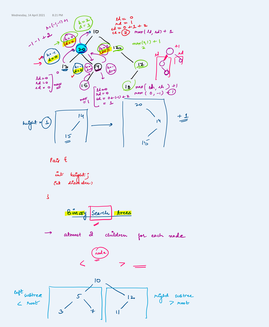


Pass 1

Step: height
Pick children

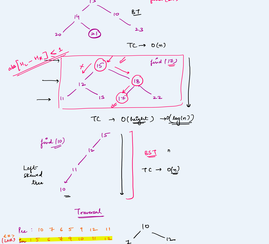
3

Binary Search Trees

→ almost 2 children for each node.

node

c ≥ ≤



find (3)

BST

TC → O(n)

find (18)

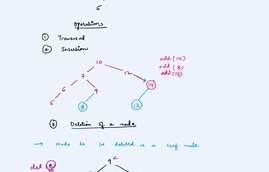
TC → O(height) → O(log(n))

Logic should flow

BST

TC → O(n)

Traversal



Operations

1. Traversal
2. Insertion
3. Deletion of a node

→ node to be deleted is a leaf node

del (9)

→ node to be deleted has only one child

→ copy the child to the node and delete the child

del (7)

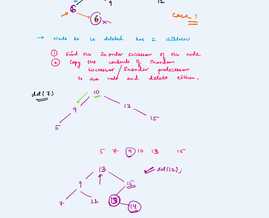
Case 1

→ node to be deleted has 2 children

1. Find the inorder successor of the node.
2. Copy the contents to the inorder successor / inorder predecessor to the node and delete it.

del (3)

5 7 9 10 13 15



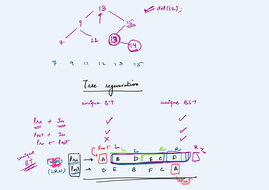
Tree representation

unique BST

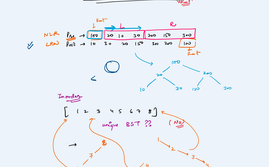
unique BST

Pass → In
Post → In
Pre ← Post

Pre: A B D E C F G
In: D E B F C G A



Pre: 100 50 25 75 150 125 175
In: 10 25 30 50 100 125 150 175



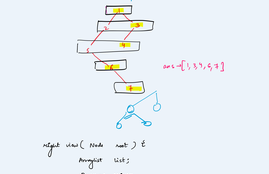
In order

[1 2 3 4 5 6 7]

unique BST ?? (yes)

Q1. given a BT, print right view of a BT.

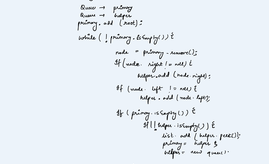
output → 1 3 5 7



ans = [1, 3, 5, 7]

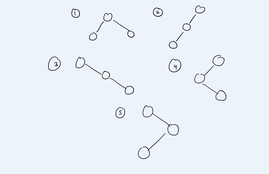
```
rightView( Node root ) {
    ArrayList list;
    Queue → primary
    Queue → helper
    primary.add (root);
    while ( ! primary.isEmpty() ) {
        node = primary.remove();
        if (node.right != null) {
            helper.add (node.right);
        if (node.left != null) {
            helper.add (node.left);
        if ( primary.isEmpty() ) {
            if ( helper.isEmpty() ) {
                list.add ( helper.peek());
                primary = helper;
                helper = new Queue();
            }
        }
    }
}
```

71, 65, 84, 69, 67, 83

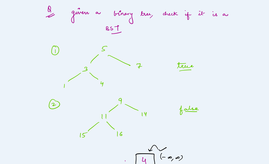


Q. given a binary tree, check if it is a BST

False



false



```
BST( node, min, max) {
    if (node == null)
        return true;
    if (node.data < min || node.data > max)
        return false;
    return BST( node.left, min, node.data-1) && BST( node.right, node.data+1, max);
}
```