Left View of Binary Tree

Given a Binary Tree, print Left view of it. Left view of a Binary Tree is set of nodes visible when tree is visited from Left side. The task is to complete the function leftView(), which accepts root of the tree as argument.

Left view of following tree is 1 2 4 8.

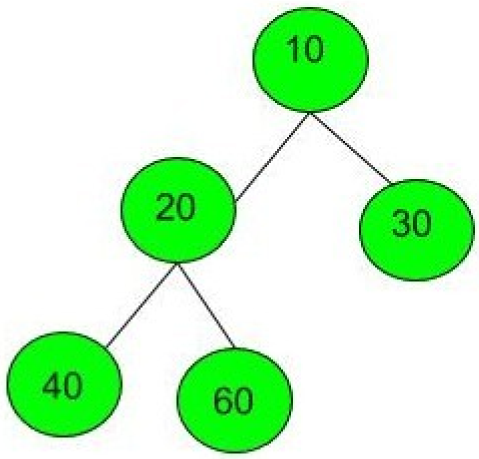
```
          1
        /    \
      2       3
    /    \   /   \
   4     5  6    7
    \
      8
```

Example 1:

```
Input:
   1
 /  \
3    2
Output: 1 3
```

Example 2:

Input:



Output: `10 20 40`

Your Task:
You just have to **complete** the function **leftView()** that prints the left view. The newline is automatically appended by the driver code.
Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Height of the Tree).

Constraints:
1 <= Number of nodes <= 100
1 <= Data of a node <= 1000

Check for BST

Given a binary tree. Check whether it is a BST or not.

Example 1:

```
Input:
    2
 /    \
1      3
Output: 1
```

Example 2:

```
Input:
  2
   \
    7
     \
      6
       \
        5
         \
          9
           \
            2
             \
              6
Output: 0
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **isBST()**

which takes the root of the tree as a parameter and returns true if the given binary tree is BST, else returns false. The printing is done by the driver's code.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Height of the BST).

Constraints:
0 <= Number of edges <= 100000

Bottom View of Binary Tree

Given a binary tree, print the bottom view from left to right.
A node is included in bottom view if it can be seen when we look at the tree from bottom.

```
              20
             /    \
           8       22
         /   \       \
       5      3       25
             /   \
           10    14
```

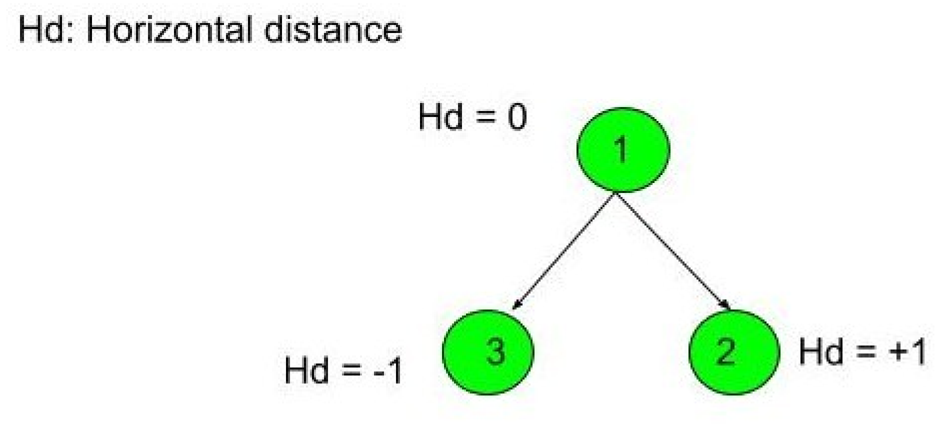
For the above tree, the bottom view is 5 10 3 14 25.
If there are **multiple** bottom-most nodes for a horizontal distance from root, then print the later one in level traversal. For example, in the below diagram, 3 and 4 are both the bottommost nodes at horizontal distance 0, we need to print 4.

