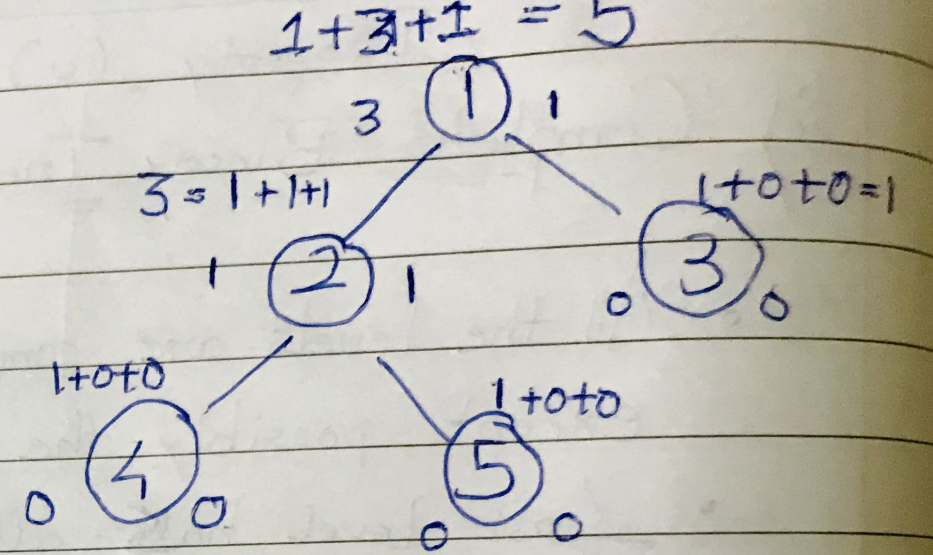
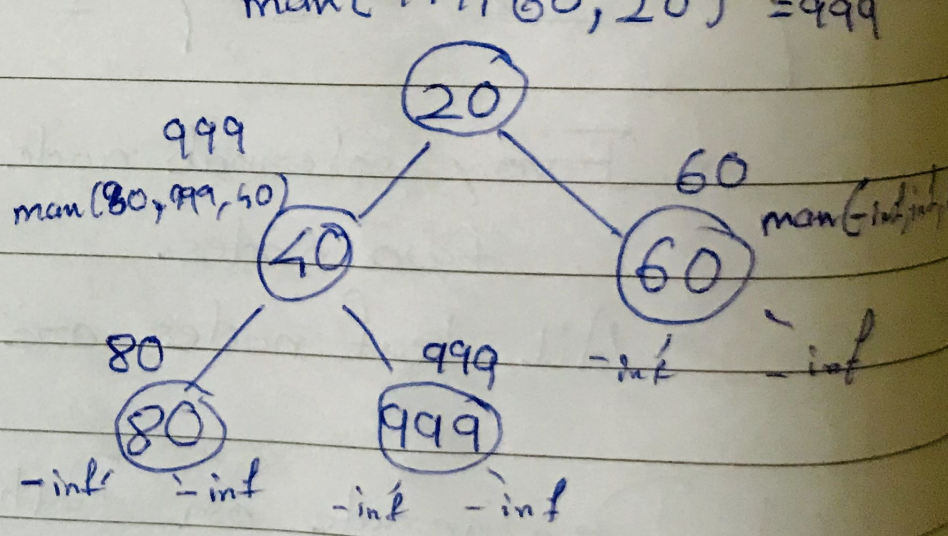
=> Size of tree :-

```
def sizeoftree(root):
    if root is None:
        return 0
    else :
        return (1+ sizeoftree(root.left)+ sizeoftree(root.right)
```



