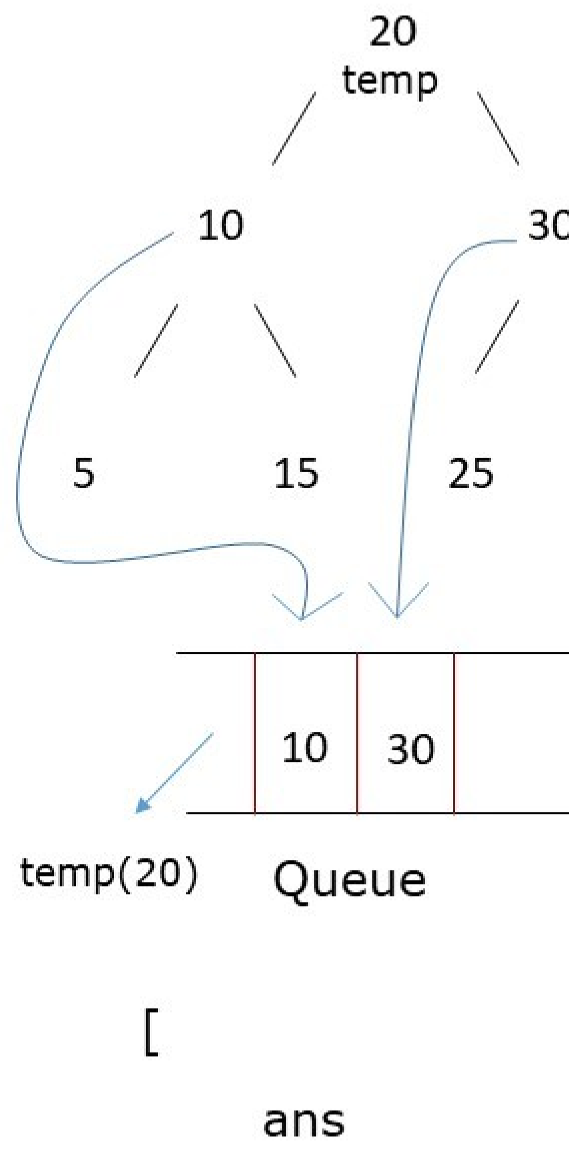
Now what about the next level? We just can’t keep writing root→left→left / root→left→right and so on to reach further levels. We need an additional data structure to store all the nodes of a level. When we are at level 1, we want its left child to be stored first, followed by the right child as the next level and store it in our data structure. Similarly at level 2, we want to access first the left child (which was added first to the data structure), to store the nodes of the next level.
As there is a first-in first-out (FIFO) operation, we will use the **queue** data structure.
Approach:
The algorithm steps are stated as:

- Take a queue data structure and push the root node to the queue.
- Set a while loop which will run till our queue is non-empty.
- In every iteration, pop out from the front of the queue and assign it to a variable (say temp).
- If temp has a left child, push it to the queue.
- If temp has a right child, push it to the queue.
- At last push the value of the temp node to our “ans” data structure.

Dry Run:
We will discuss example 1.