```
              20
             /    \
           8       22
         /   \    /   \
       5      3 4     25
              /    \
          10       14
```

For the above tree the output should be 5 10 4 14 25.

Example 1:

```
Input:
     1
   /   \
  3     2
Output: 3 1 2
Explanation:
First case represents a tree with 3 nodes
and 2 edges where root is 1, left child of
1 is 3 and right child of 1 is 2.
```



```
Thus nodes of the binary tree will be
printed as such 3 1 2.
```

Example 2:

```
Input:
       10
     /    \
    20    30
   /  \
  40   60
Output: 40 20 60 30
```

Your Task:
This is a functional problem, you **don't** need to care about input, just complete the function **bottomView**() which takes the root node of the tree as input and returns an array containing the bottom view of the given tree.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:

1 <= Number of nodes <= 10^5

1 <= Data of a node <= 10^5

Vertical Traversal of Binary Tree

Given a Binary Tree, find the vertical traversal of it starting from the leftmost level to the rightmost level.
If there are multiple nodes passing through a vertical line, then they should be printed as they appear in level order traversal of the tree.

Example 1:

```
Input:
     2
      \
       3
      /
     4
Output: 2 4 3
```

Example 2:

```
Input:
       1
     /     \
   2        3
 /    \        \
4      5        6
Output: 4 2 1 5 3 6
```

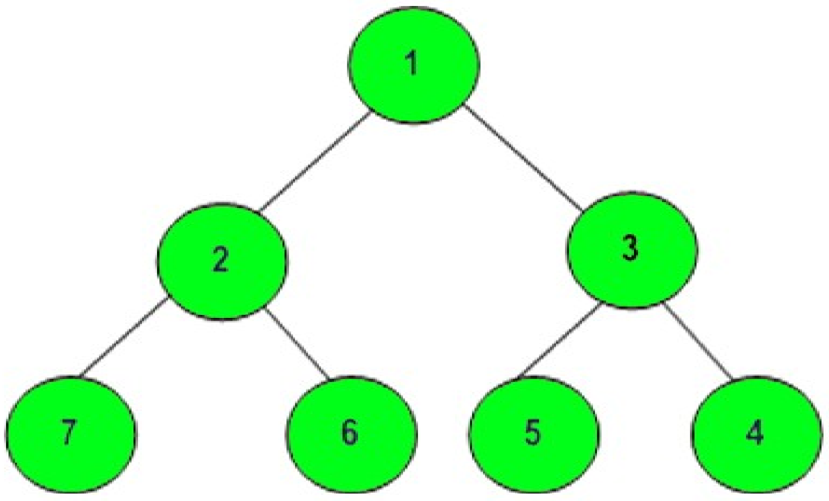
Your Task:
You don't need to read input or print anything. Your task is to complete the function **verticalOrder()** which takes the root node as input and returns an array containing the vertical order traversal of the tree from the leftmost to the rightmost level. If 2 nodes lie in the same vertical level, they should be printed in the order they appear in the level order traversal of the tree.

Expected Time Complexity: O(N log N).
Expected Auxiliary Space: O(N).

Constraints:
1 <= Number of nodes <= 5000

Level order traversal in spiral form

Complete the function to print spiral order traversal of a tree. For below tree, function should print 1, 2, 3, 4, 5, 6, 7.



Example 1:

```
Input:
     1
   /   \
  3     2
Output:1 3 2
```

Example 2:

```
Input:
         10
       /   \
      20   30
    /   \
   40   60
Output: 10 20 30 60 40
```

Your Task:
The task is to complete the function **printSpiral**() which prints the elements in spiral form of level order traversal. The newline is automatically appended by the driver code.
Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:
0 <= Number of nodes <= 10^5
1 <= Data of a node <= 10^5

Connect Nodes at Same Level

Given a binary tree, connect the nodes that are at same level. You'll be given an addition nextRight pointer for the same.

Initially, all the **nextRight** pointers point to **garbage** values. **Your function** should set these pointers to point next right for each node.

```
     10                      10 ------> NULL
    / \                     /     \
   3   5       =>     3 ------> 5 --------> NULL
  / \     \               /  \           \
 4   1   2          4 --> 1 -----> 2 -------> NULL
```

Example 1:

```
Input:
     3
   /   \
  1     2
Output:
```

```
3 1 2
1 3 2
```
Explanation:`The connected tree is`
```
      3 ------> NULL
    /    \
   1-----> 2 ------ NULL
```

Example 2:

Input:
```
     10
   /   \
  20   30
 /  \
40  60
```
Output:
```
10 20 30 40 60
40 20 60 10 30
```
Explanation:`The connected tree is`
```
       10 ----------> NULL
      /     \
    20 ------> 30 -------> NULL
  /    \
 40 ----> 60 ----------> NULL
```

Your Task:
You don't have to take input. Complete the function **connect()** that takes **root** as parameter and connects the nodes at same level. The printing is done by the driver code.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:
1 <= Number of nodes <= 100
1 <= Data of a node <= 1000

Lowest Common Ancestor in a BST

Given a Binary Search Tree (with all values unique) and two node values. Find the Lowest Common Ancestors of the two nodes in the BST.

Example 1:

```
Input:
              5
           /     \
         4         6
       /             \
      3                7
                     \
                      8
n1 = 7, n2 = 8
Output: 7
```

Example 2:

```
Input:
     2
   /   \
  1      3
n1 = 1, n2 = 3
Output: 2
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **LCA()** which takes the root Node of the BST and two integer values n1 and n2 as inputs and returns the Lowest Common Ancestor of the Nodes with values n1 and n2 in the given BST.

