PW SKILLS

Binary Search Trees

Recap

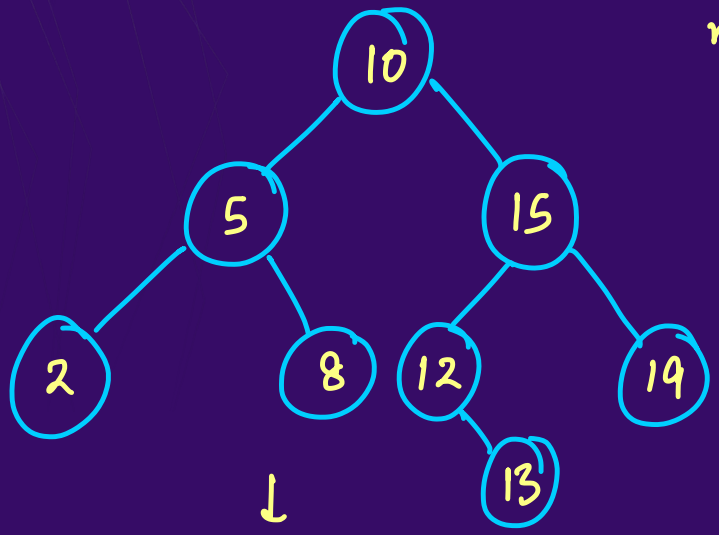
- Trees
- Interview Problems on Trees

Today's Checklist

PW SKILLS

- Why Binary Search Tree?
- What is Binary Search Tree?
- Advantages
- Disadvantages
- Applications
- Insertion
- Traversal - Inorder, Preorder, Postorder
- Searching
- Practice problems on BST

What and Why?

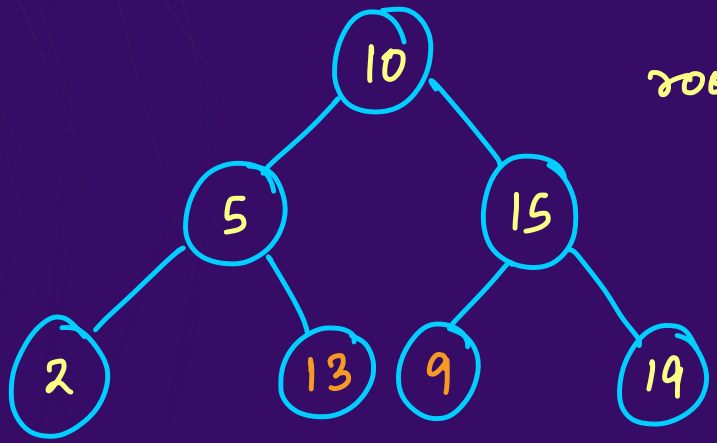


this is a VALID BST

min(RST) > node.val > max(LST)

for every 'node' in the tree

What and Why?



this is NOT a BST

Common Mistake

root.left.val < root.val < root.right.val

↓↓

wrong definition

Ques:

PW SKILLS

Q : Can a BST contain duplicate elements?

min(RST) > root.val > max(LST)

In general, a BST contains unique elements

But if it given that BST can contain duplicate, then probably see the definition of BST in question

Homework:

Q : Find the minimum and maximum elements in a given BST

↳ without traversing the entire tree

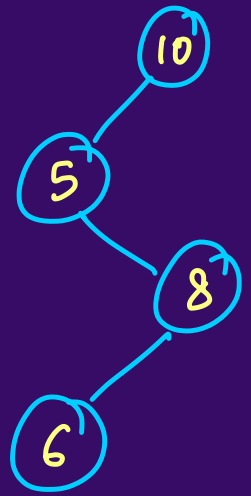
Normal BT → max, min

Advantages

- Efficient Searching
- Efficient insertion and deletion
- Usage in implementation of other data structures like sets, maps, priority queues etc.

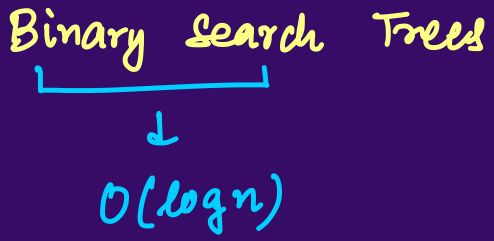
Disadvantages

- Lack of support for range queries
- Not that efficient in case of Unbalanced Trees



Applications

- Phonebook
- Dictionary
- Stock market analysis

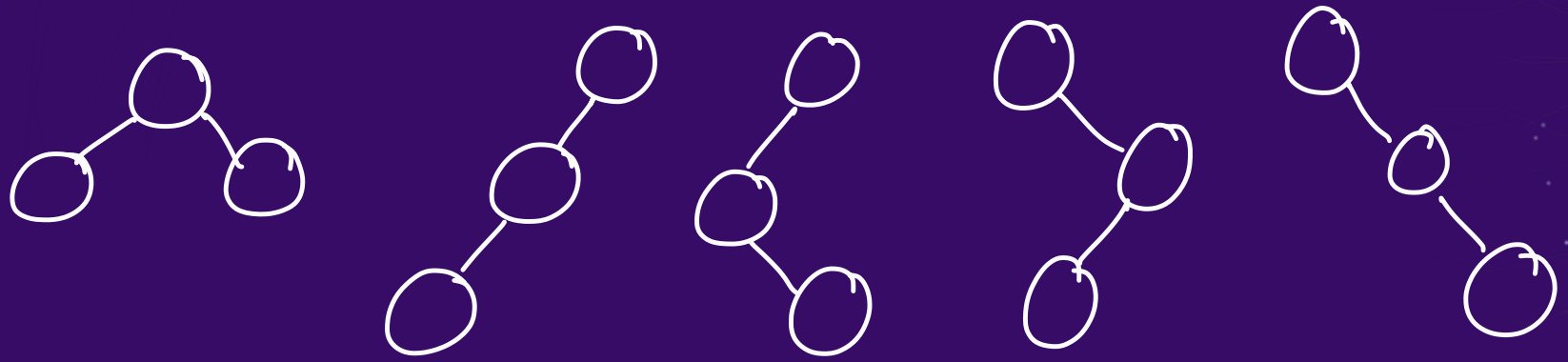


Concept Builder

PW SKILLS

- You have 3 nodes with different values. How many unique Binary Trees can be formed?

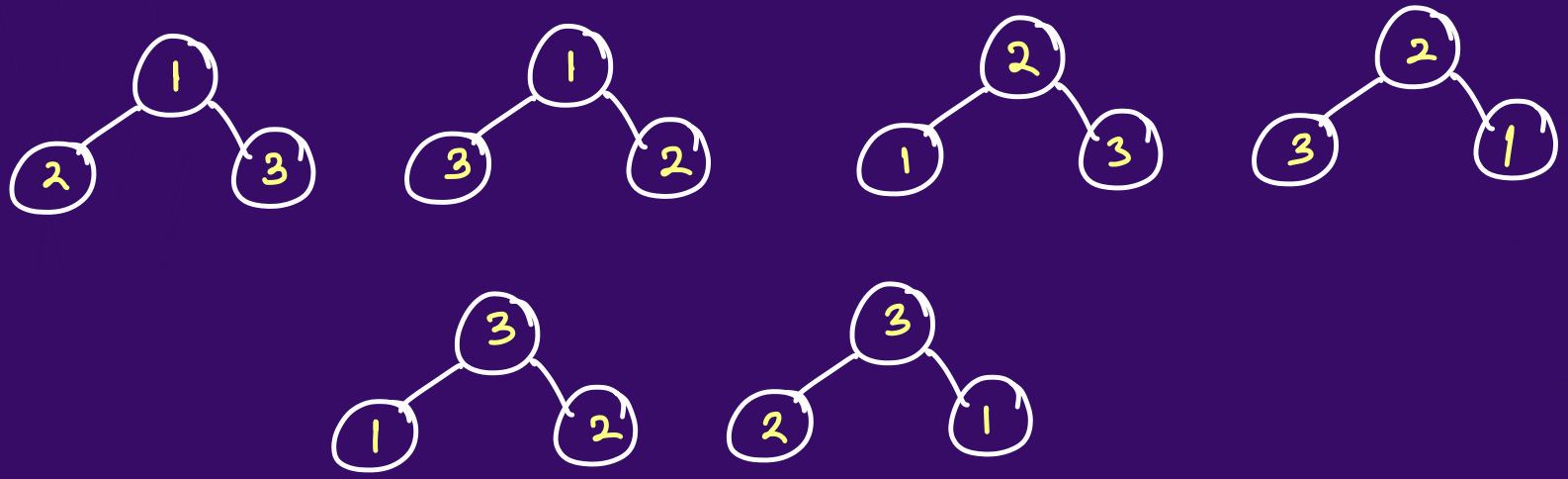
→ 1, 2, 3



$5 \times 3! = 30$

Concept Builder

- **You have 3 nodes with different values. How many unique Binary Trees can be formed?**

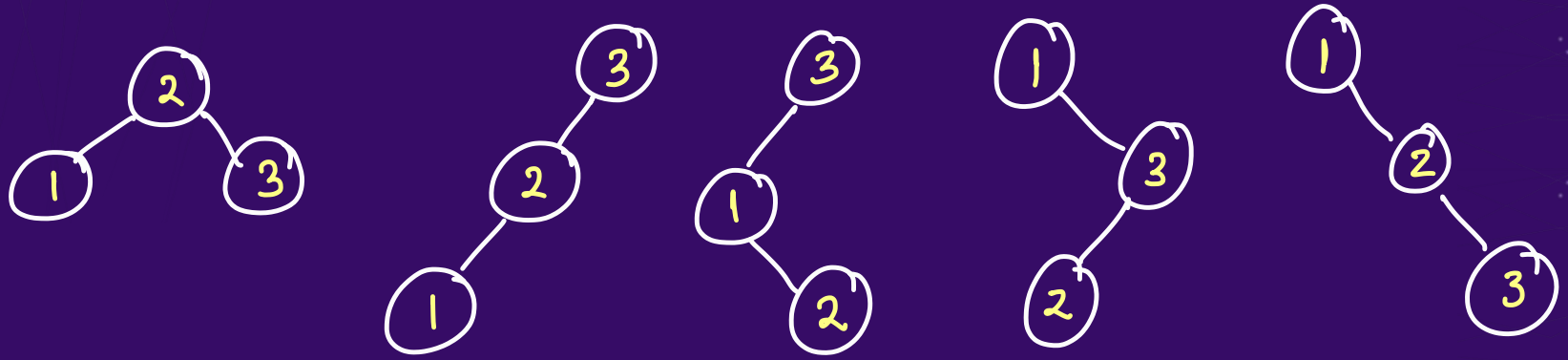


For same structure we have 6 diff Binary Trees

$\Rightarrow 3!$

Concept Builder

- You have 3 nodes with different values. How many unique Binary Search Trees can be formed?



