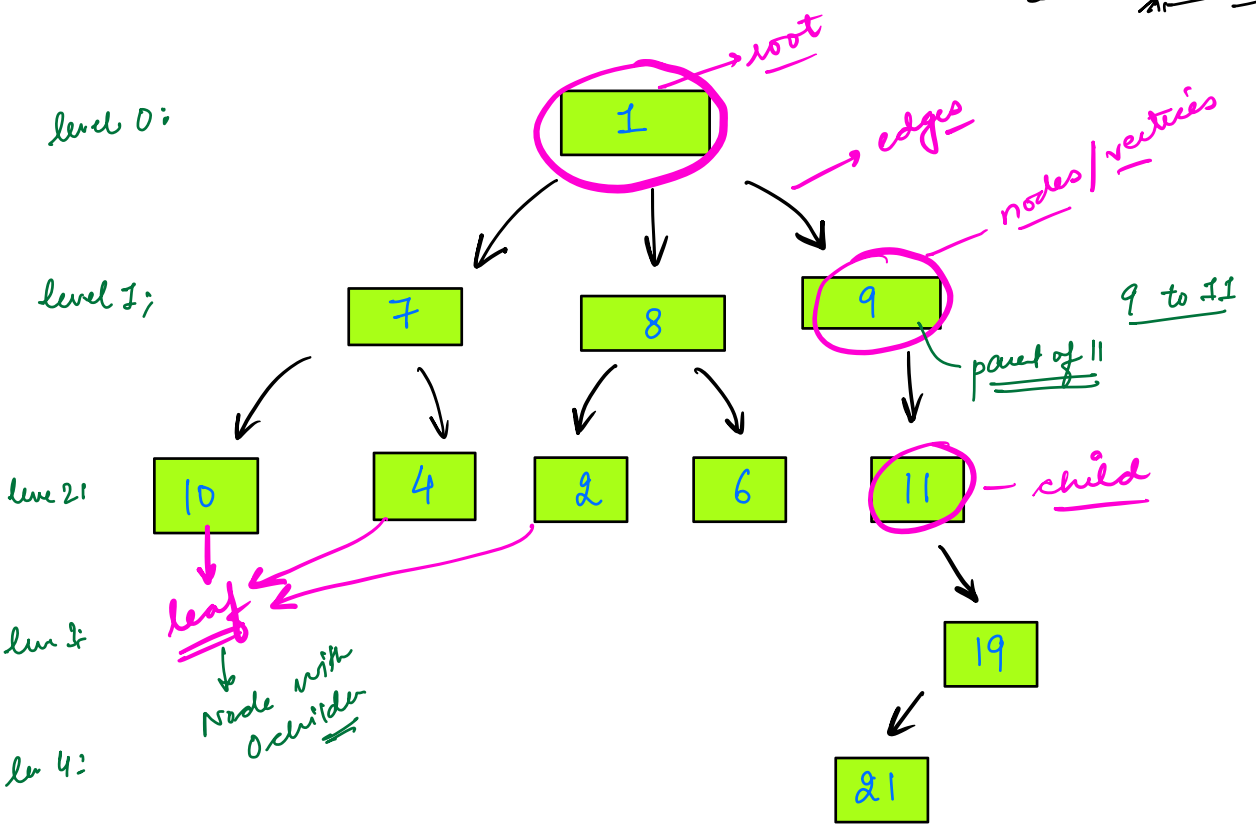
DS → array, hashmaps, LL, stack, queues } linear

