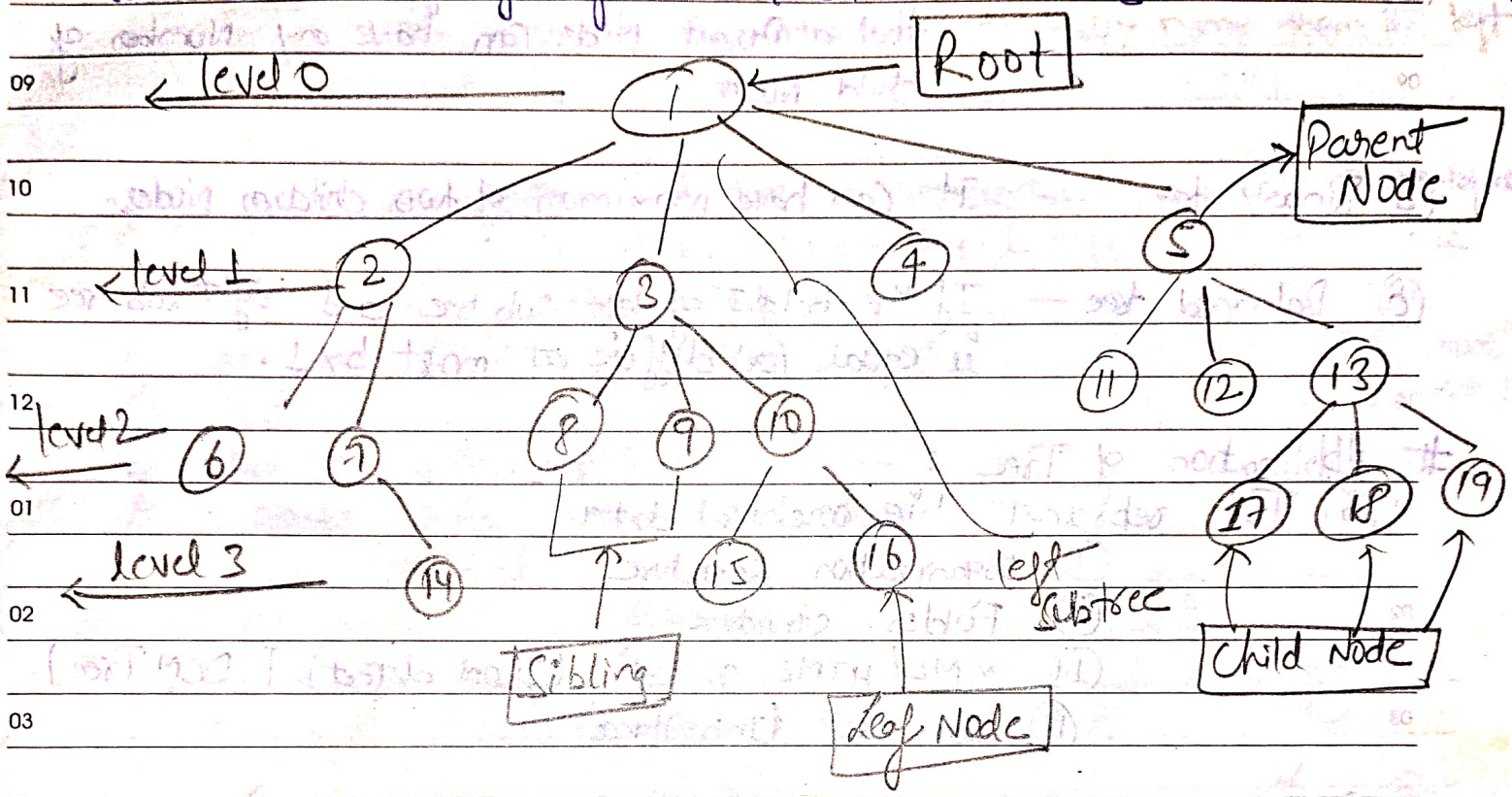
S	M	T	W	T	F	S
AUG 2024				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Tree

'24
30th Week
209-157

JULY
27
SATURDAY

→ A tree is non-linear data structure consisting of a collection of nodes such that each node of the tree ~~above~~ stores a value and a list of references to other nodes. - [hierarchical Data Str



Q Why Tree is considered as a NON linear Data structure?