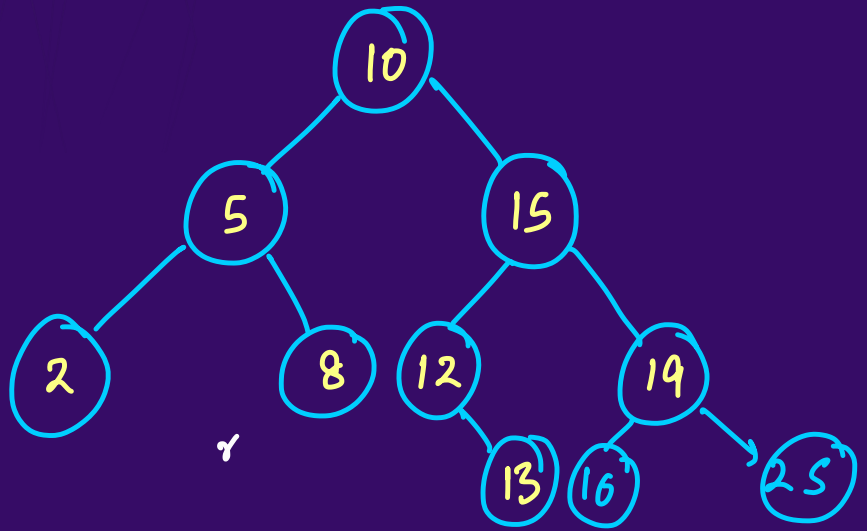
No. of B.S.T. = no. of unique structures

Ques:

PW SKILLS

Q : Search in a Binary Search Tree

- If we search the entire tree, then T.C. is O(n)
- In BST, we do not need to search the entire tree



target = 6

root.val < target
root.val > target
root.val == target

[Leetcode 700]

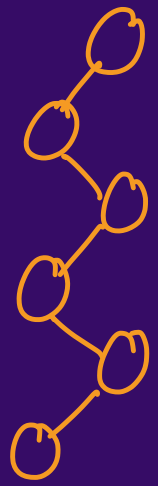
Ques:

PW SKILLS

Q : Search in a Binary Search Tree Time & Space Complexity

Best Case : $O(\log n)$ / $O(h)$

[Balanced Binary Tree]



Avg. & Worst Case : $O(h)$ → height/levels of a Tree

↓

$O(n)$

[Leetcode 700]

Homework:

PW SKILLS

Q : Range sum of BST

↓

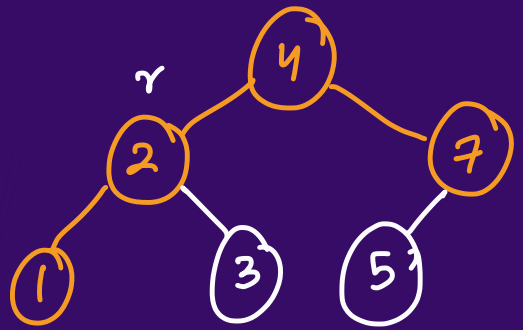
do this using the property of BST

[Leetcode 938]

Ques:

PW SKILLS

Q : Insert into a Binary Search Tree



val = 3

val < root.val

val > root.val

T.C. = O(h)

S.C. = O(h)

[Leetcode 701]

```java
public TreeNode insertIntoBST(TreeNode root, int val) {
    if(root==null) return new TreeNode(val);
    if(val<root.val) root.left = insertIntoBST(root.left,val);
    else root.right = insertIntoBST(root.right,val);
    return root;
}
```

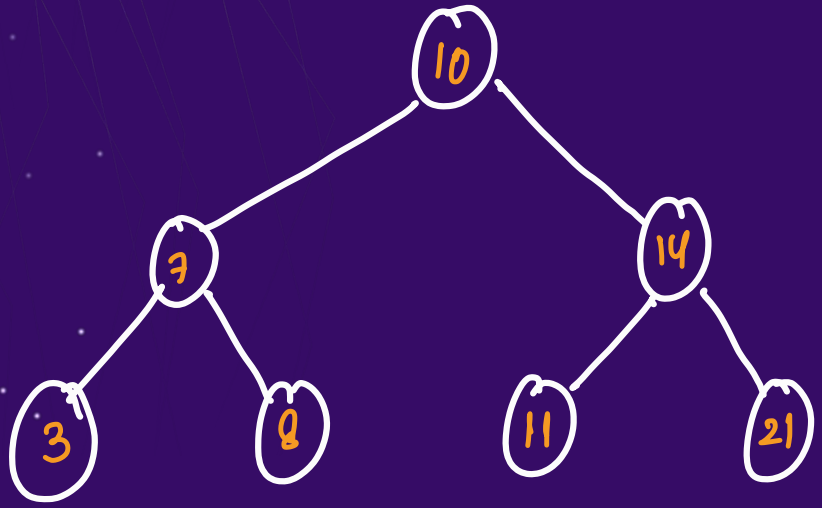
OR

```java
public void insert(TreeNode root, int val) {
    if(val<root.val){ // attach to left
        if(root.left==null) root.left = new TreeNode(val);
        else insert(root.left,val);
    }
    else{ // val>root.val  -> attach to right
        if(root.right==null) root.right = new TreeNode(val);
        else insert(root.right,val);
    }
}
public TreeNode insertIntoBST(TreeNode root, int val) {
    if(root==null) return new TreeNode(val);
    insert(root,val);
    return root;
}
```

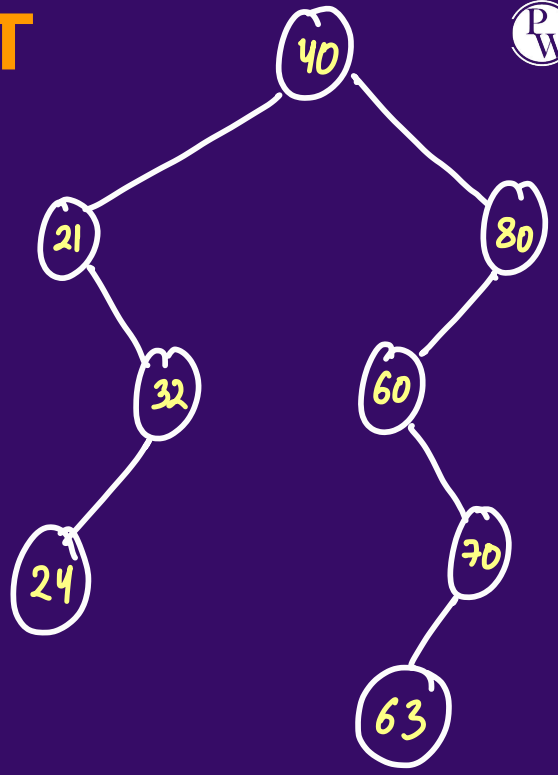
[Leetcode 701]

→ left root right

Inorder Traversal in BST

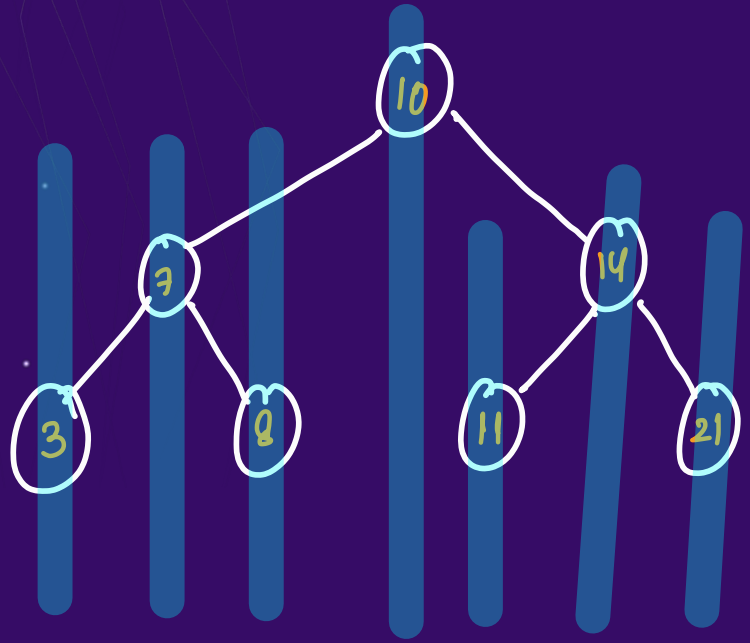


In : 3 7 8 10 11 14 21



In : 21 24 32 40 60 63 70 80

Inorder Traversal in BST is SORTED



In: 3 7 8 10 11 14 21

Sorted hai

in: 21 24 32 40 60 63 70 80

Homework:

Q : kth Smallest element in a BST

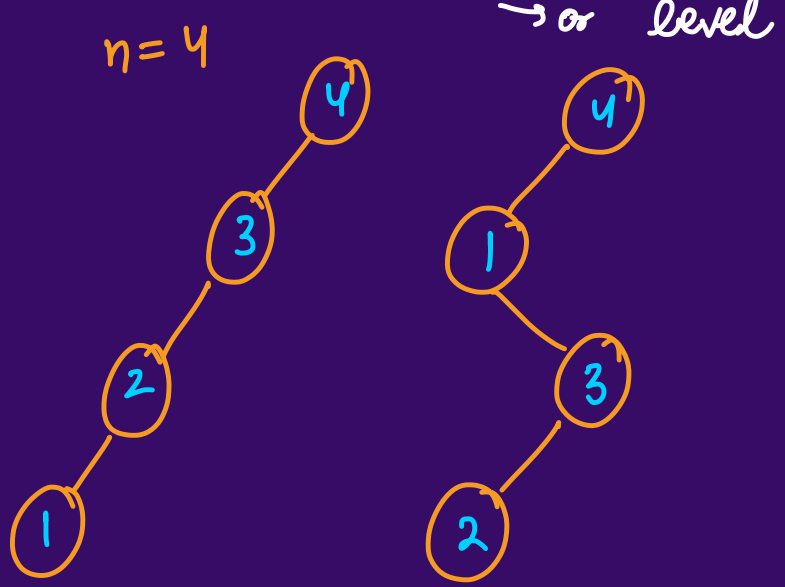
[Leetcode 230]

MCQ-1

PW SKILLS

Consider a binary search tree with n nodes.
What is the maximum possible height of the tree?

