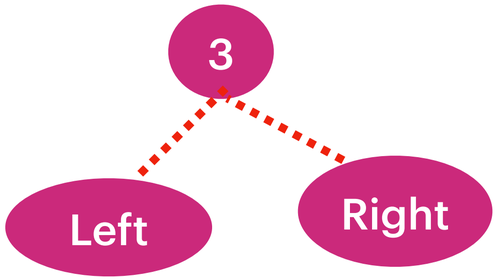
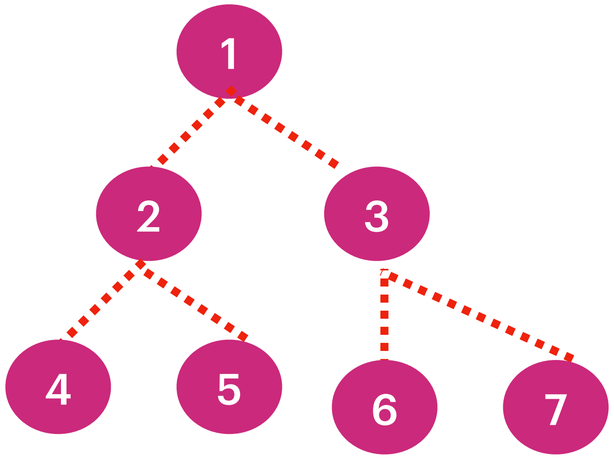
Binary Tree → In a Binary Tree a parent node at max has two Childs

Lets have a fun with trees before moving on to Weighted Graphs

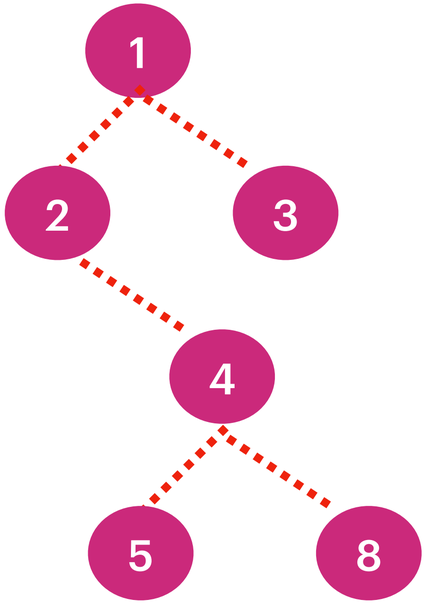
```
TreeNode :
    val : int
  left : TreeNode
 right : TreeNode
```



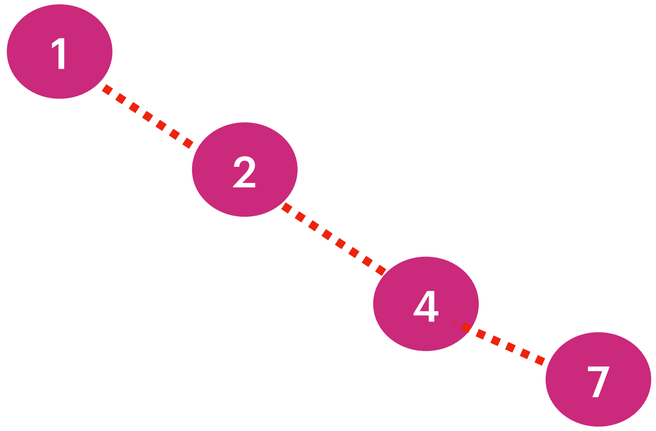
Input : [1,2,3,4,5,6,7]