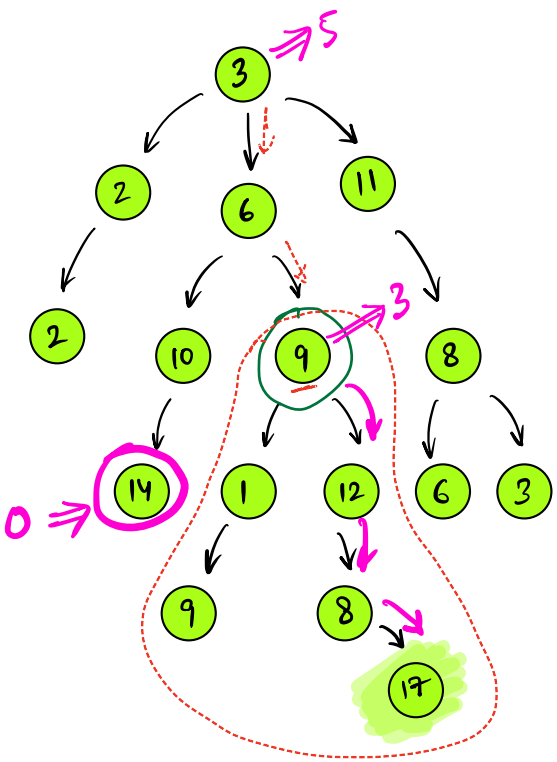
Hierarchical in nature

Tree

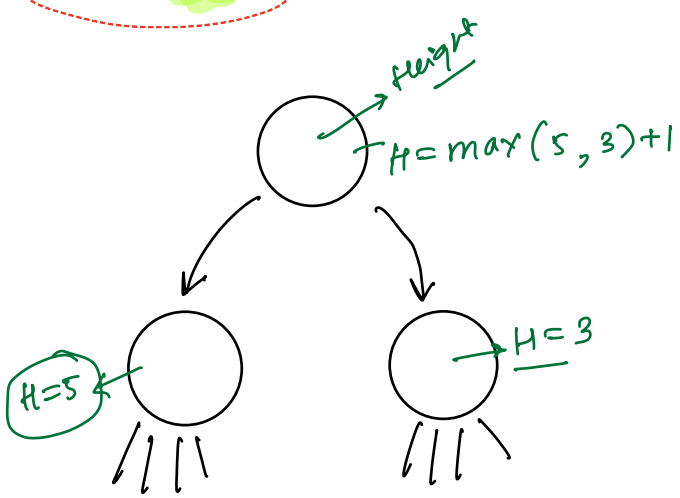


Height of a Node

→ distance from, count node to farthest leaf in it's own subtree

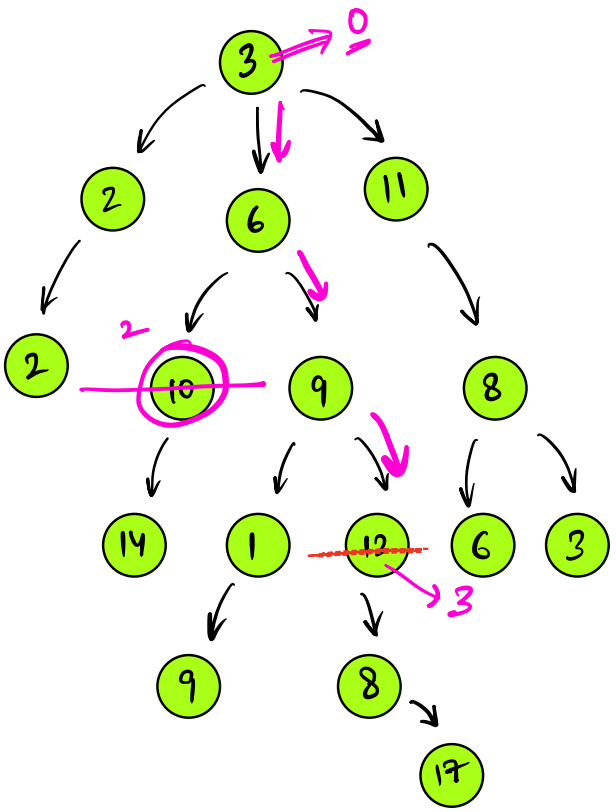


Height of root = Height of tree

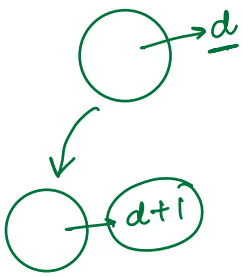


Height of a node = max(Height of its child) + 1

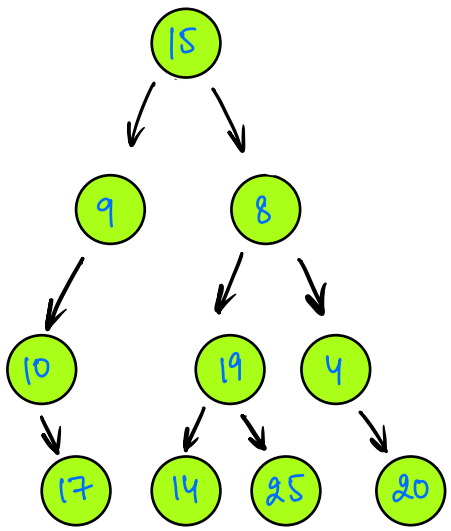
Depth of a Node → distance from root



depth = level

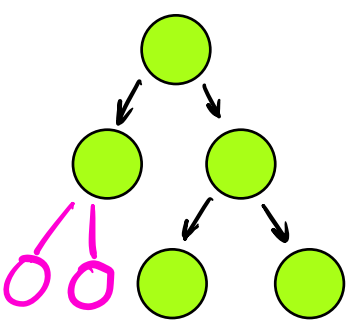


Binary Trees ⇒ Node <=2 children (or 0, 1, 2)