A. O(n)
B. O(log n)
C. O(n log n)
D. O(sqrt(n))



MCQ-2

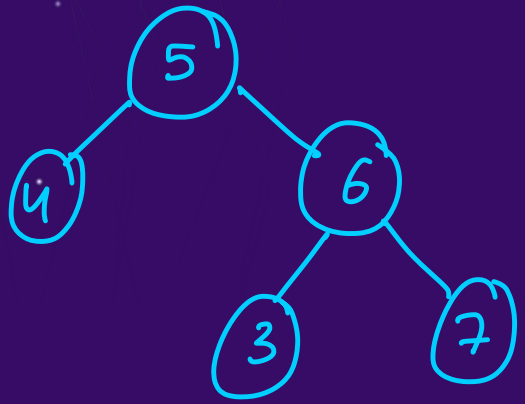
PW SKILLS

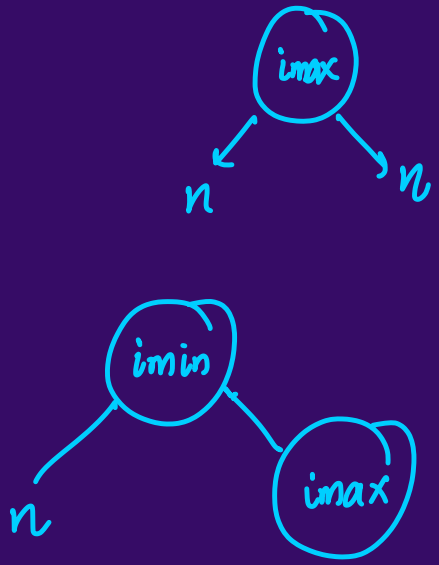
Consider a binary search tree with n nodes. What is the minimum number of comparisons required to search for a value in the worst-case scenario?

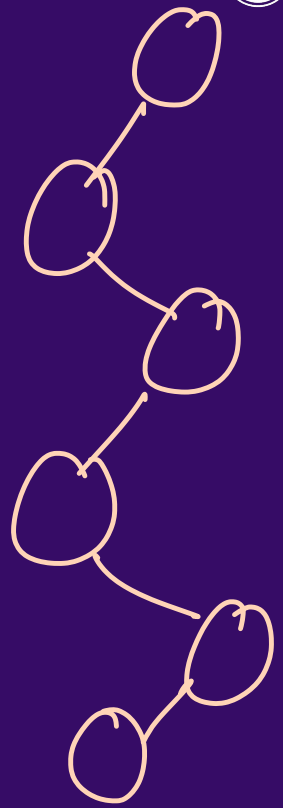
A. O(1)
B. O(log n)
C. O(n log n)
D. O(n)

Ques:

Q : Validate Binary Search Tree







$n-1 + n-2 + n-3 \cdots 1$ **[Leetcode 98]**

$= n(n-1)/2 \rightarrow_n n^2/2$

Ques:

Q : Validate Binary Search Tree

```
private long max(TreeNode root){
    if(root==null) return Long.MIN_VALUE;
    long a = root.val, b = max(root.left), c = max(root.right);
    return Math.max(a,Math.max(b,c));
}
private long min(TreeNode root){
    if(root==null) return Long.MAX_VALUE;
    long a = root.val, b = min(root.left), c = min(root.right);
    return Math.min(a,Math.min(b,c));
}
public boolean isValidBST(TreeNode root) {
    if(root==null) return true;
    if(root.val<=max(root.left)) return false;
    if(root.val>=min(root.right)) return false;
    return isValidBST(root.left) && isValidBST(root.right);
}
```

Best Case

T.C. = O(n·logn)

T.C. = O(n²) [Worst Case]

[Leetcode 98]

Ques: → Homework : Without using this arraylist use a temporary node 'prev'

Q : Validate Binary Search Tree

Method-2 : Using inorder traversal

```java
public void inorder(TreeNode root, List<Integer> arr) {
    if(root==null) return;
    inorder(root.left,arr);
    arr.add(root.val);
    inorder(root.right,arr);
}
public boolean isValidBST(TreeNode root) {
    List<Integer> arr = new ArrayList<>();
    inorder(root,arr);
    for(int i=1;i<arr.size();i++){
        if(arr.get(i)<=arr.get(i-1)) return false;
    }
    return true;
}
```

T.C. = O(n)

S.C. = O(n)

[Leetcode 98]

Ques:

Q : Validate Binary Search Tr

Method : 3 [Similar to M1]

Global variable flag = true

```
static boolean flag;
private long max(TreeNode root){
    if(root==null) return Long.MIN_VALUE;
    long leftMax = max(root.left);
    if(leftMax>=root.val) flag = false;
    long rightMax = max(root.right);
    return Math.max(root.val,Math.max(leftMax,rightMax));
}
private long min(TreeNode root){
    if(root==null) return Long.MAX_VALUE;
    long leftMin = min(root.left);
    long rightMin = min(root.right);
    if(rightMin<=root.val) flag = false;
    return Math.min(root.val,Math.min(leftMin,rightMin));
}
public boolean isValidBST(TreeNode root) {
    flag = true;
    max(root);
    min(root);
    return flag;
}
```

[Leetcode 98]

Ques:

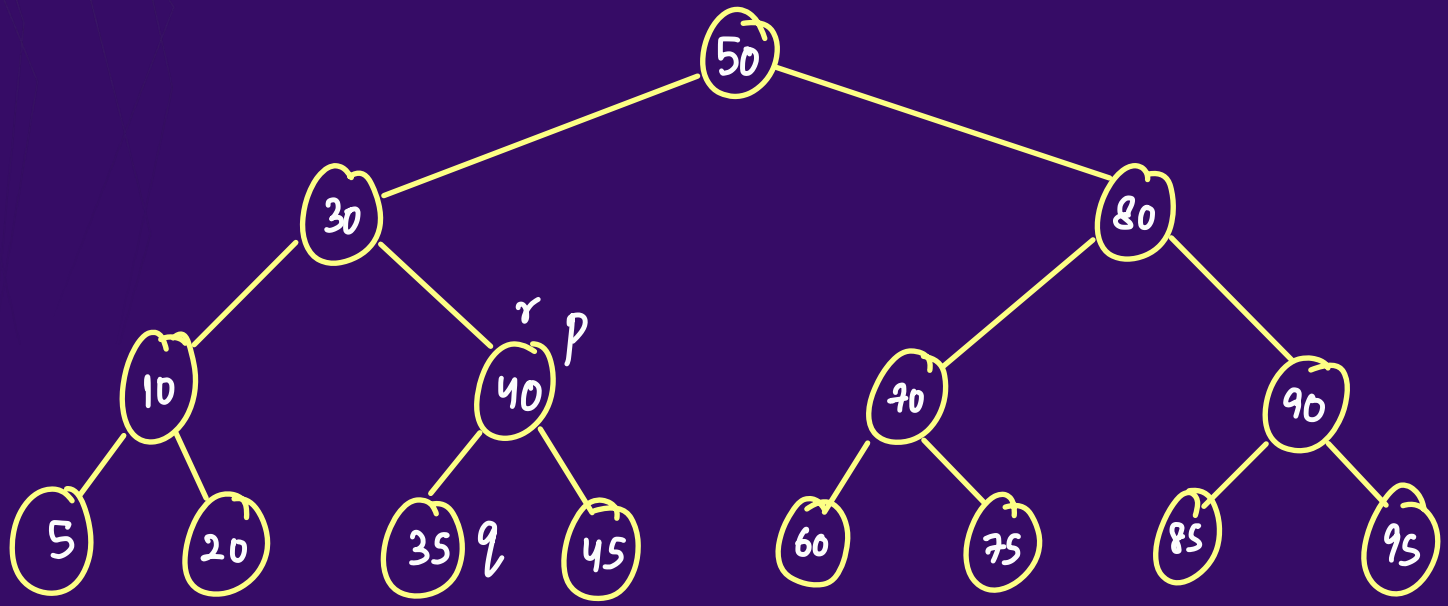
$p < r < q$ → r → LCA

$p < r, q < r$ → $r \rightarrow r.left$

$p > r, q > r$ → $r \rightarrow r.right$

PW SKILLS

Q : Lowest common Ancestor of a Binary Search Tree



[Leetcode 235]

Ques:

PW SKILLS

Q : Lowest common Ancestor of a Binary Search Tree

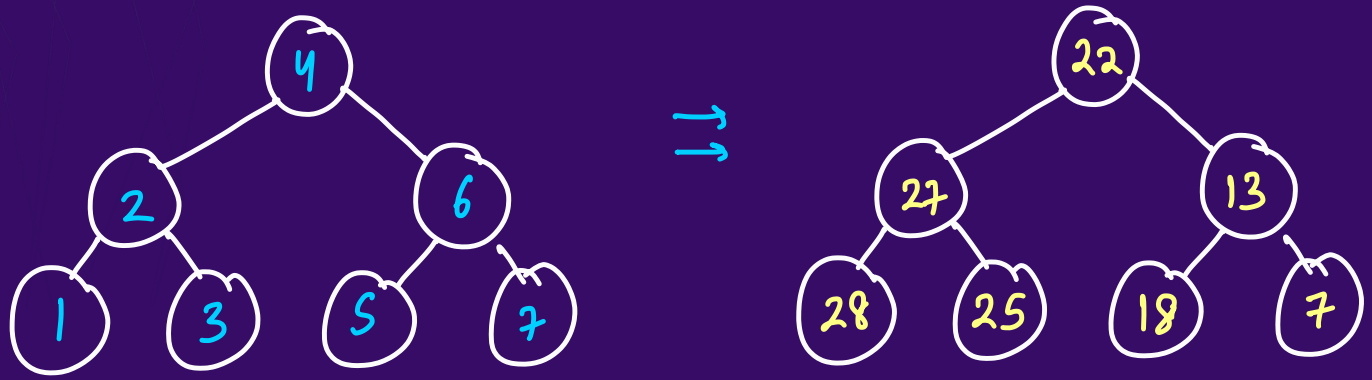
```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if(root.val<p.val && root.val<q.val) return lowestCommonAncestor(root.right,p,q);
    else if(root.val>p.val && root.val>q.val) return lowestCommonAncestor(root.left,p,q);
    else return root;
}
```

