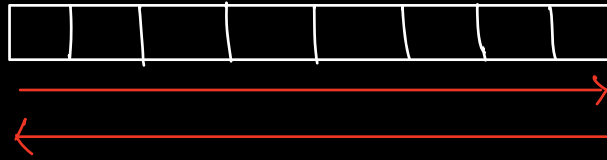
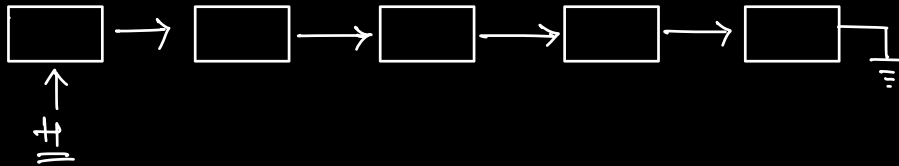


1. Array



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

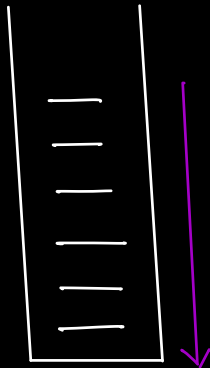
2. Linked list



- linear iteration.
- NO hierarchy.

3) Stack

LIFO



- linear iteration.
- NO hierarchy.

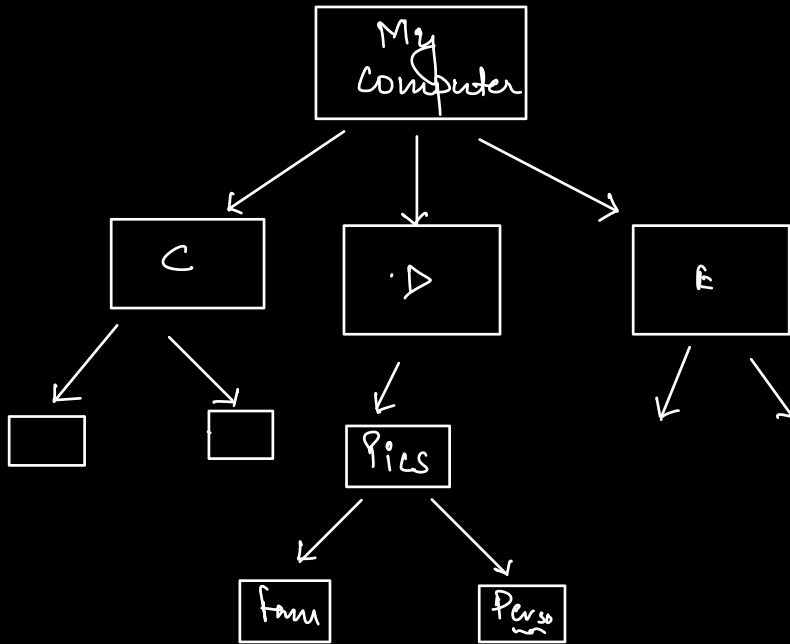
4. Queue



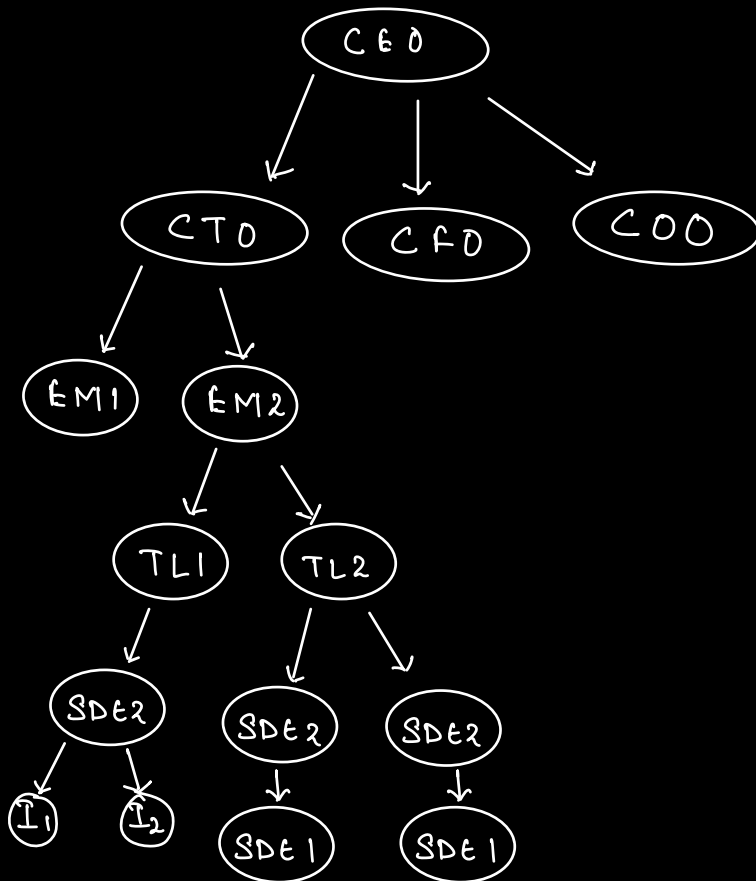
- linear iteration.
- NO hierarchy.