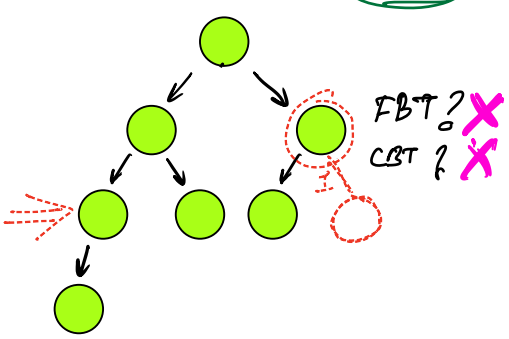


Types of BT

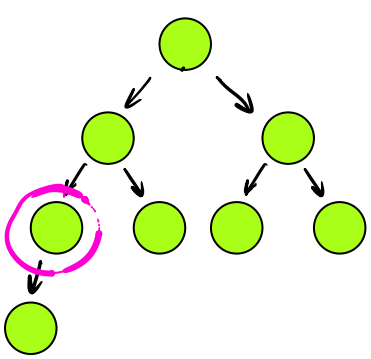
- full/proper
  - 0 or 2 child
- complete
  - every level should be completely filled
    - except for last
      - L to R



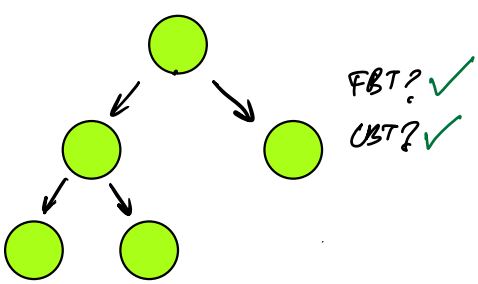
FBT? ✓
CBT? ✗



FBT? ✗
CBT? ✗



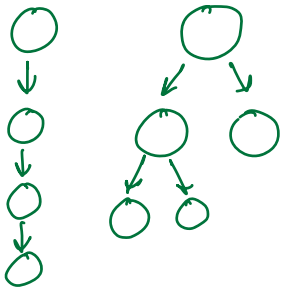
FBT? ✗
CBT? ✓

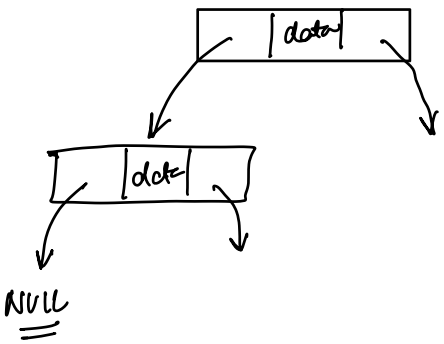


FBT? ✓
CBT? ✓

```
class Node {
    int data;
    Node left;
    Node right;
}
```

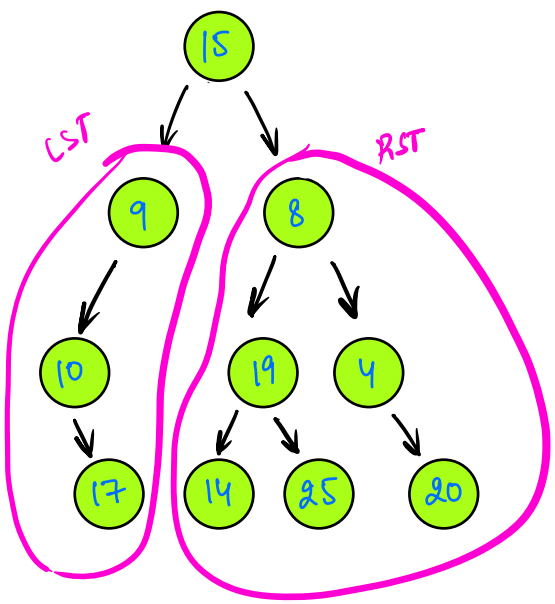
Node part;





