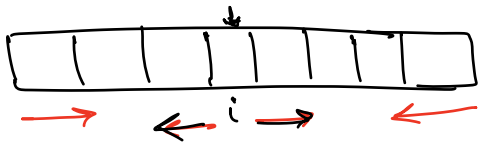
- Arrays
  - O(1) random access time



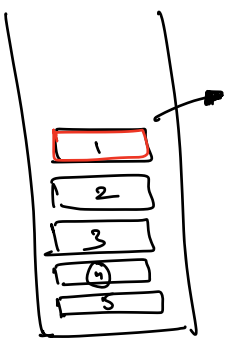
L to R
R to L.

⇒ Linear DS.

- Linked List

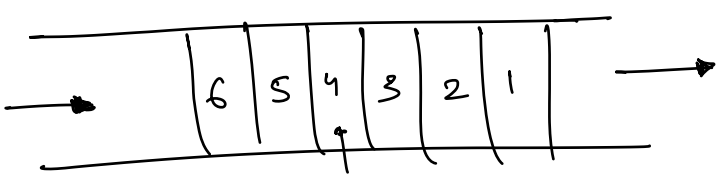
→ Linear DS.

- Stack

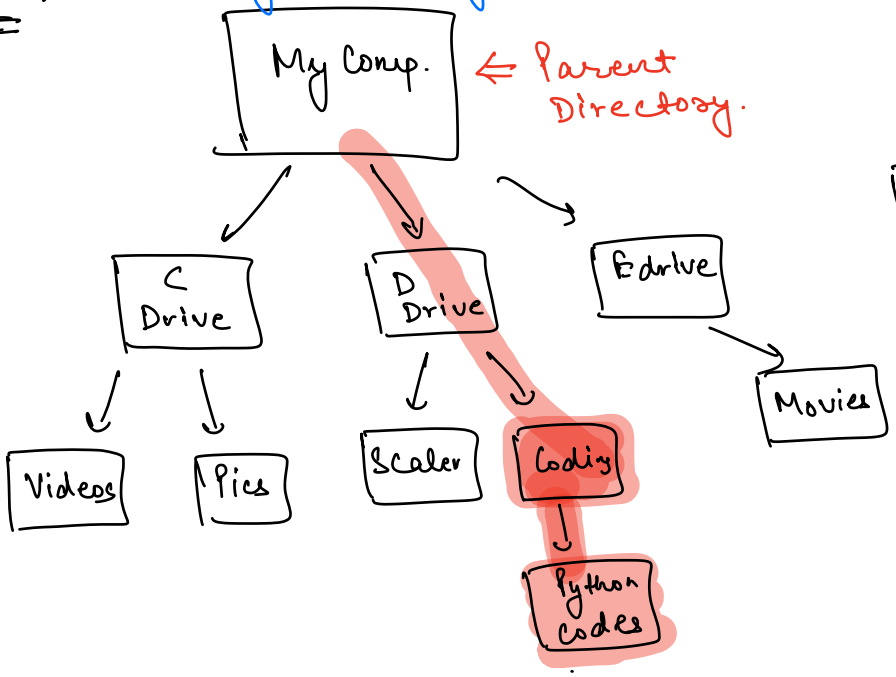


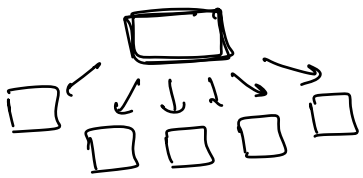
- Linear DS.

- Queue. ⇒ Linear DS.