* Hierarchical Data

File System

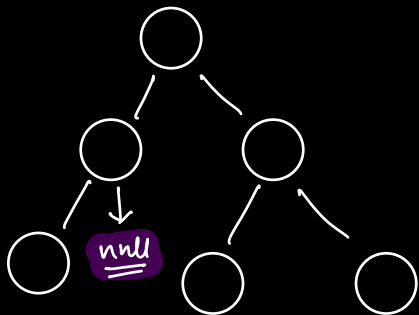


2)

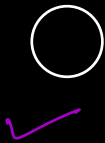


⇒ Hierarchical DS
↳ Tree.

⇒ Binary Tree
↳ Max 2 children of a node.

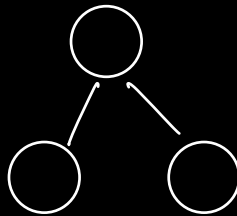


1)



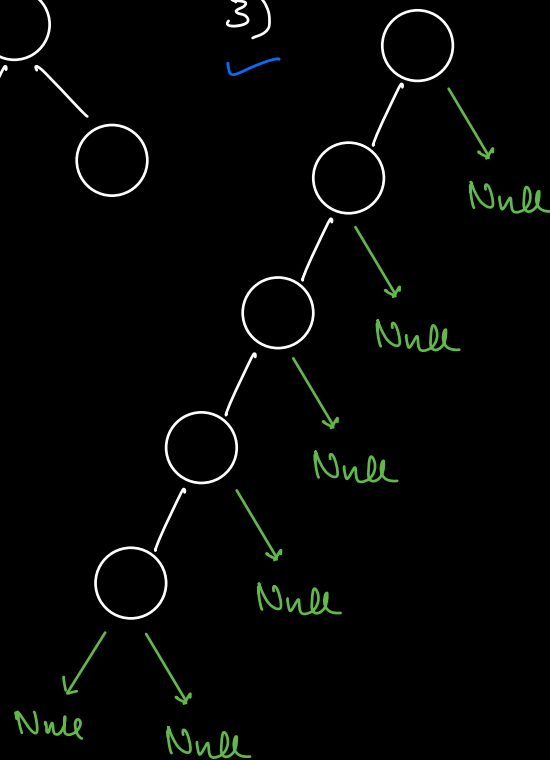
2)

✓



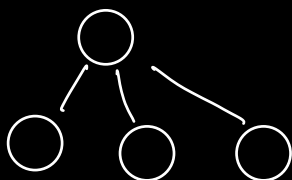
3)

✓



4)

✗



Skewed Tree

Node



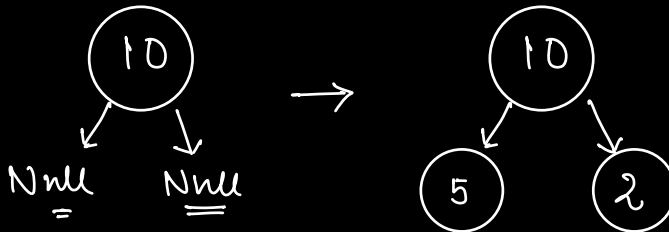
```
Class Node {  
    int data;  
    Node left;  
    Node right;  
    Node(int n) {  
        this.data = n;  
        this.left = null;  
        this.right = null;  
    }  
}
```

