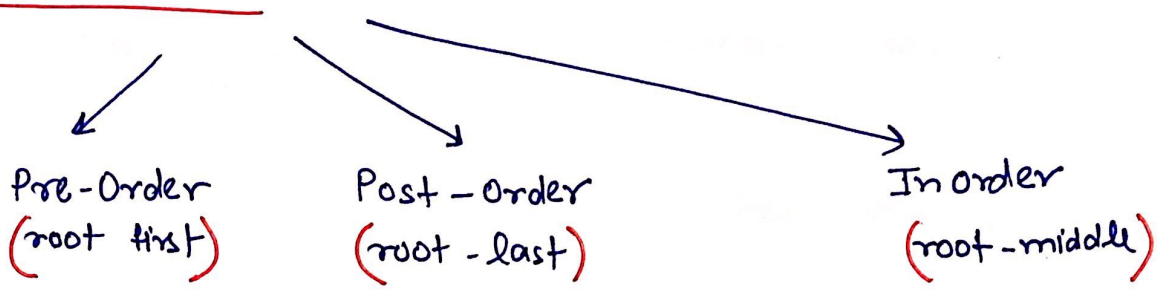
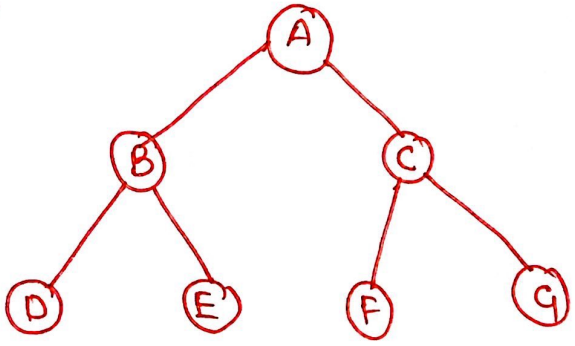
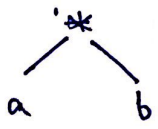
⇒ Tree Traversal :-



eg:-



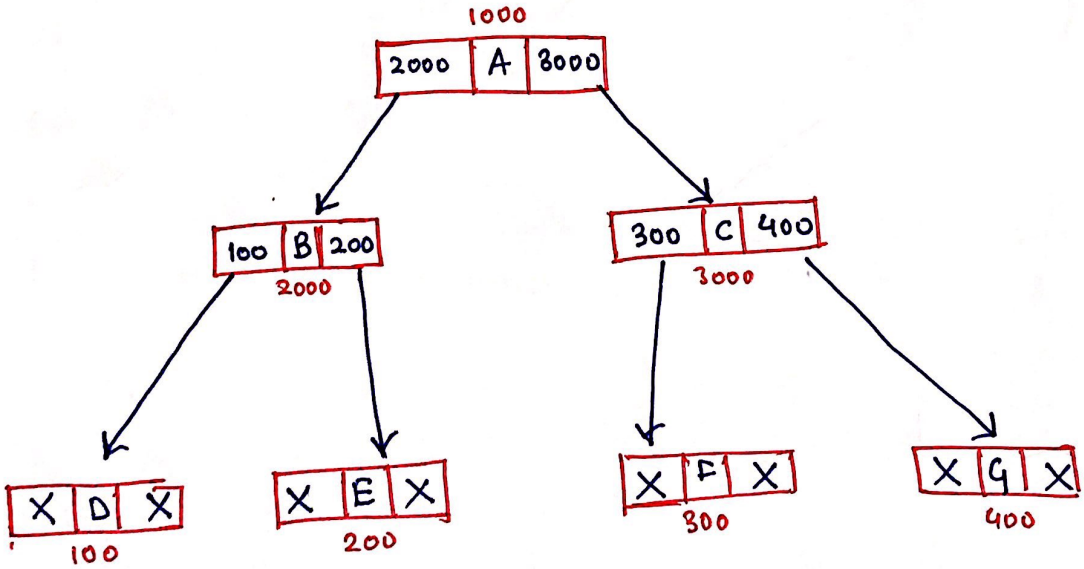


In fix : a * b
Pre-fix : * ab
Post-fix : ab *

Pre-order : A B D E C F G

Post-order : D E B A F G C A

In-order : D B E A F C G



Representation of ~~a~~ tree in memory

```
Pre-order (*root)          ← pointer
{
    print( root → data);       ← c
    pre-order( root → left);   ← T(n/2)
    pre-order( root → right);  ← T(n/2)
}
```

```
Post-order (*root)
{
    post-order( root → left);
    post-order( root → right);
    print( root → data);
}
```

