```python
class Node:
    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
# Returns true if the given tree is a binary search tree
# (efficient version)
def isBST(node):
    return (isBSTUtil(node, INT_MIN, INT_MAX))
# Retusn true if the given tree is a BST and its values
# >= min and <= max
def isBSTUtil(node, mini, maxi):
    # An empty tree is BST
    if node is None:
        return True
    # False if this node violates min/max constraint
    if node.data < mini or node.data > maxi:
        return False
    # Otherwise check the subtrees recursively
    # tightening the min or max constraint
    return (isBSTUtil(node.left, mini, node.data -1) and
          isBSTUtil(node.right, node.data+1, maxi))

# Driver program to test above function
root = Node(4)
root.left = Node(2)
root.right = Node(5)
root.left.left = Node(1)
root.left.right = Node(3)
```

```python
# Program to print binary tree in vertical order

# A binary tree
class Node:
    # Constructor to create a new node
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None

# A utility function to find min and max distances with
# respect to root
def findMinMax(node, minimum, maximum, hd):
    # Base Case
    if node is None:
        return
    # Update min and max
    if hd < minimum[0] :
        minimum[0] = hd
    elif hd > maximum[0]:
        maximum[0] = hd
    # Recur for left and right subtrees
    findMinMax(node.left, minimum, maximum, hd-1)
    findMinMax(node.right, minimum, maximum, hd+1)
# A utility function to print all nodes on a given line_no
# hd is horizontal distance of current node with respect to root
def printVerticalLine(node, line_no, hd):
    # Base Case
    if node is None:
        return
    # If this node is on the given line number
```

```
    # If this node is on the given line number
    if hd == line_no:
        print node.data,
    # Recur for left and right subtrees
    printVerticalLine(node.left, line_no, hd-1)
    printVerticalLine(node.right, line_no, hd+1)
def verticalOrder(root):
    # Find min and max distances with respect to root
    minimum = [0]
    maximum = [0]
    findMinMax(root, minimum, maximum, 0)
    # Iterate through all possible lines starting
    # from the leftmost line and print nodes line by line
    for line_no in range(minimum[0], maximum[0]+1):
        printVerticalline(root, line_no, 0)
        print


# Driver program to test above function
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)
root.right.left.right = Node(8)
root.right.right.right = Node(9)

print "Vertical order traversal is"
verticalOrder(root)
```