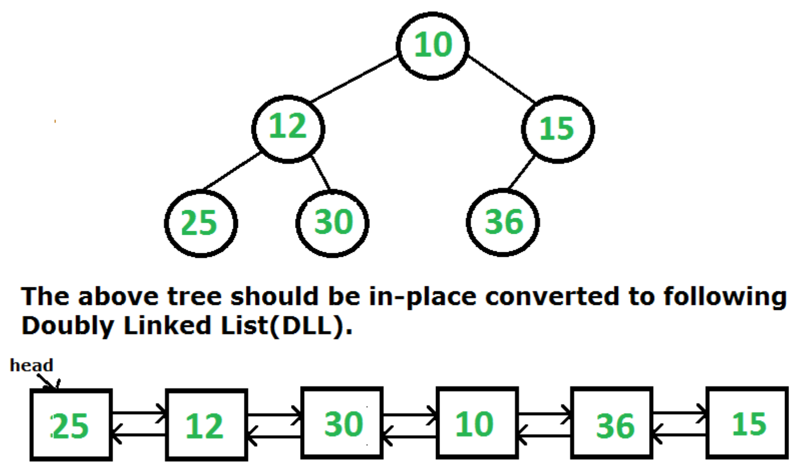
Expected Time Complexity: O(Height of the BST).
Expected Auxiliary Space: O(Height of the BST).

Constraints:

1 <= N <= 10^4

Binary Tree to DLL

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.


10
12
15
25
30
36
The above tree should be in-place converted to following Doubly Linked List(DLL).
head
25
12
30
10
36
15

Example 1:

```
Input:
      1
    /   \
   3     2
Output:
3 1 2
2 1 3
Explanation: DLL would be 3<=>1<=>2
```

Example 2:

```
Input:
       10
      /   \
     20    30
   /   \
  40    60
Output:
40 20 60 10 30
30 10 60 20 40
Explanation:  DLL would be
40<=>20<=>60<=>10<=>30.
```

Your Task:
You don't have to take input. Complete the function **bToDLL()** that takes **root** node of the tree as a parameter and returns the head of DLL . The driver code prints the DLL both ways.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(H).
Note: H is the height of the tree and this space is used implicitly for recursion stack.

Constraints:
1 <= Number of nodes <= 1000
1 <= Data of a node <= 1000

Determine if Two Trees are Identical

Given two binary trees, the task is to find if both of them are identical or not.

Example 1:

```
Input:
    1          1
  /   \      /   \
 2     3    2     3
Output: Yes
Explanation: There are two trees both
having 3 nodes and 2 edges, both trees
are identical having the root as 1,
left child of 1 is 2 and right child
of 1 is 3.
```

Example 2:

```
Input:
   1       1
  /  \     /  \
 2    3   3    2
Output: No
Explanation: There are two trees both
having 3 nodes and 2 edges, but both
trees are not identical.
```

Your task:
Since this is a functional problem you don't have to worry about input, you just have to complete the function **isIdentical()** that takes two roots as parameters and returns true or false. The printing is done by the driver code.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Height of the Tree).

Constraints:
1 <= Number of nodes <= 1000
1 <=Data of a node <= 1000

Symmetric Tree

Given a Binary Tree. Check whether it is Symmetric or not, i.e. whether the binary tree is a Mirror image of itself or not.

Example 1:

```
Input:
         5
       /   \
      1     1
     /       \
    2         2
Output: True
Explanation: Tree is mirror image of
itself i.e. tree is symmetric
```

Example 2:

```
Input:
         5
       /   \
      10     10
     /  \     \
    20  20     30
Output: False
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **isMirror()** which takes the root of the Binary Tree as its input and returns True if the given Binary Tree is a same as the Mirror image of itself. Else, it returns False.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Height of the Tree).

Constraints:
1<=Number of nodes<=100