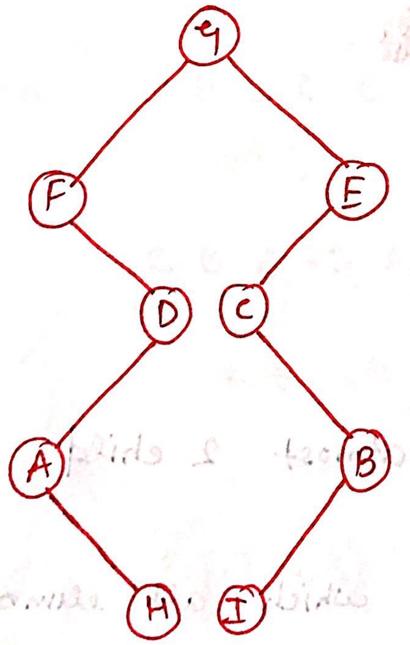
```
In-order( *root)
{
    In-order( root → left);
    print( root → data);
    In-order( root → right);
}
```

$T(n) = 2T(n/2) + c$

$T(n) = O(n)$

NOTE:- Preorder, Postorder, & Inorder all have same time-complexity in each case i.e $O(n)$. (all nodes visited once)

eg:-

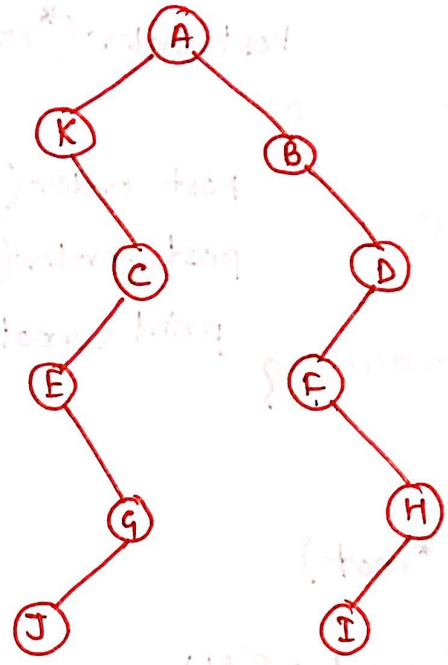


Inorder :- F A H D G C B I B

Preorder :- G F D A H E C B I

Postorder :- H A D F I B C E G

eg:-

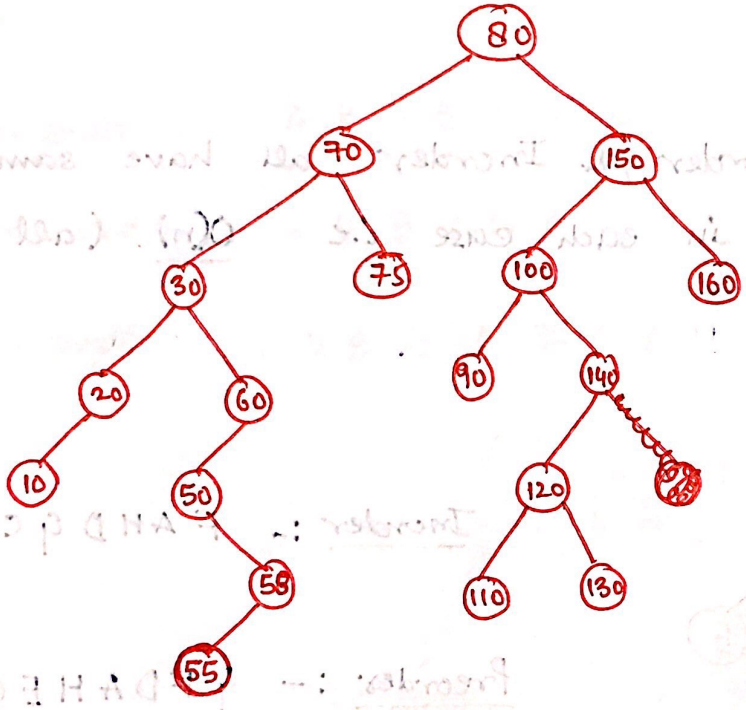


Preorder: A K C E G J B D F H I

Post order: J G E C K I H F D B A