Ans The data in a tree are not stored in a sequential manner. i.e, they are not stored linearly. Instead, they are arranged on multiple levels (or) we can say it is hierarchical structure.

→ Properties :-

(a) Traversing in a tree is done by dfs (or) bfs.
(b) It has no loop and no circuit.
(c) It has no self loops.
(d) It's hierarchical model.

JULY
28 SUNDAY
'24
31st Week
210-156

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

Types of Tree :—

(A) General tree — No restriction on number of Nodes, It means that a Parent Node can have any Number of child Nodes.

(B) Binary tree — It can have maximum of two children Nodes.

(C) Balanced tree — If the height of left sub tree and right sub tree is equal (or) differ at most by 1.

Application of Tree : —

(a) To represent hierarchical data
  (i) Organization Structure
  (ii) Folder structure
  (iii) XML/HTML content (JSON object) [DOM Tree]
  (iv) In OOP (Inheritance)

(b) Binary Search trees.

(c) Binary heaps — Priority Queue.

(d) B & B+ Trees in DBMS [for Indexing]. ☺

(e) Spanning tree & shortest path trees in computer network. → router, Bridges

(f) Parse tree, Expression tree in compilers. → Compiler

(g) Trie [It is used to represent Dictionary].

(h) Suffix Tree.

(i) Segment Tree.

and so one.

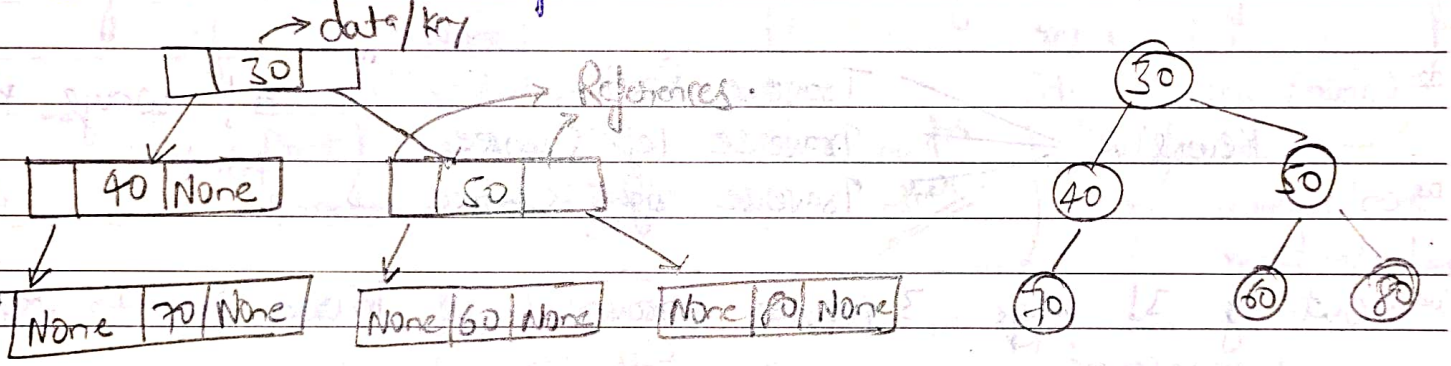
Binary Tree in Python :—

Binary Tree: A tree whose elements have atmost 2 children is called a binary Tree. Since each elements in a binary tree can have only 2 children, we typically name them the left and right child.

Binary Tree Representation: A tree is represented by a pointer to the topmost node of the tree. If the tree is empty then the value of the root is NULL.

