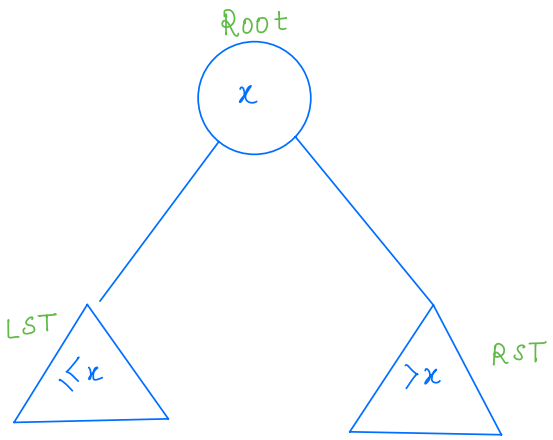
Lecture :- BST

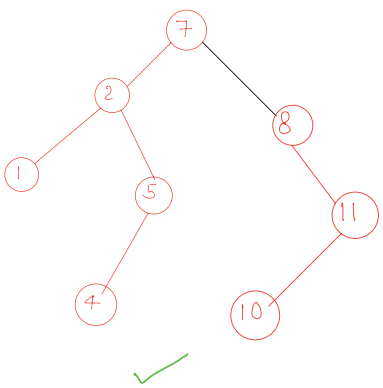
Agenda

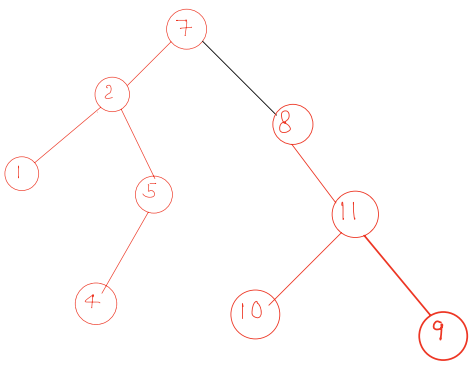
- Introduction
- Operations
- Construct BST from sorted array.
- Check BST.

Binary search tree

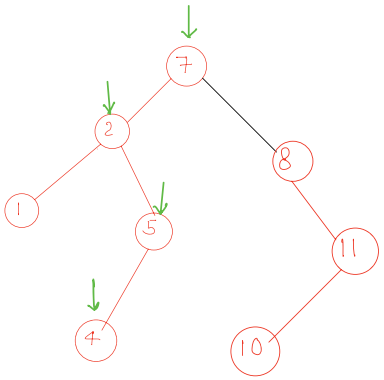


Ex:



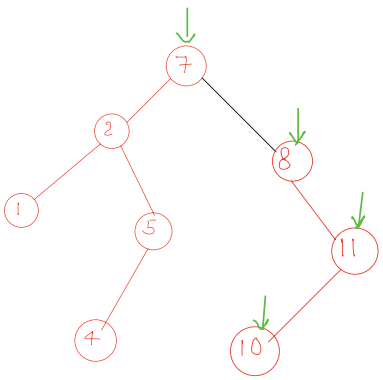


Searching in binary search tree



true.

K = 4



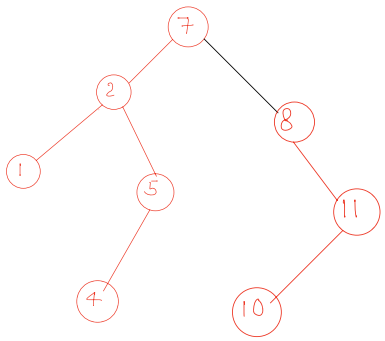
true

K = 10

Pseudocode

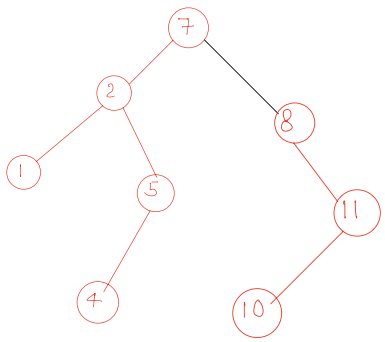
```
boolean search ( TreeNode root , int k)  {
        if ( root == null) {
            return false;
        }
        if ( root. data == k) {
            return true;
        }
        if ( root. data > k) {
            return search( root. left, k);
        }
        return search( root. right, k);
}
```

