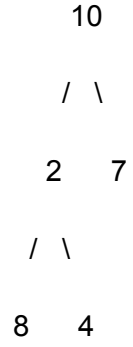
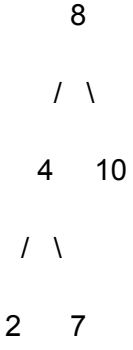
Question-1

You are given a binary tree. The binary tree is represented using the TreeNode class. Each TreeNode has an integer value and left and right children, represented using the TreeNode class itself. Convert this binary tree into a binary search tree.

Input:



Output:



Soln:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def inorderTraversal(root, values):
    if root is None:
        return

    inorderTraversal(root.left, values)
    values.append(root.val)
```

```
    inorderTraversal(root.right, values)


def convertToBST(root, values):
    if root is None:
        return

    convertToBST(root.left, values)
    root.val = values.pop(0)
    convertToBST(root.right, values)


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

    values = []
    inorderTraversal(root, values)
    values.sort()
    convertToBST(root, values)

    return root
```

Question 2:
Given a Binary Search Tree with all unique values and two keys. Find the distance between two nodes in BST. The given keys always exist in BST.

SOln:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def findDistance(root, node1, node2):
    if root is None:
        return 0

    if node1 < root.val and node2 < root.val:
        return findDistance(root.left, node1, node2)
    elif node1 > root.val and node2 > root.val:
```

```
        return findDistance(root.right, node1, node2)
    else:
        return distanceFromNode(root, node1) + distanceFromNode(root, node2)


def distanceFromNode(root, target):
    if root.val == target:
        return 0

    if target < root.val:
        return 1 + distanceFromNode(root.left, target)
    else:
        return 1 + distanceFromNode(root.right, target)
```

Question-3:

Write a program to convert a binary tree to a doubly linked list.

Input:

```
        10

      /   \

     5     20

          /   \

        30     35
```

Output:

5 10 30 20 35

Soln:

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

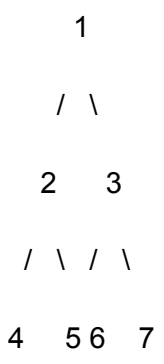
```
def convertToDoublyLinkedList(root):
    if root is None:
        return None

    # Create a dummy node to represent the head of the doubly linked list
    dummy = Node(None)
    prev = dummy

    # Perform in-order traversal
    stack = []
    current = root

    while current or stack:
        while current:
            stack.append(current)
            current = current.left

        current = stack.pop()

        # Update the pointers for the doubly linked list
        prev.right = current
        current.left = prev
        prev = current

        current = current.right

    # Set the last node's right pointer to None
    prev.right = None

    # Set the left pointer of the head to None
    dummy.right.left = None

    # Return the head of the doubly linked list
    return dummy.right

# Create the binary tree
root = Node(10)
root.left = Node(5)
root.right = Node(20)
root.right.left = Node(30)
root.right.right = Node(35)

# Convert the binary tree to a doubly linked list
head = convertToDoublyLinkedList(root)
```

```
# Traverse the doubly linked list and print the values
current = head
while current:
    print(current.val, end=" ")
    current = current.right
```

Question-4:

Write a program to connect nodes at the same level.

Input:



Output:

1 → -1

2 → 3

3 → -1

4 → 5

5 → 6

6 → 7

7 → -1

Soln:

class Node:

```
    def __init__(self, value):
        self.val = value
        self.left = None
        self.right = None
        self.next = None

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

    # Initialize the queue with the root node
    queue = [root]

    while queue:
        level_size = len(queue)

        # Connect the nodes at the same level
        for i in range(level_size):
            current = queue.pop(0)

            # Set the next pointer to the next node in the queue
            if i < level_size - 1:
                current.next = queue[0]

            # Add the left and right child nodes to the queue
            if current.left:
                queue.append(current.left)
            if current.right:
                queue.append(current.right)

# Create the binary tree
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

# Connect the nodes at the same level
connectNodes(root)

# Print the connections
current = root
```

```
while current:
    temp = current
    while temp:
        if temp.next:
            print(temp.val, "→", temp.next.val)
        else:
            print(temp.val, "→ -1")
        temp = temp.next
    current = current.left
```