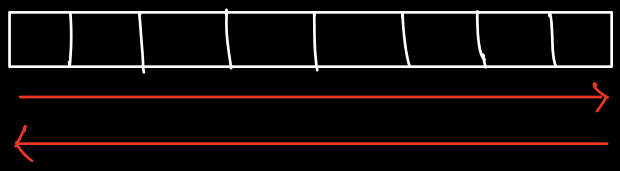
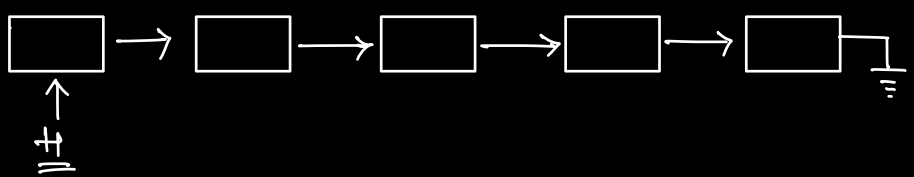
1. Array



- Iteration : L to R or R to L.
- O(1) random access.
- NO hierarchy.

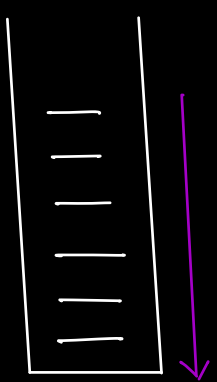
2. Linked List



- linear iteration.
- NO hierarchy.

3) Stack

LIFO



- linear iteration.
- NO hierarchy.

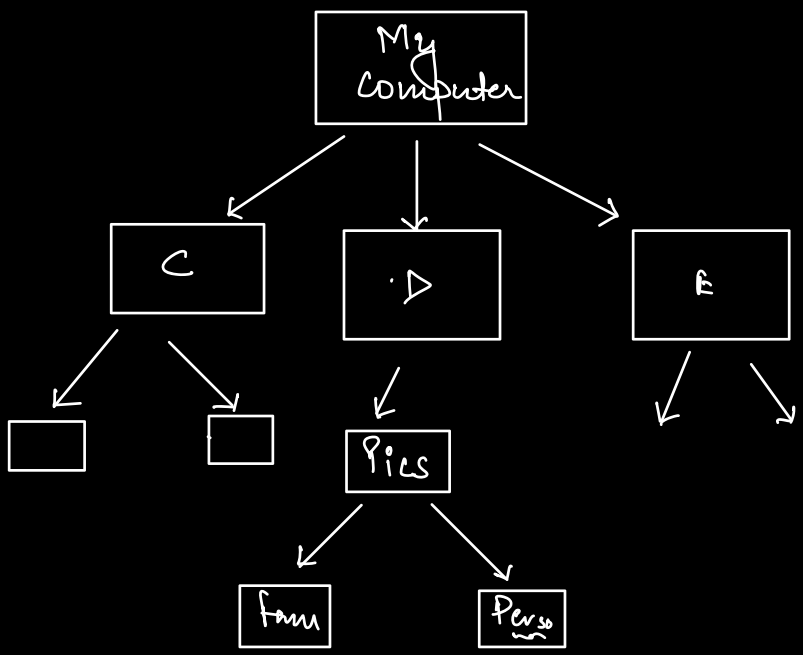
4. Queue

FIFO

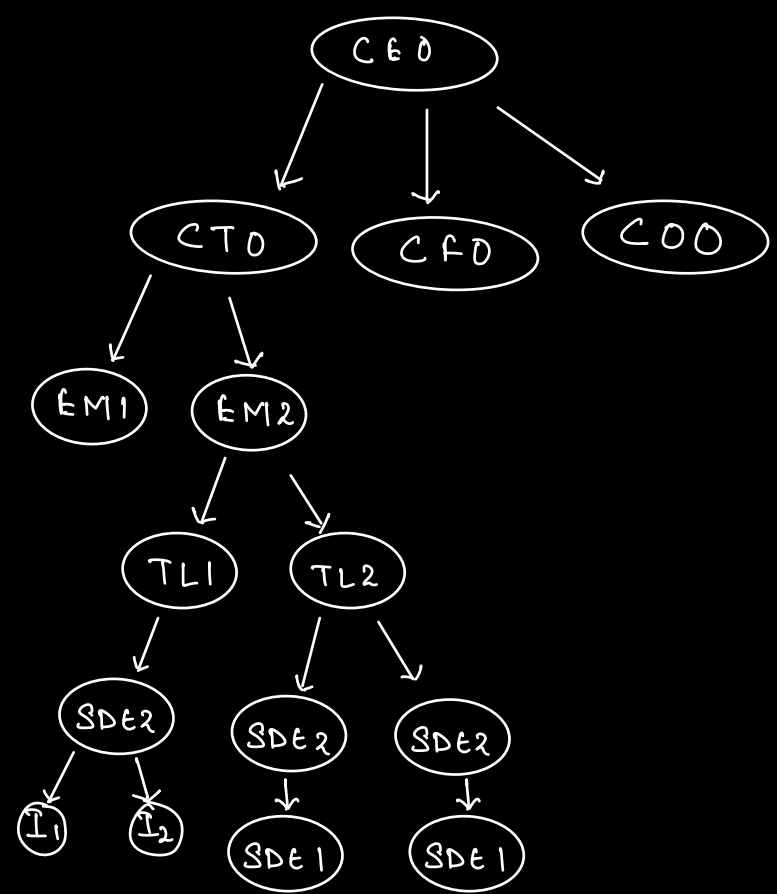
- linear iteration.
- NO hierarchy.

