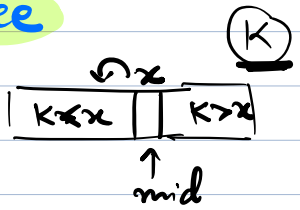
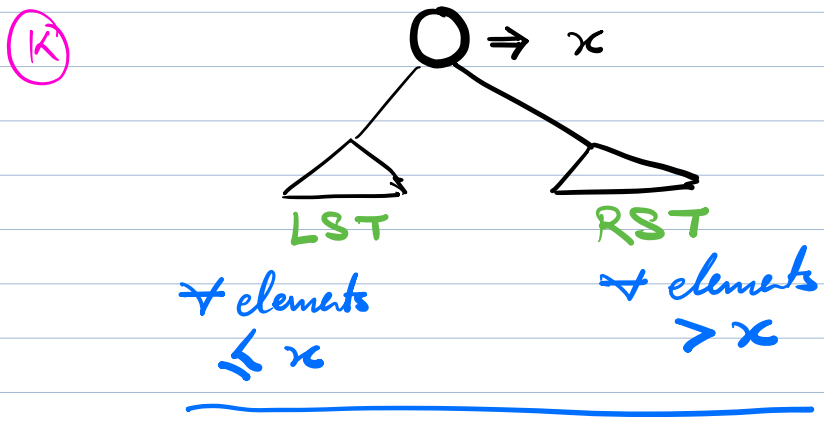
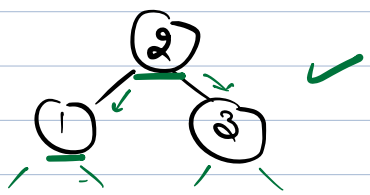
BST ⇒ Binary Search Tree

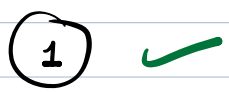




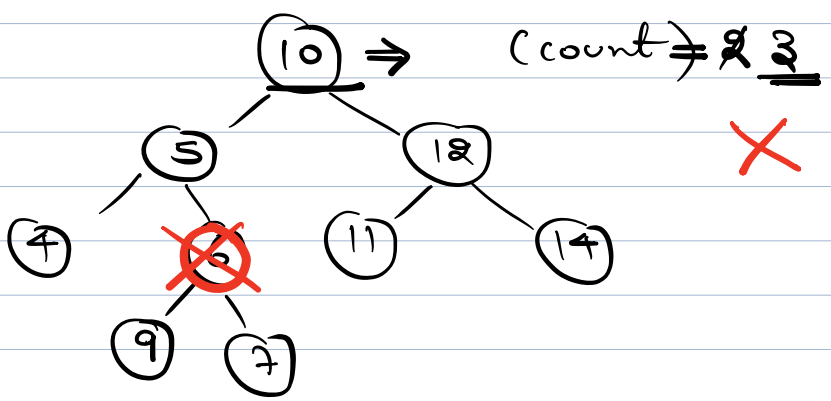
∀ nodes in the tree

1) 

✓

2) 

✓

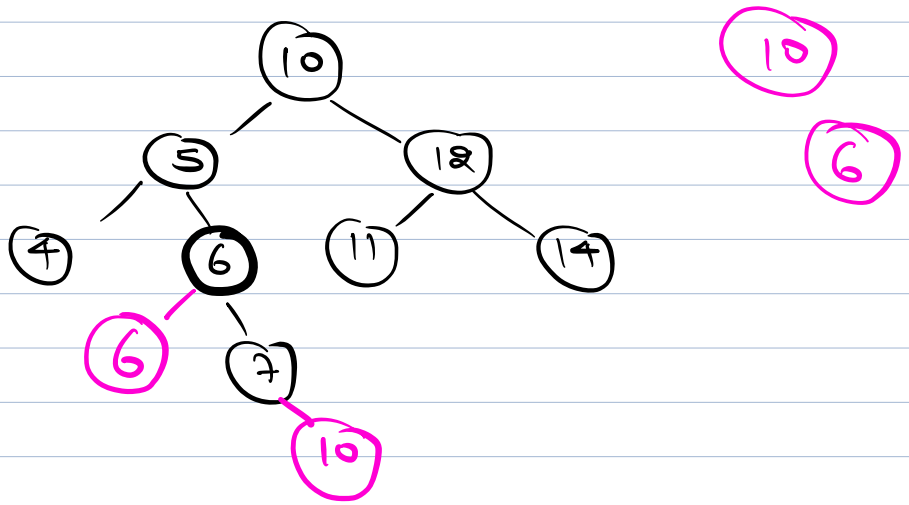
3) 