In-order: K E J G C A B F I H D

eg:-



Binary tree :- Tree with atmost 2 child

Binary search tree:- Tree in which all elments present at LHS are smaller than root & RHS are larger than root.

Preorder: 80 70 30 20 10 60 50 58 55 75 150
100 90 140 120 110 130 160

Postorder: 10 20 55 58 50 60 30 75 70 90 110
130 120 140 100 160 150 80

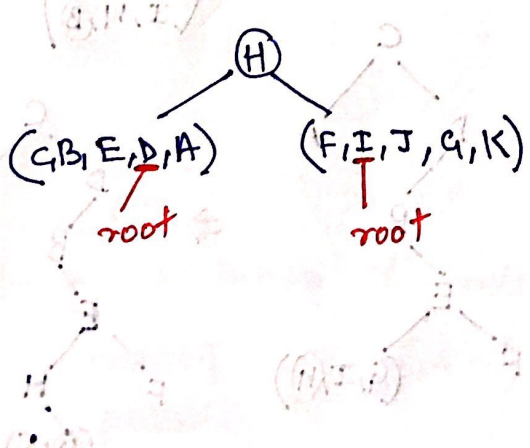
Inorder: 10 20 30 50 55 58 60 70 75 80 90
{Always produce ascending order}
100 110 120 130 140 150 160

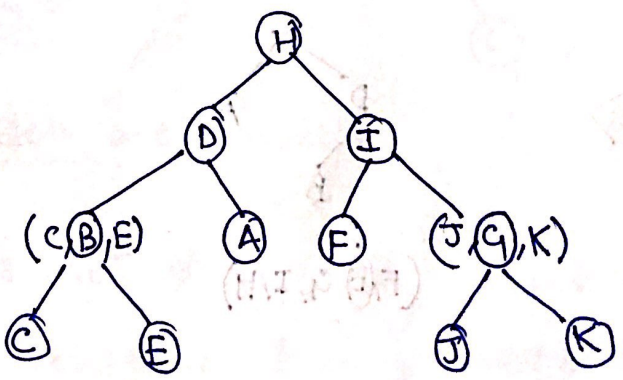
* To sort an ~~array~~ BST in ascending order, apply In-order i.e O(n).

