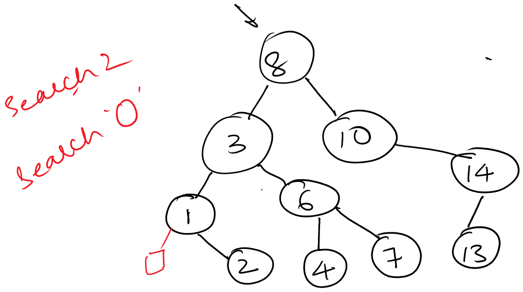
Binary Search Trees

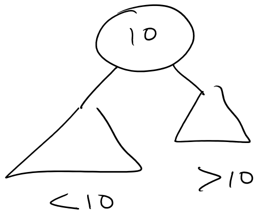
values
left < Node < Right
ST ST

- values in the left ST are always less than the current node
- Values in the right ST are always more than the current node

✱ No identical values



Inorder Traversal



1 2 3 4 6 7 8 10 13 14
Sorted sequence

I Find an Element

II Inserting an Element

III Deleting an Element

Best - Root - O(1)
Worst - O(h)

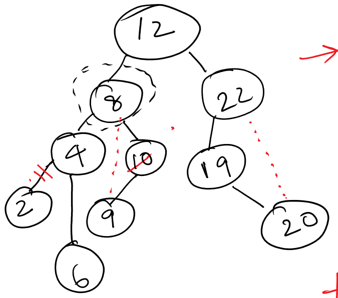
I Find an element

```
bool ItemExists ( Tree tree, int Data)
{
    if(tree==0) return false;

    else if ( Data == tree→data)
                return true;

    else if (Data < tree → data)
        return ( tree → left, Data)
}   else return (tree → right, Data)
```

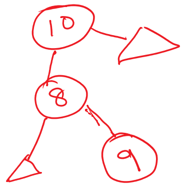
II Insert an Element

```
void InsertItem( Tree tree, int Data)
{
    if (tree == 0)
            tree = MakeNode (Data);
    else if (Data < tree → data)
            InsertItem( tree → left, Data)
    else if (Data > tree → data)
            InsertItem (tree → right, Data)
}   else
            Error Duplicates
```

#g 12, 22, 8, 19, 10, 9, 20, 4, 2, 6



→



#1 Deleting 2

#2 Deleting 10

#3 Deleting 19

#4 Deleting 12 2 4 6 8 9 10 $\boxed{12}$ 19 20 22

Replace 12 by 10, Delete 10

III Deleting An Element – Node to be deleted

1. Leaf Node – No children

1 child
2. Has a left child. (no right child)
3. Has a right child (no left child)

4. Has both left & right child.

1s. Set parent's pointer child to NULL
Release the memory of the leaf node

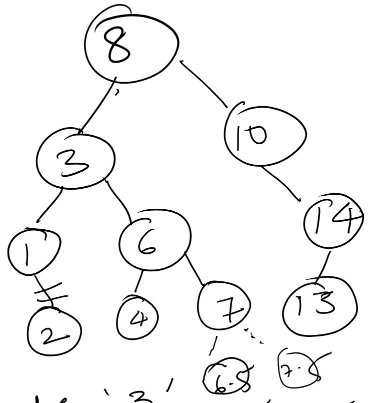
2s. Replace the deleted node with its left child

3s. Replace the deleted node with its

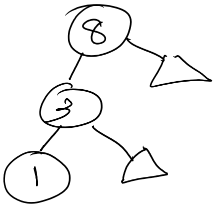
right child

4s. Replace the ~~deleted~~ deleted node with its <u>predecessor</u> in the <u>INORDER</u> traversal. Delete the node that holds the predecessor

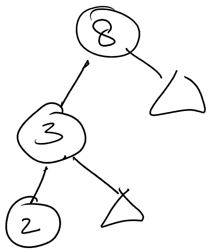
g.