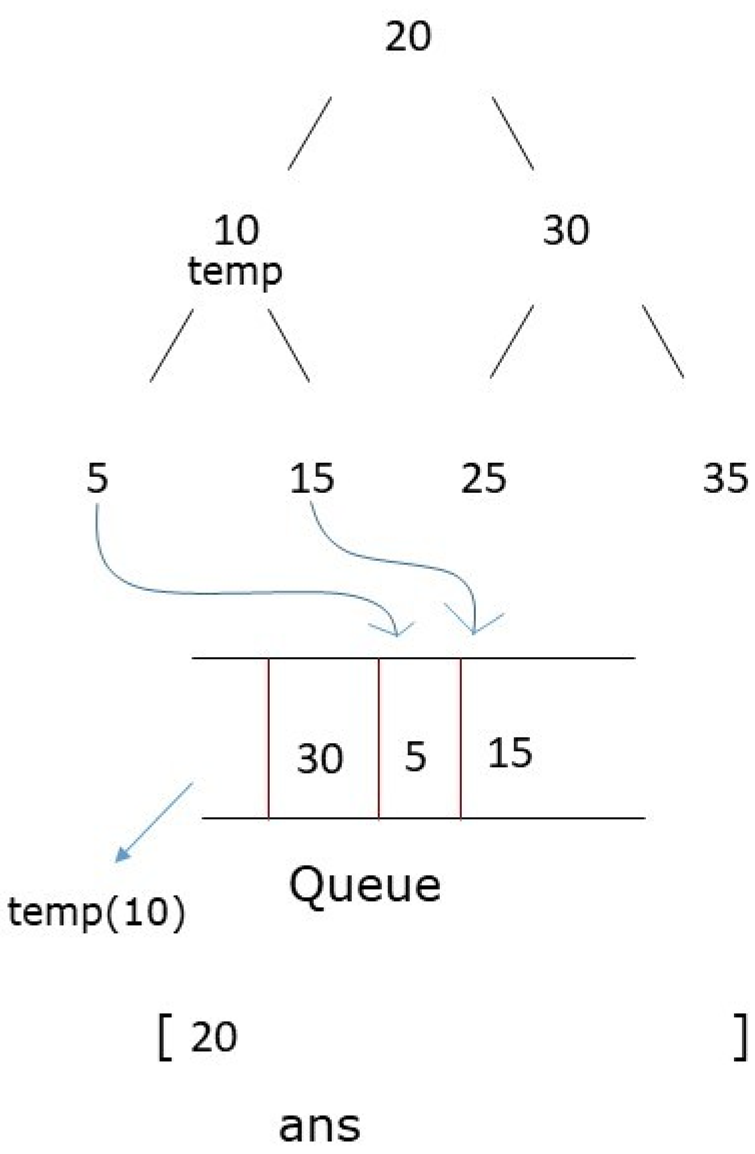
(i)



(ii)

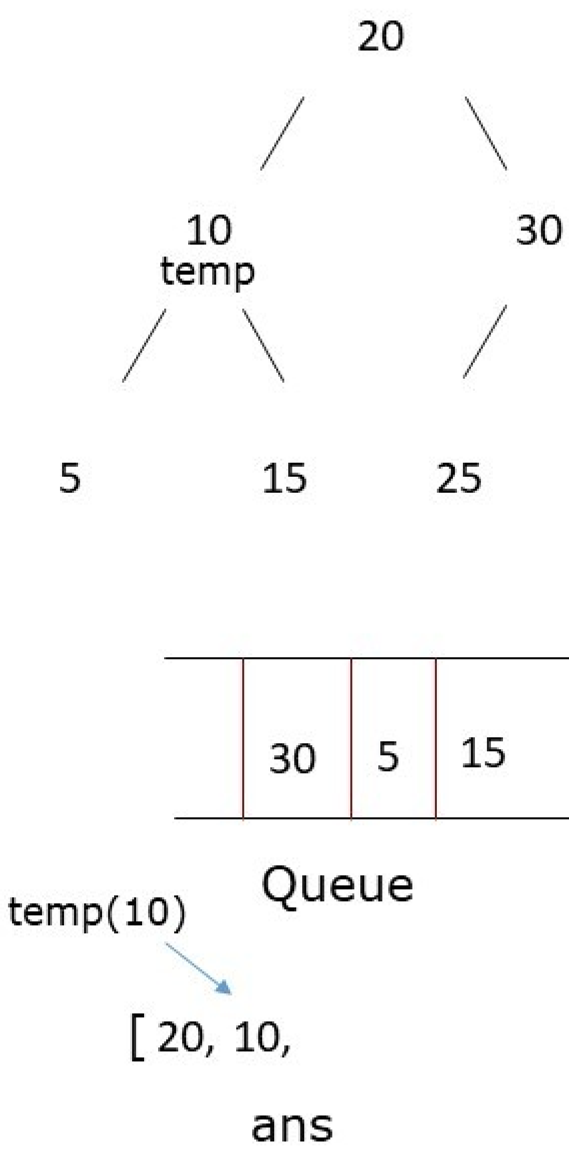
1. First we insert the root node to the queue.
2. We set our while loop and pop its front element as temp Then we inserted the left child followed by the right child of temp to the queue.
3. Then we push the temp value to our ans list.