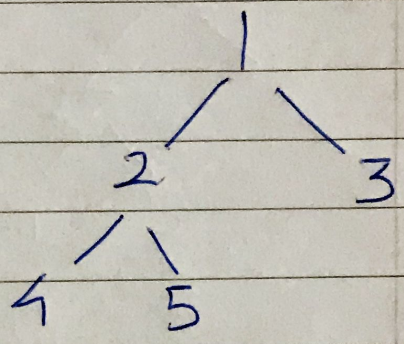

=> Man of tree :-

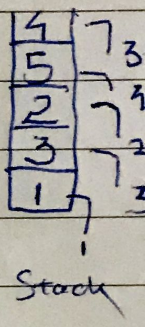
```
def Man(root):
    if root is None:
        return float('-inf')
    else :
        return man( Man(root.left), Man(root.right), root.data)
```

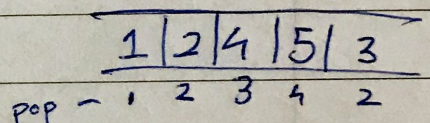


## Preorder Without recursion :

```
def pre(root):
    stack = []
    stack.append(root)
while stack:
    temp = stack.pop()
    print(stack.data)
    if temp.right != None:
        stack.append(temp.right)
    if temp.left != None:
        stack.append(temp.left)
```



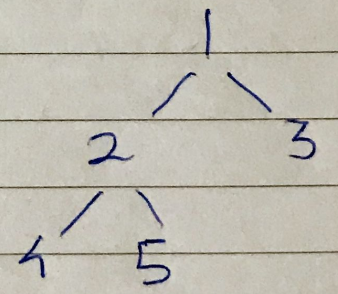



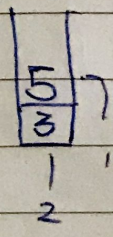



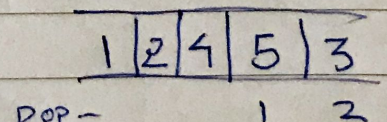

Time & Space – O(N)

## Space Optimized soln :-

```
while temp!=None or stack:
    while(temp):
        print(temp.data)
    if  if temp.right != None:
            stack.append(temp.right)
        temp = temp.left
    if stack:
        temp = stack.pop()
```



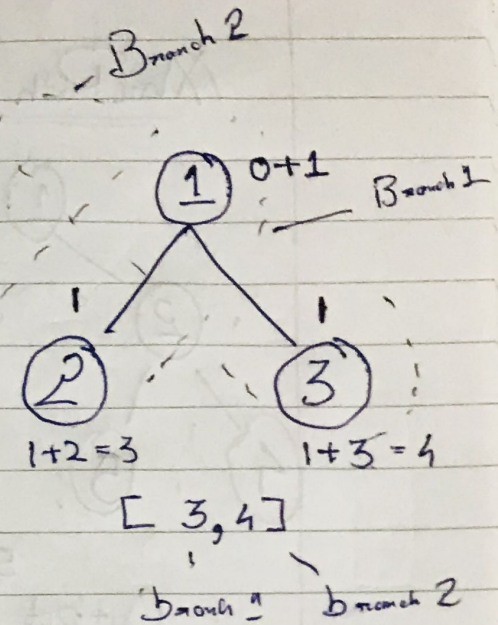



# Branch Sum :-

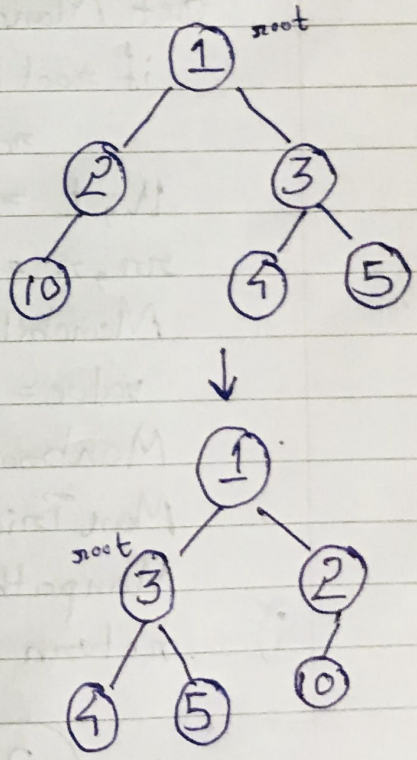
```
def branch(root, sum, sumlist)
    if
        newsum = sum + root.data
Leaf Node ← if root.left & right is None:
            sumlist.append(newsum)
        branch(root.right, newsum, sumlist)
        branch(root.left, newsum, sumlist)
```