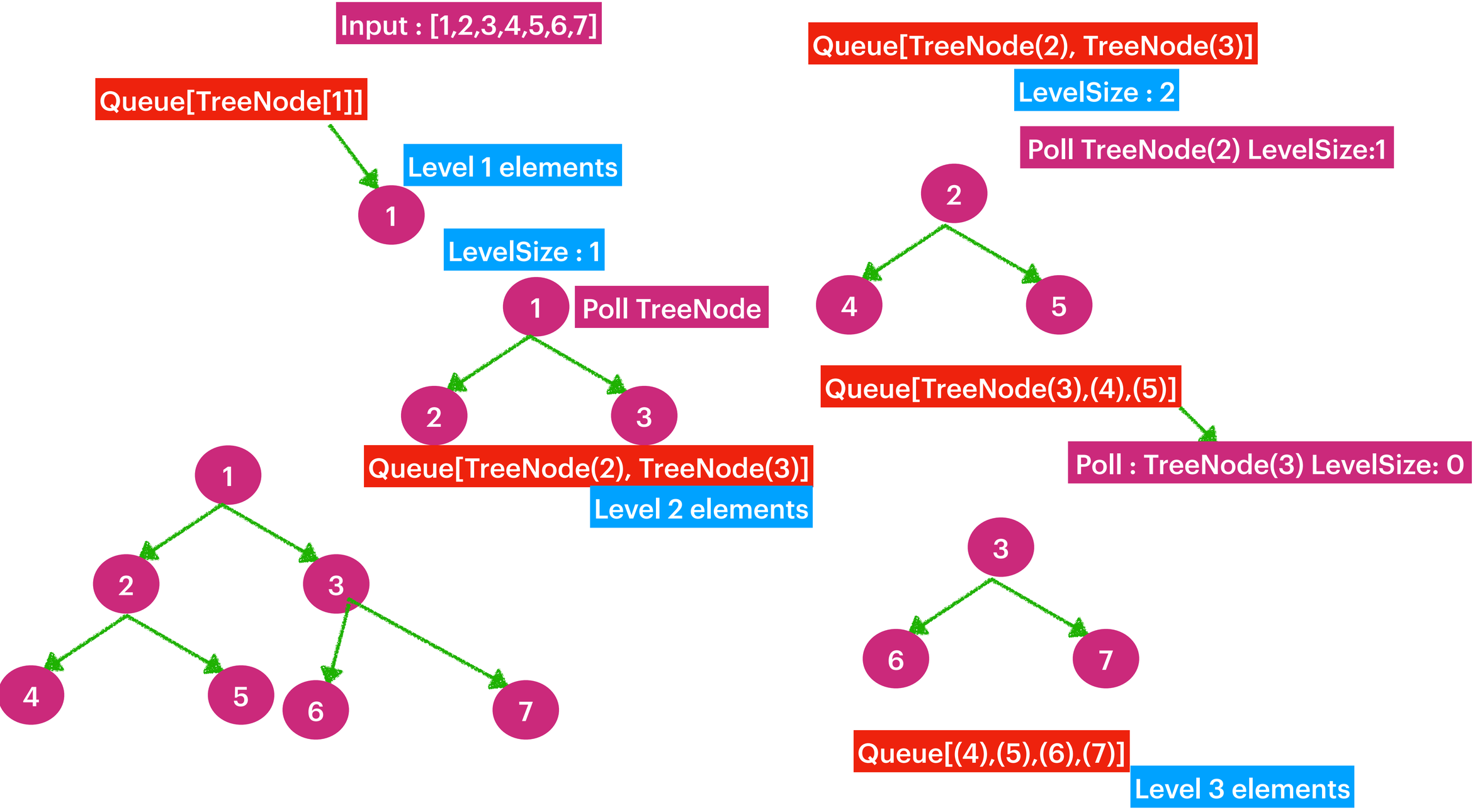


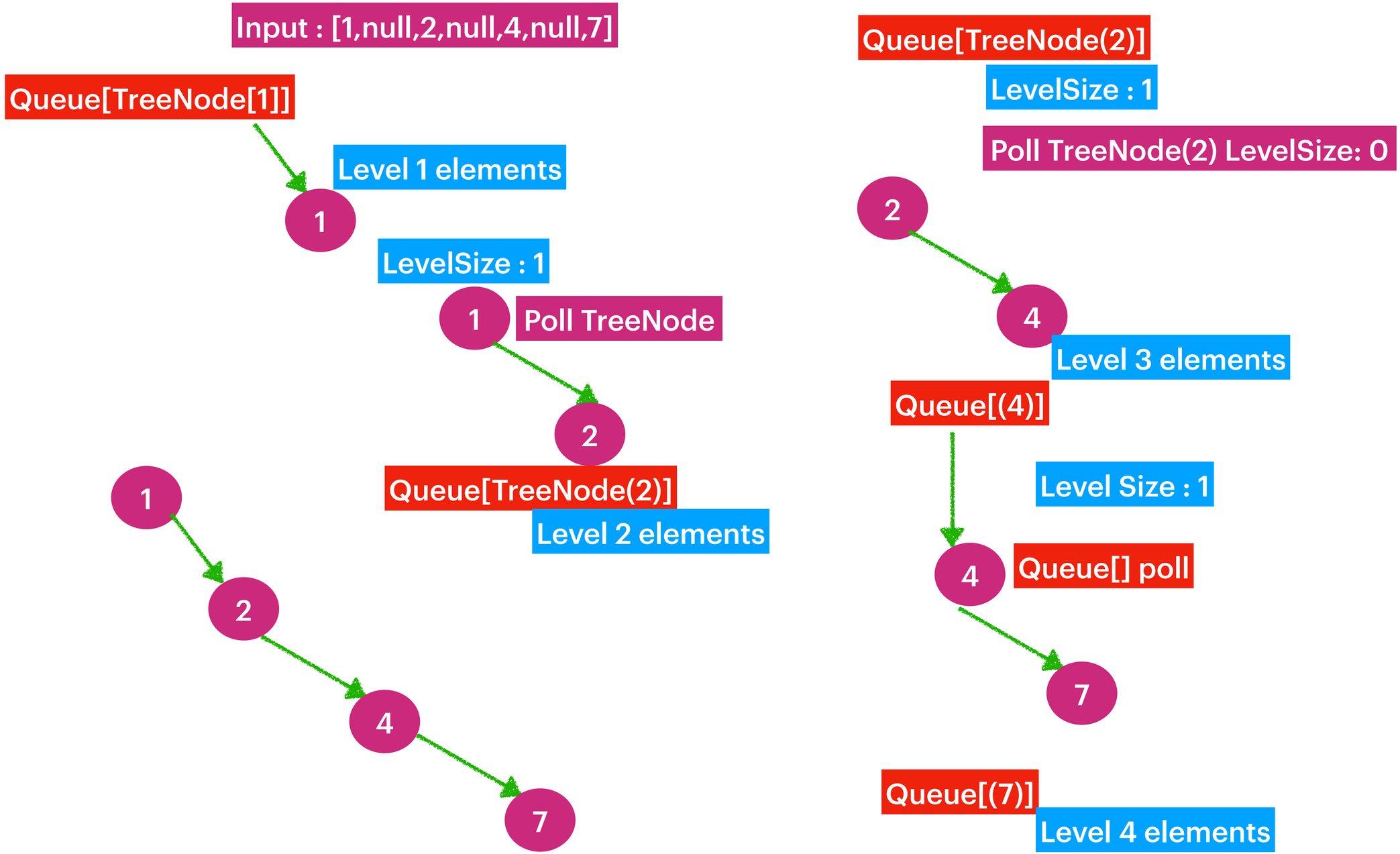
Input : [1,2,3,null,4,null,null,5,8]



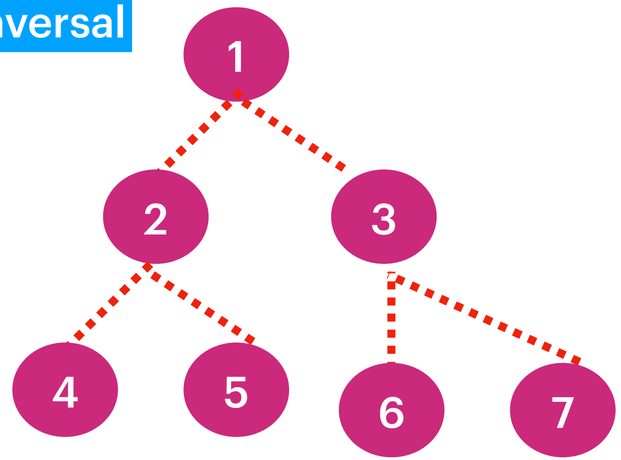
Input : [1,null,2,null,4,null,7]




Input : [1,2,3,4,5,6,7]
Queue[TreeNode[1]]
Level 1 elements
1
LevelSize : 1
1
Poll TreeNode
2
3
Queue[TreeNode(2), TreeNode(3)]
Level 2 elements
Queue[TreeNode(2), TreeNode(3)]
LevelSize : 2
Poll TreeNode(2) LevelSize:1
2
4
5
Queue[TreeNode(3),(4),(5)]
Poll : TreeNode(3) LevelSize: 0
1
2
3
4
5
6
7
3
6
7
Queue[(4),(5),(6),(7)]
Level 3 elements