A Tree node contains the following parts. { In binary tree degree of a Node is atmost 2 }

(a) Data.
(b) Pointer to the left child.
(c) Pointer to the right child.




```
Class Node :
    def __init__(self, k):
        self.left = None
        self.right = None
        self.key = k
```

```
root = Node(10)
root.left = Node(20)
root.right = Node(30)
root.left.left = Node(40)
```

→ Tree is Hierarchical data structure, Main uses of trees include maintaining hierarchical data, providing moderate access and insert/delete operation. Binary trees are special cases of tree where every node has atmost two childrens.

→ Empty binary tree is represented by simply None.

JULY
30
TUESDAY
'24
31st Week
212-154

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

Q. Why do we study binary Tree? Why not Terniary Tree.

Ans The most commonly tree DS are binary search, binary heap, segment tree all these are binary tree, all these tree are atmost 2 children.

Tree Traversal [printing every key of the tree exactly once]

(a) BFT (or level order Traversal)

(b) Depth First
- Inorder
- Pre order
- Post order

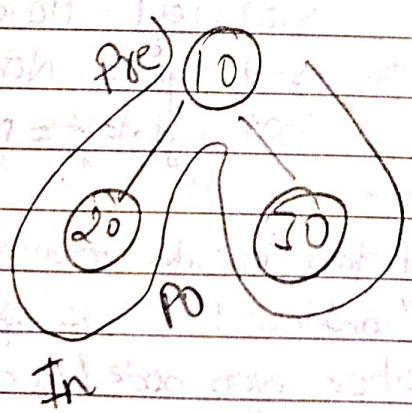
We go to one side and finish it completely. Then we go to other side.

Recursive
- Traverse Root
- Traverse left subtree
- Traverse right subtree

3! ways to print

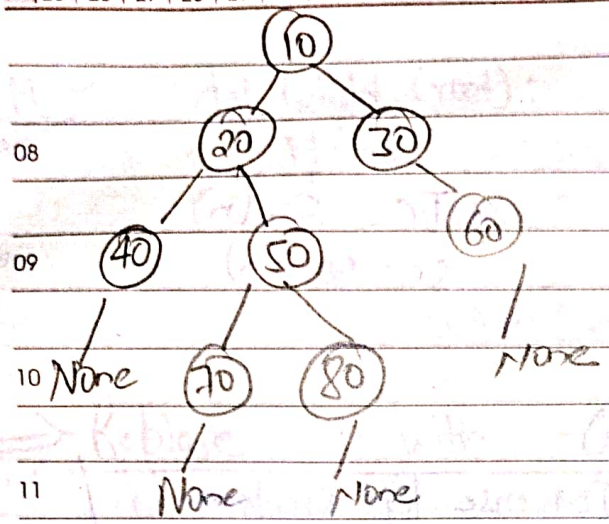
Out of 3! ways, 3 Popular traversal is inorder, Preorder, Postorder

Inorder (Left → Root → Right)
Pre Order (Root → left → Right)
Post order (left → Right → Root)



Inorder : 20 10 30
Pre order : 10 20 30
Post Order : 20 30 10

Try to implement new function inside the Given function of Questions



Inorder : 40, 20, 70, 50, 80, 10, 30, 60
Preorder : 10, 20, 40, 50, 70, 80, 30, 60
Postorder : 40, 70, 80, 50, 20, 60, 30, 10

Post = Not tail recursive
In, Pre = tail recursive

Inorder Traversal :-

```
def Inorder(root):
    if root != None:
        Inorder(root.left)
        Print(root.key)
        Inorder(root.right)
```

```
def Inorder(self, root)
    result = []
    def traverse(node):
        if node:
            traverse(node.left)
            result.append(node.data)
            traverse(node.right)
    traverse(root)
    return result
```

checking tree is empty (or) not.

same for other

TC = O(n)
SC = O(h)

Traverse left subtree
Visit the tree
Traverse right subtree

Preorder Traversal :-

```
def preOrder(root):
    if root :
        Print(root.val)
        Preorder(root.left)
        Preorder(root.right)
```

Visit the root.
Traverse the left subtree.
Traverse the right subtree.