Qu find smallest in binary search tree



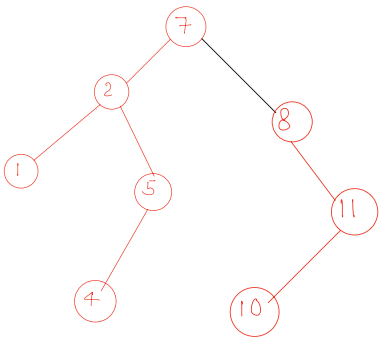

```
Node findSmallest(Node root) {
    temp = root;
    while(temp.left != null) {
        temp = temp.left;
    }
    return temp;
}
```

Qu find largest in binary search tree

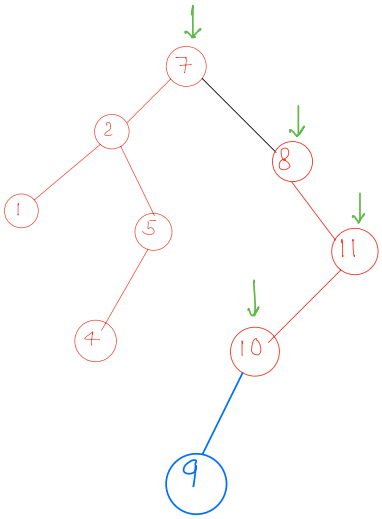


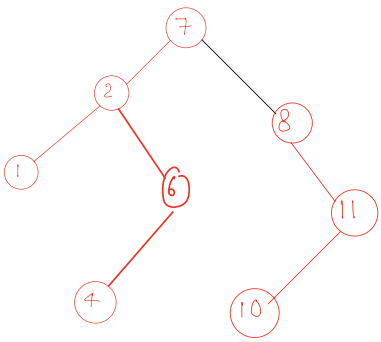

```
Node findLargest (Node root) {
    temp = root;
    while( temp.right != null) {
        temp = temp.right;
    }
    return temp;

}
```