* Hierarchical Data

File System



2)

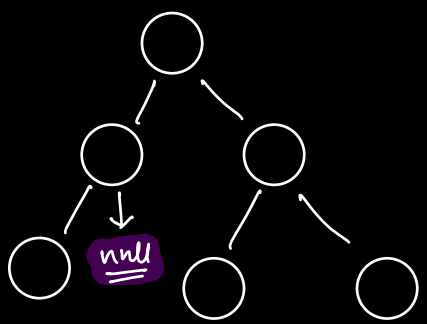


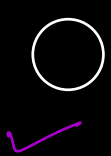
⇒ Hierarchical DS

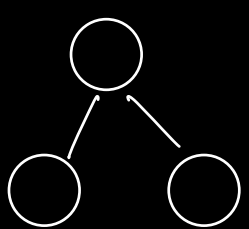
↳ Tree.

⇒ Binary Tree

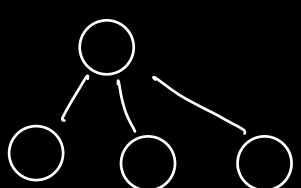
↳ Max 2 children of a node.



1) 

2) 

3)

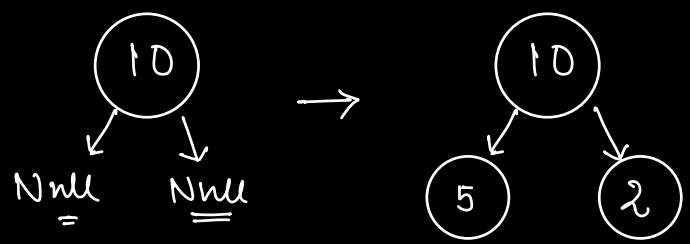
4) 

Node

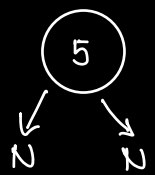

```
Class Node {
    int data;
    Node left;
    Node right;
    Node(int x) {
        this.data = x;
        this.left = null;
        this.right = null;
    }
}
```

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

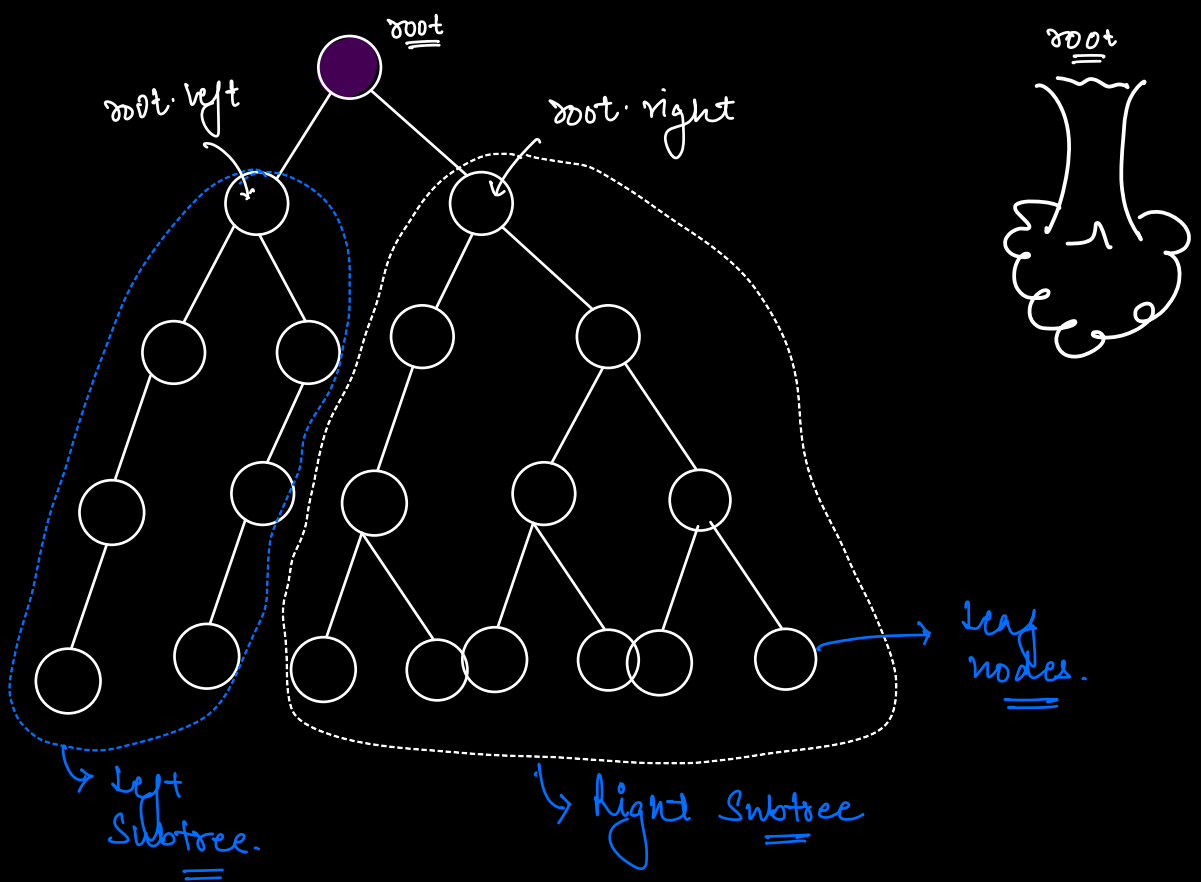
```
TreeNode node = new TreeNode(10);
```




```
TreeNode l = new TreeNode(5);
node.left = l;
```




```
node.right = new TreeNode(2);
```



