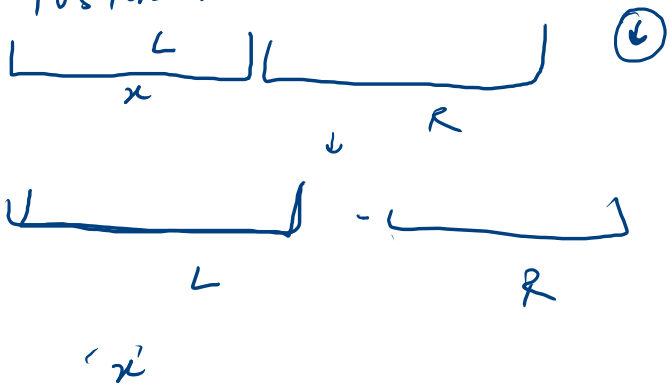
Starts @ 9:10 pm

→ Construction

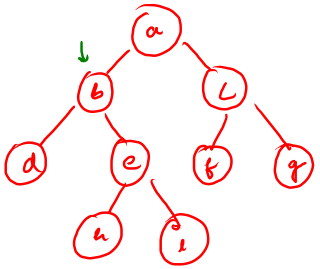
↳ Postorder & Inorder.

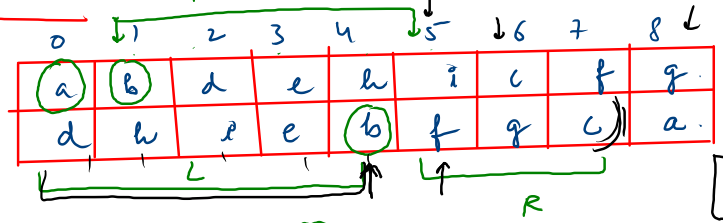
isi > iei



Preorder and Postorder Traversal

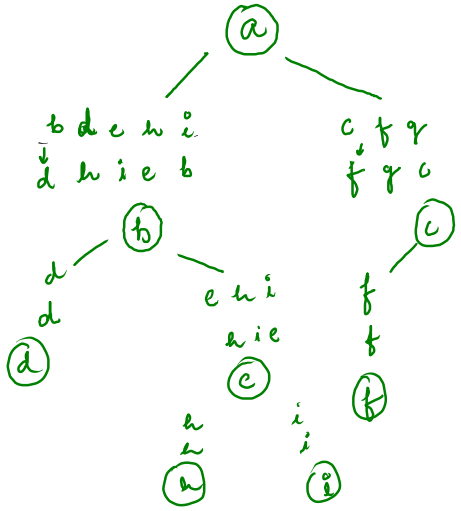
N L R





posi > pei → stop

10 mins



right side

psi → psi + tnn + 1
pei → pei
posi → idx + 1
poci → poci - 1

idx = 0

tnn = idx - posi + 1

4 - 0 + 1 = 5

left side

psi → psi + 1
pei → psi + tnn

posi → posi
poci → idx

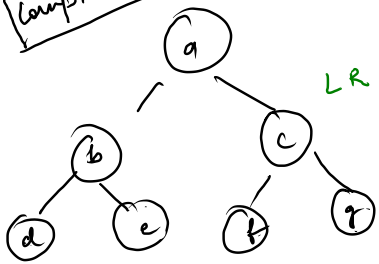
9:50pm

#

Complete binary Tree

N LR

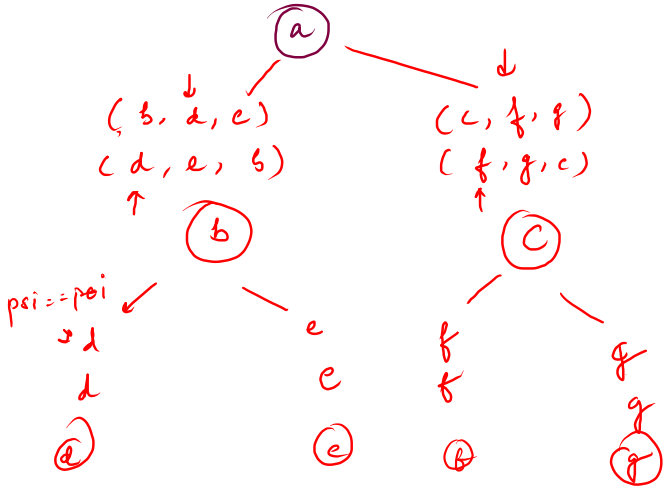
LR N

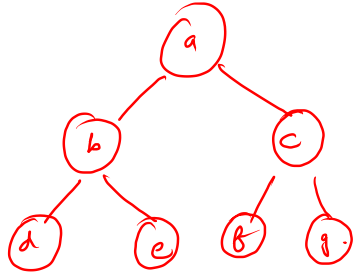


pre

post

0	1	2	3	4	5	6
a	b	d	e	c	f	g
d	e	b	f	g	c	a



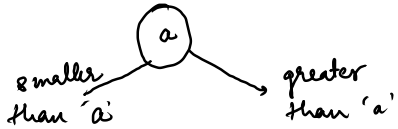


```java
public static Node helper(int [] post,int posi, int poei,int [] pre, int psi,int