TC = O(n)
SC = O(n) → used to create copy of a tree.

AUGUST
01
THURSDAY
'24
31st Week
214-152

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

AUG 2024

Post order traversal :-

```
def postorder(root):
    l = []
    def traverse(node)
        traverse(node.left)
        traverse(node.right)
        l.append(node.data)
    traverse(root)
    return l
```

TC = O(n)
SC = O(n)

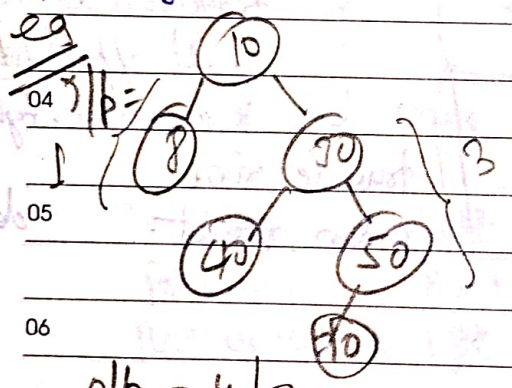
Traverse left subtree
Traverse right subtree
Visit the node

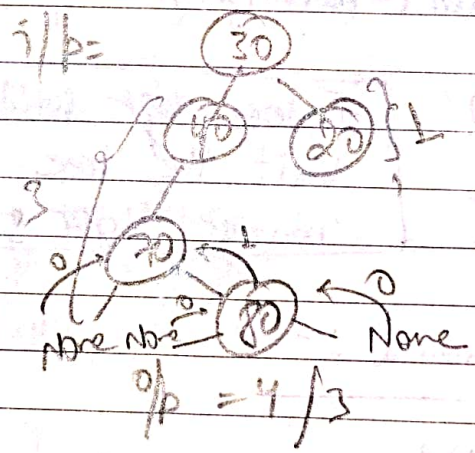
→ used to delete a tree.

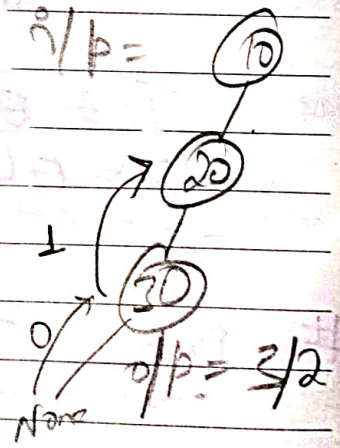
height of binary tree :-

height is maximum No. of node from root to leaf. → Height = h
C-1 // C-2 // longest root to leaf path = height = (h-1)

eg







C-1 // M1

```
def height(root):
    if root == None:
        return 0
    else:
        lh = height(root.left)
        rh = height(root.right)
        return max(lh, rh) + 1
```

C-2 == -1

TC = O(n)
SC = O(n)

Fear of ideas makes us impotent and ineffective

M-2

```
def height(root):
    if root == None:
        return 0
    else:
        return (max(height(root.left), height(root.right))+1
```

⇒ Replace 0 with -1 we will get result according to convention 2.

- Recursively do DFS:—
  - If tree is empty return -1
  - otherwise,
    - get the max depth of the left subtree recursively.
    - " " " " " " right " " "
    - " " " " " " left & right " " " add 1 on it.

Check if trees are identical

```
def isIdentical(root1, root2, self):
    if root1 is None and root2 is None:
        return True
    if root1 is not None and root2 is not None:
        return ((root1.data == root2.data) and
                self.isIdentical(root1.left, root2.left) and
                self.isIdentical(root1.right, root2.right))
```

One liner!—

```
return str(root1) == str(root2)
```

AUGUST
03
'24
31st Week
216-150
SATURDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

AUG 2024

Maximum in Binary Tree.