[Leetcode 235]

Ques:

PW SKILLS

Q : Binary Search Tree to Greater Sum Tree



List < TreeNode > arr = { 28, 27, 25, 22, 18, 13, 7 }

[Leetcode 1038]

Ques:

PW SKILLS

Q : Binary Search Tree to Greater Sum Tree

M-I :

```
public void inorder(TreeNode root, List<TreeNode> arr) {
    if(root==null) return;
    inorder(root.left,arr);
    arr.add(root);
    inorder(root.right,arr);
}
public TreeNode bstToGst(TreeNode root) {
    List<TreeNode> arr = new ArrayList<>();
    inorder(root,arr);
    int n = arr.size();
    for(int i=n-2;i>=0;i--){
        arr.get(i).val += arr.get(i+1).val;
    }
    return root;
}
```

T.C. = O(n)

S.C. = O(n)

We have used an Extra Array

[Leetcode 1038]

Ques:

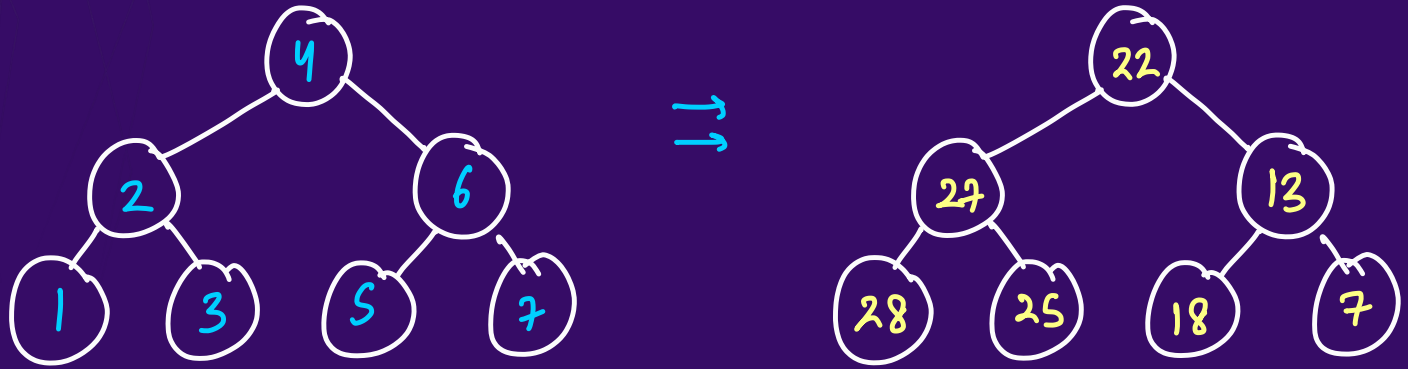
Inorder : Left Root Right

Reverse : Right Root left

PW SKILLS

Q : Binary Search Tree to Greater Sum Tree

M-2



Sum = 0 7 13 18 22 25 27 28

root.val += sum

Sum = root.val

[Leetcode 1038]

Ques:

PW SKILLS

Q : Binary Search Tree to Greater Sum Tree

```
static int sum;
public void reverseInorder(TreeNode root) {
    if(root==null) return;
    reverseInorder(root.right);
    root.val += sum;
    sum = root.val;
    reverseInorder(root.left);
}
public TreeNode bstToGst(TreeNode root) {
    sum = 0;
    reverseInorder(root);
    return root;
}
```

T.C. = O(n)

S.C. = O(h)

↓

h is levels/height of tree

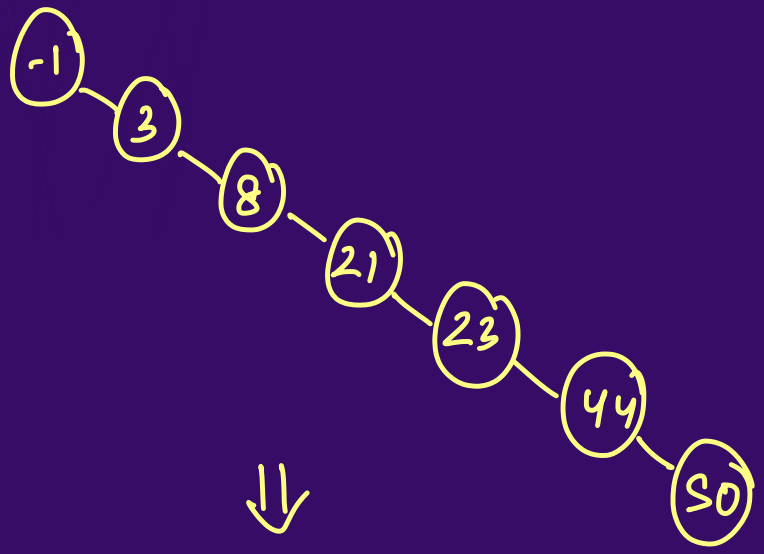
[Leetcode 1038]

Ques:

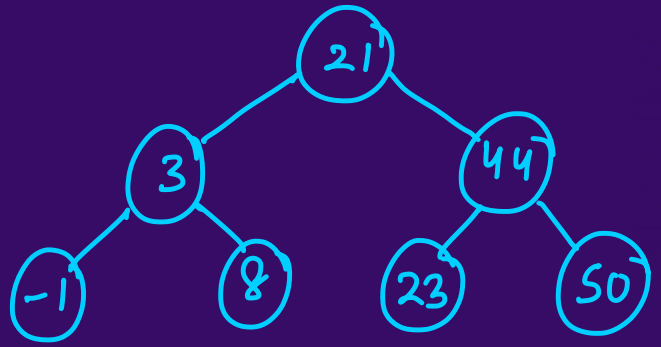
PW SKILLS

Q : Convert Sorted Array to Balanced Binary Search Tree

arr = { -1, 3, 8, 21, 23, 44, 50 }



ye nahi karna



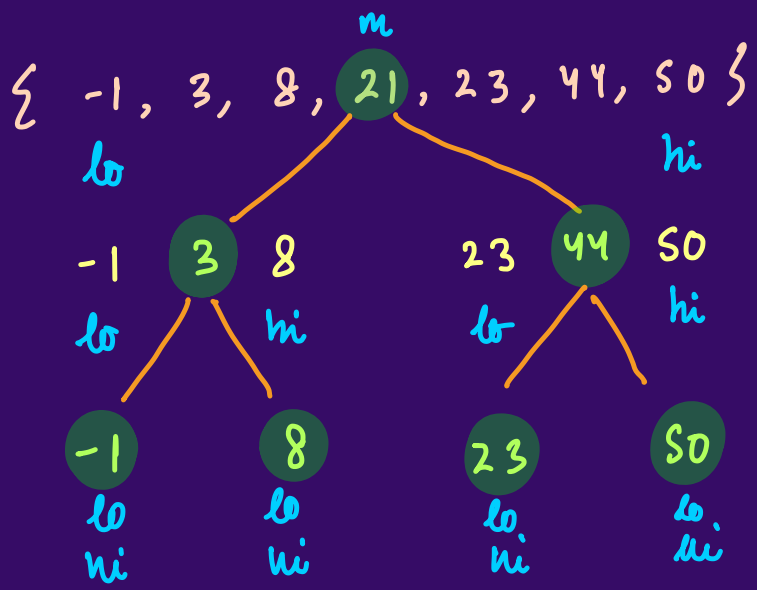
ye karna hai

[Leetcode 108]

Ques:

PW SKILLS

Q : Convert Sorted Array to Binary Search Tree



[Leetcode 108]

Homework:

Q : Balance a Binary Search Tree

[Leetcode 1382]

Ques:

PW SKILLS

Q : Construct Binary Search Tree from Preorder traversal

pre : 8 5 1 7 10 2

in : 1 2 5 7 8 10

Summarise : BST properties

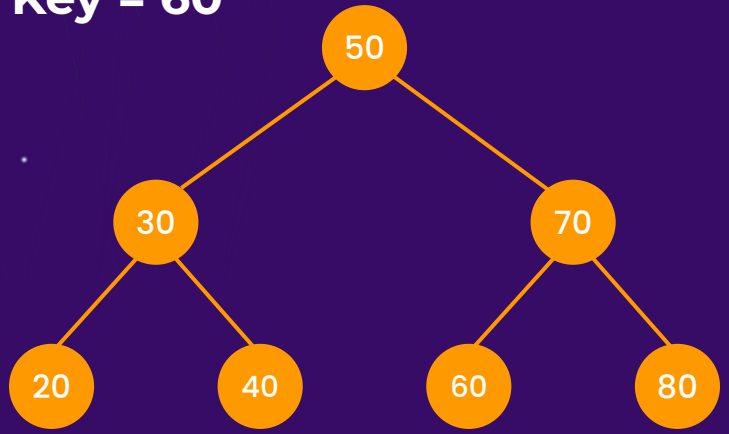
- max(LST) < root.val < min(RST)
- Inorder traversal of BST is sorted

[Leetcode 1008]

Inorder predecessor and successor for a given key in BST

Input:

Key = 60



inorder = { 20, 30, 40, 50, 60, 70, 80 }

p

n 20, 30, 40, 50, 60, 70, 80

r

pred = p

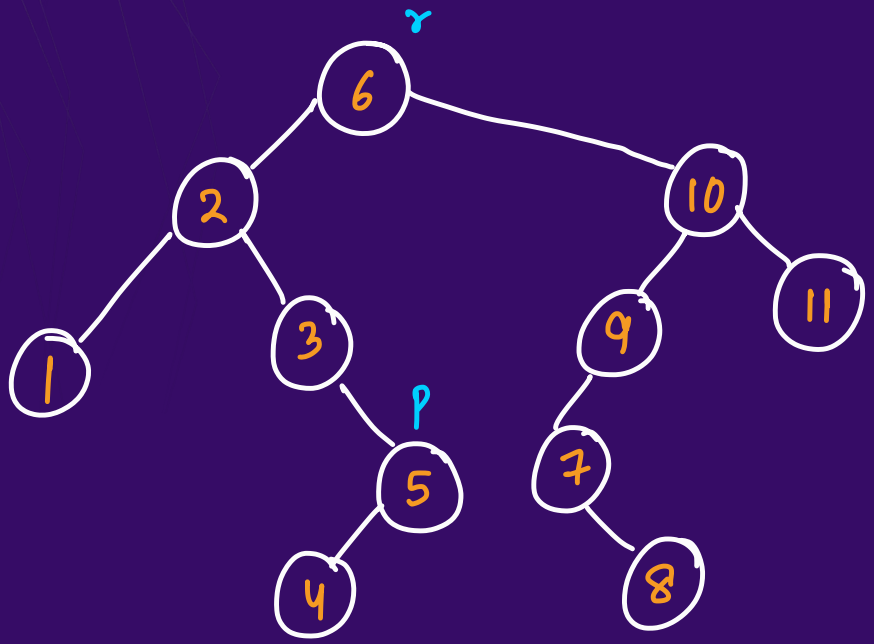
succ = r

Output:

Inorder predecessor is 50

Inorder successor is 70

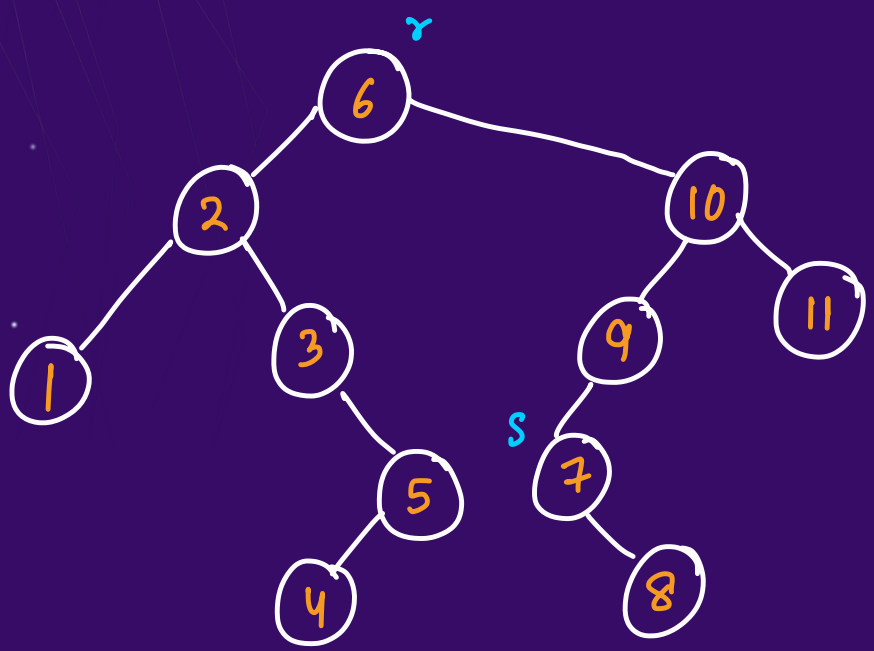
Inorder predecessor



6 Ka pred

```
pred = root.left
while (pred.right != null)
    pred = pred.right
```

Inorder successor



go right once

go left jab tak null

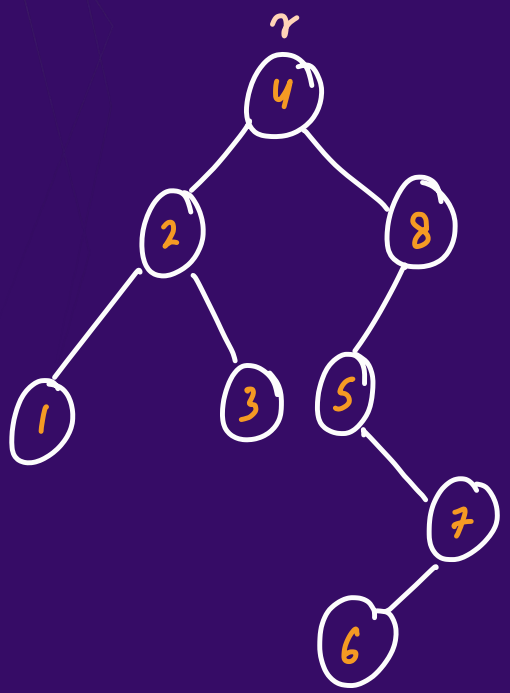
rahi aata

```
s = r.right
while(s.left != null)
    s = s.left
```

Ques:

PW SKILLS

Q : Delete Node in a BST

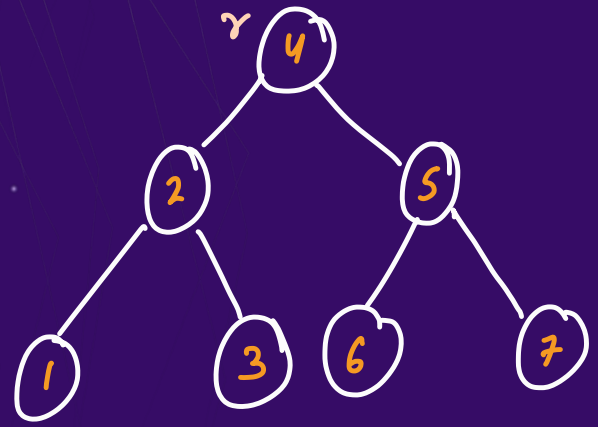


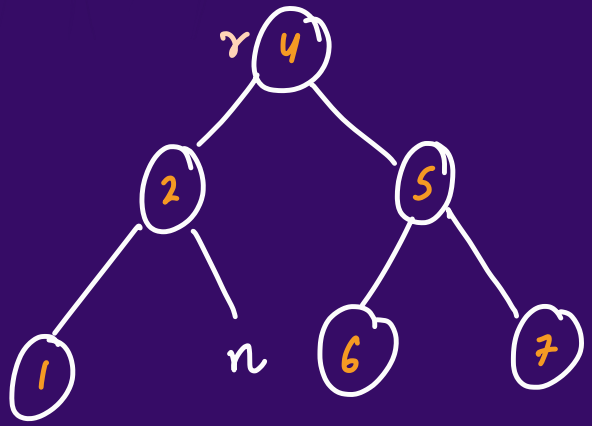
integer

delete (root, key)

[Leetcode 450]

Deletion: The node has 0 child / leaf node

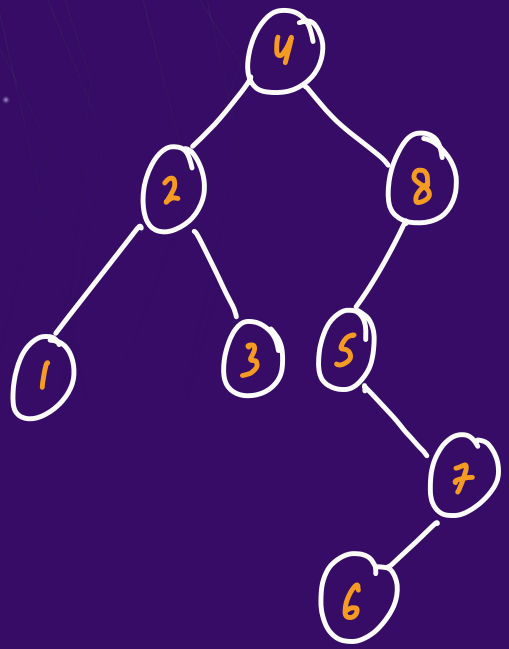




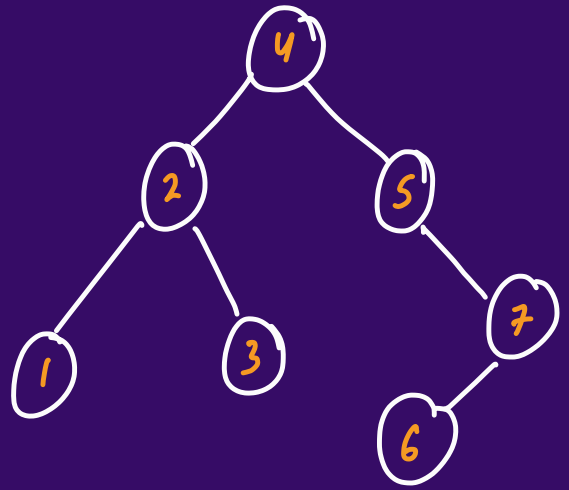
key = 3

```
if (root.val > key)  // LST will change
    root.left = delete(root.left, key);

if (root.val < key)  // RST will change
    root.right = delete(root.right, key)

if (root.val == key) return null
```

