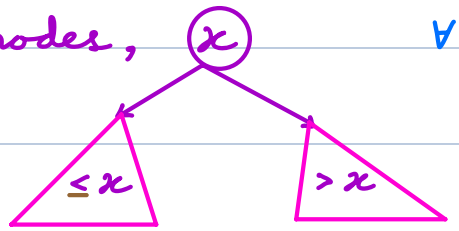
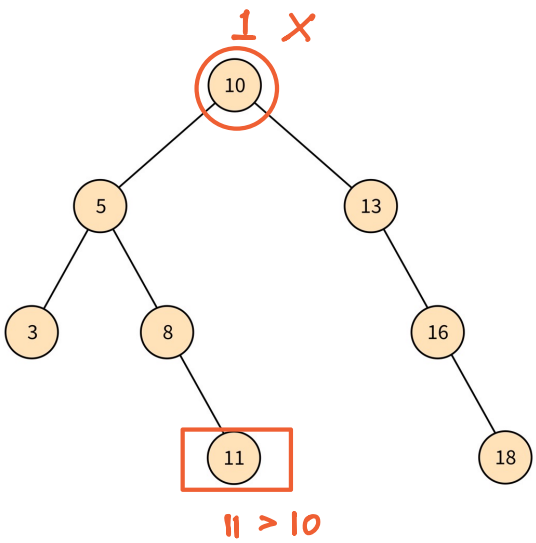
Binary Search Tree [BST]