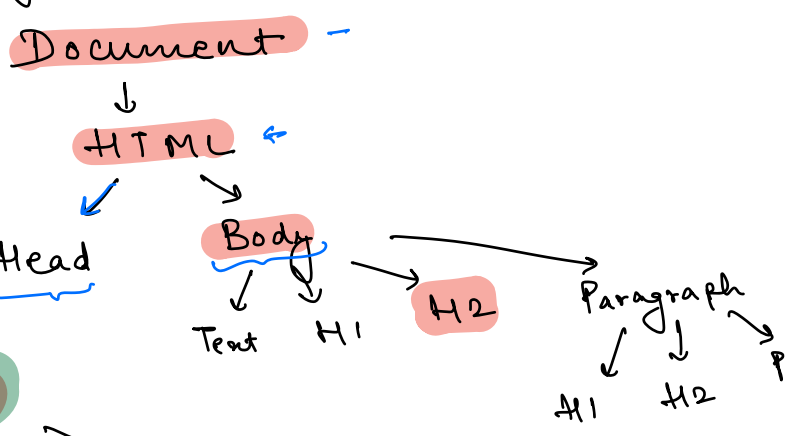


Ex 1 :- Directory Hierarchy.

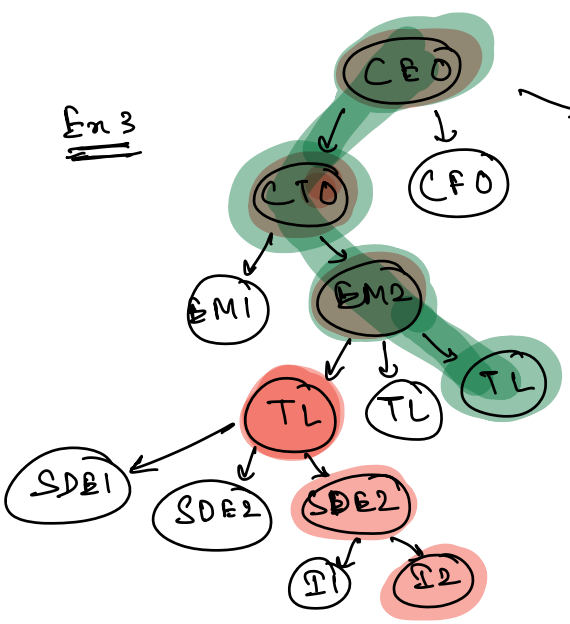




Ex 2 :- HTML Page :-



Ex 3



Hierarchical Data

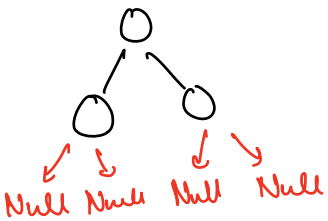
Tree

Tree :-

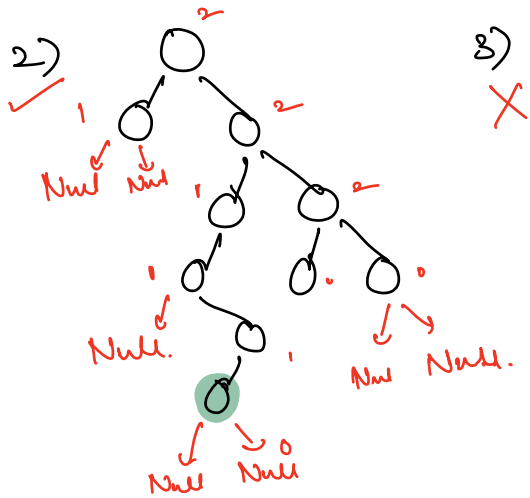
→ Binary Trees.

↳ Every node can have 0/1/2 children.

