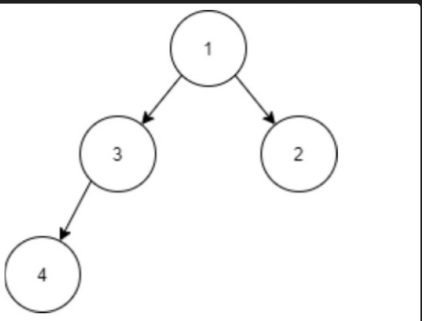
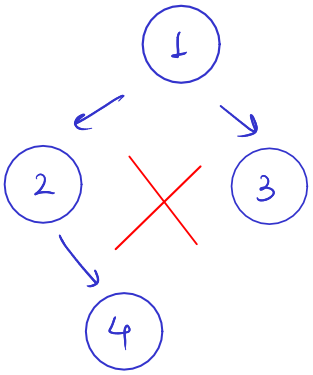
If Check if binary tree is complete

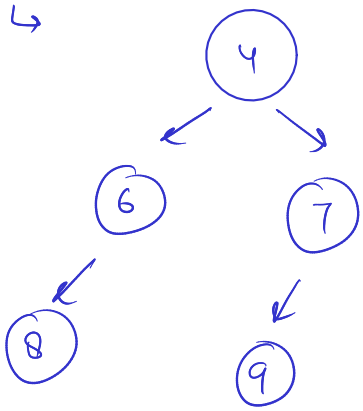
You are given a binary tree. Your task is to check whether the given binary tree is a Complete Binary tree or not.
A Complete Binary tree is a binary tree whose every level, except possibly the last, is completely filled, and all nodes in the last level are placed at the left end.
Example of a complete binary tree :



→ Approach 1 :-



↳




```
#include <bits/stdc++.h>

int isCompleteBinaryTree(TreeNode<int> *root) {
  queue<TreeNode<int> *> q;

  q.push(root);
  bool end = false;

  while (!q.empty()) {
    TreeNode<int>* temp = q.front();
    q.pop();

    if(end && (temp->left || temp->right)) return false;
    if(temp->right && !temp->left) return false;
    if(!temp->left || !temp->right) end = true;

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

  }
  return true;
}
```

4 | 6 7 | 8 9

↓

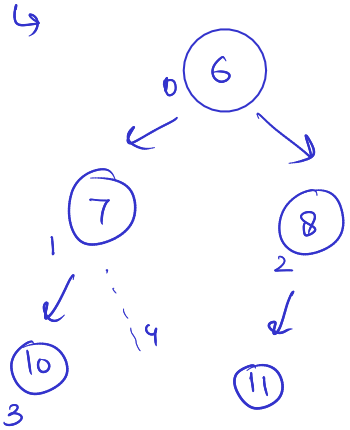
does not have right child ie no next element should be present.
But we have 9 after that
∴ It's not complete.

Time Complexity: O(N)
Space Complexity: O(N)

→ Approach 2 :-

↳ Complete binary tree should follow left & right children $2*i+1$ & $2*i+2$ (0 based) & the value of left & right should lie under the node count.

↳



Node count = 5

6	7	8	10		11

But we have array of size 5 only and for 11 we must have size of 6.
ie. It's not a complete binary tree.

```cpp
#include <bits/stdc++.h>

int size(TreeNode<int>* root) {
  if(root == NULL) return 0;

  int left = size(root->left);
  int right = size(root->right);

  return 1 + left + right;
}

bool isCBT(TreeNode<int>* root, int i, int nodeCount) {
  if(root == NULL) return 1;
  if(i >= nodeCount) return 0;
  else {
    bool left = isCBT(root->left, 2 * i + 1, nodeCount);
    bool right = isCBT(root->right, 2 * i + 2, nodeCount);
    return left && right;
  }
}

int isCompleteBinaryTree(TreeNode<int> *root) {
  int nodeCount = size(root);

  return isCBT(root, 0, nodeCount);
}
```

Time Complexity : O(n)

Space Complexity : O(n)