Todays Content

a. Sum()/ Size()/ Max()/ Height()

b. Balanced Tree()

c. Diametre Tree()

Implement below functions using recursion

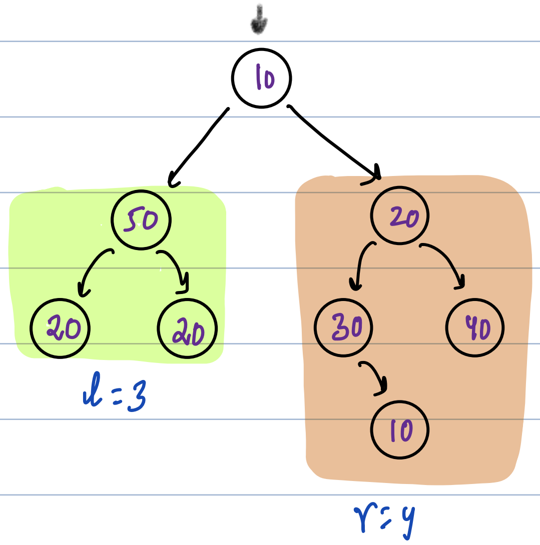
```
int  size(Node *root)
int  sum (Node *root)
int  max (Node *root)
int  height(Node *root)
```

Implement below functions

Ass: Given root of BT : calculate & return size of BT

TODO: do dry run

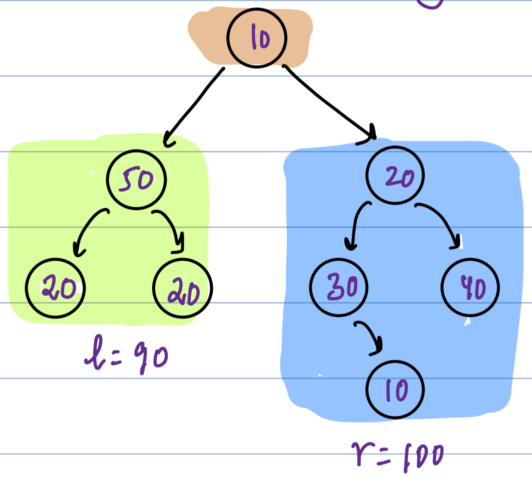
```
int size(Node root){
    if( root==nullptr ){
    } return 0;
    int l= size(root->left);
    int r= size(root->right);
    return l+r+1;
}
```



Ass: Given root of BT : calculate & return sum of BT

TODO: do dry run

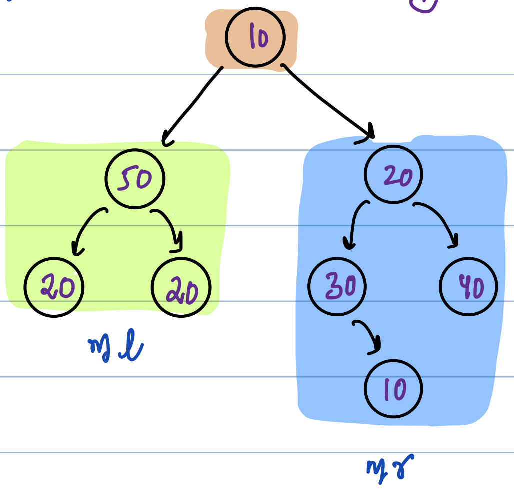
```
int sum(Node root){
    if( root==nullptr ){
    } return 0;
    int l = sum(root->left);
    int r= sum(root->right);
    return l+r+root->data;
}
```



Ass: Given root of BT: calculate & return max of BT

TODO: do dry run

```
int max(Node root){
    if(root==nullptr){
    }   return INT_MIN;
    int ml = max(root.left);
    int mr = max(root.right);
    return max(ml, mr, root->data);
}
```