3

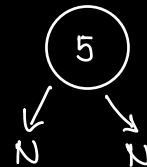
Class TreeNode:

```
def __init__(self, n):  
    self.data = n  
    self.left = None  
    self.right = None
```

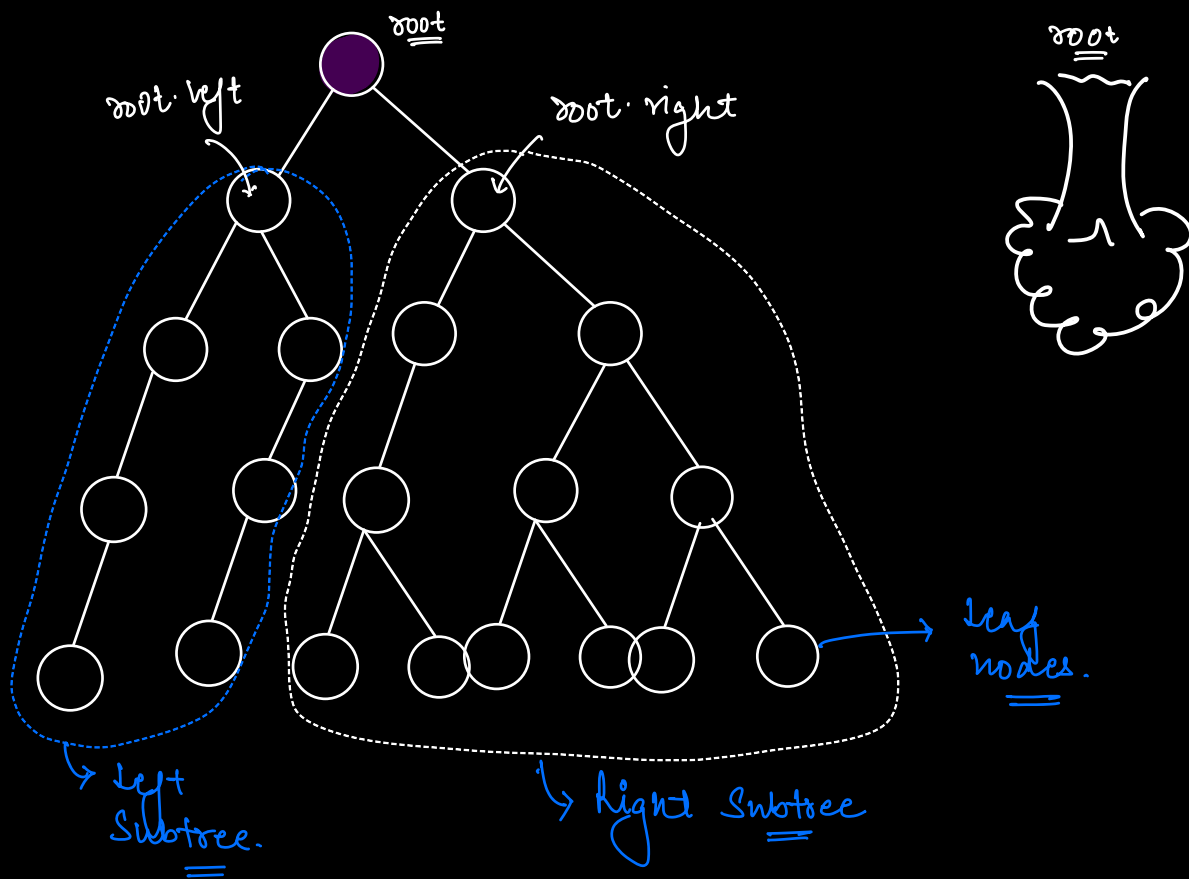
TreeNode node = new TreeNode(10);



```
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.

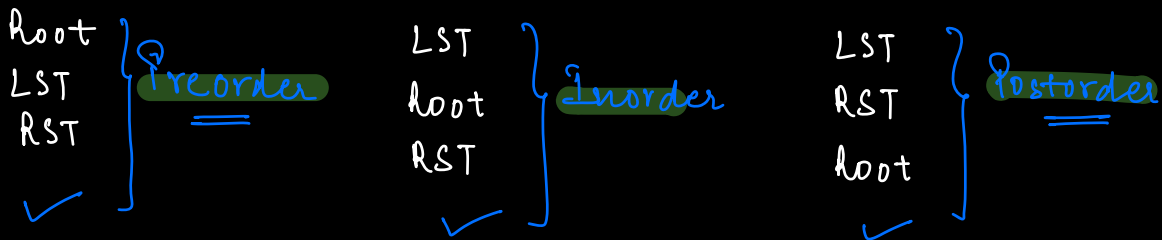
LST Root RST

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.