   Then we follow these steps for all nodes till the time our queue is non-empty, as shown in the figures below.



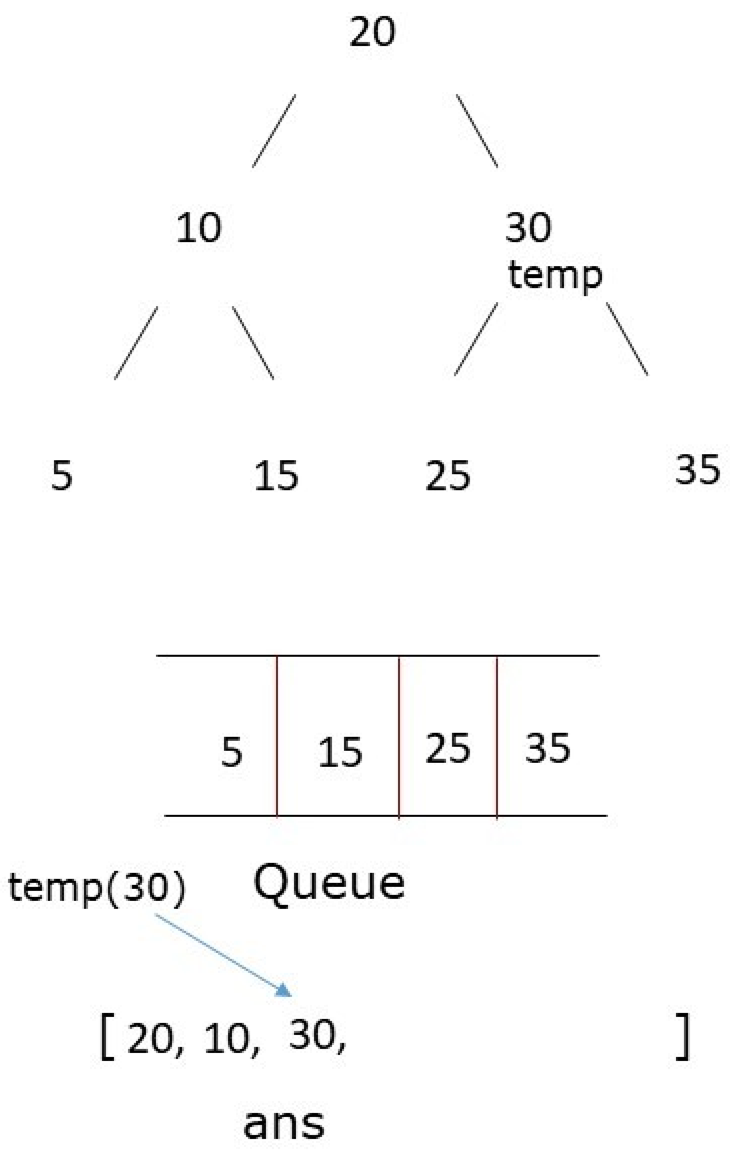
20
10
temp
30
5
15
25
35
30
5
15
Queue
temp(10)
[20]
ans

(iv)

