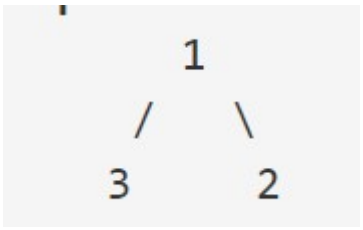
Bottom view of a Binary Tree
Problem Statement: Given a binary tree, print the bottom view from left to right. A node is included in the bottom view if it can be seen when we look at the tree from the bottom.
Example 1:

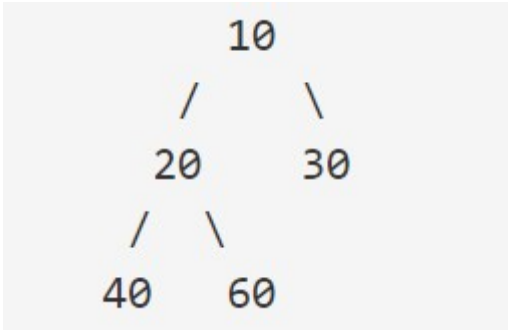
Input:



Output: 3 1 2

Explanation:
Explanation:
If you look up from the bottom from left to right then first we get 3, then 1 and 2.
Example 2:

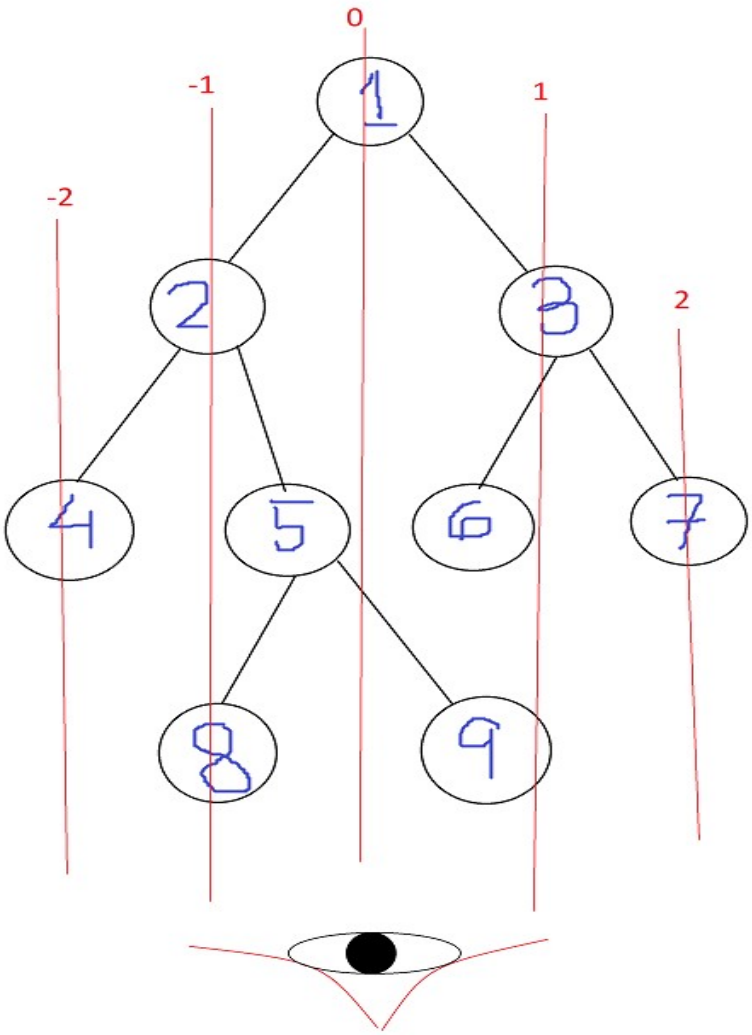
Input:



Output: 40 20 60 30

Explanation:
If you look up from the bottom from left to right then first we get 40, then 20, 60(it blocks 10), and 30.
Disclaimer: *Don't jump directly to the solution, try it out yourself first*

Solution
Intuition: We can mark straight lines like in the image below and mark them with +ve and -ve indexes. The Last node of every line will be my Bottom view.



Approach:

- First we have to make a queue of pair which have nodes and their respective +ve and -ve indexes.
- Then we need a map data structure to store the lines and the nodes. This map will store the data in the form of sorted orders of keys(Lines).
- Here we will follow the level order traversal.
- Traverse through the nodes starting with root,0 and store them to the queue.
- Until the queue is not empty, store the node and line no. in 2 separate variables .
- Then store the line and the node->val to the map, if there will be any node value present that corresponds to a line in the map , it will be replaced by the new node value and by this we will get the last node of each line.
- Store the node->left and node->right along with their line nos. to the queue.
- Then print the node->val from the map

Tip: Here there is only 1 small difference from the Top View of the Tree. Here we don't need to check whether the node is previously present on the map or node before entering it. We have to replace the node of each line if that was previously present on the map.

Code:

- C++ Code

- Java Code

```
class Solution {
  public:
    vector <int> bottomView(Node *root) {
        vector<int> ans;
        if(root == NULL) return ans;
        map<int,int> mpp;
        queue<pair<Node*, int>> q;
        q.push({root, 0});
        while(!q.empty()) {
            auto it = q.front();
            q.pop();
            Node* node = it.first;
            int line = it.second;
            mpp[line] = node->data;

            if(node->left != NULL) {
```

```
                q.push({node->left, line-1});
            }
            if(node->right != NULL) {
                q.push({node->right, line + 1});
            }

        }

        for(auto it : mpp) {
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Time Complexity: O(N)

Space Complexity: O(N)