```

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

```

3

```

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

```

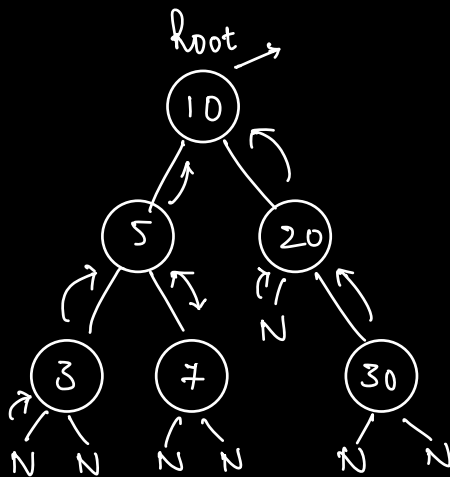
3

```

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

```

3



PreOrder

10, 5, 3, 7, 20, 30

PostOrder

3, 7, 5, 30, 20, 10

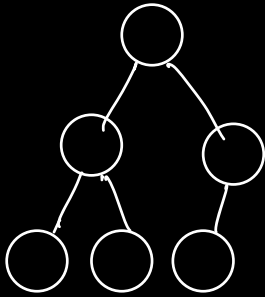
HW Dry run for Inorder & Postorder.

TC: $O(N)$ ^{no. of nodes}

SC: $O(N)$ (Worst Case)

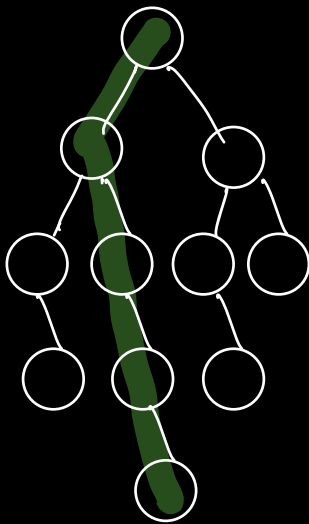
Skewed Tree.

Q. Given a Binary Tree, find its height.



$H = 2 \mid 3$
↓ ↓
Edges nodes.

length of largest path
from root to leaf.



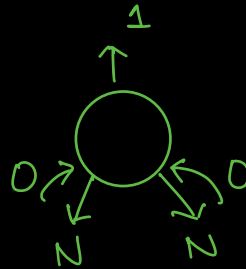
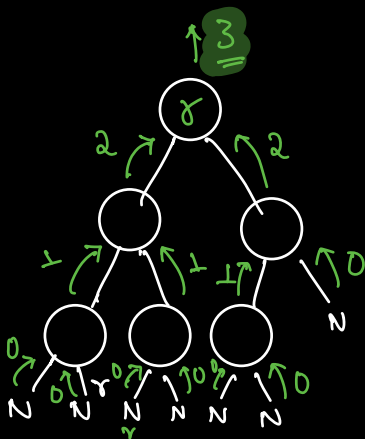
$H = 4 \mid 5$

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

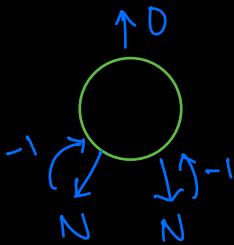
`int height (TreeNode root) {`
`if (root == Null) Return -1 to return height in terms of edges`
`return 0;`
`leftHeight = height (root.left);`
`rightHeight = height (root.right);`
`return max (leftHeight, rightHeight) + 1;`

No. of nodes in the longest path. $\underline{3}$

Post order traversal.



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



Height (No. of edges) = 0.

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

———— * ————