Conversion of Pre-order ↔ Post-order :-

Pre-order :- H D B C E A I F G J K
root

In-order :- (C B E D A) H (F I J G K)
left subtree right subtree





- Pre-order ⟶ root of tree
- In-order ⟶ left and right sub-tree of root

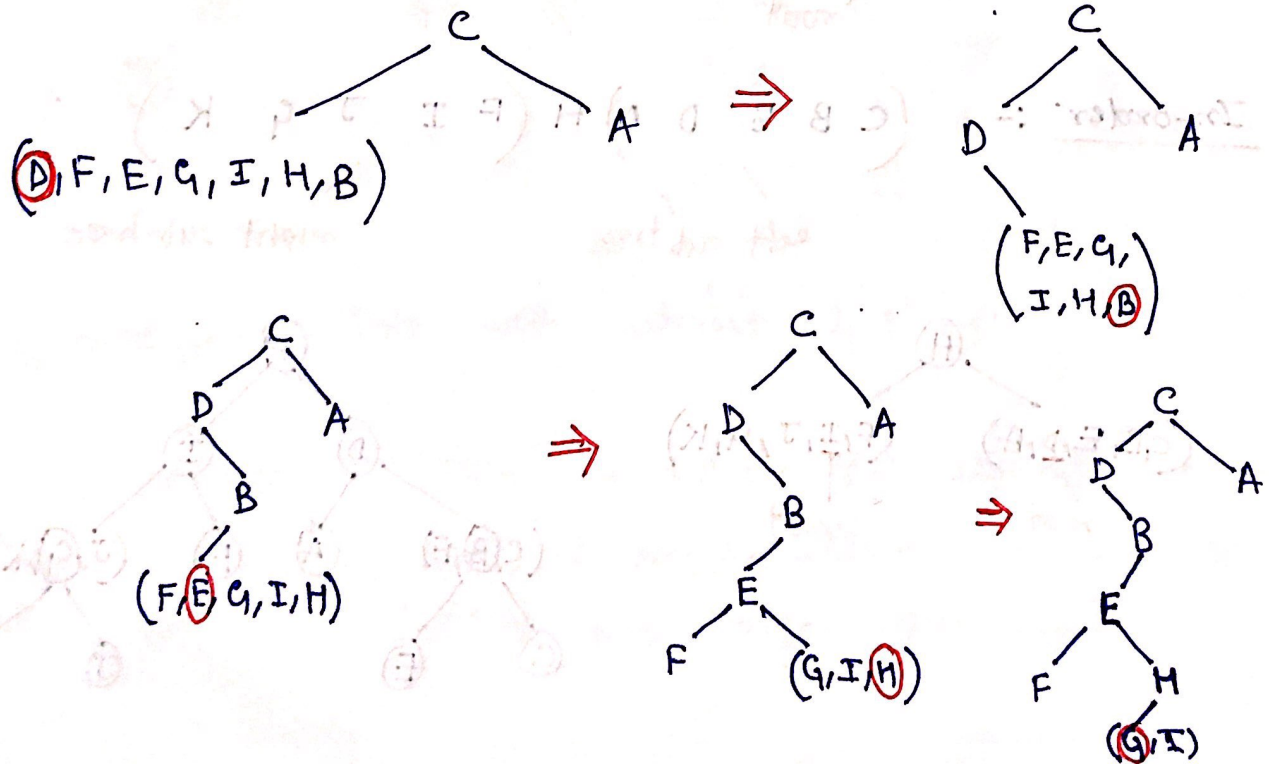
⇒ Time-complexity to build binary tree :-

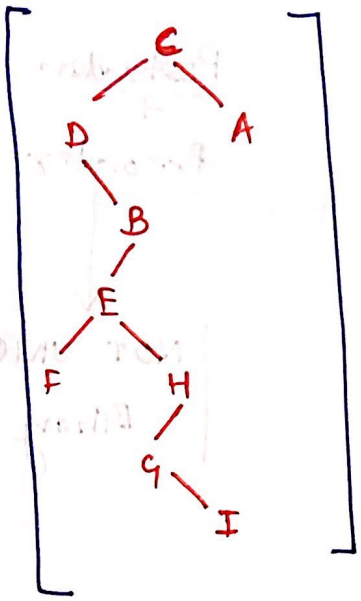
we have to apply linear search for root in inorder

Best	Avg	Worst
O(n)	O(n log n)	O(n²)
root at first position	root in middle	root at last position

eg:- Post-order : F I G H E B D A C (root)

In-order : (D F E G I H B) C (A)





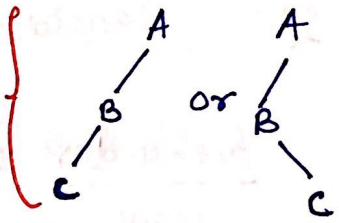
NOTE:- To create binary tree for (Preorder – inorder) or (Postorder – inorder), takes $O(n^2)$