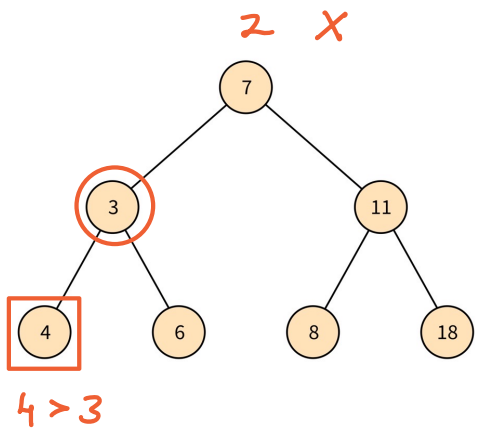


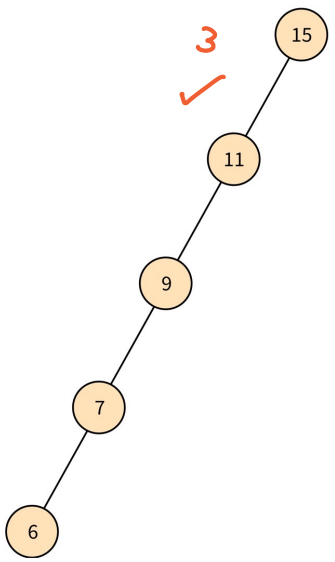
∀ nodes x,

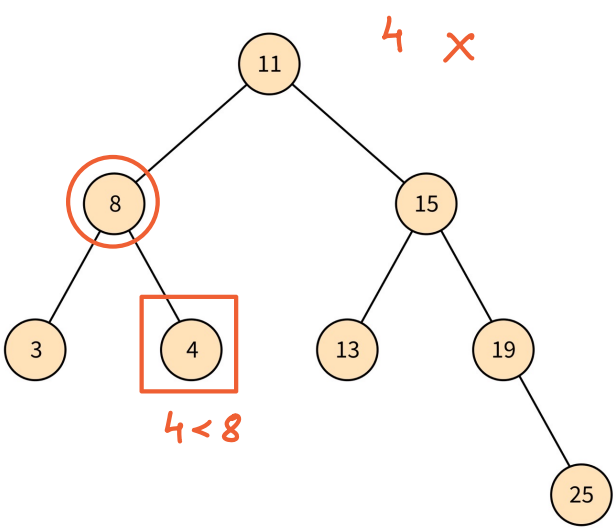
left subtree data <= x

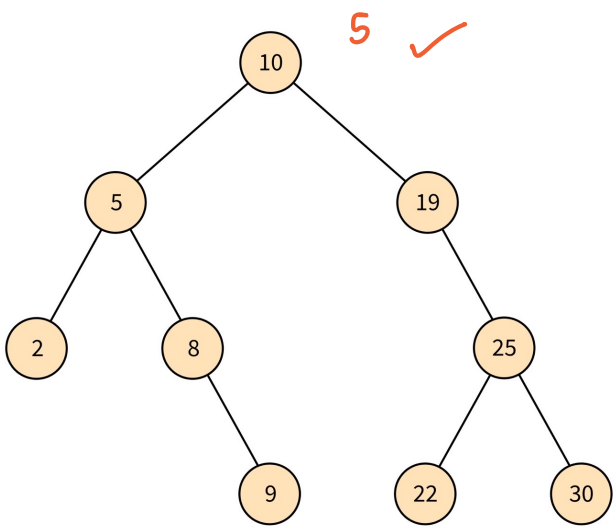
right subtree data > x











Scenario

Flipkart stores and manages the history of all the orders that were processed and stores them efficiently such that whenever some information is required regarding any order, it can be fetched easily.

Problem

Develop a system for Flipkart that efficiently stores and manages the history of all processed orders. The system should allow easy access to information regarding any specific order when needed.

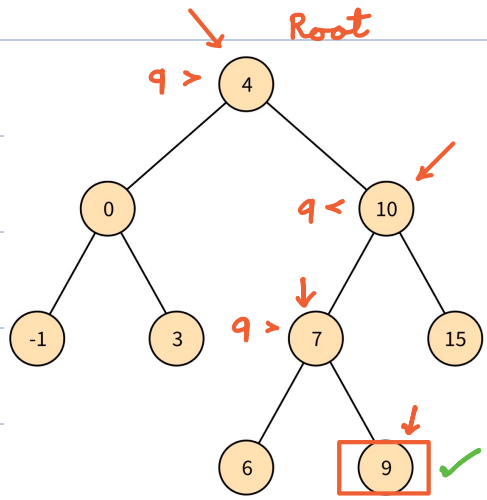
Requirements

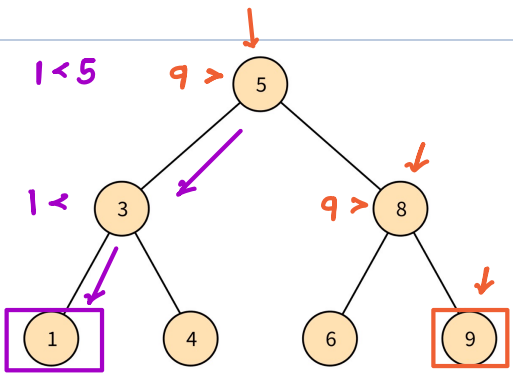
There are two types of requirements :

- **Add Order :** Insert a new record.
- **Find Order Time :** The system should be able to quickly retrieve the time of placing the order, given the unique order ID.

< Question > : Search an element K in Binary Search Tree.

K = 9





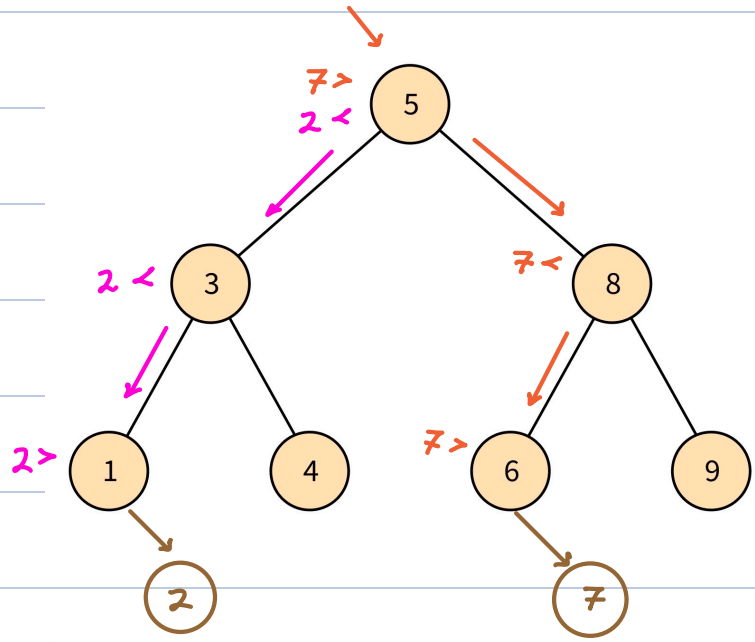

```
temp = root
while (temp != null) {
    if (temp.val == K)  return temp
    if (temp.val > K)   temp = temp.left
    else                temp = temp.right
}

return null
```