⇒ Left subtree is a B.T

⇒ Right subtree is a B.T

⇒ Binary Tree is a recursive data structure.

LST Root RST

Tree traversals.

⇒

Root	Root	LST	LST	RST	RST
LST	RST	Root	RST	LST	Root
RST	LST	RST	Root	Root	LST
✓	✗	✓	✓	✗	✗

Rule : Visit LST before RST.

Root
LST
RST
✓ — Preorder

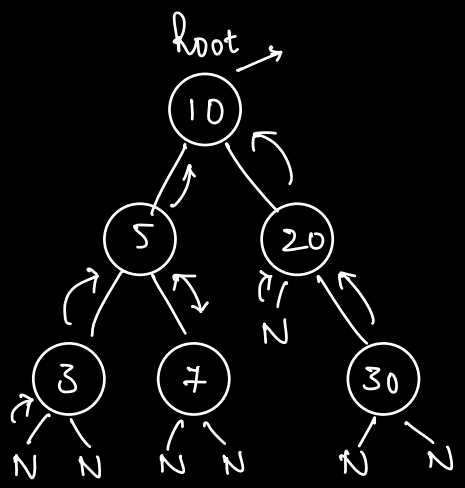
LST
Root
RST
✓ — Inorder

LST
RST
Root
✓ — Postorder

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

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

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



PreOrder

10, 5, 3, 7, 20, 30

PostOrder

3, 7, 5, 30, 20, 10

HW Dry run for Inorder & Postorder.

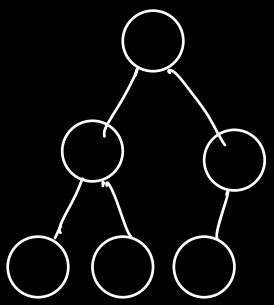
TC: O(N) — no. of nodes.

SC: O(N) (Worst Case)

Skewed Tree.

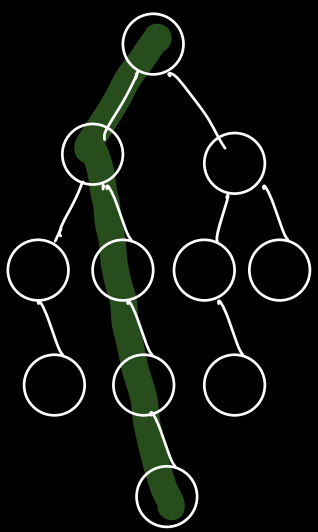
Q: Given a Binary Tree, find its height.

height = length of largest path from root to leaf.



H = 2 | 3

Edges | nodes.



H = 4 | 5

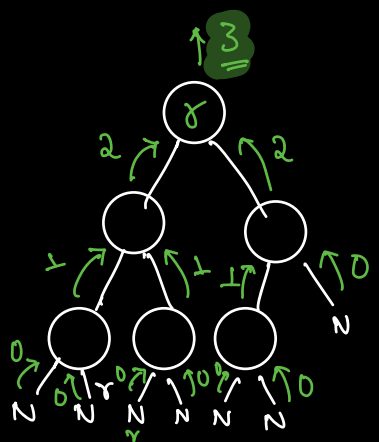
$$H = \max(\text{height}(LST), \text{height}(RST)) + 1$$

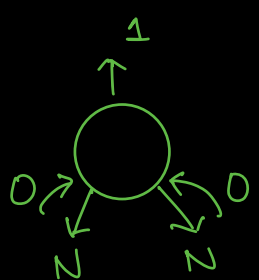
```
int height(TreeNode root) {
    if(root == Null)
        return 0;
    leftHeight = height(root.left);
    rightHeight = height(root.right);
    return max(leftHeight, rightHeight) + 1;
}
```

No. of nodes in the longest Path.

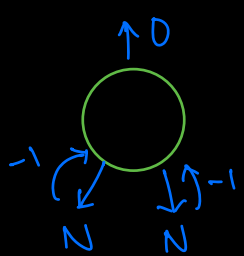
3

Post order traversal.





TC: O(N), SC: O(N) (Worst Case)



Height (No. of edges) = 0.

```
if(root == Null)
    return -1;
```
