```
class Node:

  def __init__(self, data):

    self.left = None

    self.right = None

    self.data = data
class tree:

  def creatingNode(self, data):

    return Node(data)

  def inserting(self, node, data):

    if node is None:

      return self.creatingNode(data)

    if data < node.data:

      node.left = self.inserting(node.left, data)

    else:

      node.right = self.inserting(node.right, data)

      return node
```

```
#task 01
def height(self, rootoftree):
  if rootoftree is None:
    return -1

    return max(self.height(rootoftree.left), self.height(rootoftree.right))+1

# #Task 02
def node_lev(self, rootoftree, key, level):

  if rootoftree is None:
    return -1

  if rootoftree.data == key:
    return level

  lev = self.node_level(rootoftree.left, key, level+1)

  if lev != -1:
    return lev

    return self.node_lev(rootoftree.right, key, level+1)
```

```
# #Task 03

def preordering_traversal(self, rootoftree):
  print(rootoftree.data, end=" ")
  self.preordering_traversal(rootoftree.left)
  self.preordering_traversal(rootoftree.right)


# #Task 04

def inordering_traversal(self, rooofftree):

  if rootoftree is not None:
    self.inordering_traversal(rootoftree.left)

    print(rootoftree.data, end=" ")

    self.inordering_traversal(rootoftree.right)


# #Task 05

def postordering_traversal(self, rootoftree):
  self.postordering_traversal(rootoftree.left)
  self.postordering_traversal(rootoftree.right)
  print(rootoftree.data, end=" ")
```

```
# # Task 06

def tree_identify(self, rootoftree, rootoftree1):
  if rootoftree == None and rootoftree1 == None:
    return True

  if rootoftree != None and rootoftree1!= None and rootoftree.data == rootoftree1.data:
    x = self.tree_identify(rootoftree.left, rootoftree1.left)
    y = self.tree_identify(rootoftree.right, rootoftree1.right)

  if x and y:
    return True
    return False




# #Task 07


def new_tree(self, rootoftree):
  ref = Node(rootoftree.data)

  if rootoftree.left is not None:
    ref.left = self.clone(rootoftree.left)
```

```
  if rootoftree.right is not None:
    ref.right = self.clone(rootoftree.right)

    return ref
#testing

  treetest = Tree()
#Create a tree
  rootoftree = treetest.createNode(5)

  treetest.insert(rootoftree, 3)
  treetest.insert(rootoftree, 5)
  treetest.insert(rootoftree, 17)
  treetest.insert(rootoftree, 22)
  treetest.insert(rootoftree, 32)
  treetest.insert(rootoftree, 37)
  treetest.insert(rootoftree, 9)

  print("The height is",treetest.height(rootoftree))
  print("The level of 37:", treetest.nodelevel(tree, 37, 0))
  print("Pre Order: ")
  treetest.preordering_traversal(tree)
  print()
  print("Inorder: ")
```

```
  treetest.inordering_traversal(tree)
  print()
  print("Post Order: ")
  treetest.postordering_traversal(rootoftree)
  print()

#2nd tree For new_tree checking
  rootoftree1 = treetest.createNode(5)

  treetest.insert(rootoftree1, 3)
  treetest.insert(rootoftree1, 5)
  treetest.insert(rootoftree1, 17)
  treetest.insert(rootoftree1, 22)
  treetest.insert(rootoftree1, 32)
  treetest.insert(rootoftree1, 37)
  treetest.insert(rootoftree1, 9)

  print("Two trees are Same:", treetest.new_tree(rootoftree,rootoftree1))

  z = testtree.new_tree(rootoftree)
  print("New tree is:",end=" ")
  treetest.traverse_Inorder(z)
```