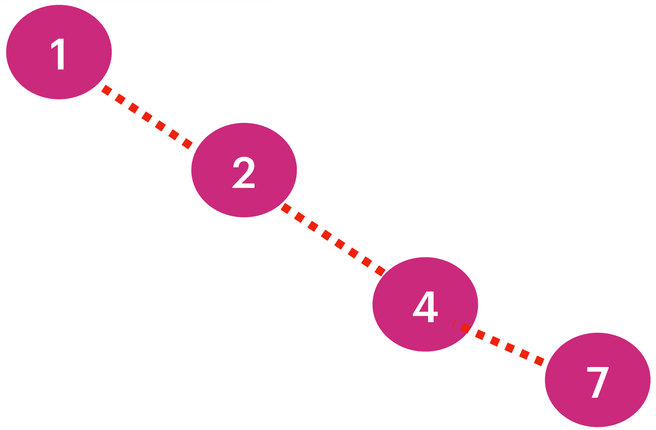

Input : [1,null,2,null,4,null,7]
Queue[TreeNode[1]]
Level 1 elements
1
LevelSize : 1
1
Poll TreeNode
2
Queue[TreeNode(2)]
Level 2 elements
1
2
4
7
Queue[TreeNode(2)]
LevelSize : 1
Poll TreeNode(2) LevelSize: 0
2
4
Level 3 elements
Queue[(4)]
Level Size : 1
4
Queue[] poll
7
Queue[(7)]
Level 4 elements

LevelOrder Traversal



Level Order : [[1] , [2,3] , [4,5,6,7]]

Input : [1,null,2,null,4,null,7]

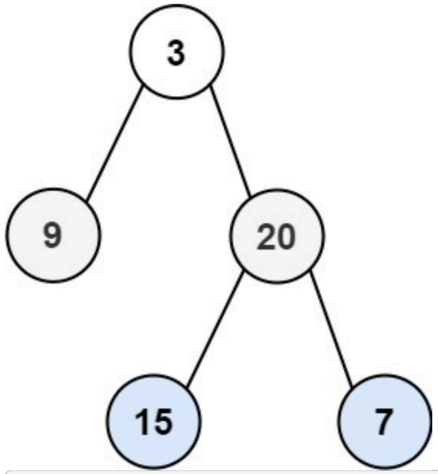


Level Order : [[1] , [2] , [4] ,[7]]

Binary Tree Level Order Traversal

Given the `root` of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

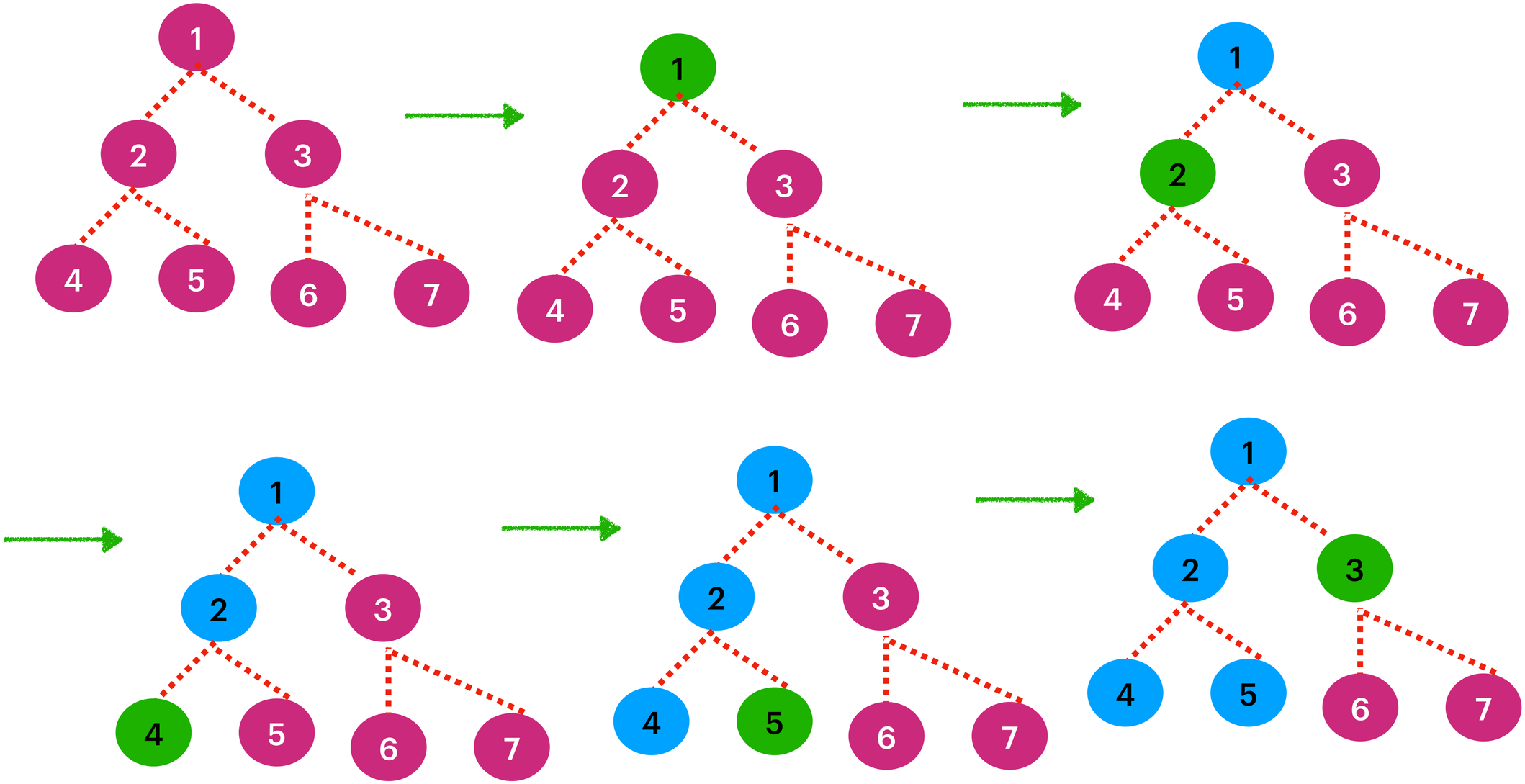
Example 3:

```
Input: root = []
Output: []
```