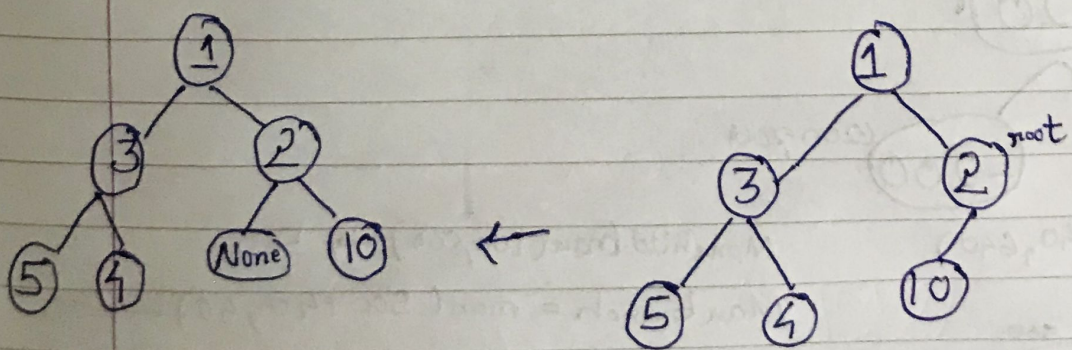
Time & Space - O(N)




# Invert Binary Tree :-

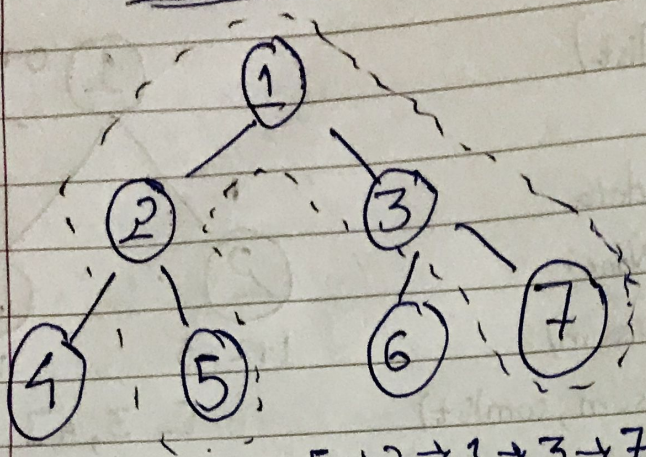
```
def InvertTree(root):
    if root is None:
        return
    Swap(root)
    InvertTree(root.left)
    InvertTree(root.right)

def Swap(root):
    root.left, root.right = root.right, root.left
```



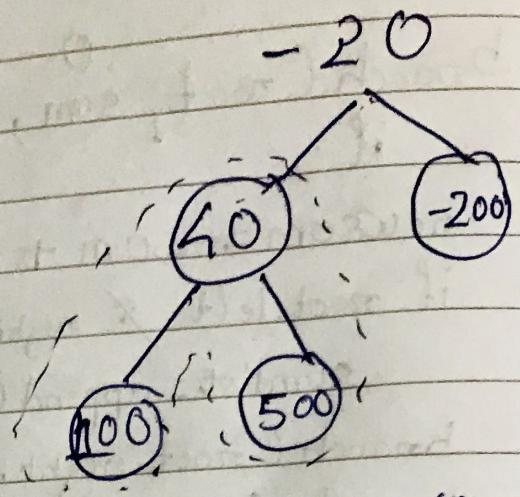

Time & Space Complexity - T → O(N)
S → O(H)
Log(N)