eg:-

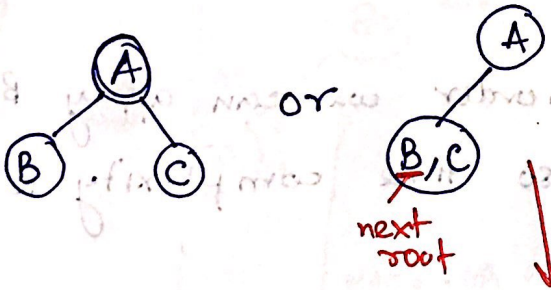
Pre-order :- A B C

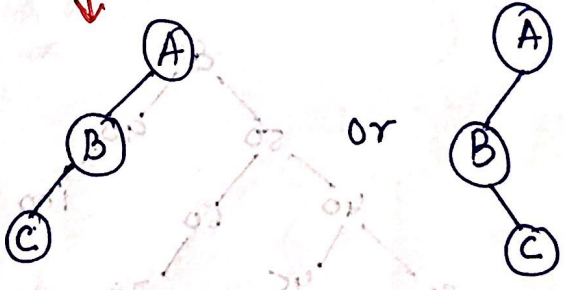
Post-order :- C B A

root



Since we have no inorder so we can't find right & left sub-tree directly, so some cases are generated we have to check for all,





Both are correct

NOTE :- • When in-order is not given (only post & pre order) are given) so multiple Binary trees are possible.

Pre order + In order → [UNIQUE Binary Tree]

Postorder + inorder → [UNIQUE Binary Tree]

Postorder + Pre order → [NOT UNIQUE Binary Tree]

eg:- Consider following BST :

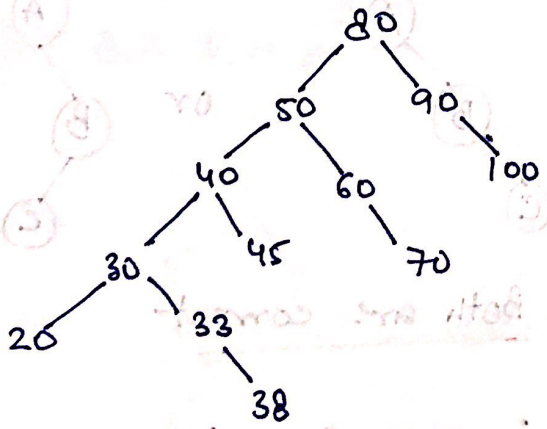
Pre-order : 80 50 40 30 20 33 38 45 60 70 90 100

(80 is root)

Find Post-order)

In-order :- (20 20 33 38 40 45 50 60 70) 80 (90 100)

To search root inside inorder we can apply Binary search as it is BST, so time complexity is O(logn)

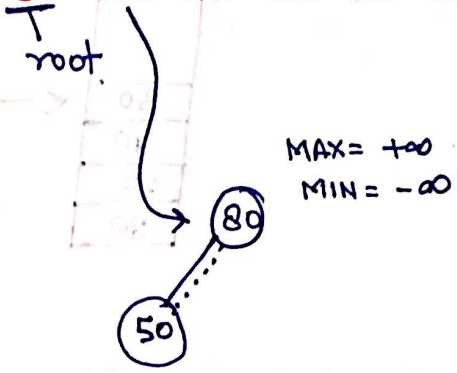


So, for n-vertices take logn for each

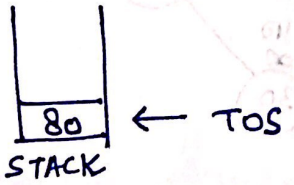
$T(n) = n \log n$

ALTER :-

Pre-order : 80 50 40 30 20 33 38 45 60 70 90 100



PUSH 80



- compare ~~that~~ coming node ' n ' i.e $\{MAX \leq n \leq MIN\}$

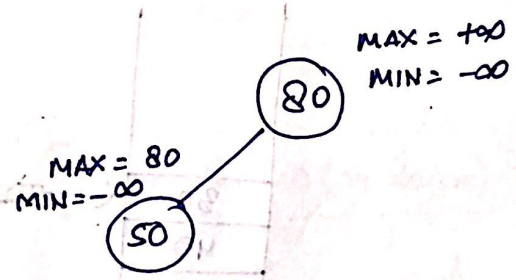
n is in range of root 80.

(if coming node is in range of existing node (TOS) then place it in BST and perform PUSH op^n.)