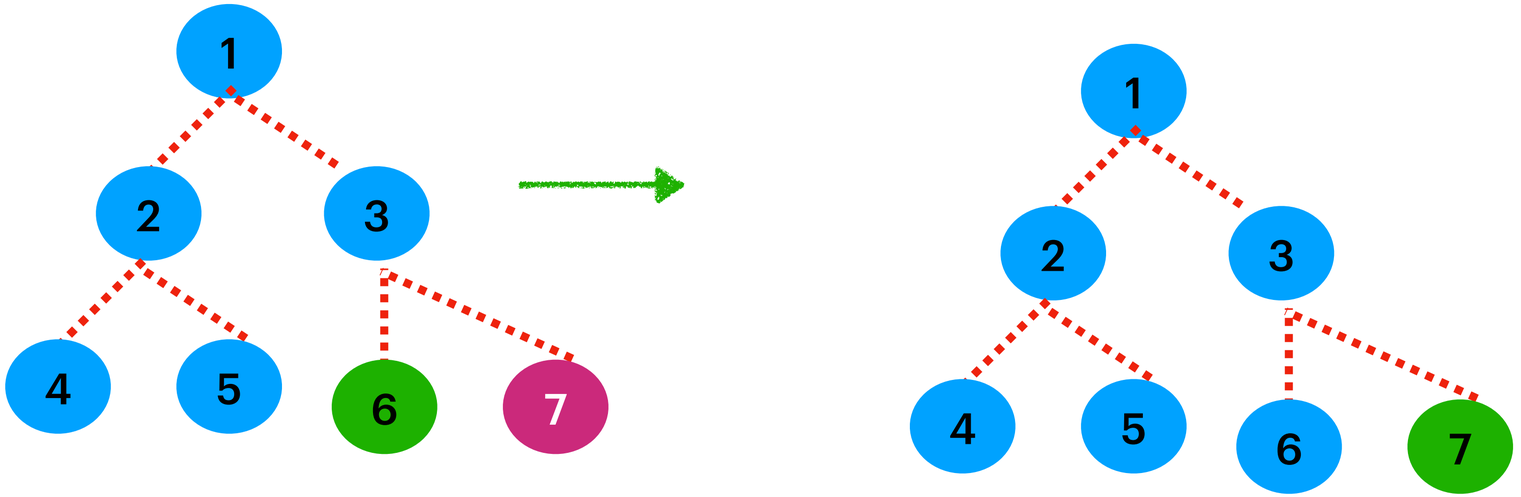
Constraints:

- The number of nodes in the tree is in the range `[0, 2000]`.
- `-1000 <= Node.val <= 1000`