# Max Path Sum :-




5 → 2 → 1 → 3 → 7

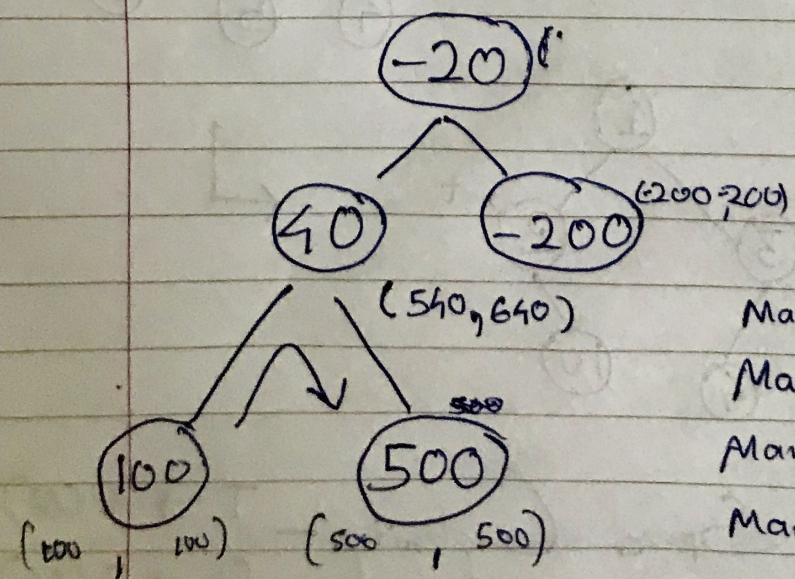
Output : 18




100 → 40 → 500

Output : 640

```
def Manpath Sum(root):
    if root is None:
        return [0,0]
    ll, l = ManpathSum(root.left)
    rr, r = Manpath Sum(root.right)
    Manchild = max(ll, rr)
     value = root.data
    Manbranch = max( Manchild+Value , Value)
    Max Triangle = max( ll+rr+value, Manbranch)
    Manpath = max( l, r, Max Triangle)
    return ( Manbranch, Max Triangle)
```




Manchild=tran(100,500) → 500

Manbranch = max(500+40, 40) → 540

Max Triangle = max(root+40+500, 540) → 640

Manpath = max(100,500, 640) → 640

## Efficient sol$^n$ for Counting Nodes of Perfect/Complete Binary Tree.

```
count(root):
    curr = root
    while curr not None:
        lh += 1 ; curr = curr.left
    while curr not None:
        rh += 1 ; curr = curr.right
    if (lh = rh)
        return 2 ** lh or rh - 1
    return count(root.left) + count(root.right)
```