count ≠ ~~2~~ 3
✗

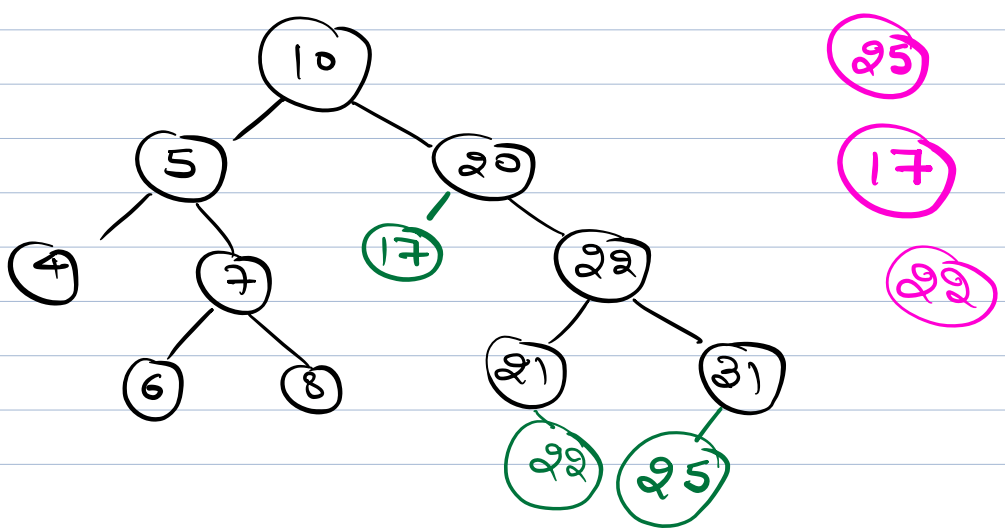
Duplicates

1) Add a variable count to the node structure.

2) Choose one side (LST or RST) & remain consistent for all nodes



Given the root node of a BST insert a new node s.t. the property of BST is maintained.

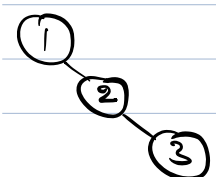


Code

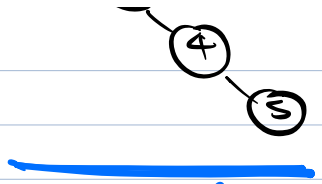
```
Node insert (root, K) {
// Ass: Insert K at the correct position in the tree rooted at the root node & returns the updated root.
if (root == NULL) {
        return new Node (K); }
if (root.val >= K) {
        root.left = insert (root.left, K);
}
else {
        root.right = insert (root.right, K);
}
return root;
}
```

T.C. = O(H) H = height

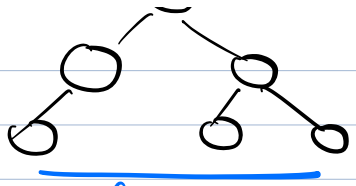
S.C. = O(N) ↓ N



N



Skewed Tree



Balanced BST

$\log N$

Given a BST & a value K

Return true if K is present in the given BST

Code

```
boolean search (root, K) {
    if (root == NULL) {
        return false;
    }
    if (root.val == K) {
        return True;
    }
    if (root.val > K) {
        return search (root.left, K);
    }
```