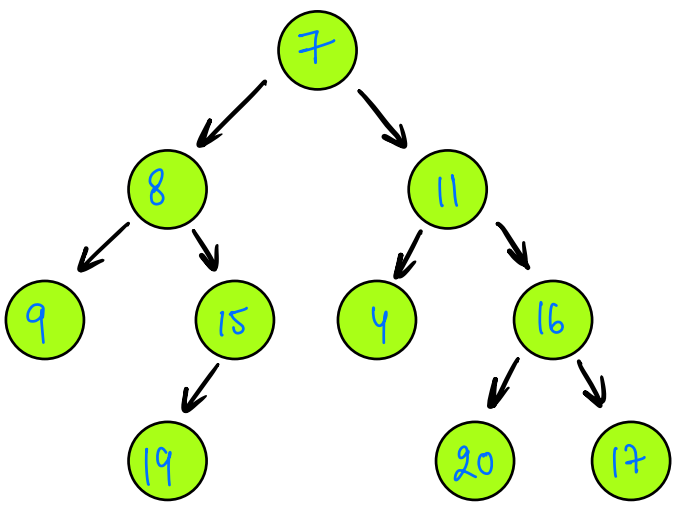
root

Recursion



Traversal of a BT

- inorder
- preorder
- postorder

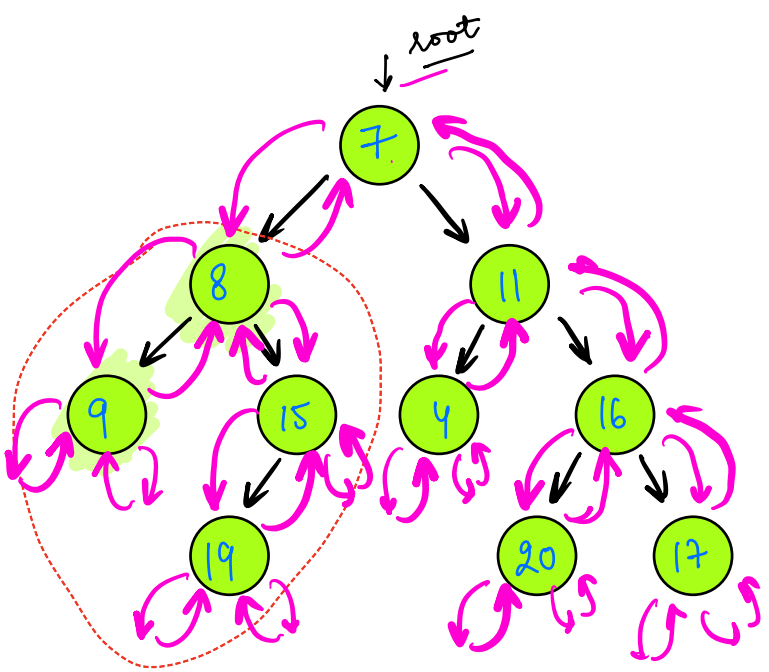


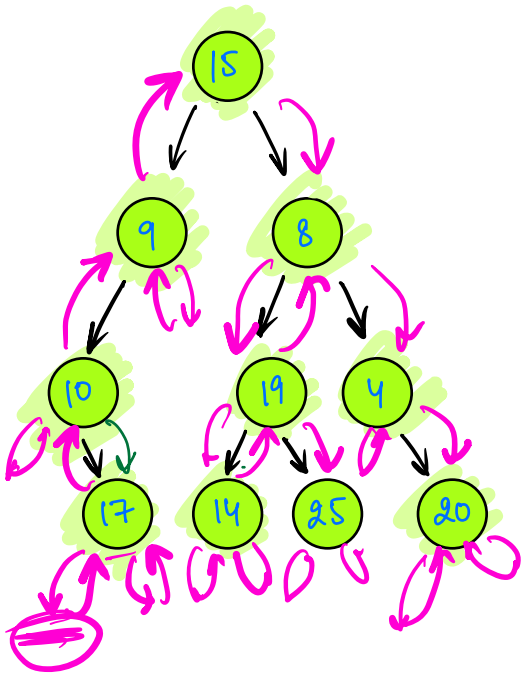
inorder

⇓

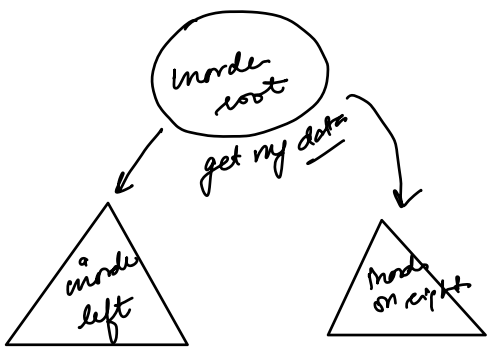
left subtree
root
Right subtree

9 8 19 15 7 4 11 20 16 17

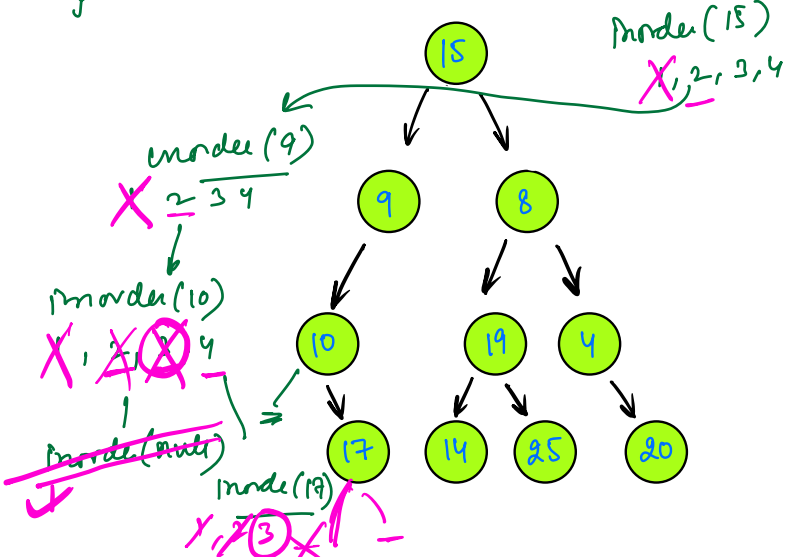