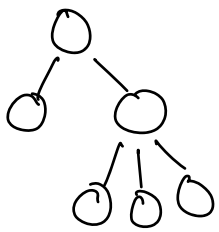
1) ✓



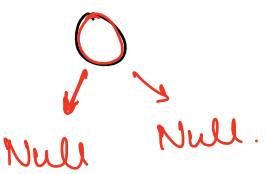
2) ✓

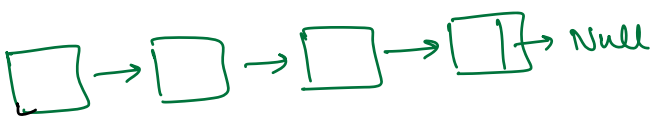


3) ✗



4) ✓





Tree Node class :-

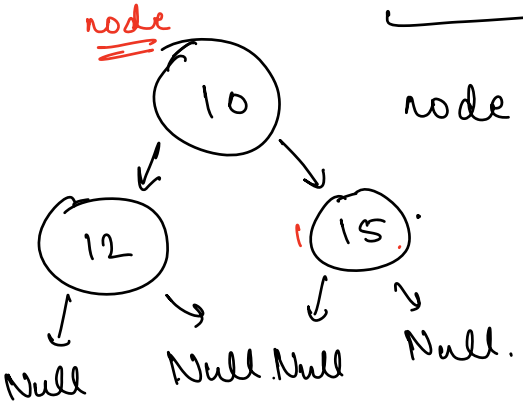
```
Class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;
    TreeNode (int x) {
        this.data = x;
        this.left = null;
        this.right = null;
    }
}
```

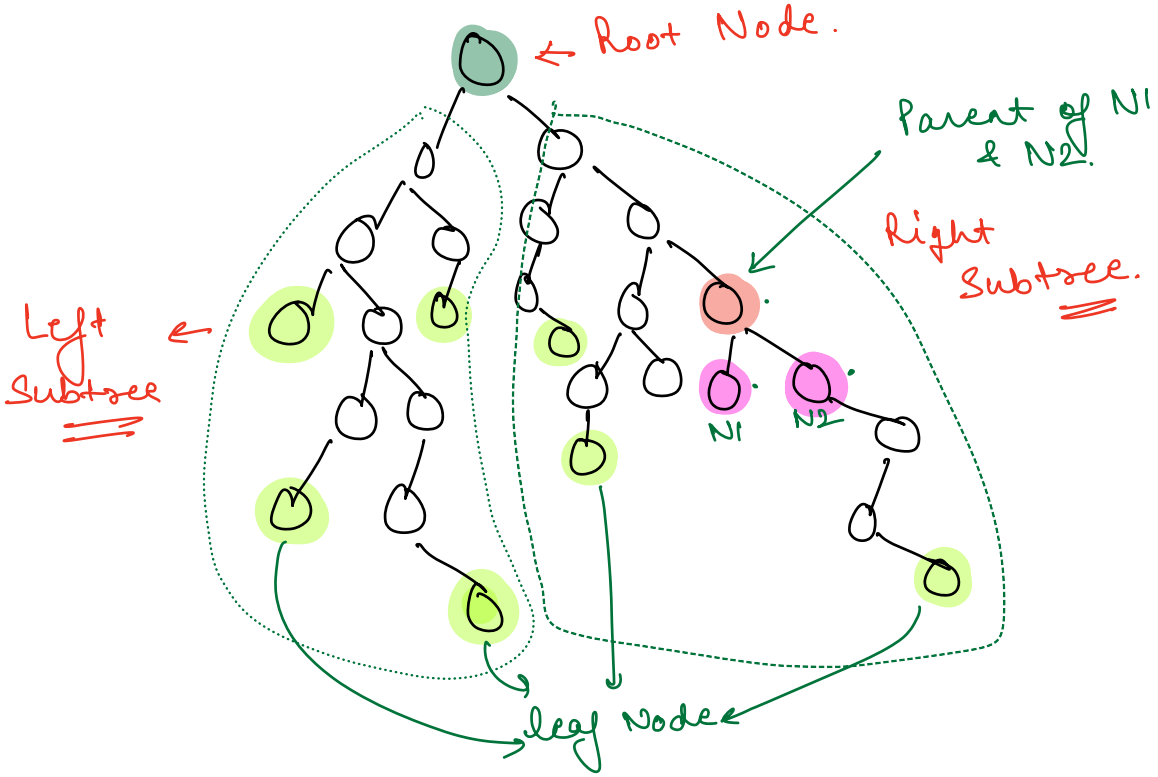
```
Class TreeNode:
    def __init__(self, x):
        self.data = x
        self.left = None
        self.right = None
```