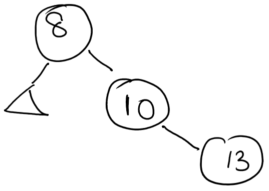
1. Delete '2'



2. Delete '1'



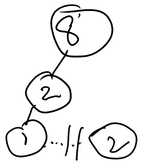
3. Delete '14'



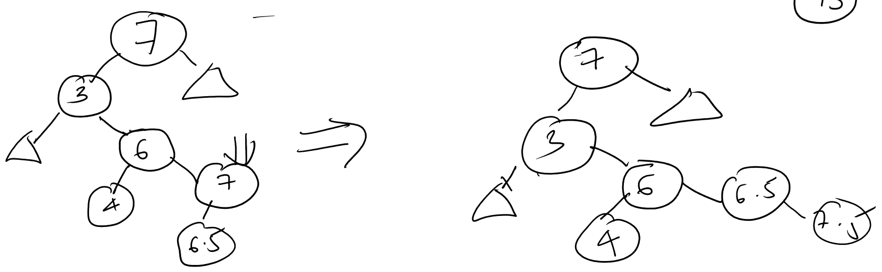
4. Delete '3'

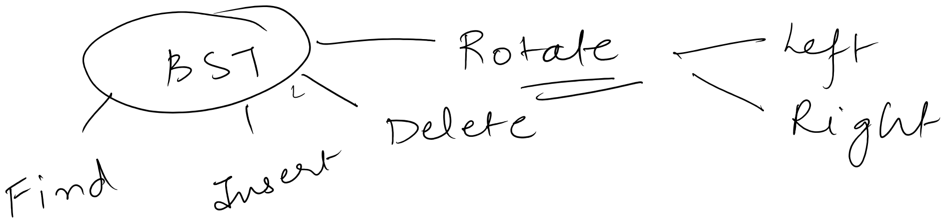
6.5 7.5

Inorder - 1 2 [3] 4 6 | 7 | 8 10 13 14

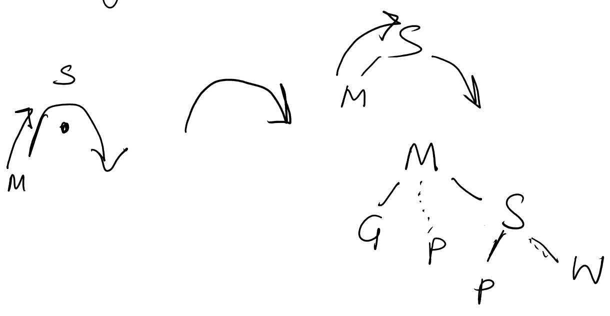


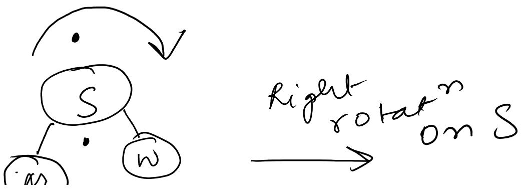
5. Delete '8'



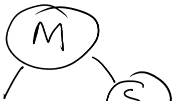


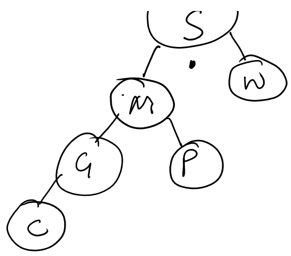
I Right Rotation





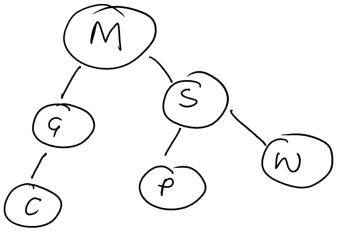
Right rotation on S



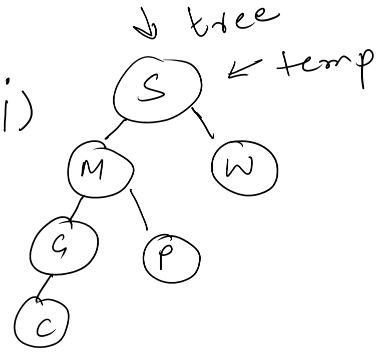


Right rotatⁿ on S $\longrightarrow$

Promote node M



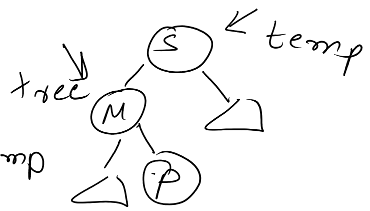
i)