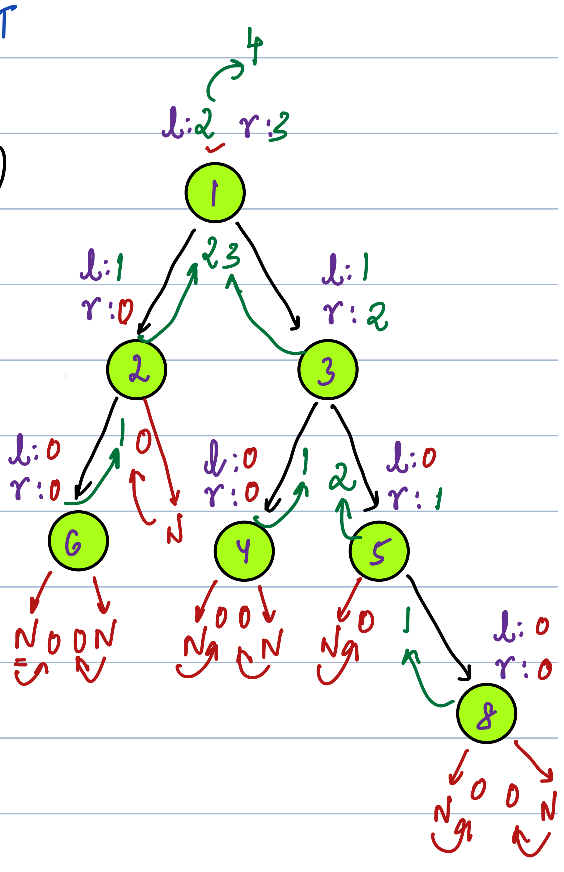
Note: Height based on nodes

Ass: Given root of BT: Calculate & return height of BT

#obs1: height of Tree = height(root)

#obs2: height(node) = 1+ max(height of child node)

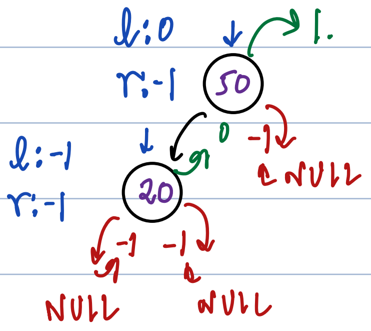
```
int height(Node root){
    if(root==nullptr){
    }   return 0;
    int lh = height(root.left);
    int rh = height(root.right);

    return max(lh,rh)+1; # height of root node
}
```



#Note: If we want height based on edge:

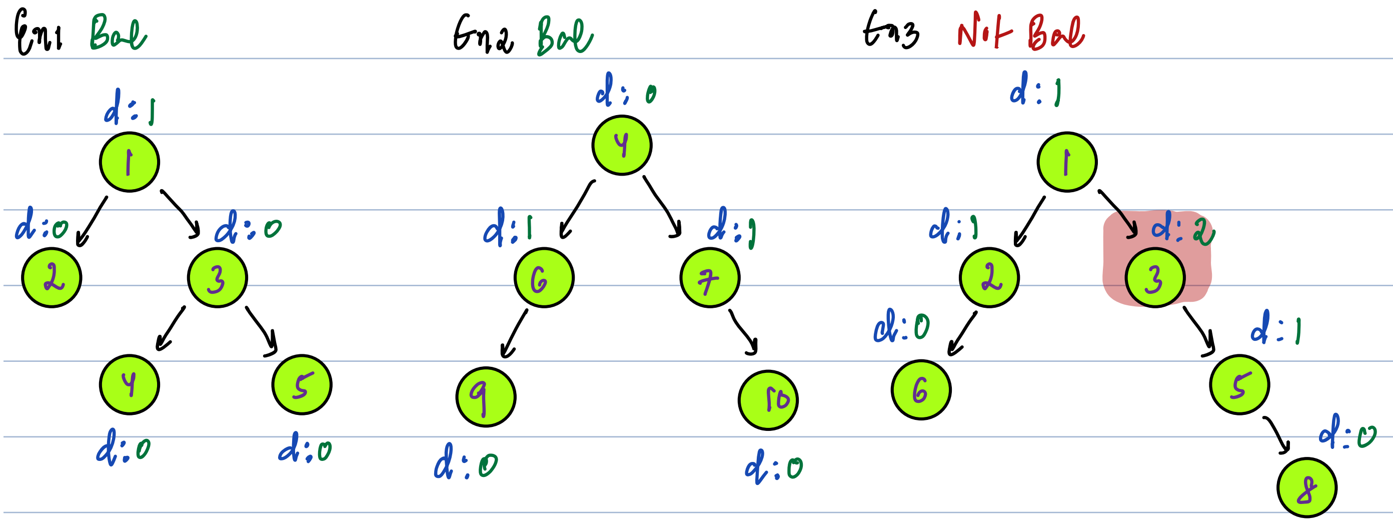
```
if(root==nullptr){ return -1;}
```

DryRun:



48 Given a BT, check if it's Balanced or Not.

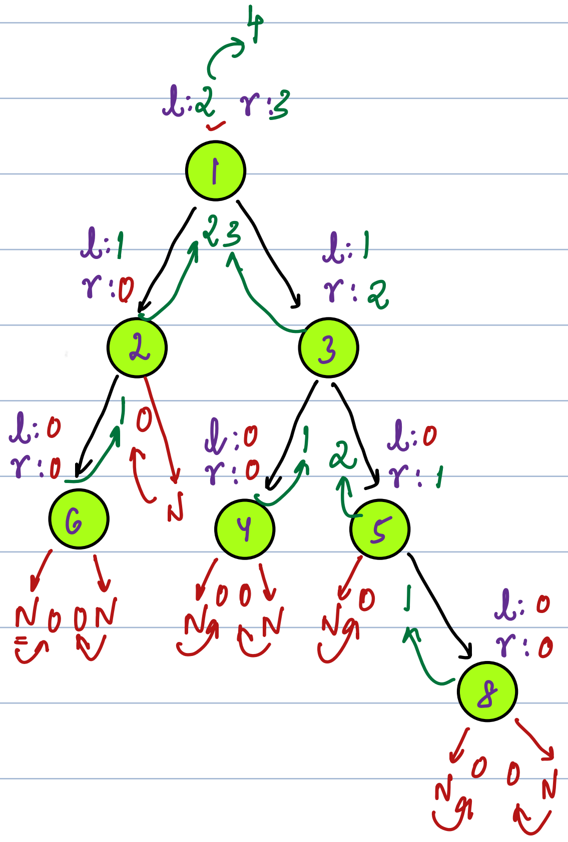
Note: A Binary Tree is said to balanced if
    For every node : abs| Height(LST) - Height(RST) | <= 1.



```
Idea: for every node in BT:
    l = height(node->left);
    r = height(node->right);
    if( abs(l-r) > 1){ #Not balanced
        return false;
    }
}
return True;
```

TC: O(N*{N+N}) = O(N2)   SC: O(H)
↳ stack size

Optimization: If we call height(root) on a, It will calculate height of l & r subtree for each node?



```
bool isbalanced = True;

int Height(Node root){
    if(root==nullptr){
        return 0;
    }

    int lh = Height(root.left);
    int rh = Height(root.right);
    #Check if node is balanced using lh & rh
    if(abs(lh-rh)>1){ isbalanced = false }
    return max(lh,rh)+1;
}

boolean isBalancedTree(Node root){
    isbalanced = True;   #reinitialize at start
    Height(root);
    return isbalanced;
}
```