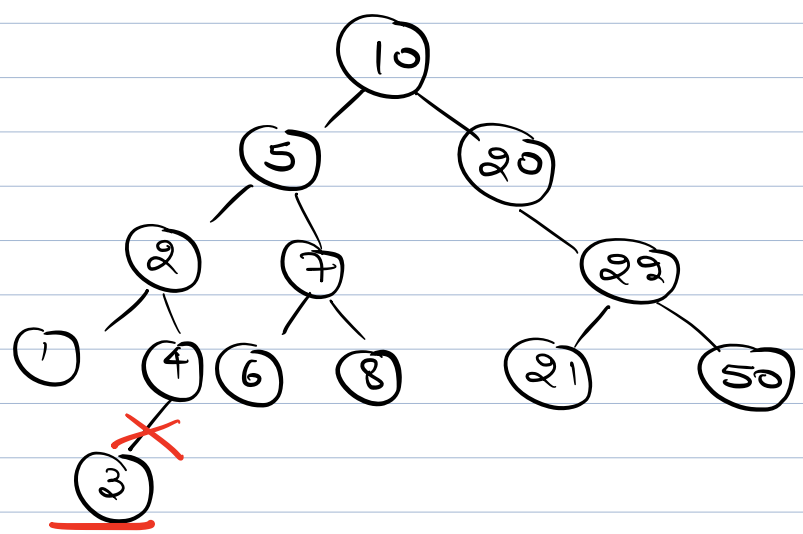
```
else {
    return search(root.right, K);
}
}
```

T.C. = O(H) = O(N)

S.C. = O(N)

Q Given a BST. Delete node with a value K (No duplicates)

K ↓ present

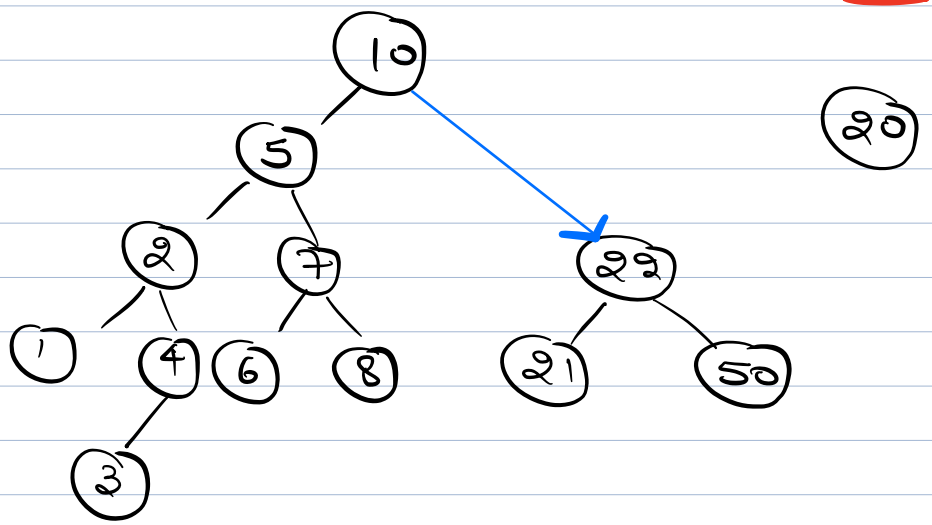


Case I ⇒ (If K is present at a leaf node)

(3)