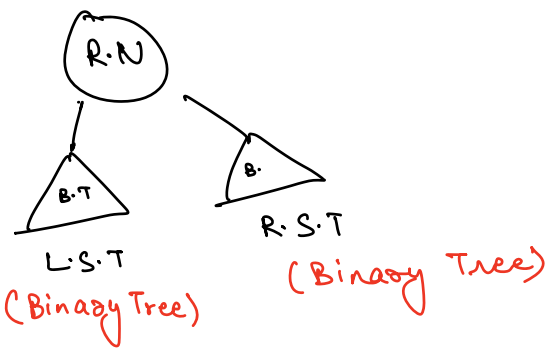
TreeNode node = new TreeNode(10)

node.left = new TreeNode(12)

node.right = new TreeNode(15);

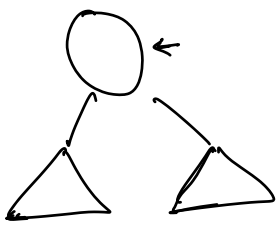






- Property of Binary Tree :-
  - ↳ L.S.T is also a Binary Tree
  - → R.S.T is also a Binary Tree.

⇒ Programming paradigm :- Recursion.

⇒ 

- Root. value
- L.S.T
- R.S.T

Root. data L.S.T R.S.T Pre order	Root. data R.S.T L.S.T ✗	L.S.T ← Root. data R.S.T ← Inorder	L.S.T R.S.T Root. data Post Order	R.S.T Root. data L.S.T ✗ ----

* L.S.T before R.S.T.

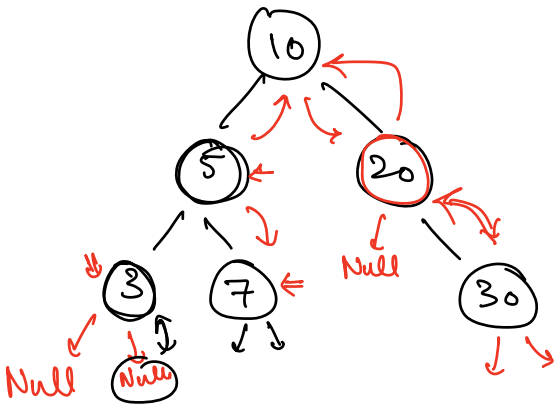
Pre Order :-

Root. data ←
L.S.T
R.S.T.

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

Void inOrder(TreeNode root) {
    if(root == Null) return;
    inOrder(root.left);
    print(root.data);
    inOrder(root.right);
}