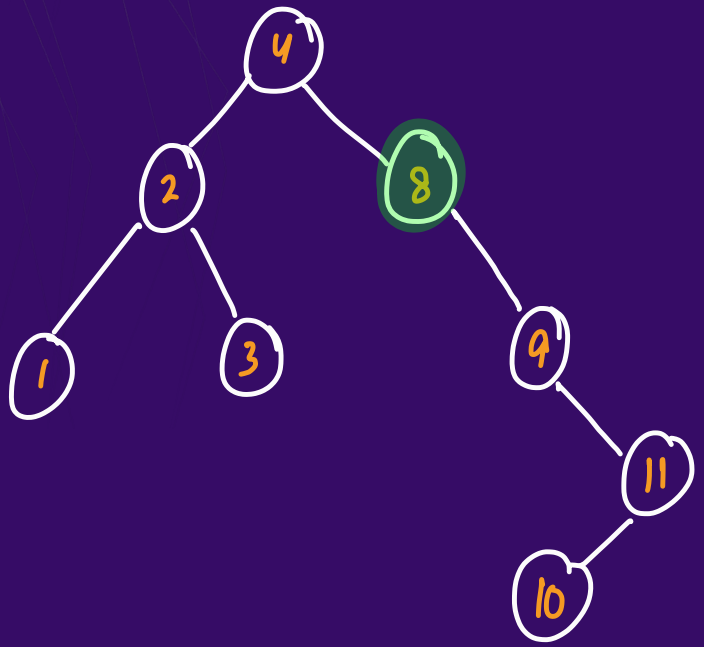
Deletion: The node has 1 child

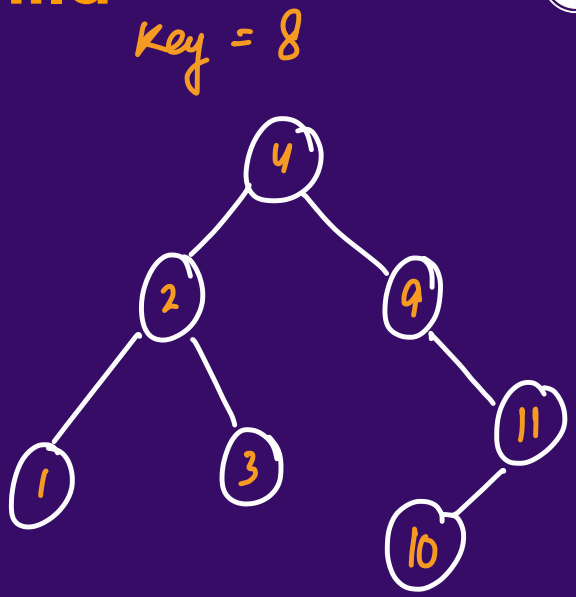


key = 8

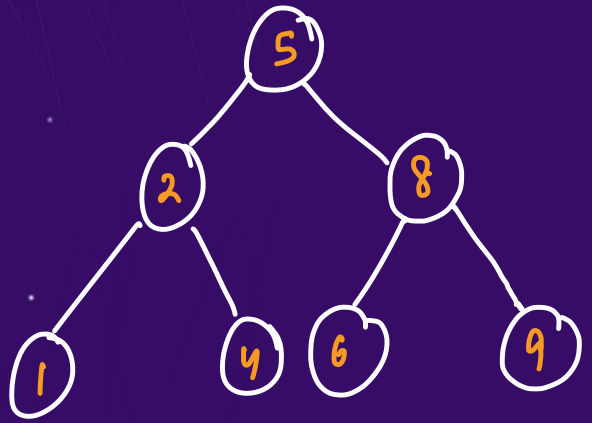


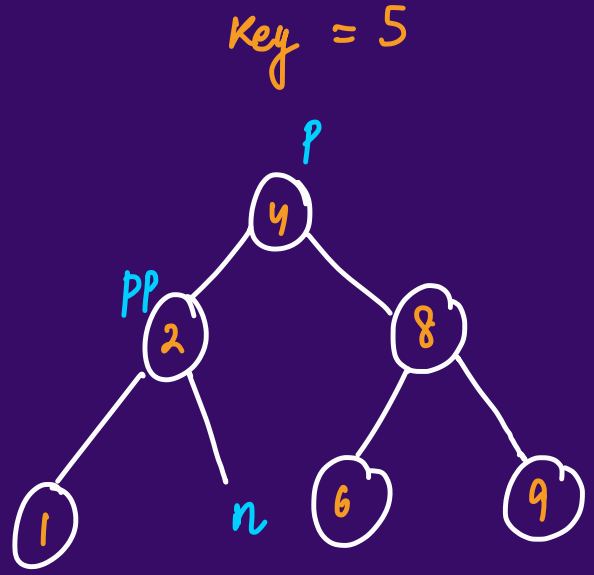
Deletion: The node has 1 child





Deletion: The node has 2 children

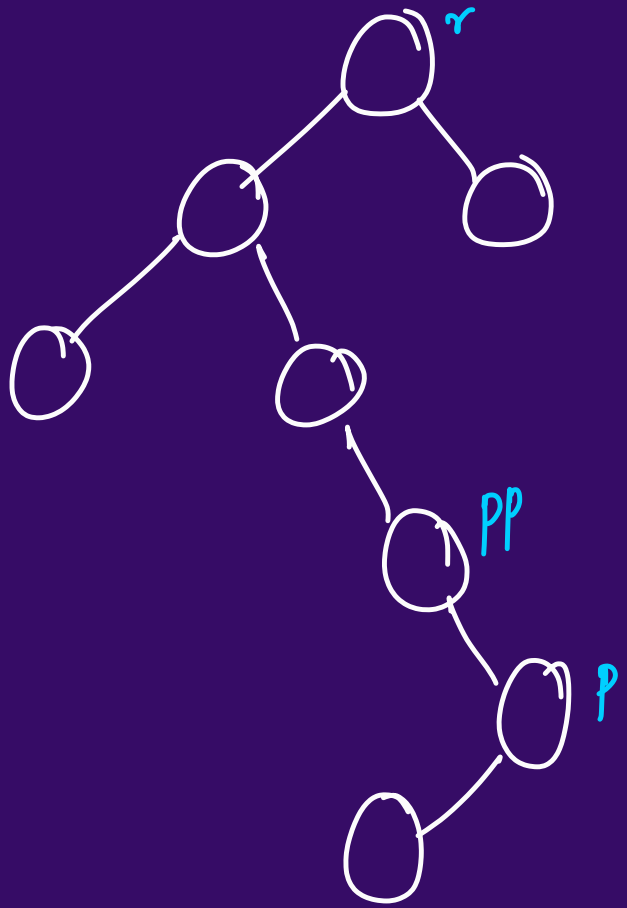




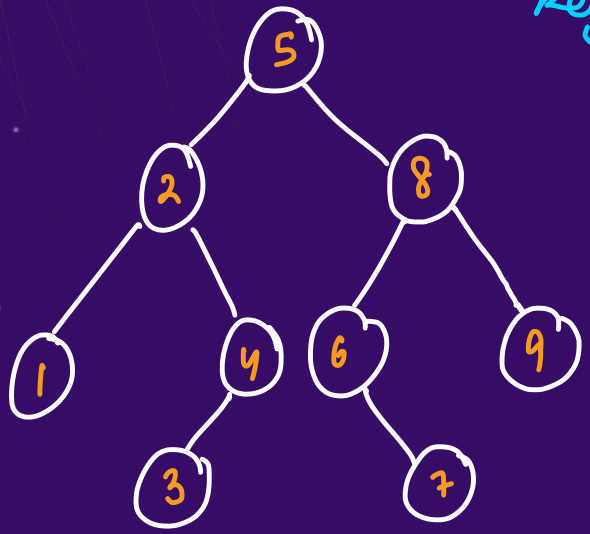
pp.right = null
p.left = r.left
p.right = r.right
return pred

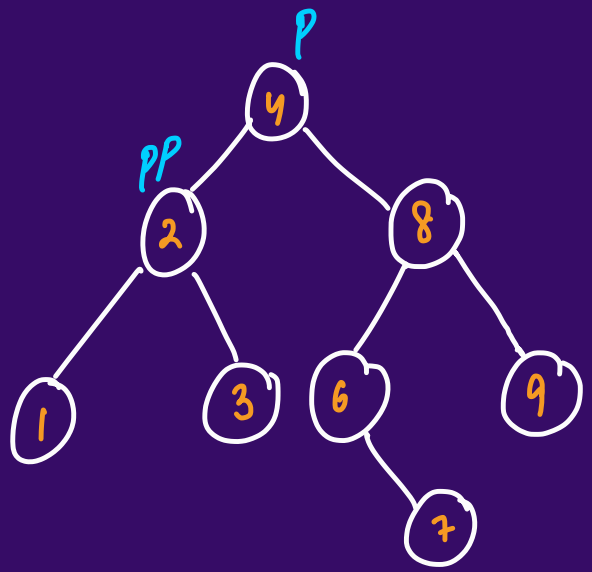
Concept :
replace 'key' node with its inorder predecessor

Deletion: The node has 2 children



Deletion: The node has 2 children





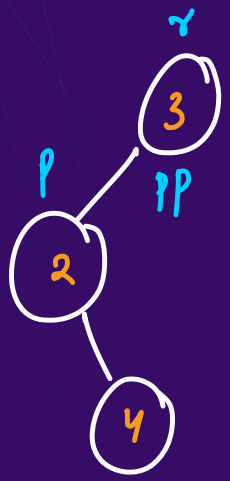
pp.right = p.left

p.left = r.left

p.right = root.right

predecessor ka koi right nahi hota

Deletion: The node has 2 children



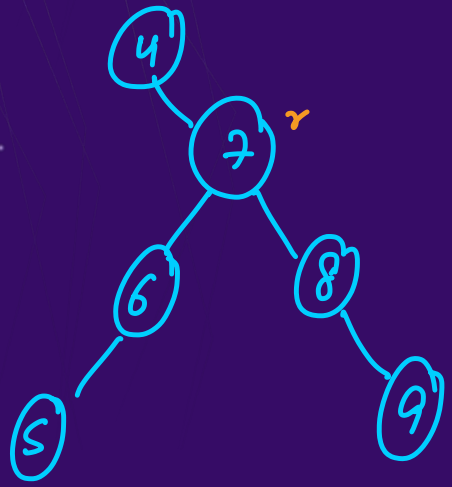
key = 3

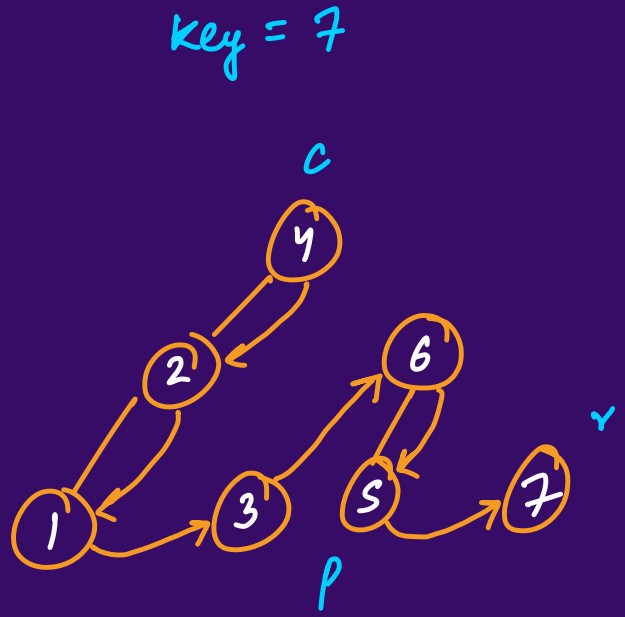
```
predParent.right = pred.left;
pred.left = root.left; pred.right = root.right;
return pred;
```

] galat

Case → if pred is child of the node which you want to delete

Deletion: The node has 2 children





```java
public TreeNode deleteNode(TreeNode root, int key) {
    if(root==null) return null;
    if(root.val==key){ // deletion
    }
    else if(root.val>key){ // LST will change
        root.left = deleteNode(root.left,key);
    }
    else{ // root.val<key : RST will change
        root.right = deleteNode(root.right,key);
    }
    return root;
}
```