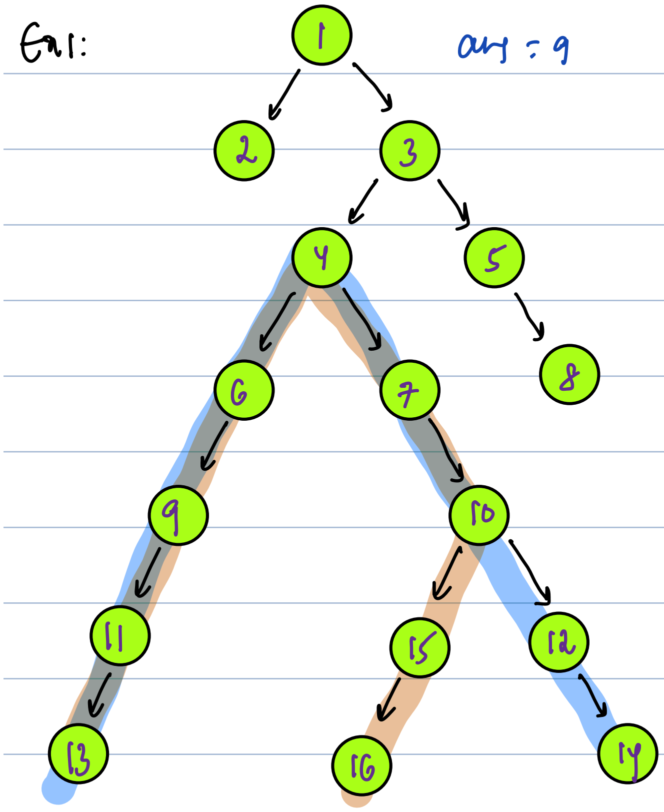
#obs:

Diameter of Binary Tree

Given a binary Tree, find the length of longest path between any two nodes in the tree.

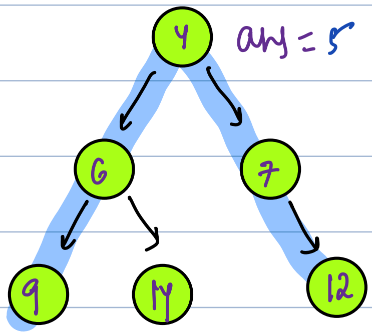
Note: Path length is calculated based on no: of nodes

Ex1:



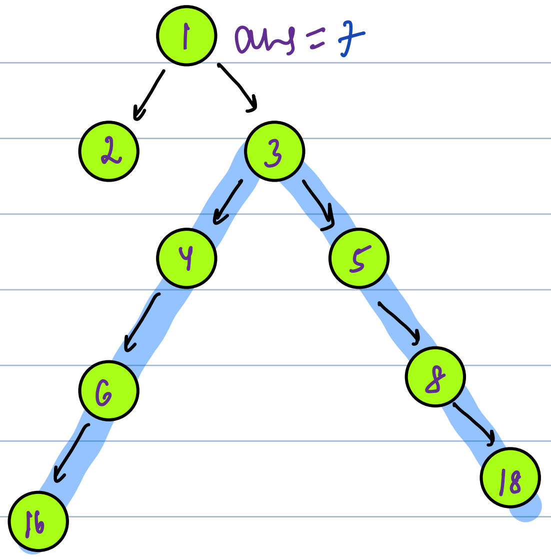
ans = 9

Ex2:



ans = 5

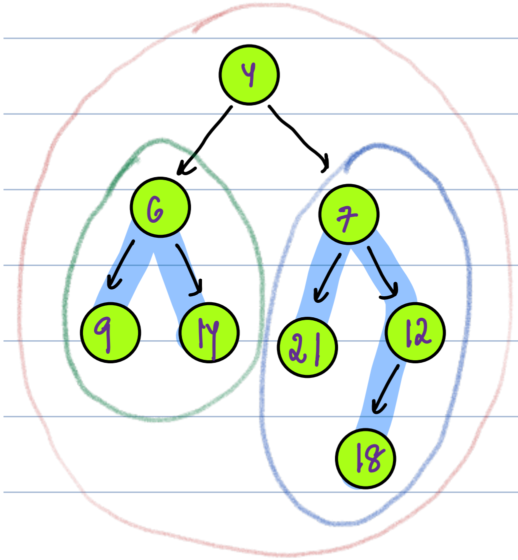
Ex3:



ans = 7

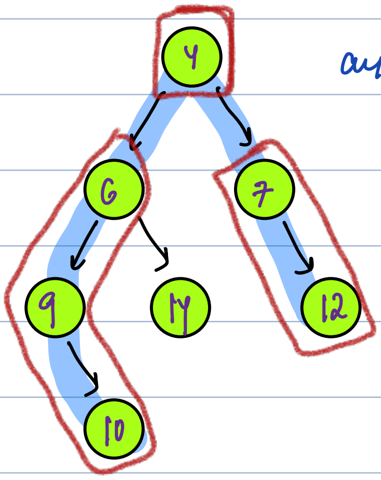
Idea1: Wrong ✗ length of longest path in BT