M-1

```
08 def getmax(root):
   import math
09 if (root == None):
       return = -math.inf  # = infinity from math module
10 else:
       lm = getmax(root.left)
11     rm = getmax(root.right)
       return max(root.key, lm, rm)
12
```

TC = $\theta(h)$
SC = $\theta(h)$

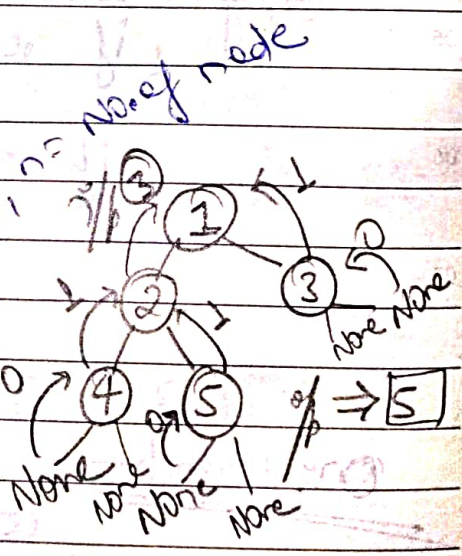
M-2 My way

```
def getmax(root):
    def Traverse(node):
        l = []
02      if node != None:
            return None
03          Traverse(node.left)
            l.append(node.data)
04          Traverse(node.right)
    Traverse(root)
05  return max(l)
```

Calculate size of binary tree:-

```
06 def treesize(root):
07     if root == None:
           return 0
       else:
           ls = treesize(root.left)
           rs = treesize(root.right)
           return (ls + rs + 1)
```

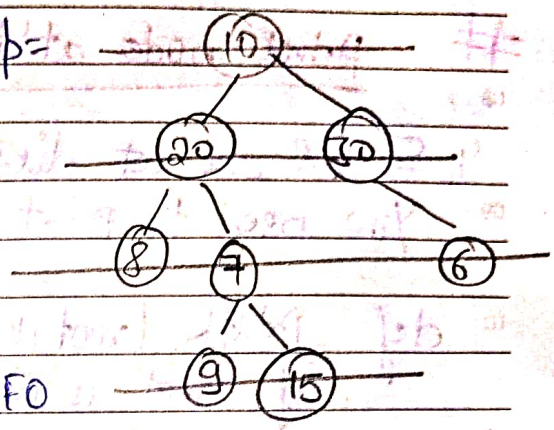
TC = $\theta(n)$
SC = $\theta(n)$



return 1 + treesize(root.left) + treesize(root.right)

Level order traversal

The task is to print the level order traversal of a tree is breadth first traversal for the tree.



i/p =

⟹ For each node, first the node is visited and then it's child nodes are put in a FIFO queue. Then again the first Node is poped out and the it's child nodes are put in a FIFO queue and repeat unitil queu becomes empty.

o/p = 10 20 30 8 7 6 9 15

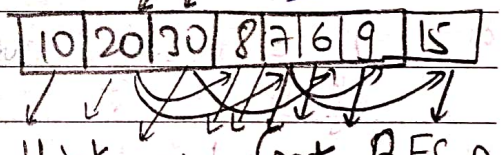
```
from collections import deque
def printLevel(root):
    if root is None:
        return
    q = deque()
    q.append(root)
    while len(q) > 0:
        node = q.popleft()
        print(node.key)
        if node.left is not None:
            q.append(node.left)
        if node.right is not None:
            q.append(node.right)
```

Initially : q = {10}

1st iteration print(10)
q = {20, 30}

2nd iteration print(20)
q = {30, 40, 50}

3rd iteration print(30)
q = {40, 50, 60}



think as Crack BFS problem

{TC = O(n), N = No. of Nodes in binary tree.
SC = O(n), N = " " " " " ".