10 17 9 15 14 19 25 8 4 20



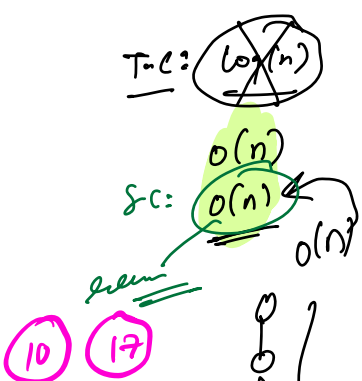

```
void inorder(Node root)
{
    if(root==null) return;          (1)
    inorder(root.left);             (2)
    print(root.data);               (3)
    inorder(root.right);            (4)
}
```



T.C: O(n)

S.C: O(n)

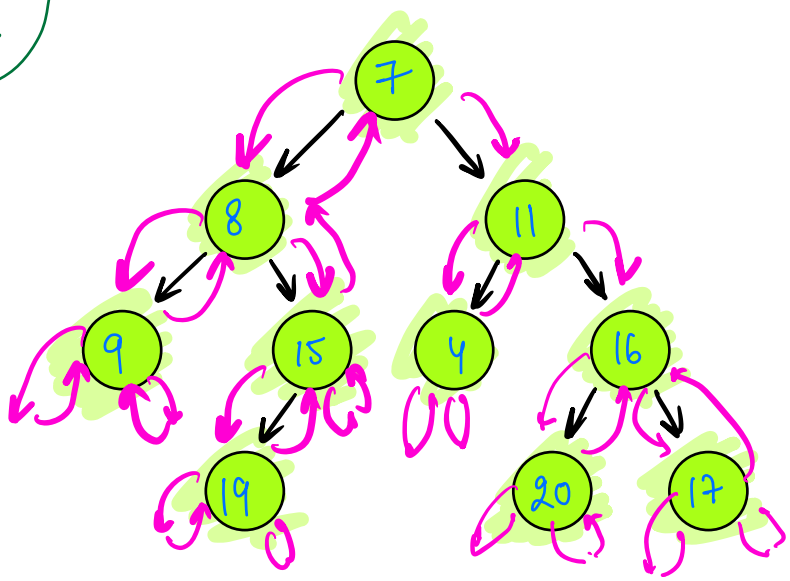
O(n)