= length of longest path in LST + length of longest path in RST + 1



#obs: longest path will pass through rootnode of a subtree

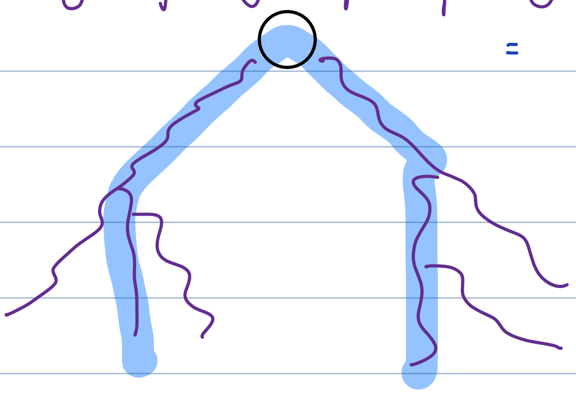
Ex: length of longest path passing through 4?



ans = 6 = hl + hr + 1 = 3+2+1

#Con: length of longest path passing through rootnode of a subtree?

= height(LST) + height(RST) + 1



Idea2:

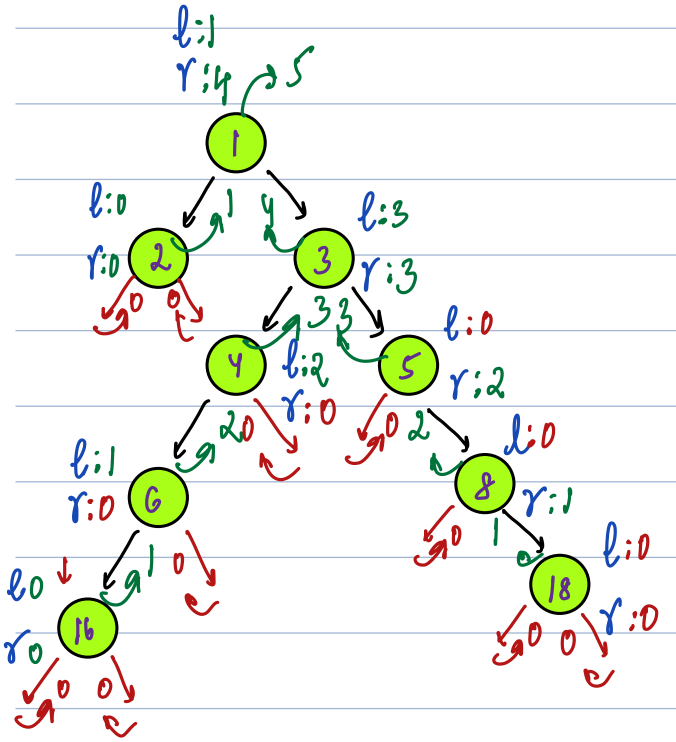
```
For every node:
|   Calculate length of longest path through node as root of subtree.
|       l = height(root->left)
|       r = height(root->right)
|       ans = max(ans, l+r+1);
}
```

return ans;

TC: $O(N*N) = O(N^2)$ SC: $O(1)$

Idea3: If we calculate height(tree):

For every node we will calculate height of LST & RST.



```
int ans=0;    TC:O(N) SC:O(H)
int Height(Node root){
    if(root==nullptr){ return 0;}
    int l= Height(root->left);
    int r= Height(root->right);
     #length of longest path passing through
      node as {root of subtree}  l+r+1
    ans = max(ans, l+r+1);
    return max(l,r)+1;
}

int diameter(Node root){
    ans=0; # Reinitialize at start
    Height(root);
    return ans;
}
```

#obs