

Making binary tree manually

```
In [1]: 1 def first_tree():
2         root=Node(10,None,None,None)
3         n1=Node(20,None,None,None)
4         n2=Node(30,None,None,None)
5         n3=Node(40,None,None,None)
6         n4=Node(50,None,None,None)
7         n5=Node(70,None,None,None)
8         n6=Node(80,None,None,None)
9         n7=Node(90,None,None,None)
10        n8=Node(100,None,None,None)
11        root.left=n1
12        root.right=n2
13
14        n1.left=n3
15        n3.parent=n1
16        n1.right=n4
17        n4.parent=n1
18        n1.parent=root
19
20        n2.left=n5
21        n5.parent=n2
22        n2.right=n6
23        n6.parent=n2
24        n2.parent=root
25
26        n6.left=n7
27        n7.parent=n6
28        n6.right=n8
29        n8.parent=n6
30        return root
31
32 def second_tree():
33     root=Node(100,None,None,None)
34     n1=Node(200,None,None,None)
35     n2=Node(300,None,None,None)
36     n3=Node(400,None,None,None)
37     n4=Node(500,None,None,None)
38     n5=Node(700,None,None,None)
39     n6=Node(800,None,None,None)
40     n7=Node(900,None,None,None)
41     n8=Node(1000,None,None,None)
42     root.left=n1
43     root.right=n2
44
45     n1.left=n3
46     n3.parent=n1
47     n1.right=n4
48     n4.parent=n1
49     n1.parent=root
50
51     n2.left=n5
52     n5.parent=n2
53     n2.right=n6
54     n6.parent=n2
55     n2.parent=root
56
57     n6.left=n7
```

```

57     n7.parent=n6
58     n6.right=n8
59     n8.parent=n6
60     return root

```

In [39]:

```

1  #obj binary tree
2  #      10
3  #    /  \
4  #   20   30
5  #  / \  / \
6  # 40 50 70 80
7  #      /  \
8  #     90 100
9
10 #obj2 binary tree
11 #      100
12 #    /   \
13 #   200   300
14 #  /  \  /  \
15 # 400 500 700 800
16 #      /   \
17 #     900 1000

```

```
In [2]: 1 class Node:
2         def __init__(self,element, left , parent, right):
3             self.element=element
4             self.left=left
5             self.parent=parent
6             self.right=right
7     class binary_tree:
8         def __init__(self,root):
9             self.root=root
10        def height(self,root):
11            if root== None:
12                return -1
13            else:
14                a,b=(self.height (root.left), self.height(root.right))
15                if a>b:
16                    return 1+ a
17                else:
18                    return 1+b
19        def level(self,root):
20            if root.parent== None:
21                return 1
22            else:
23                return 1+self.level(root.parent)
24        def preorder(self,root):
25            if root !=None:
26                print(root.element,end=" ")
27                self.preorder(root.left)
28                self.preorder(root.right)
29            else:
30                return
31        def inorder(self,root):
32            if root !=None:
33                self.inorder(root.left)
34                print(root.element,end=" ")
35                self.inorder(root.right)
36            else:
37                return
38        def postorder(self,root):
39            if root !=None:
40                self.postorder(root.left)
41                self.postorder(root.right)
42                print(root.element,end=" ")
43            else:
44                return
45        def same_or_not(self,root1,root2):
46            if root1 !=None:
47                if root1.element==root2.element:
48                    self.same_or_not(root1.left,root2.left)
49                    self.same_or_not(root1.right,root2.right)
50
51            else:
52                return "\nNot Same"
53
54            return "\nSame"
55        def copy_bt(self,root):
56            obj4=binary_tree(root)
```

```
57         return obj4
58 root=first_tree()
59 root2=second_tree()
60 root3=first_tree()
61 obj=binary_tree(root)
62 obj2=binary_tree(root2)
63 obj3=binary_tree(root3)
64 print("Height :",obj.height(obj.root))
65 print("Level  :",obj.level(obj.root.left.right))
66 print("Level  :",obj.level(obj.root.right.right.right))
67 print("Pre-Order")
68 obj.preorder(obj.root)
69 print("\nIN-Order")
70 obj.inorder(obj.root)
71 print("\npost-Order")
72 obj.postorder(obj.root)
73 print(obj.same_or_not(obj.root,obj2.root))
74 print(obj.same_or_not(obj.root,obj3.root))
75 obj4=obj.copy_bt(obj.root)
76 print("Copy of Binary Tree:")
77 obj4.preorder(obj4.root)
78 print("\nMain Binary Tree")
79 obj.preorder(obj.root)
```

```
Height : 3
Level  : 3
Level  : 4
Pre-Order
10 20 40 50 30 70 80 90 100
IN-Order
40 20 50 10 70 30 90 80 100
post-Order
40 50 20 70 90 100 80 30 10
Not Same
```

```
Same
Copy of Binary Tree:
10 20 40 50 30 70 80 90 100
Main Binary Tree
10 20 40 50 30 70 80 90 100
```

In []:

```
1
2
```