Void postOrder(TreeNode root) {
    if(root == Null) return;
    postOrder(root.left);
    postOrder(root.right);
    print(root.data);
}
```



Pre Order

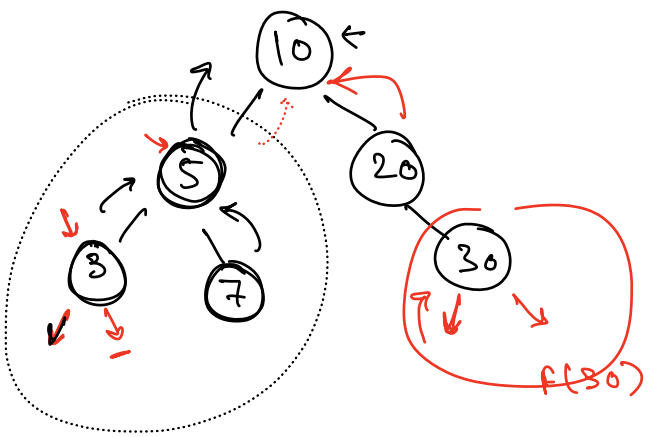
↳ Root.data ←

L.S.T

R.S.T.

10, 5, 3, 7, 20, 30

In Order

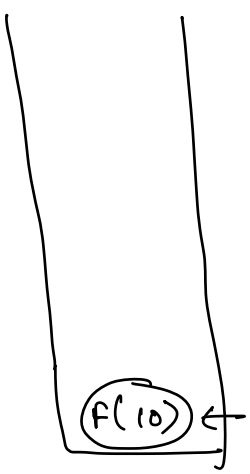


3, 5, 7, 10, 20, 30

→ L.S.T

→ Root.data

→ R.S.T



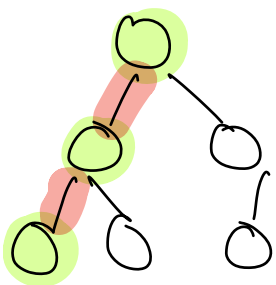
HW :- Dry run for Post Order Traversal.

Post order :- 3, 7, 5, 30, 20, 10

Q. Given a Binary tree, find its height.

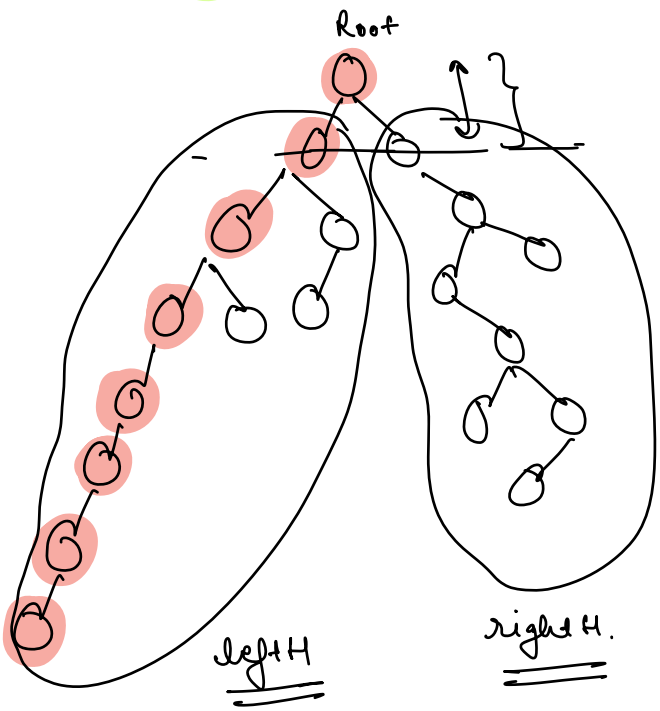
Root is given.

length of the largest path from root to leaf.



no. of nodes

H = 3 | 2 → no. of edges.

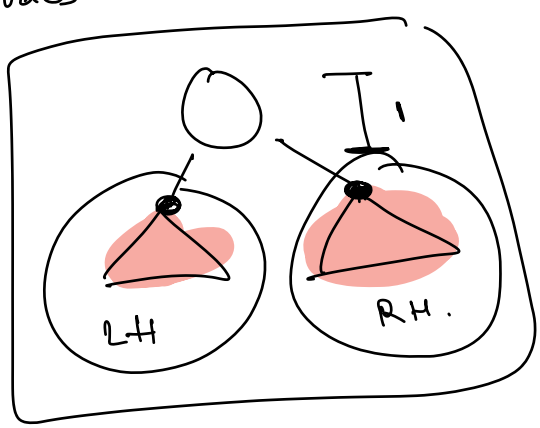


max(leftH, rightH) + 1

H = 8 | 7

↓ ↳ edges.

Nodes



max(LH, RH) + 1

Height of B.T = max (left Height , right Height) + 1.

```
int height (TreeNode root) {
    if (root == Null) return 0;

    int leftH = height (root.left);   LST
    int rightH = height (root.right); RST

    return max (leftH, rightH) + 1;   Root
}
```

LST
RST
root.data
} → Post Order