PreOrder Traversal : Root -> Left -> Right [Recursively]



PreOrder Traversal : Root -> Left -> Right [Recursively]



Point Out the Green Nodes [1,2,4,5,3,6,7]

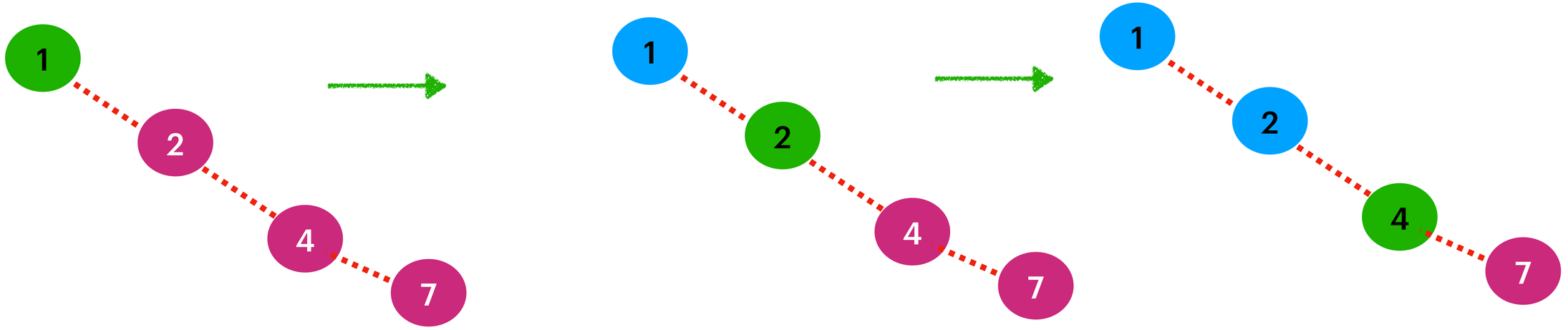
[1],[2,[4],[5]],[3,[6],[7]]

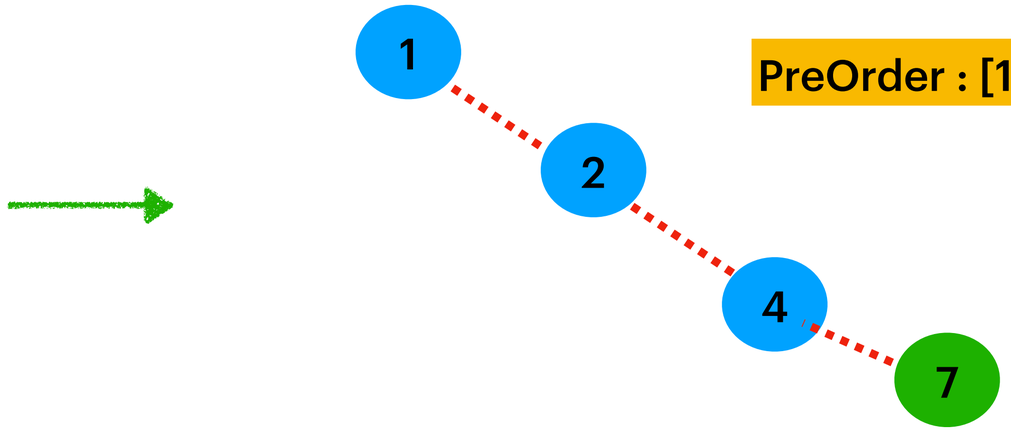
Time Complexity : root[1] + left [n/2] + right[n/2] = 1+2n/2 = O(n)
Space Complexity : O(n) [For skew tree n stack frames active]

PreOrder Recursion Logic :

```
public void preOrder(TreeNode root)
{
if(root == null)
{
return;
}
print(root.value);
preOrder(root.left);
preOrder(root.right);
}
```

PreOrder Traversal : Root -> Left -> Right [Recursively]