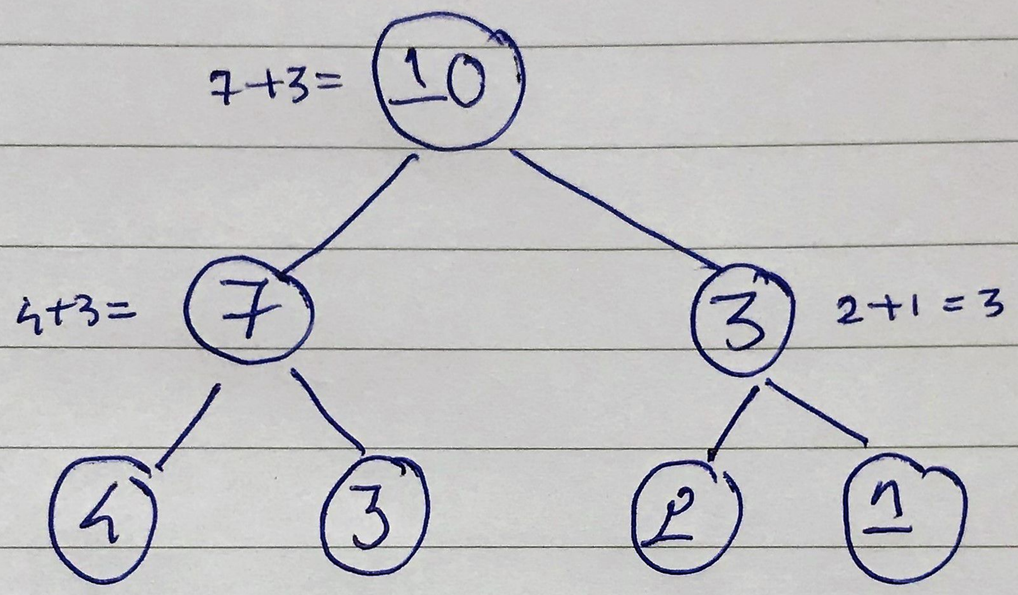
Time complexity -

$T(n) < \Theta(H) + \Theta(H) + T(2n/3)$

$T(n) = T(2n/3) + \Theta(\log H)$

T.C $\rightarrow O(\log n \cdot \log n)$

# Children Sum Property



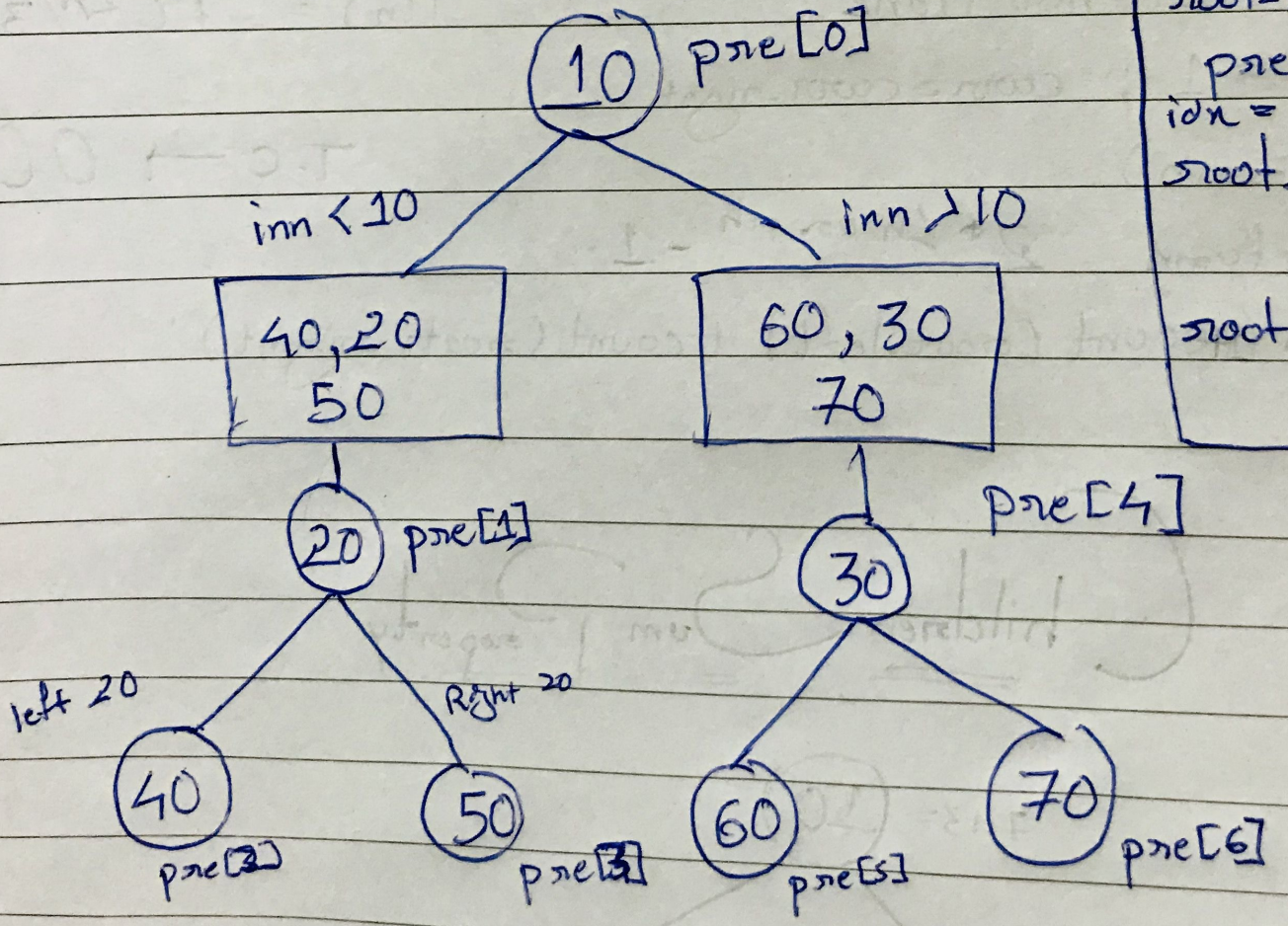

```
def ChildSum(root):
    if root.left & root.right is None:
        return True
    if root.left!=None ; suml= root.left.data
    if root.right!=None; sumr = root.right.data

    return ( ( root.data == suml+sumr ) and
                ChildSum(root.left) and ChildSum(root.
                right))
```