TC = O(H) SC = O(1)

Insertion in B.S.T

insert (7)

insert (2)



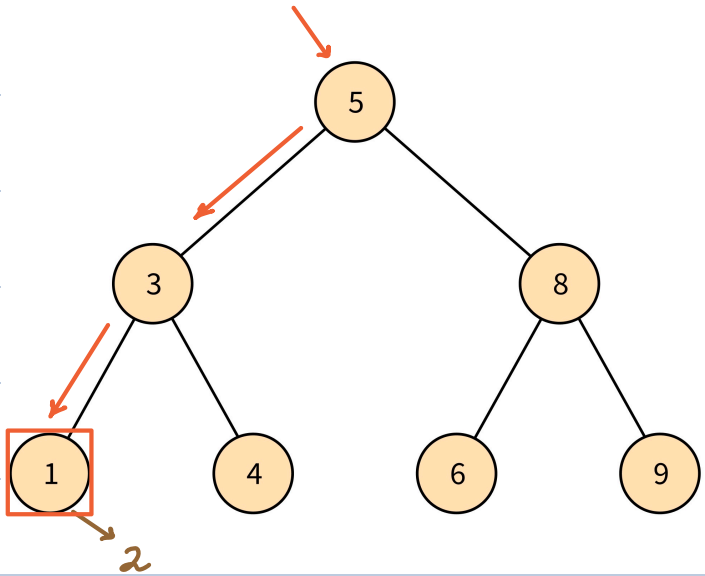
insert (7)

```
temp = root
while (temp != null) {
    if (K <= temp) {
        if (temp.left == null) {
            temp.left = new TreeNode(K)
            break
        } temp = temp.left
    } else {
        if (temp.right == null) {
            temp.right = new TreeNode(K)
            break
        } temp = temp.right
    }
}
```

TC = O(H) SC = O(1)

Find the Smallest Element in B.S.T

```
temp = root
while (temp.left != null) {
    temp = temp.left
}
return temp.
```



TC = O(H)

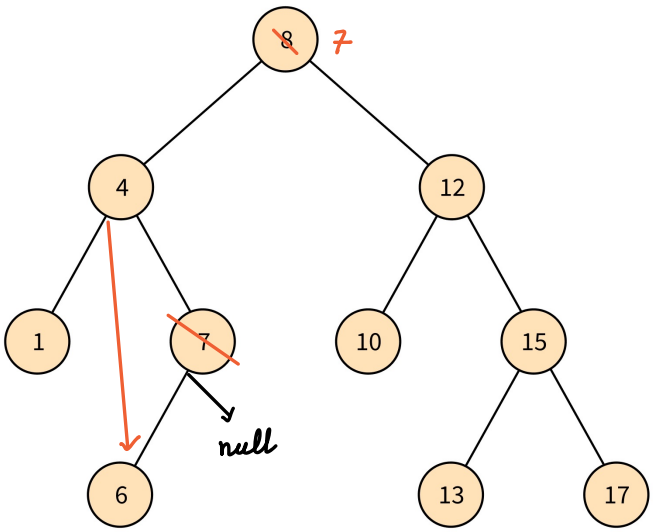
SC = O(1)

< *Question* > : How to find the largest element in Binary Search Tree.

```
temp = root
while (temp.right != null) {
    temp = temp.right
}
return temp
```

TC = O(H) SC = O(1)