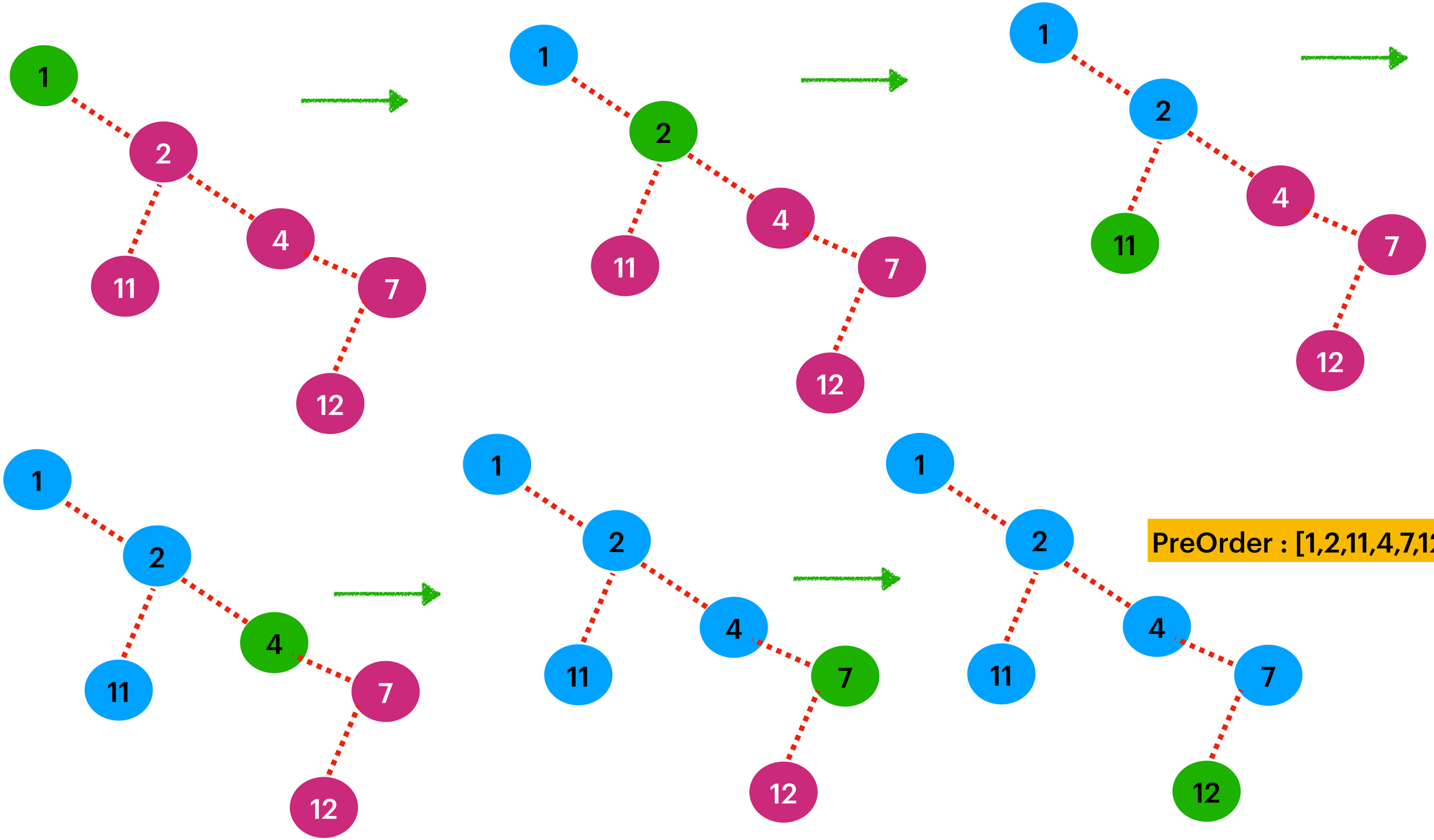
PreOrder : [1,2,4,7]

Time Complexity : O(n)
Space Complexity : O(n)

PreOrder Iterative Logic :

```
public void preOrder(TreeNode root)
{
Stack<TreeNode> stack = new Stack<>();
stack.push(root);
while(!stack.isEmpty())
{
TreeNode current  = stack.pop();
Print(current.val);
if(current.right != null )
stack.push(current.right);
if(current.left != null)
    stack.push(current.left);
}
}
```

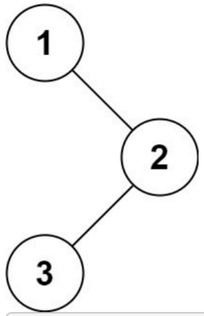
PreOrder Traversal : Root -> Left -> Right [Recursively]



Binary Tree Preorder Traversal

Given the `root` of a binary tree, return *the preorder traversal of its nodes' values*.

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,2,3]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
Output: [1]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

Follow up: Recursive solution is trivial, could you do it iteratively?