Question-1:

Given preorder of a binary tree, calculate its depth(or height) [starting from depth 0]. The preorder is given as a string with two possible characters.

1. ‘l’ denotes the leaf
2. ‘n’ denotes internal node

The given tree can be seen as a full binary tree where every node has 0 or two children. The two children of a node can ‘n’ or ‘l’ or mix of both.

Examples :

Input : nlnll Output : 2

Solution:

```
def calculate_tree_depth(preorder):
    # Base case: if the preorder string is empty, return -1
    if not preorder:
        return -1

    # Get the first character of the preorder string
    root = preorder[0]

    # If it's a leaf node, return 0
    if root == 'l':
        return 0

    # If it's an internal node, recursively calculate the depth of the left and right subtrees
    left_subtree = preorder[1:]
    right_subtree = ""

    # Find the index where the right subtree starts
    count = 0
    for i in range(1, len(preorder)):
        if preorder[i] == 'n':
            count += 1
        else:
            count -= 1
        if count == 0:
            right_subtree = preorder[i+1:]
            break
    # Calculate the depths of the left and right subtrees recursively
    left_depth = calculate_tree_depth(left_subtree)
    right_depth = calculate_tree_depth(right_subtree)

    # Return the maximum depth of the left and right subtrees, plus 1 for the root
    return max(left_depth, right_depth) + 1
```

Question-2:

Given a Binary tree, the task is to print the left view of the Binary Tree. The left view of a Binary Tree is a set of leftmost nodes for every level.

Examples:

Input:

```
          4

        /   \

       5     2

          /   \

          3     1

        /  \

        6    7
```

Output: 4 5 3 6

Solution:

```
class Node:
    def __init__(self, value):
        self.data = value
        self.left = None
        self.right = None

def print_left_view(root):
    if root is None:
        return

    # Create an empty queue for level order traversal
    queue = []
    # Append the root node to the queue
    queue.append(root)

    # Iterate over the nodes in the queue
    while queue:
        # Get the number of nodes in the current level
        level_size = len(queue)

        # Traverse the current level and print the leftmost node
        for i in range(level_size):
            node = queue.pop(0)

            # Print the leftmost node of the current level
            if i == 0:
                print(node.data, end=' ')

            # Append the left and right child nodes to the queue
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
```

Question-3:

Given a Binary Tree, print the Right view of it.

The right view of a Binary Tree is a set of nodes visible when the tree is visited from the Right side.

Examples:

Input:

```
     1

  /     \

2        3
```

```
/   \      /  \

4    5   6    7

            \

              8
```

Output:

Right view of the tree is 1 3 7 8

Input:

```
        1

      /

   8

 /

7
```

Output:

Right view of the tree is 1 8 7

Solution:

```
class Node:
    def __init__(self, value):
        self.data = value
        self.left = None
        self.right = None

def print_right_view(root):
    if root is None:
        return

    # Create an empty queue for level order traversal
    queue = []
    # Append the root node to the queue
    queue.append(root)
```

```
# Iterate over the nodes in the queue
while queue:
    # Get the number of nodes in the current level
    level_size = len(queue)

    # Traverse the current level and print the rightmost node
    for i in range(level_size):
        node = queue.pop(0)

        # Print the rightmost node of the current level
        if i == level_size - 1:
            print(node.data, end=' ')

        # Append the left and right child nodes to the queue
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
```

Question-4:

Given a Binary Tree, The task is to print the **bottom view** from left to right. A node **x** is there in output if x is the bottommost node at its horizontal distance. The horizontal distance of the left child of a node x is equal to a horizontal distance of x minus 1, and that of a right child is the horizontal distance of x plus 1.

Examples:

Input:

```
          20

        /    \

      8        22

   /     \        \

5        3        25

      /    \
   10       14
```

Output: 5, 10, 3, 14, 25.

Input:

```
             20

           /     \

        8         22

    /      \      /   \

 5         3    4     25

         /    \

     10       14
```

Output:

5 10 4 14 25.

Explanation:

If there are multiple bottom-most nodes for a horizontal distance from the root, then print the later one in the level traversal.

3 and 4 are both the bottom-most nodes at a horizontal distance of 0, we need to print 4.

Solution:

```
class Node:
    def __init__(self, value):
        self.data = value
        self.left = None
        self.right = None
        self.hd = 0

def print_bottom_view(root):
    if root is None:
        return
```

```
# Create an empty queue for level order traversal
queue = []
# Append the root node to the queue
queue.append(root)

# Create an empty dictionary to store the nodes with their horizontal distance
bottom_view_dict = {}

# Perform level order traversal
while queue:
   node = queue.pop(0)
   hd = node.hd

   # Update the node value for the horizontal distance in the dictionary
   bottom_view_dict[hd] = node.data

   # Append the left and right child nodes to the queue
   if node.left:
      node.left.hd = hd - 1
      queue.append(node.left)
   if node.right:
      node.right.hd = hd + 1
      queue.append(node.right)

# Print the values in the dictionary in ascending order of their horizontal distance
for value in sorted(bottom_view_dict.keys()):
   print(bottom_view_dict[value], end=' ')
```