Convert a normal BST to Balanced BST

Given a BST (**B**inary **S**earch **T**ree) that may be unbalanced,
convert it into a balanced BST that has minimum possible height.
Input:
4
/
3
/
2
/
1
Output:
3 3 2
/ \ / \ /
1 4 OR 2 4 OR 1 3 OR ..
\ /
2 1 4

Input:
4
/
3 5
/
2 6
/
1 7
Output:
4
/
2 6
/ \ /
1 3 5 7

An **Efficient Solution** can construct balanced BST in O(n) time with minimum possible height.

Below are steps.

1.Traverse given BST in inorder and store result in an array. This step takes O(n) time.

Note that this array would be sorted as inorder traversal of BST always produces sorted sequence.

2.Build a balanced BST from the above created sorted array using the recursive approach discussed here.

This step also takes O(n) time as we traverse every element

exactly once and processing an element takes O(1) time.

```
import sys
import math

# A binary tree node has data, pointer to left child
# and a pointer to right child
class Node:
    def __init__(self,data):
        self.data=data
        self.left=None
        self.right=None

# This function traverse the skewed binary tree and
# stores its nodes pointers in vector nodes[]
def storeBSTNodes(root,nodes):

    # Base case
    if not root:
        return

    # Store nodes in Inorder (which is sorted
    # order for BST)
    storeBSTNodes(root.left,nodes)
    nodes.append(root)
    storeBSTNodes(root.right,nodes)

# Recursive function to construct binary tree
def buildTreeUtil(nodes,start,end):

    # base case
    if start>end:
        return None

    # Get the middle element and make it root
    mid=(start+end)//2
    node=nodes[mid]
```

```python
    # Using index in Inorder traversal, construct
    # left and right subtress
    node.left=buildTreeUtil(nodes,start,mid-1)
    node.right=buildTreeUtil(nodes,mid+1,end)
    return node

# This functions converts an unbalanced BST to
# a balanced BST
def buildTree(root):

    # Store nodes of given BST in sorted order
    nodes=[]
    storeBSTNodes(root,nodes)

    # Constucts BST from nodes[]
    n=len(nodes)
    return buildTreeUtil(nodes,0,n-1)

# Function to do preorder traversal of tree
def preOrder(root):
    if not root:
        return
    print("{} ".format(root.data),end="")
    preOrder(root.left)
    preOrder(root.right)
```