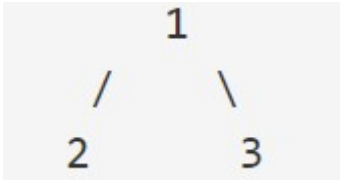
Top view of a Binary Tree
Problem Statement: Given below is a binary tree. The task is to print the top view of the binary tree. The top view of a binary tree is the set of nodes visible when the tree is viewed from the top.
Example 1:

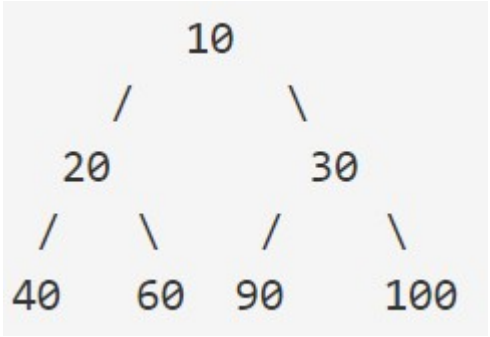
Input:



Output: 2 1 3

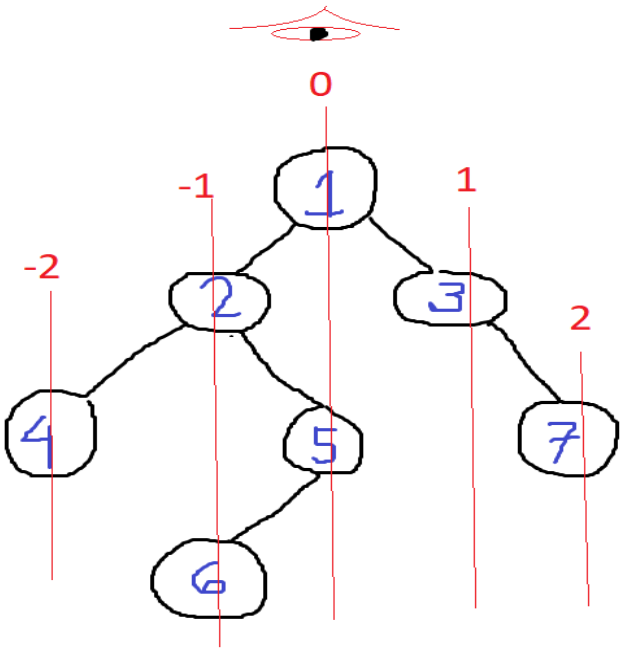
Example 2:

Input:



Output: 40 20 10 30 100

Solution:
Intuition: We can mark straight lines like in the image below and mark them with +ve and -ve indexes. The first node of every line will be my top 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 variable .
- Then check if that line is present in the map or not
- If not present then store the line and the node->val to the map
- Otherwise store the node->left and node->right along with there line nos. to the queue.
- Then print the node->val from the map

Code:

- C++ Code
- Java Code

```
class Solution
{
    public:
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    vector<int> topView(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;
            if(mpp.find(line) == mpp.end()) 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)