# Binary tree from inorder and preorder Array

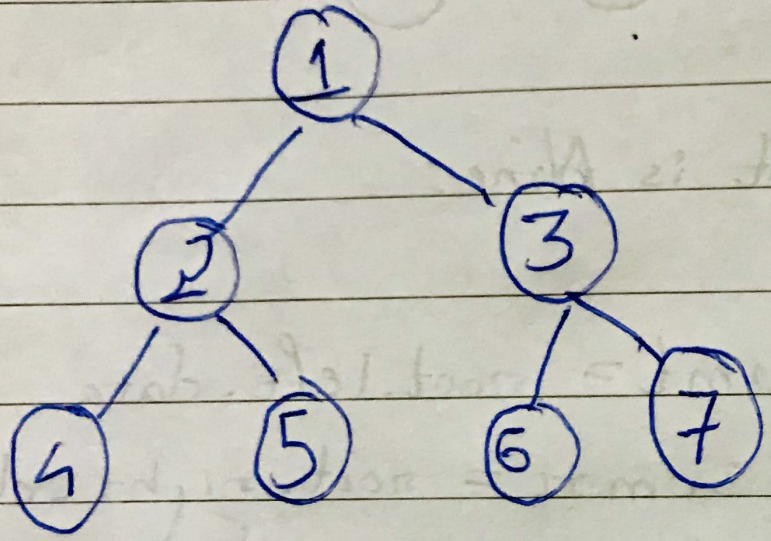
inn = [ 40, 20, 50, 10, 60, 30, 70 ]

pre = [ 10, 20, 40, 50, 30, 60, 70 ]
0 1 2 3 4 5 6

```
def BT(inn, pre, start, end)
root = Node(pre[pre_idx])
  pre_idx += 1
idx = Search(inn, start, end, root.d)
root.left = BT(inn, pre, start, idx-1)

root.right = (inn, pre, idx+1, end)
```



# Least Common Ancestor




n1 = 7  n2 = 5

path1 = [1, 3, 7]

path2 = [1, 2, 5]

if path[i+1] != path2[i+1]:
    return path[i]

```
def path(root, path, n):
    path.append(root.data)
    if root.data == n: return True
    if ~~find~~ path(root.left, path, n) or path(root.right, path, n):
        return True
    path.pop()
    return False
```