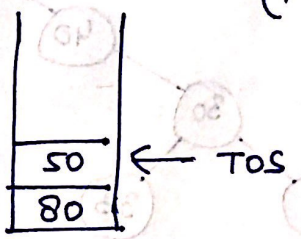
if (n < root) ⟶ Left child

if (n > root) ⟶ right child

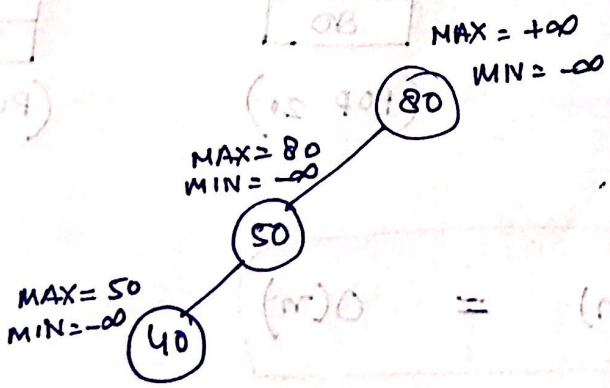
(i) 50,



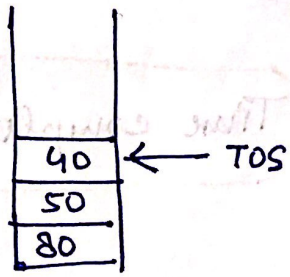
(PUSH 50)



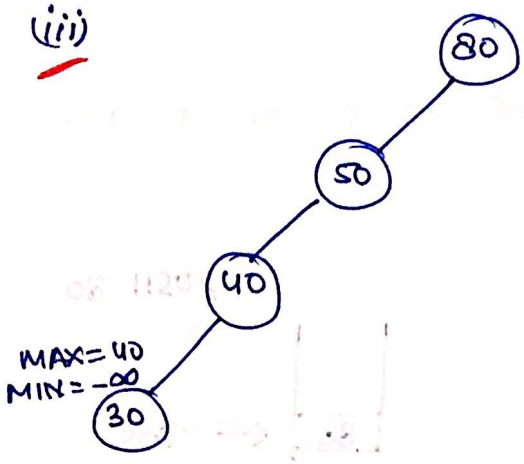
(ii)

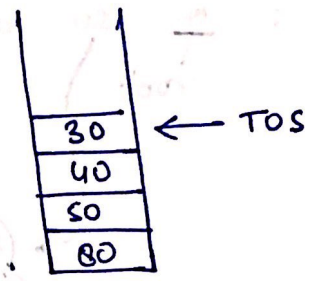


(PUSH 40)



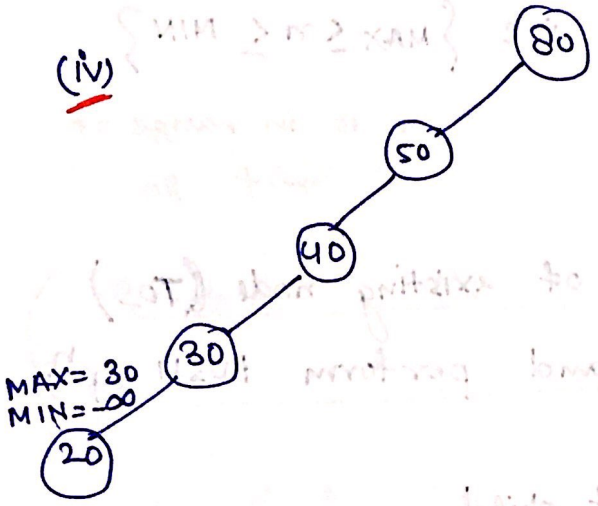
(iii)

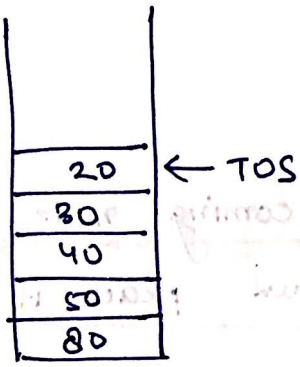




(PUSH 30)