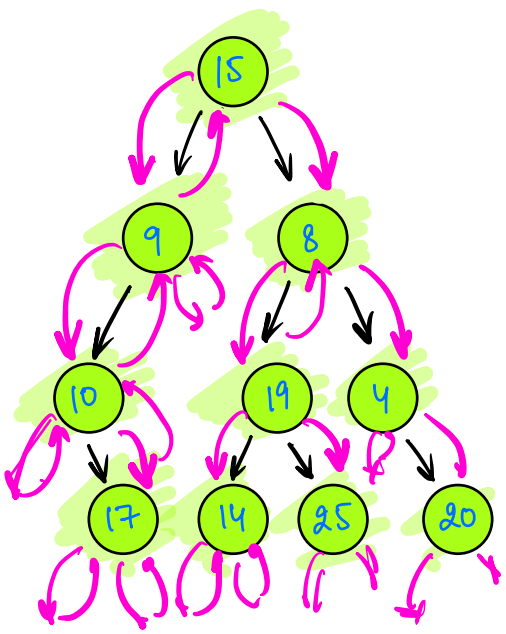


preorder

root
LST
RST

7 8 9 15 19 11 4





15 9 10 17 8 19 14 25 4 20

```
void preorder(Node root)
{
    if(root == null) return;
    print(root.data)
    preorder(root.left);
    preorder(root.right);
}
```