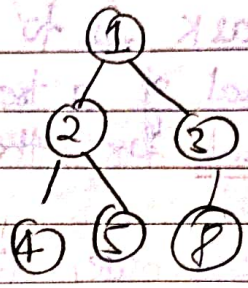
AUGUST
05
'24
32nd Week
218-148
MONDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

AUG 2024

Print nodes at 'K' distances from root.

4, 5 & 8 are at distances 2 from Root.
You need to print the $(k+1)^{th}$ level




```
def PrintK (root, K):
    if root is None:
        return
    if K == 0:
        Print (root.key, end = " ")
    else:
        PrintK( root.left , K-1)
        PrintK ( root.right, K-1)
```

{TC = O(n), SC = O(h)}

Determine binary tree is height balanced or Not?

```
def isBalanced (root):
    if root == None:
        return 0
    lh = isBalanced (root.left)
    if lh == -1:
        return -1
    rh = isBalanced(root.right)
    if rh == -1:
        return -1
    if abs (lh-rh)>1:
        return -1
    return max(lh, rh) +1
```

TC = O(n)

```
def isBalancedMain(root):
    if isBalanced (root) == -1:
        return False
    return True.
```

isBalanced(): return -1 if the tree is balanced else return height.

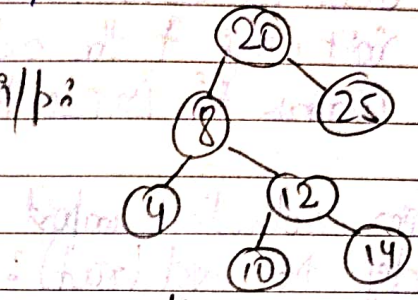
isBalancedMain(): return True if tree is balanced else false.

Level order Traversal line by line :-

08 → Given root of a binary Tree, the task is to Print level order traversal in a way that nodes are printed in separate lines.

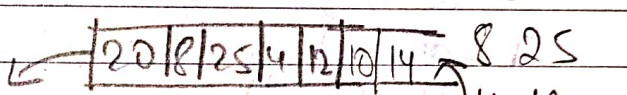
09

→ For each node, first, the node is visited and then its child nodes are put in a FIFO Queue. Then again the first node is popped out and the it's child nodes are put in a FIFO queue and repeat until queue becomes empty.



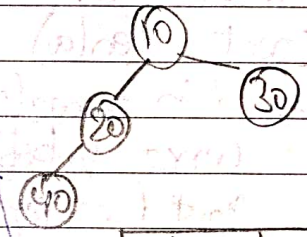
i/p:

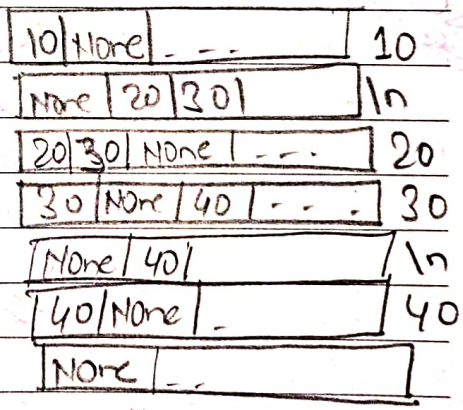
o/p:
20
8 25
4 12
10 14




```
def printLevel(root):
    if root is None:
        return
    q = deque()
    q.append(root)
    q.append(None)
    while len(q) > 1:
        curr = q.popleft()
        if curr == None:
            print()
            q.append(None)
            continue
        print(curr.key, end = " ")
        if curr.left is not None:
            q.append(curr.left)
        if curr.right is not None:
            q.append((curr.right)
```

O(n)
O(width)
+
O(width)





AUGUST
07 '24
32nd Week
220-146
WEDNESDAY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

M2 # Level order traversal line by line:-

→ First insert the root and a null element into the queue. This null element acts as a delimiter. Next, pop from the top of queue and add its left and right nodes to the end of the queue and then print at the top of the queue. Continue this process till the queues become empty.