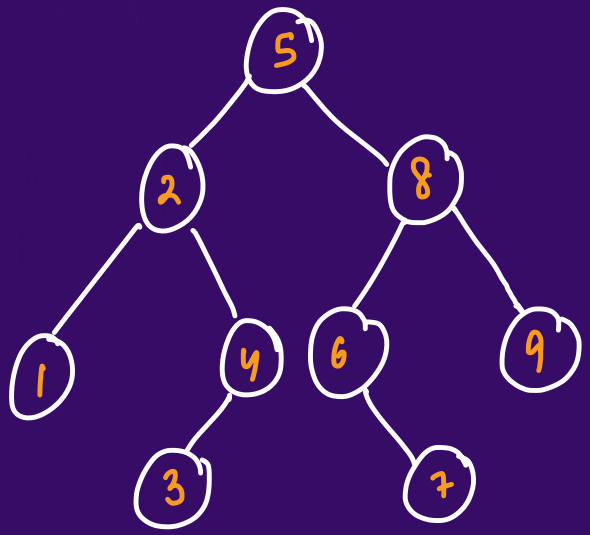
```java
public TreeNode iop(TreeNode root){
    TreeNode temp = root.left;
    while(temp.right!=null) temp = temp.right;
    return temp;
}
public TreeNode parent(TreeNode root, TreeNode pred){
    if(root.left==pred || root.right==pred) return root;
    TreeNode temp = root.left;
    while(temp.right!=pred) temp = temp.right;
    return temp;
}
```

2 c

```java
if(root.val==key){ // deletion
    // Case 1 : 0 child nodes
    if(root.left==null && root.right==null) return null;

    // Case 2 : 1 child node
    else if(root.left==null || root.right==null){
        if(root.left==null) return root.right;
        else return root.left;
    }

    // Case 3 : 2 child nodes
    else{
        TreeNode pred = iop(root);
        TreeNode predParent = parent(root,pred);
        if(root==predParent){
            pred.right = root.right;
            return pred;
        }
        predParent.right = pred.left;
        pred.left = root.left; pred.right = root.right;
        return pred;
    }
}
```

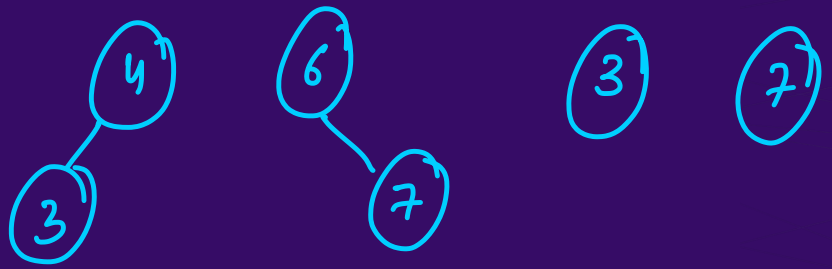
Homework:

Q : Count BST Subtrees that lie in given range



root, lo, hi
↓ ↓
2 8

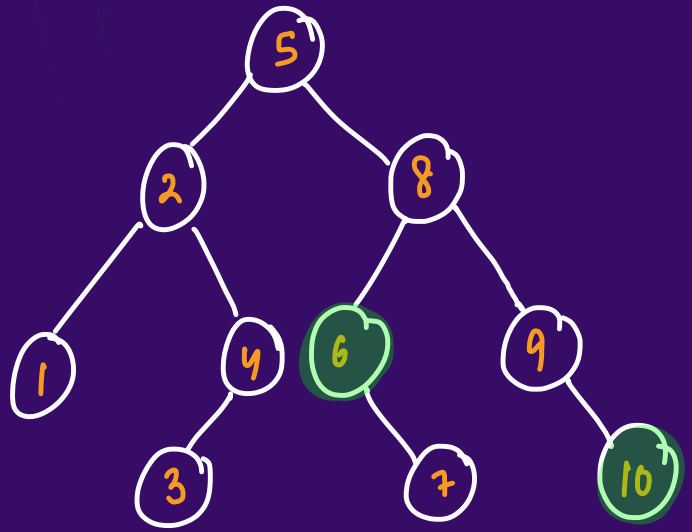
Count = 4



Homework:

PW SKILLS

Q : Shortest distance between two nodes in BST



root, p, q

6 10

dist = 3

Morris Traversal [Google]

PW SKILLS

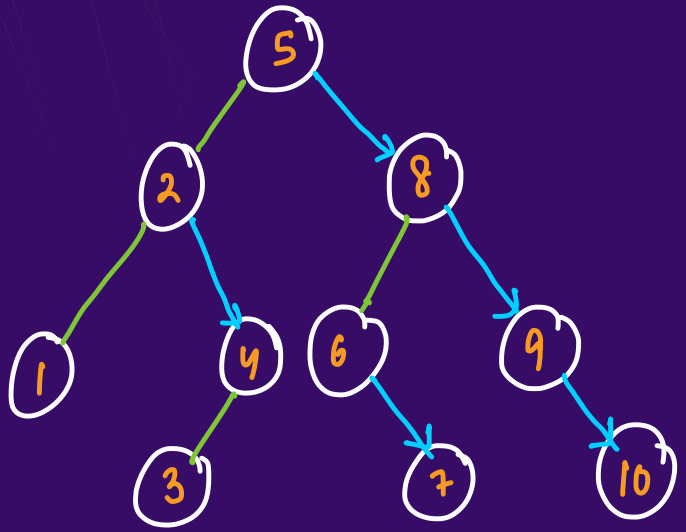
Traversals → preorder, postorder, inorder [recursive] → T.C. = O(n)
S.C. = O(h)
↓
h = log n
h = n

Morris → inorder traversal in O(1) space
↓
Before your interview → ek baar solve

You need to Know INORDER PRED

Concept : Linking & Unlinking
↓
change tree structure

Morris Traversal



c

ans = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

```
while (curr != null) {
    if (curr.left != null) {
        Node pred = curr.left
        while (pred.right != null && pred.right != c)
            pred = pred.right
        if (pred.right == null)
            pred.right = curr
            curr = curr.left
        }
        else {
            visit(curr)
            curr = curr.right
            pred.right = null
        }
    }
    else
        visit(curr)
        curr = curr.right;
}
```

link

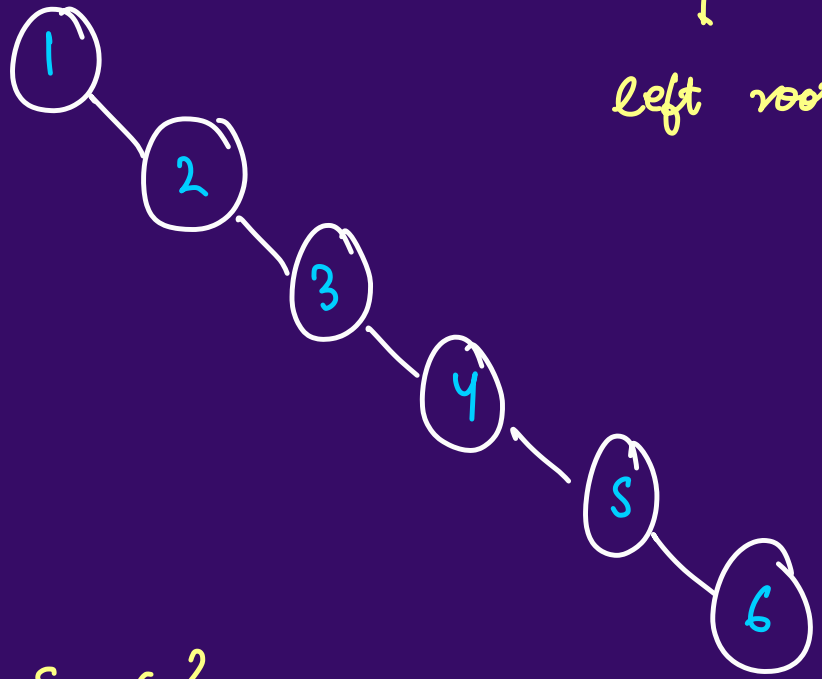
unlink

SKILLS

Morris Traversal

PW SKILLS

Inorder
{
left root right



{1, 2, 3, 4, 5, 6}

C

```java
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> ans = new ArrayList<>();
    TreeNode curr = root;
    while(curr!=null){
        if(curr.left!=null){ // left hai
            TreeNode pred = curr.left;
            while(pred.right!=null && pred.right!=curr)
                pred = pred.right;
            if(pred.right==null){ // link
                pred.right = curr;
                curr = curr.left;
            }
            else{ // pred.right = curr : Unlink
                pred.right = null;
                ans.add(curr.val);
                curr = curr.right;
            }
        }
        else{ // left nahi hai
            ans.add(curr.val);
            curr = curr.right;
        }
    }
    return ans;
}
```

PW SKILLS

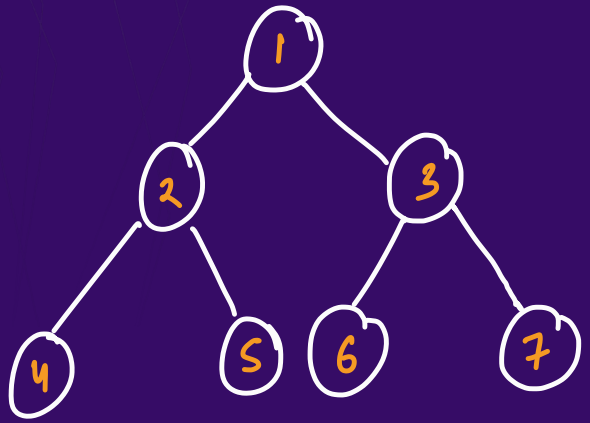
Homework :