Deletion in B.S.T



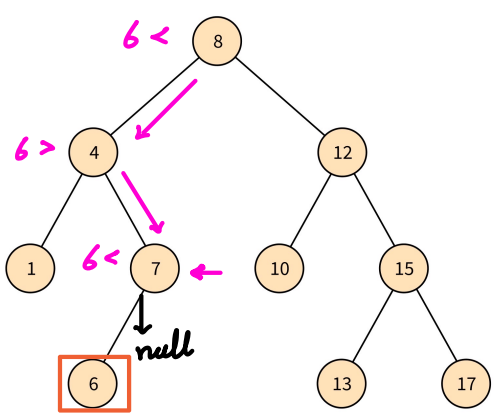
delete (6) ✓

delete (7) ✓

delete (8) ✓

1. Leaf node ✓
2. Node with single child ✓
3. Node with both the children ✓

Leaf Node



1) Travel till parent of K.

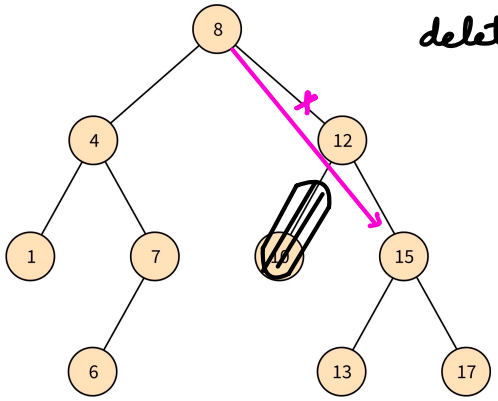
2) Update the child to null.

delete (6) ✓

~~delete (7)~~

~~delete (8)~~

Node with single child



delete (12)

delete (6)

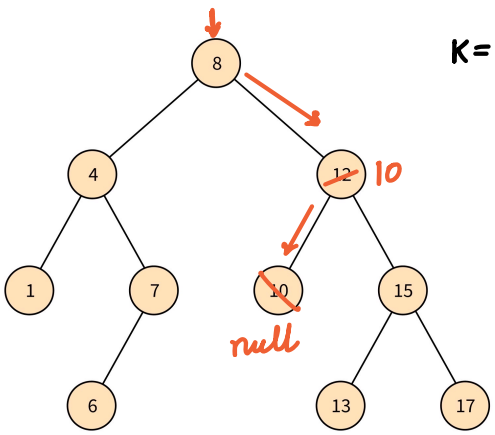
delete (7)

delete (8)

1) Travel till parent of K.

2) Update link to K with link to child of K.

Node with both children



K = 12

~~delete (6)~~

~~delete (7)~~

~~delete (8)~~

1) Travel till node K.

2) Find largest node in left subtree & delete it.

3) Replace K with largest node in left subtree.

```
root = delete ( root , K)
```

```
                                    int
TreeNode  delete ( root , K ) {
    if ( root == null )  return null
    if ( K < root . data )
        root . left = delete ( root . left , K)  ✓
    else if ( K > root . data )
```

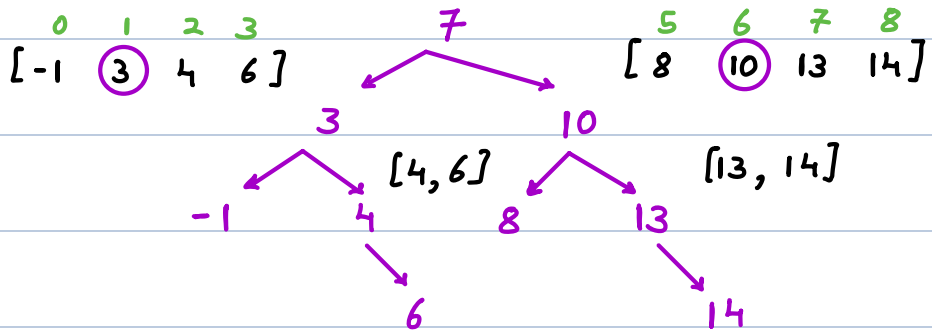
```
    root.right = delete(root.right, K) ✓
else {  // K == root.data
    if (root.left == null &&
        root.right == null)   return null
    if (root.left == null)   return root.right
    if (root.right == null)  return root.left

    temp = root.left
    while (temp.right != null)
        temp = temp.right
    }
    root.data = temp.data
    root.left = delete(root.left, temp.data)
    return root
}
```

TC = O(H) SC = O(H)

H.W → Read about Red-Black Tree & AVL Tree } Balanced BST.