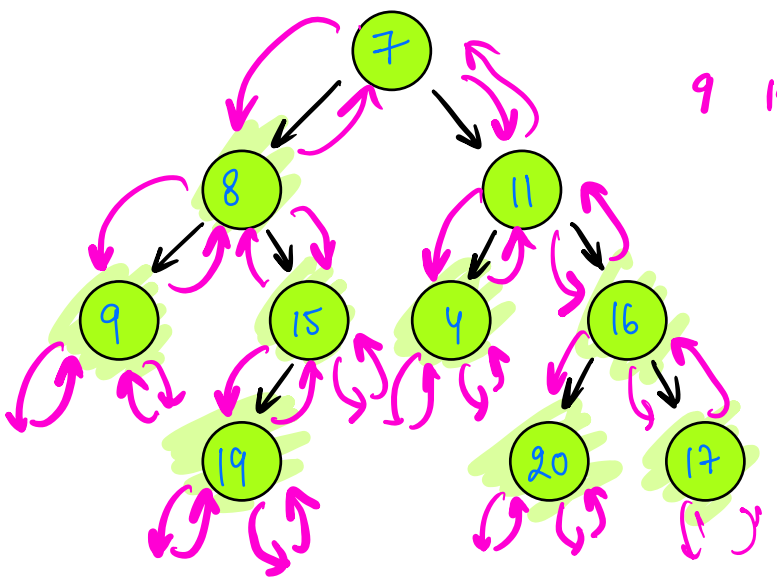
#

postorder

LST

RST

root



9 19 15 8 4 20 17 16

11 7

post

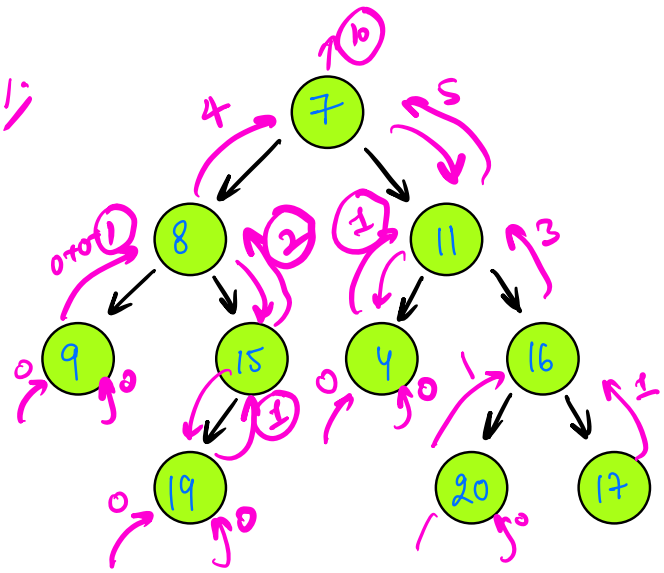
post

root.data

Calculate size of the tree :- total no of nodes

size(tree) = size(LST) + size(RST) + 1;

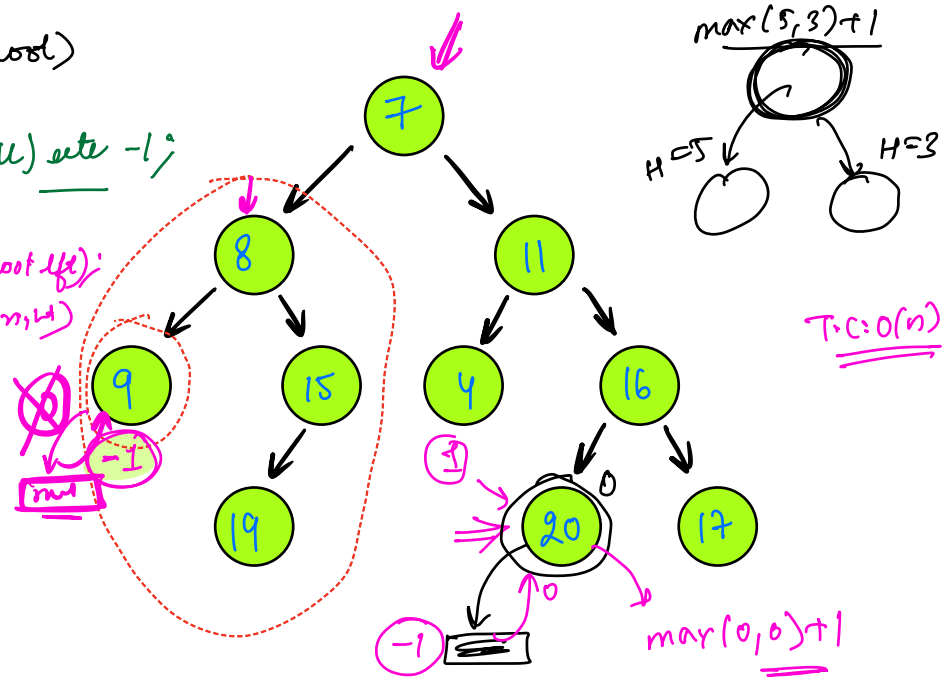
```
int size(Node root)
    if(root==null)
        return 0;
    int ls = size(root.left);
    int rs = size(root.right);

    return ls+rs+1;
}
```



Q height of a tree

height(root) = max(height(LST), height(RST)) + 1

```
int height(Node root)
{
    if(root == NULL) return -1;
    int hl = height(root.left);
    int hr = height(root.right);
    return max(hl,hr) + 1;
}
```

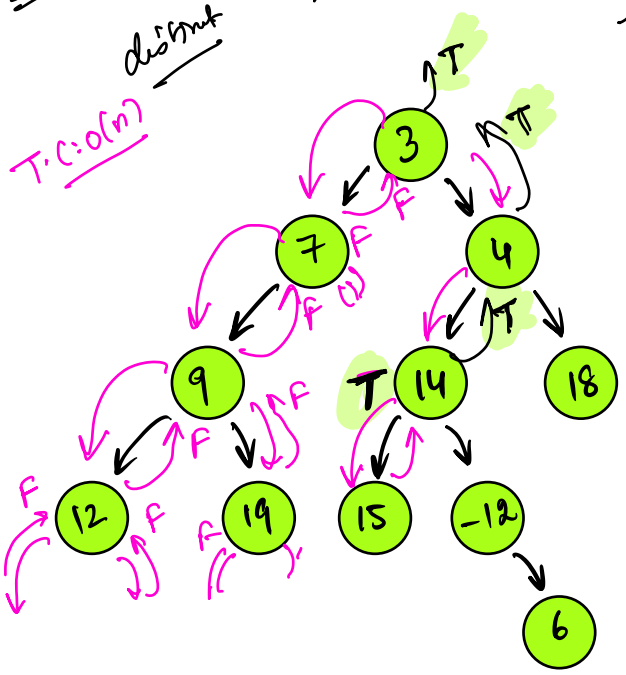


T.C: O(n)