   if(psi>pei){
       return null;
   }

    Node node = new Node(pre[psi]);

    if(psi == pei){
        return node;
    }

    int idx = posi;
    while(post[idx] != pre[psi+1]){
        idx++;
    }

    int tnn = idx-posi+1;

    node.left = helper(post,posi,idx,pre,psi+1,psi+tnn);
    node.right = helper(post,idx+1,poei-1,pre,psi+tnn+1,pei);

    return node;

}

public static Node constructTree(int [] post, int [] pre){
    return helper(post,0,post.length-1, pre, 0,pre.length-1);
}
```

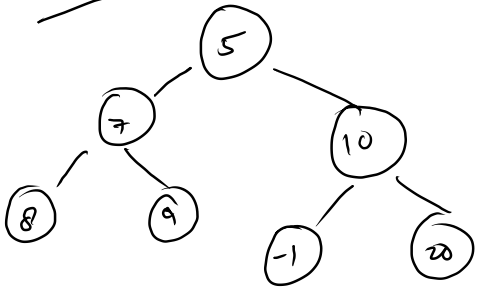
Binary Search Tree (BST)

↳ Similarly to Binary Tree (at most 2 children $\begin{matrix}0\\1\\2\end{matrix}$)

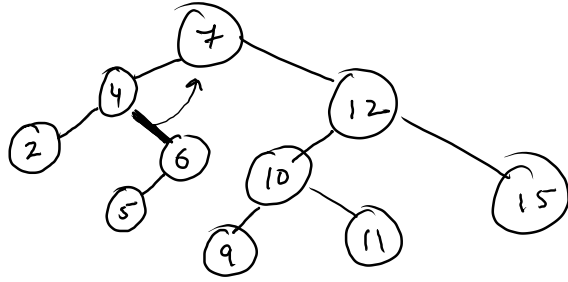
↳ For a particular



B.T

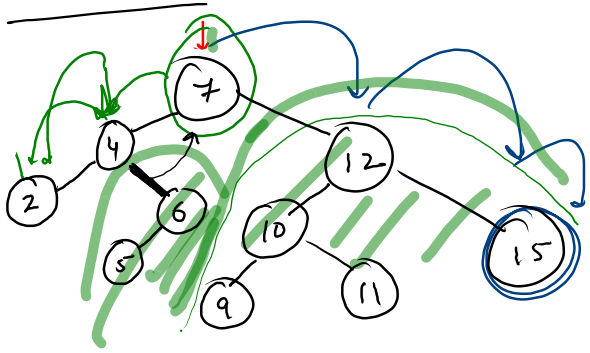


B.S.T



B.T: O(n)

B.S.T



T.C.