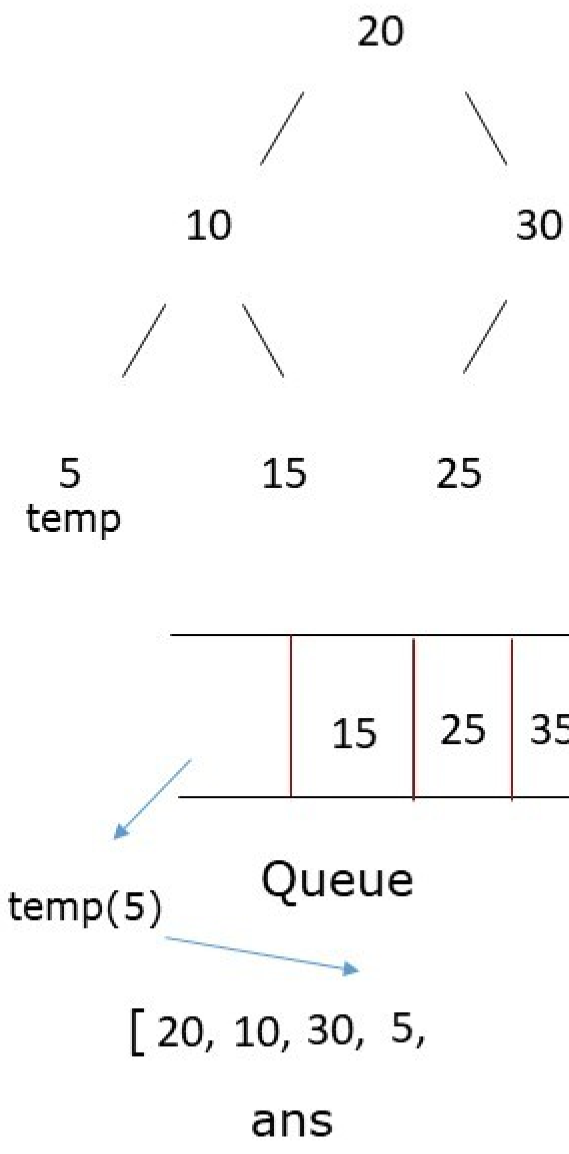


20
10
temp
30
5
15
25
30
5
15
Queue
temp(10)
[20, 10,
ans

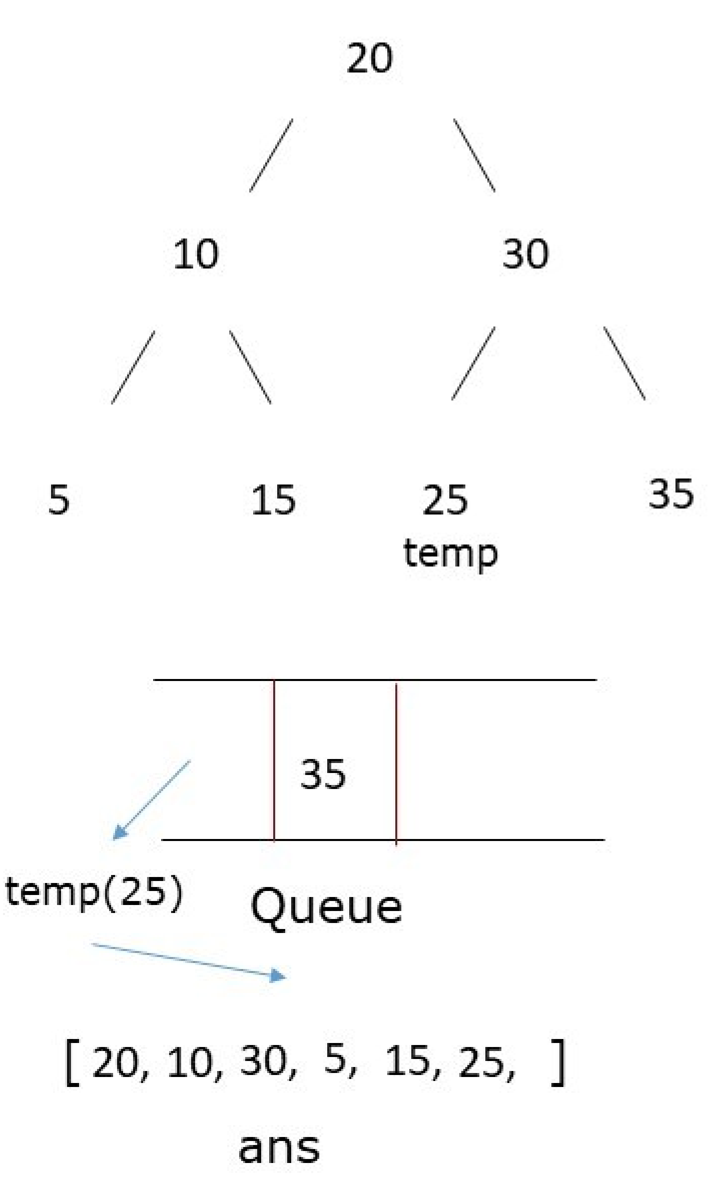
(v)