Construct B.S.T from sorted array

arr → [-1 , 3 , 4 , 6 , 7 , 8 , 10 , 13 , 14]
0 1 2 3 4 5 6 7 8

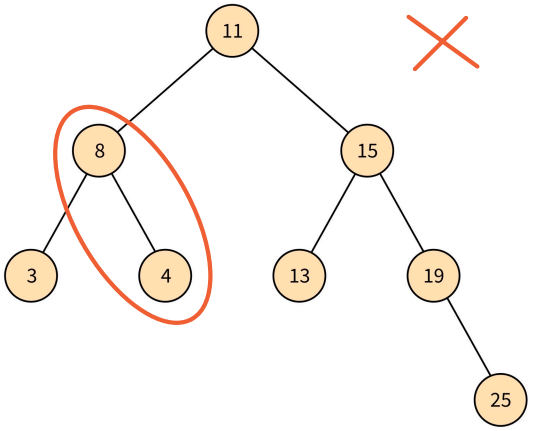


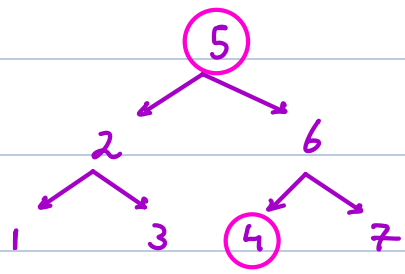

```
TreeNode build ( A[], L, R) {
    if ( L > R )  return null
    m = (L+R) / 2
    root = new TreeNode(A[m])
    root.left = build (A, L, m-1)
    root.right = build (A, m+1, R)
    return root
}
```

(L: 0, R: N-1)

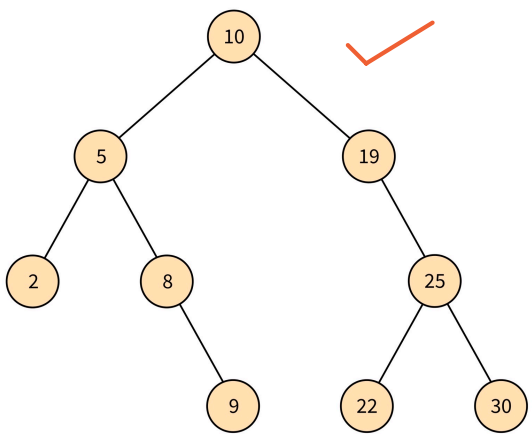
TC = O(N) SC = O(log (N))

Check if given Binary Tree is a B.S.T





Not a BST



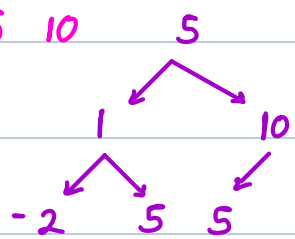
Sol 1 → Left Node Right

< >

Inorder traversal is sorted.

Unique data ✓

→ in = -2 1 5 5 5 10



Not a BST

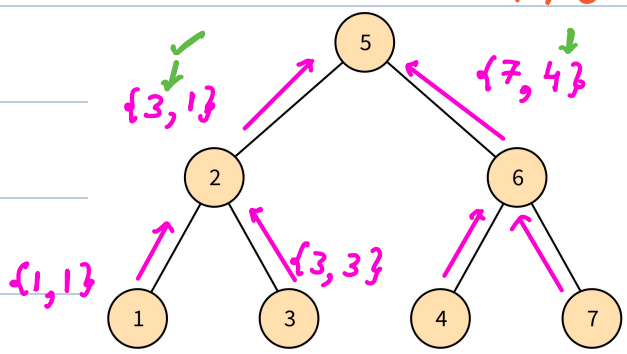
4 ≯ 5

Sol 2 →

∀ nodes x,

left subtree data <= x

right subtree data > x



largest in left subtree <= x {max, min}

& smallest in right subtree > x ✓