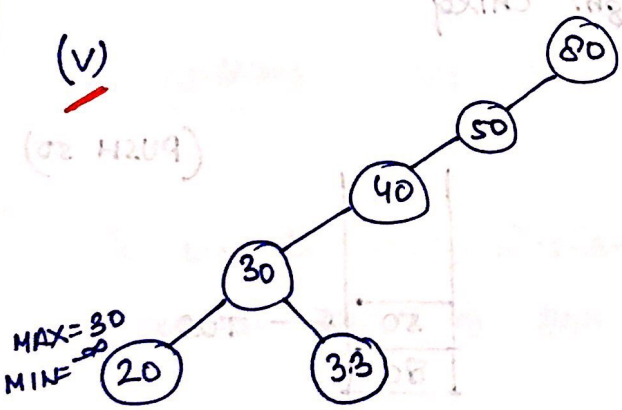
(iv)

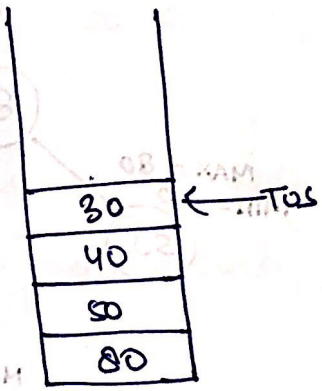




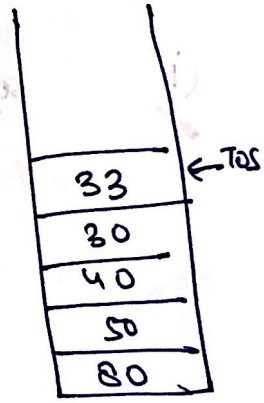
(PUSH 20)

(v)





(POP 20)



(PUSH 33)

Time complexity $T(n) = O(n)$

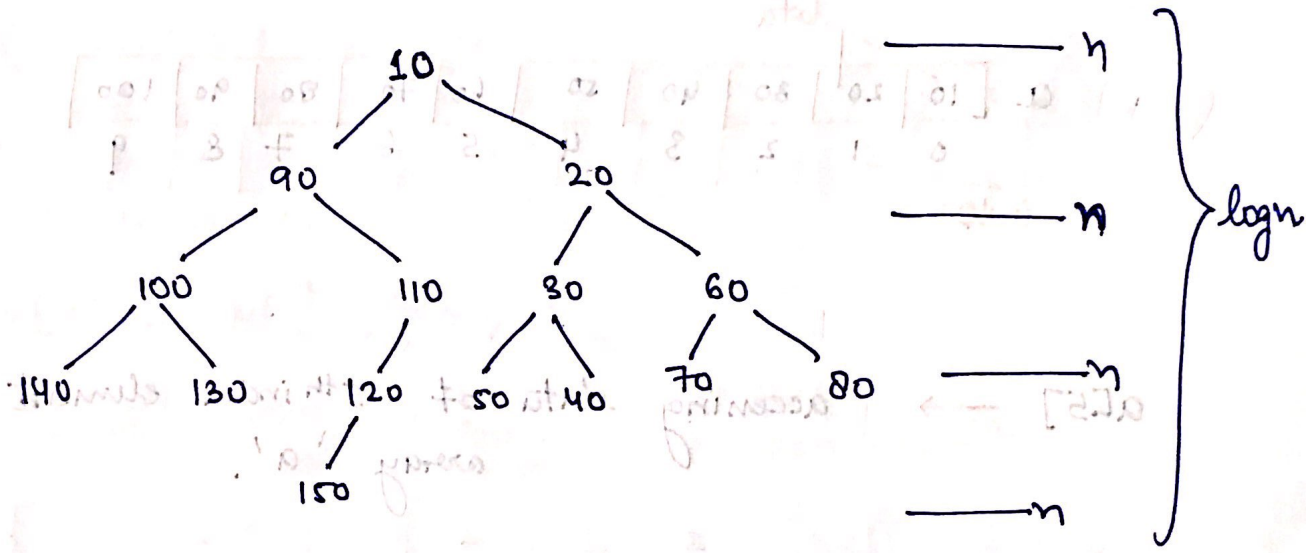
Q. Consider following minheap data:

Inorder: (140 100 130 90 150 120 110) (10) (50 30 40 20 70 60 80)

Find Preorder?

In minheap a root element is the minimum element:

(root = 10)



So, $T(n) \doteq O(n \log n)$