20
10
30
temp
5
15
25
35
5
15
25
35
temp(30)
Queue
[20, 10, 30,]
ans

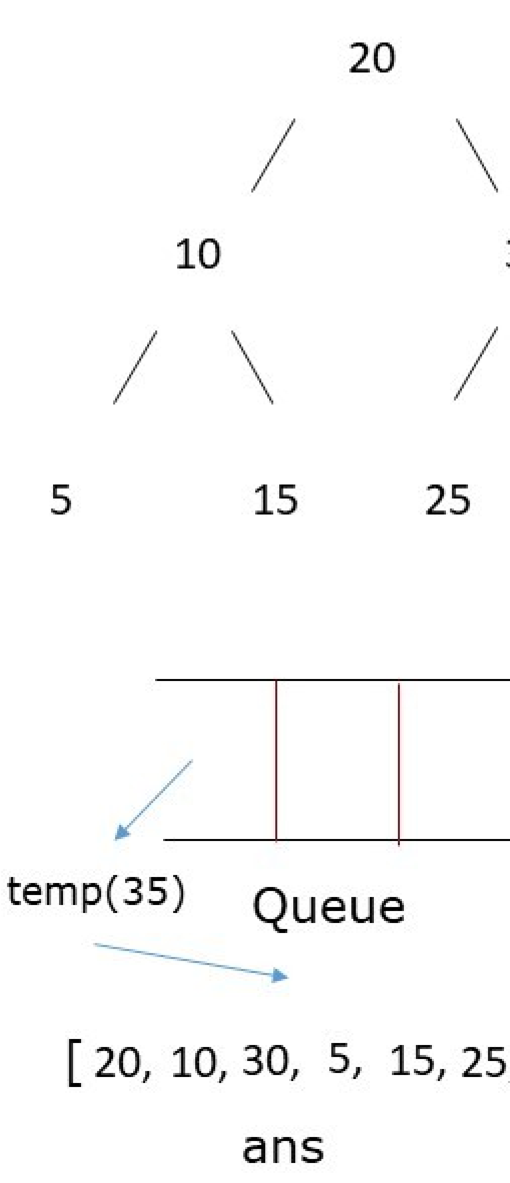
(vii)


20
10
30
5
temp
15
25
15
25
35
temp(5)
Queue
[20, 10, 30, 5,
ans

(viii)



(x)



(xi)

Code:

- C++ Code
- Java Code

```
class Solution {
public:
    vector<int> levelOrder(TreeNode* root) {
        vector<int> ans;

        if(root == NULL)
            return ans;

        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()) {

            TreeNode *temp = q.front();
            q.pop();

            if(temp->left != NULL)
                q.push(temp->left);
            if(temp->right != NULL)
                q.push(temp->right);

            ans.push_back(temp->val);
        }
        return ans;
    }
};
```

Time Complexity: O(N)

Space Complexity: O(N)

What if we have to print the level numbers as well?

In the above approach we print the nodes level-wise but we can't differentiate from our ans that whether two nodes are from the same level or not.

To store the level-order traversal along with individual levels stored together ([[20],[10,30],[5,15,25,35]]), we need to make the following changes:

- First we need to declare a 2d array to store our answer(vector<vector<int>> in C++ and List<List<int>> in Java).
- Inside the while loop, first we declare a list to store nodes of a level (say level), then we need to set another for loop, which iterates for the size of the queue and inside this for loop we need to write the logic which we had discussed in the first approach
- The for loop ensures that all the nodes of a particular level are inserted together and when the iteration of the for loop ends, the queue contains the elements of only one level at a time.
- Inside the for loop we push the value of temp to 'level'.
- After the for loop ends, we push 'level' to the answer.