tree = S

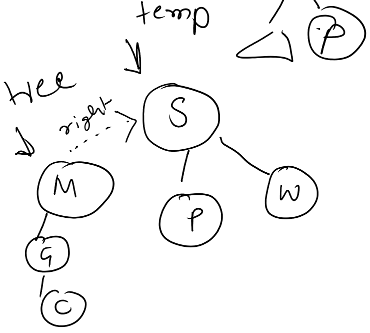
temp = S

ii) tree = tree → left

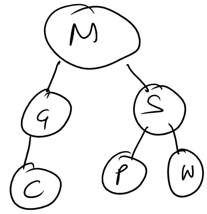


III temp → left = tree → right

make left of S ⇒ P

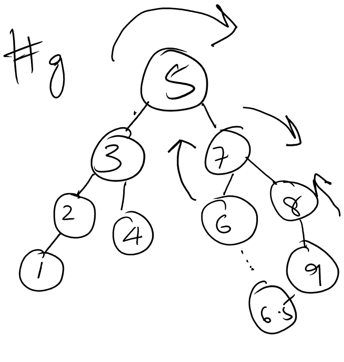


IV tree → right = temp



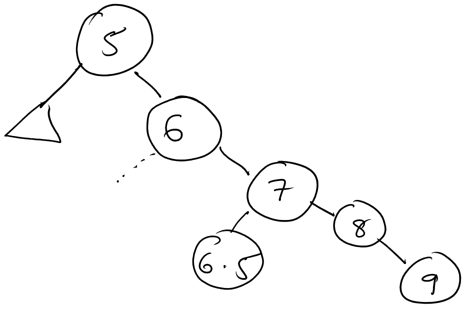
Promoting a node - Same as rotating the parent node, Makes the posⁿ of node as the parent's position.

Eg



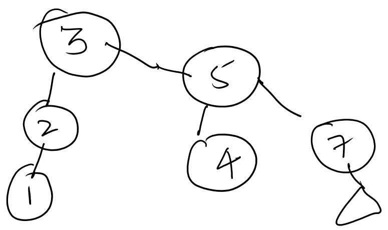
Promote 6 ⇒ make it at posⁿ 7

Right rotatⁿ 7



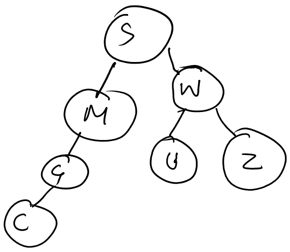
Right rotatⁿ at node 5

Right rotat^n at node 5



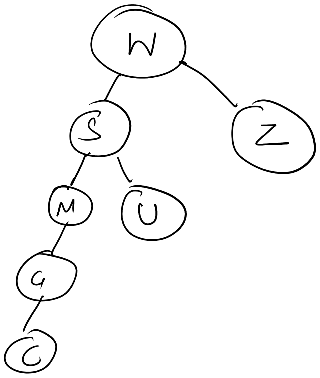
= Promote 3

II Left Rotation

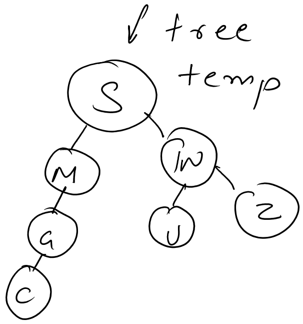


Left rotat^n on S →

Promote W

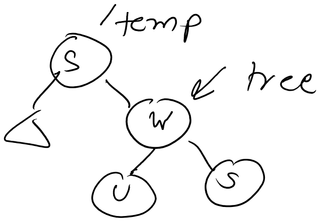


I

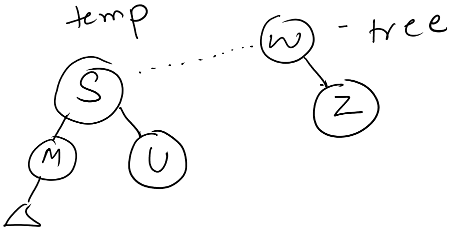


tree = temp = S

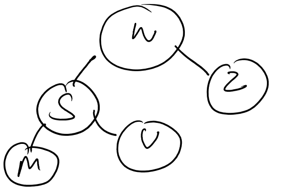
II tree = tree→right



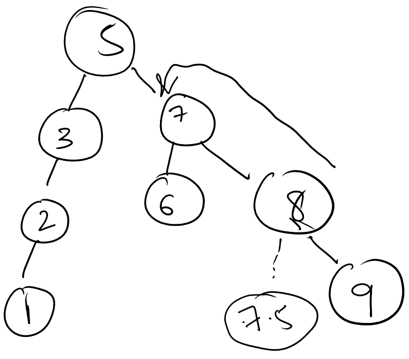
III temp→ right = tree→ left



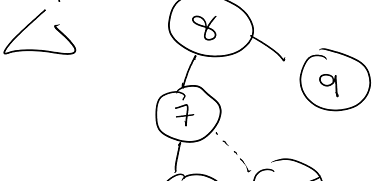
IV tree→left=temp



#eg



Promote 8 = Left Rotation at 7



7.5
7
6
7.5