Min^n value node:- 2

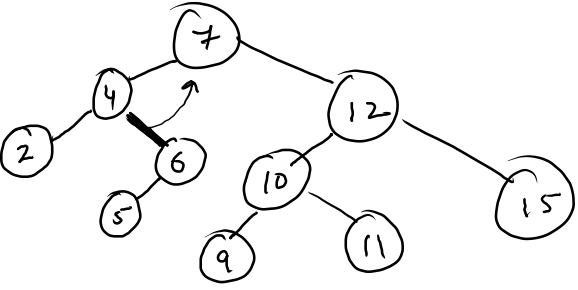
↳ node.left == null && node.right == null

node.left

Max^n value node:- 15

B.S.T → O(log^n)

B.S.T



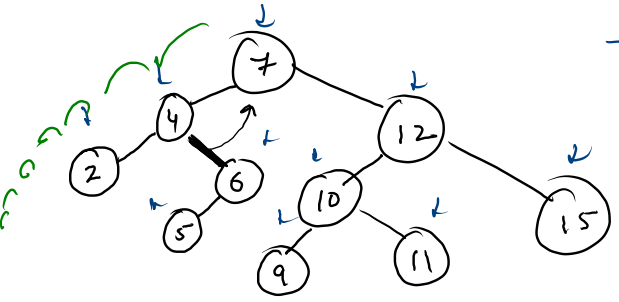
Preorder :- 7, 4, 2, 6, 5, 12, 10, 9, 11, 15

Postorder :- 2, 5, 6, 4, 9, 11, 10, 15, 12, 7

Inorder :- 2, 4, 5, 6, 7, 9, 10, 11, 12, 15

Obs :- Inorder traversal of B.S.T in always sorted

B.S.T



size :- exactly similar to B.T. T.C. O(n)

sum :- ——— X ———

min :- B.T → O(n) | B.S.T → O(log n)

max :- B.T → O(n) | B.S.T → O(log n)

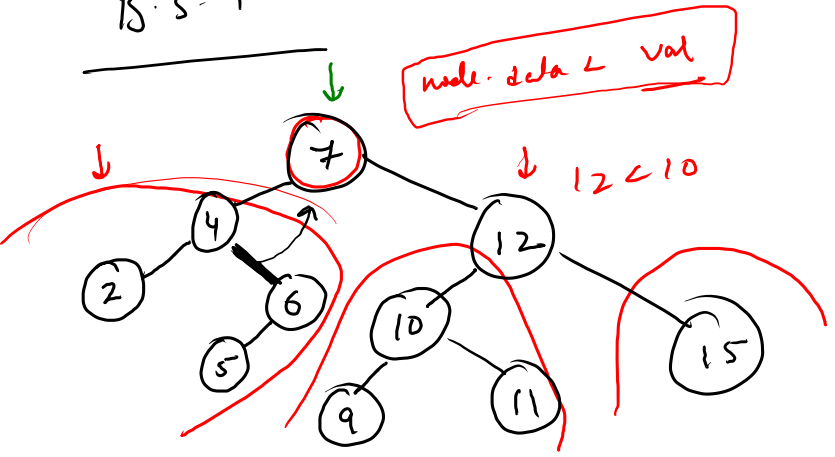
15

9 11

B.T. → O(n)

val → 10

B.S.T




```
if (node == null)
    return false;
→ if node.data == val
    return true;
else if node.data < val.
    find (node.right)
else
    find (node.left)
```

⇒ find a node in B.S.T = O(log n)

```java
public static int size(Node node) {
  // write your code here
    if(node == null) return 0;

    int leftSize = size(node.left);
    int rightSize = size(node.right);

    return leftSize + rightSize+1;
}

public static int sum(Node node) {
  // write your code here
    if(node == null) return 0;

    int leftSum = sum(node.left);
    int rightSum = sum(node.right);

    return leftSum + rightSum + node.data;
}

public static int max(Node node) {
  // write your code here
    if(node == null){
        return Integer.MIN_VALUE;
    }
    if(node.left == null && node.right == null){
        return node.data;
    }

    return max(node.right);
}
```

```java
public static int min(Node node) {
  // write your code here
    if(node == null) {
        return Integer.MAX_VALUE;
    }

    if(node.left == null && node.right == null){
        return node.data;
    }

    return min(node.left);
}

public static boolean find(Node node, int data){
  // write your code here
    if(node == null){
        return false;
    }

    if(node.data == data){
        return true;
    } else if(node.data>data){
        return find(node.left,data);
    } else {
        return find(node.right,data);
    }


}
```