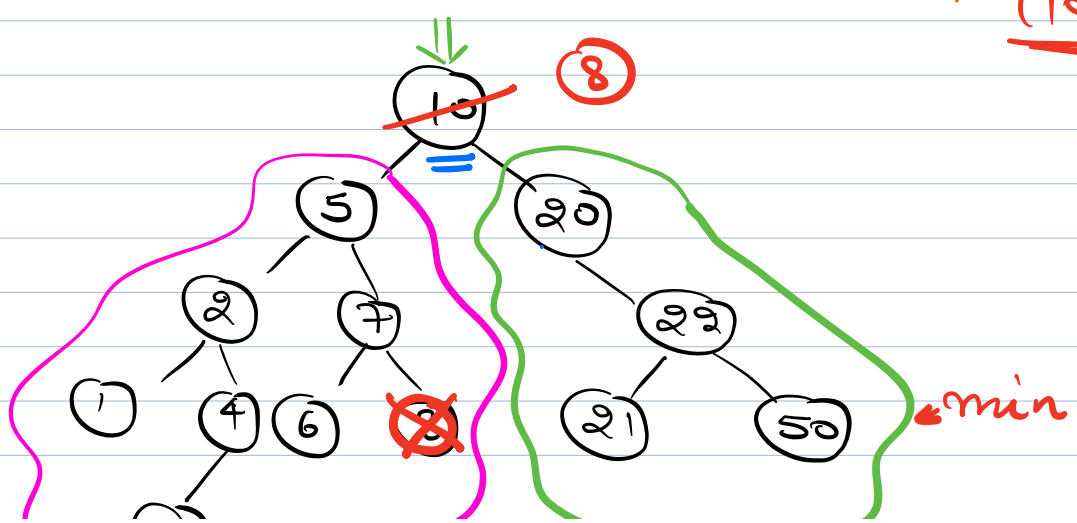
Make it NULL

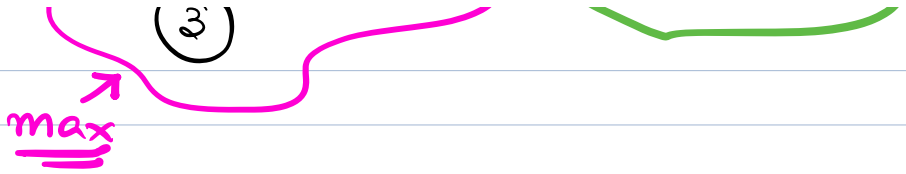
Case II ⇒ (If Node to be deleted has one child) (20)



⇒ Replace the node with the non NULL child.

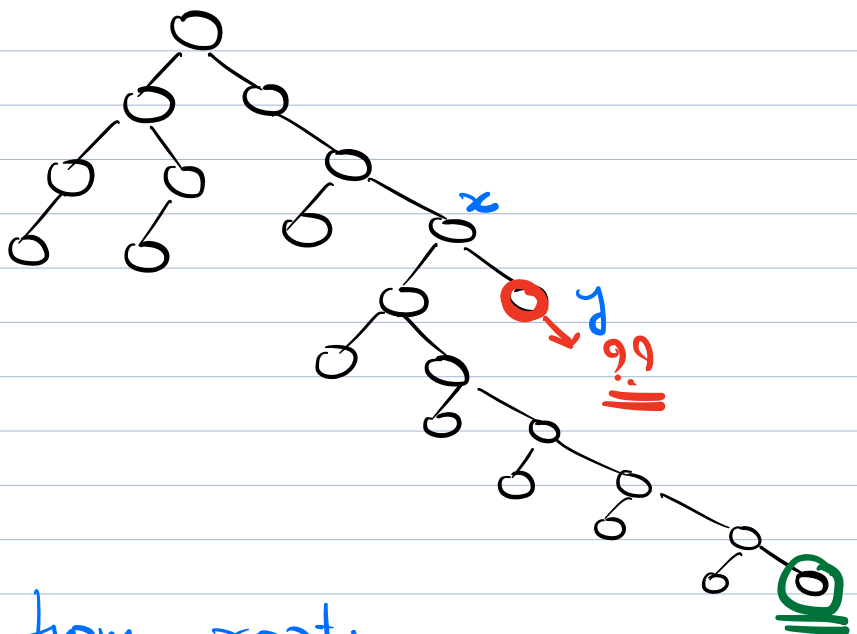
Case 3 ⇒ (Both children are present) (10)





$\forall$ elt LST $\leq x <$ $\forall$ RST

(BST)



1) Max

⇒ Start from root.

⇒ Keep going right till you find NULL

* Max of BST will never have a right child.

2) Min

⇒ Start from root

⇒ Keep going left till you find NULL

Min will never have left child

Code

```
Node delete (root, K) {
    if (root == NULL) {
        return NULL;
    }

    if (root.val > K) {
        root.left = delete(root.left, K);
    }
    else if (root.val < K) {
        root.right = delete(root.right, K);
    }

    else {  // (root.val == K).
```

Case I

```
if (root.left == NULL && root.right == NULL) {
    return NULL;
}
```

Case II

```
else if (root.left == NULL) {
    return root.right;
}
else if (root.right == NULL) {
    return root.left;
}
```

Case III

```
else {
    Node max = getMax(root.left);   // ???
    root.val = max.val;
    root.left = delete(root.left, max.val);
}
return root;
}
```

T.C. = O(H) = O(N)

Doubt

Check

1) Pre & Level → Min Height