K^{th} smallest element in A BST using morris traversal

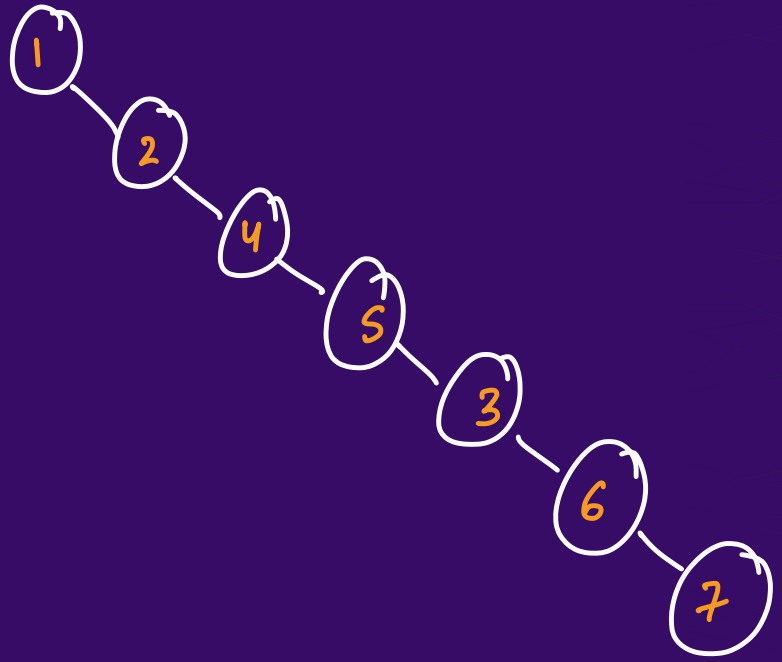
Ques:

PW SKILLS

Q : Flatten Binary Tree to Linked List (Interview Question)



⇒



pre: 1 2 4 5 3 6 7

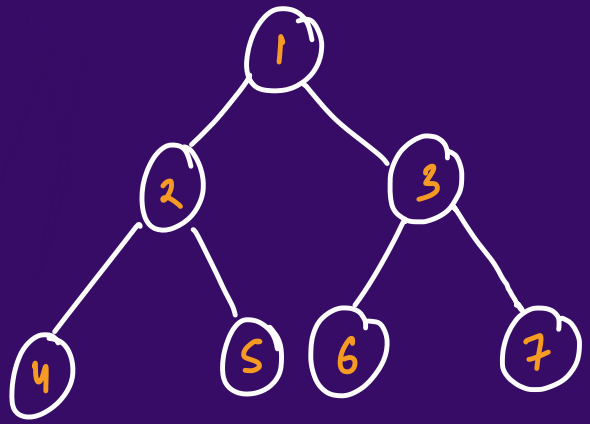
[Leetcode 114]

Ques:

PW SKILLS

Q : Flatten Binary Tree to Linked List

M-I : Create preorder array of 'TreeNodes'



1) put null in left of every node

2) ans[i].right = ans[i+1]

T.C. = O(n)

S.C. = O(n)

ans : { 1 2 4 5 3 6 7 }

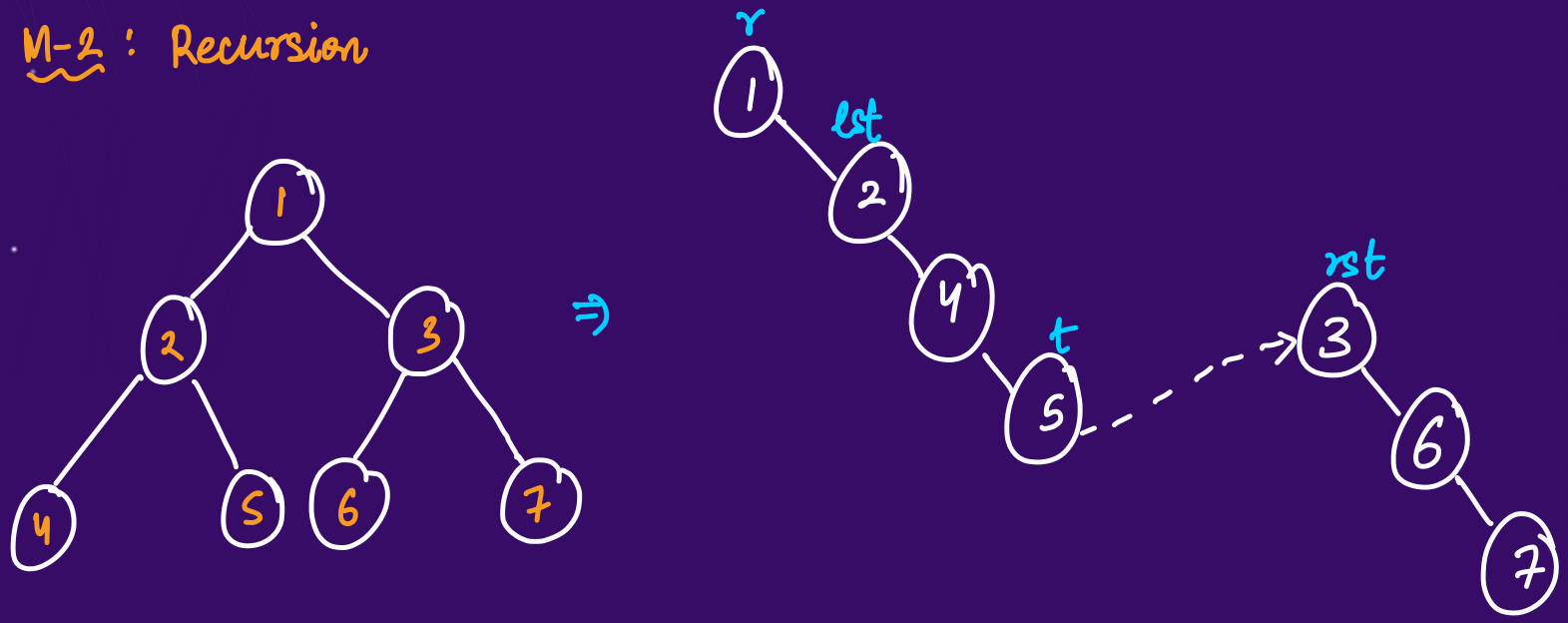
[Leetcode 114]

Ques: Preorder → Root left right

PW SKILLS

Q : Flatten Binary Tree to Linked List

M-2 : Recursion



[Leetcode 114]

Ques:

Q : Flatten Binary Tree to Linked List

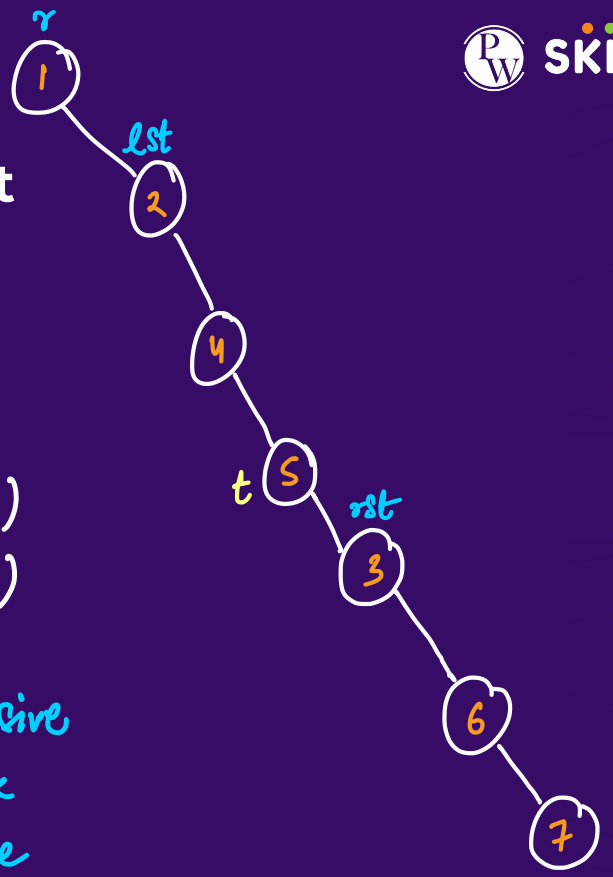
```java
public void flatten(TreeNode root) {
    if(root==null) return;
    if(root.left==null && root.right==null) return;
    TreeNode lst = root.left;
    TreeNode rst = root.right;
    flatten(lst);
    flatten(rst);
    root.left = null;
    root.right = lst;
    TreeNode temp = root;
    while(temp.right!=null) temp = temp.right;
    temp.right = rst;
}
```

T.C. = O(n)

S.C. = O(n)

↓

Recursive Stack Space



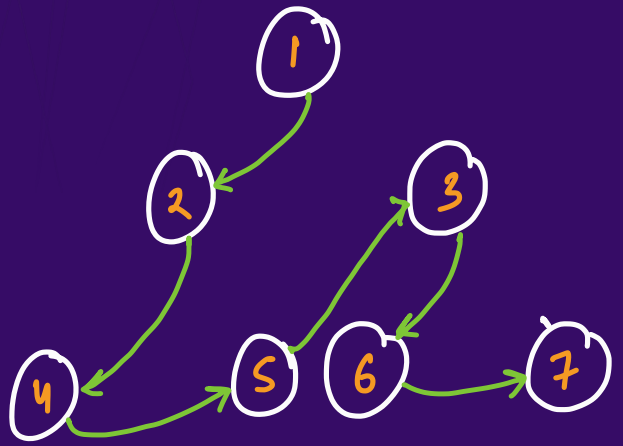
[Leetcode 114]

Ques:

PW SKILLS

Q : Flatten Binary Tree to Linked List

M-3 : Something like Morris Traversal [curr, pred, r]



green → right
blue → left

```
if(curr.left != null){
    TreeNode r = c.right
    curr.right = curr.left
    TreeNode pred = curr.left
    while(pred.right != null) pred = pred.right
    pred.right = r
    curr.left = null
    curr = curr.right
```

c

Pr 3

[Leetcode 114]

Ques:

PW SKILLS

Q : Flatten Binary Tree to Linked List

```java
public void flatten(TreeNode root) {
    TreeNode curr = root;
    while(curr!=null){
        if(curr.left!=null){
            TreeNode r = curr.right;
            curr.right = curr.left;
            TreeNode pred = curr.left;
            while(pred.right!=null) pred = pred.right;
            pred.right = r;
            curr.left = null; // IMPORTANT
            curr = curr.right;
        }
        else{
            curr = curr.right;
        }
    }
}
```

T.C. = O(n)

S.C. = O(1)

Bonus: Pre/In/Post → O(n) space Recursive sol"

In → morris → O(1) space

Pre → morris → O(1) space

(But structure of tree is permanently changed)

[Leetcode 114]

Next Lecture

- Sets, Maps, Heaps

THANK YOU