Q Search if k elements exists in your tree. return true;

distinct

T.C: O(n)



3 → 4 → 14 → 15

9? true
15? Yes

```
bool search( root)
{   if(root==null) return false;
    if(root.data ==k)
        return true;
    bool l = search(root.left)
    if(!l)
        bool r= search(root.right)
    return l || r
}
```

return search(left) || search(right);

F

Q path from root → element

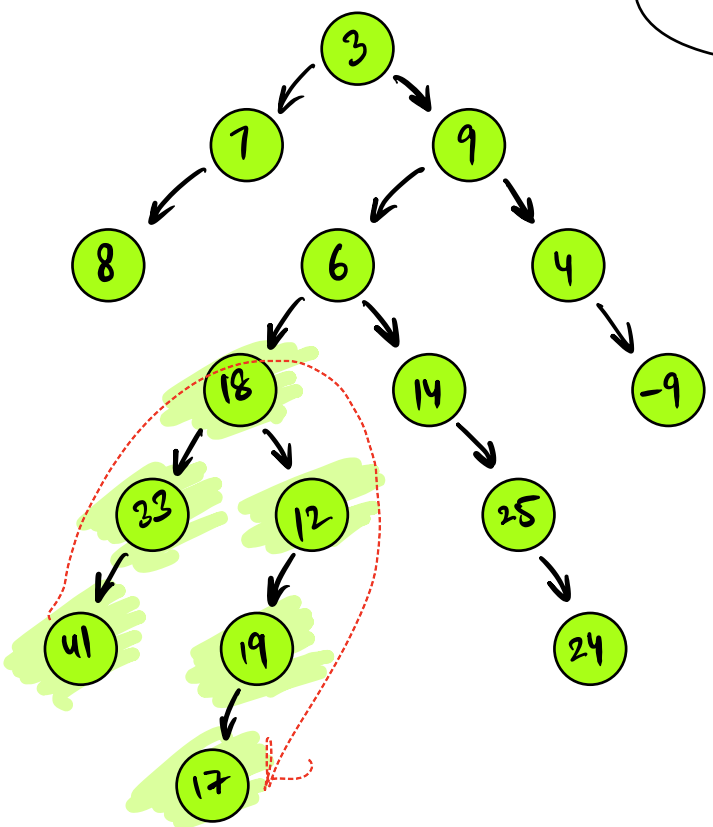
by reference

T.C: O(n)

```
path( root, k, path)
{   if(root==null) return false;
    if( root.data==k)
    {
        path.insert(root.data);
        return true;
    }
    if( path(left) || path(right))
    {   path.insert(root.data)
        return true;
    }   return false;
}
```

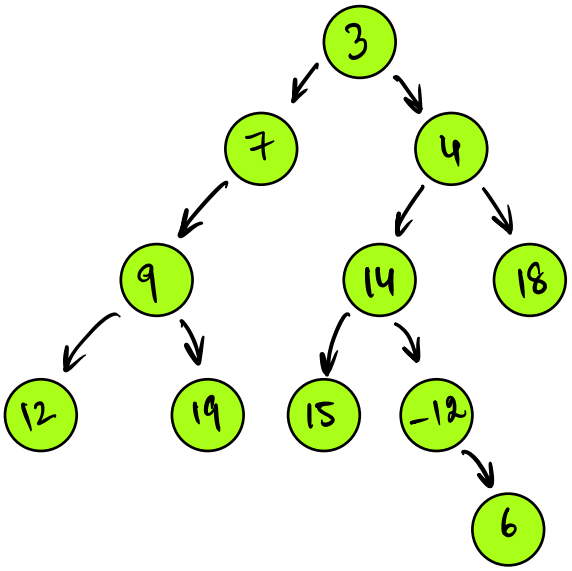
recursive path

8



path b/w any two nodes!

41 → 17


3
7
4
9
14
18
12
19
15
-12
6