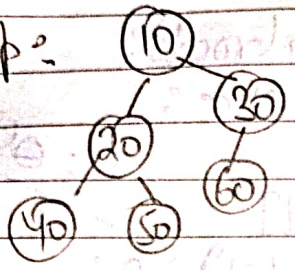
```
from collection import deque
def printlevel(root):
    if root is None:
        return
    q = deque()
    q.append(root)
    while len(q) > 0:
        count = len(q)
        for i in range(count):
            curr = q.popleft()
            print(curr.key, end=" ")
            if curr.left is not None:
                q.append(curr.left)
            if curr.right is not None:
                q.append(curr.right)
        print()
```

TC: O(n)
SC: O(n)

Example is the lesson that all men can read

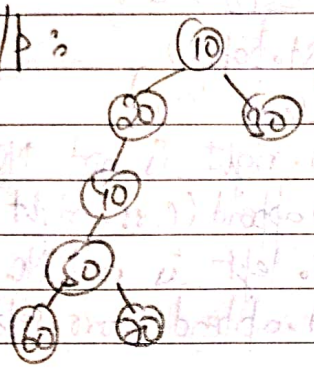
Iterative Inorder Traversal

i/p:



o/p: 40, 20, 50, 10, 60, 30

i/p:



o/p: 60, 50, 70, 40, 20, 10, 30

TC = $\theta(n)$
SC = $\theta(h)$

```
def itrInorder(root):
    if root is None:
        return
    st = []
    curr = root
    while curr is not None:
        st.append(curr)
        curr = curr.left
    while len(st) > 0:
        curr = st.pop()
        print(curr.key)
        curr = curr.right
    while curr is not None:
        st.append(curr)
        curr = curr.left
```

Initially: curr = 10, st = [], st = [10, 20, 40]
1st, curr = 40, st = [10, 20], print(40)
2nd, curr = 20, st = [10], print(20), st = [10, 50]
3rd, curr = 50, st = [10], Print(50), st = [10]
4th, curr = 10, st = [], print(10), st = [30]
5th, curr = 30, st = [], print(30), st = []

Using stack is the obvious way to traverse tree w/o recursion.

1. Create an empty stack S.
2. Initialize current node as root.
3. Push the current node to S and set current = current -> left until current is Null.
4. If current is Null and stack is not empty then
   a. Pop the top item from stack.
   b. Print the popped item, set current = popped_item -> right.
   c. Go to step 3.
5. If current is Null and stack is empty then we are done.

AUGUST
09
FRIDAY
'24
32nd Week
222-144

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Preorder Traversal

Following is a simple stack based iterative process to print Preorder traversal.

(1) Create an empty stack nodeStack and push root node to stack.

(2) Do the following while nodeStack is not empty.
(1) pop an item from the stack & print it.
(2) Push right child of a popped item to stack.
(3) Push left child of a popped item to stack.

M-1

```
def itrPreorder(root):
    if root is None:
        return
    St = [root]
    while len(St) > 0:
        Curr = St.pop()
        Print(Curr.key)
        if Curr.right is not None:
            St.append(Curr.right)
        if Curr.left is not None:
            St.append(Curr.left)
```

TC: $\Theta(n)$

M-2

The idea is to start traversing the tree from the root node, and keep printing the left child while exist and simultaneously Push the right child of every node in an auxillary Stack.
once we reach a null node, pop a right child from the auxiliary stack and repeat the process while the auxillary stack is not empty.

```
def Preorder(root):
    if (root == None):
        return
    St = []
    Curr = root
    while (len(St) or curr != None):
        while (curr != None):
            Print(Curr.data, end = " ")
            if (curr.right != None):
                St.append(Curr.right)
        if (len(St) > 0):
            Curr = St[-1]
            St.pop()
```