Insertion in Binary search tree

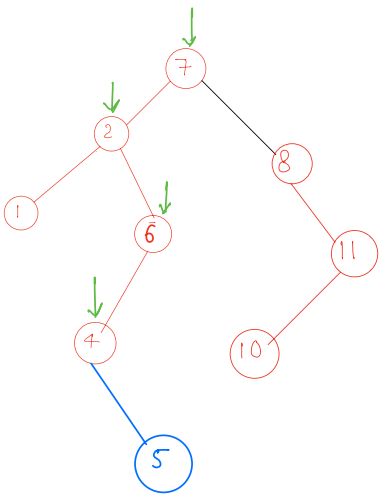


K = 9



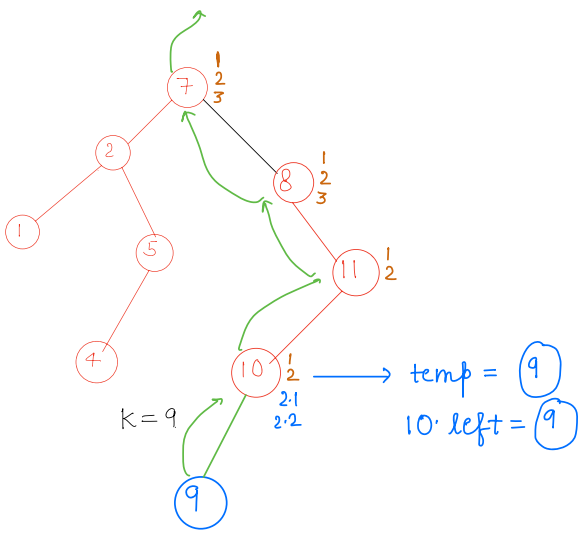


K = 5



Pseudocode

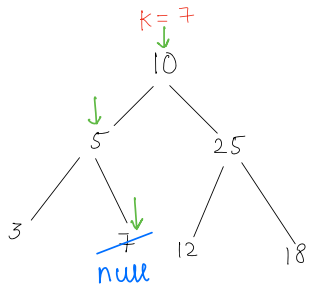
```
TreeNode insert ( TreeNode root, int K) {
    if (root == null) {
        return new TreeNode(K);
    }
    if ( K < root.data) {
      root.left = insert(root.left, K);
    } else {
      root.right = insert(root.right, K);
    }
    return root;
}
```



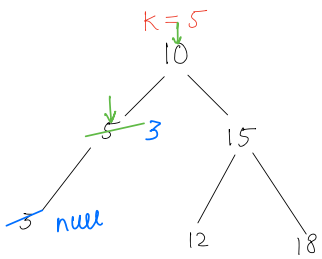

```
TreeNode insert ( TreeNode root, int K) {
1   if (root == null) {
        return new TreeNode(K);
    }
2   if ( K < root.data) {
2-1   temp = insert(root.left, K);
2-2   root.left = temp;
3   } else {
      root.right = insert(root.right, K);
    }
4   return root;
}
```

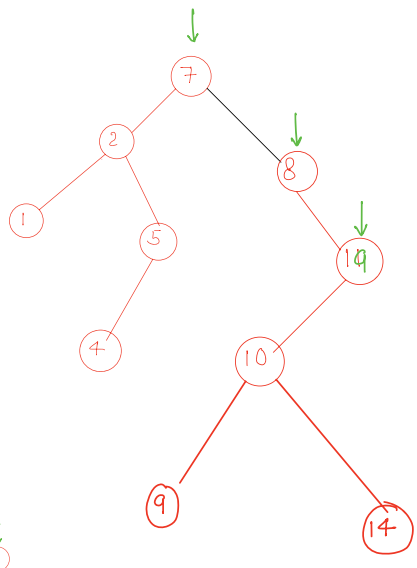
Deletion in Binary search tree.

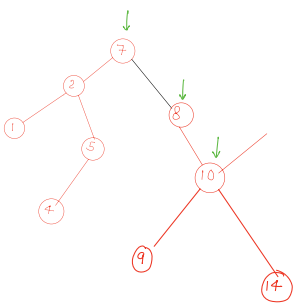
Case1 Leaf node [Node with no children]



Case2: Node with one child.



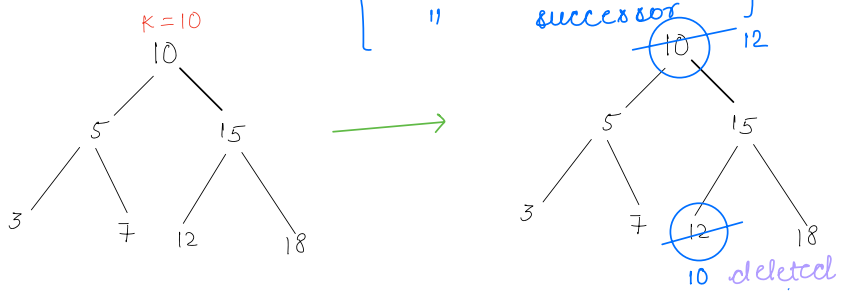


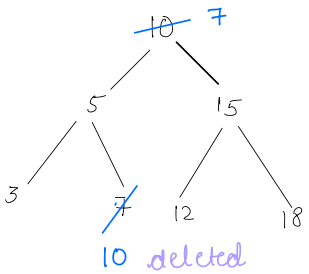


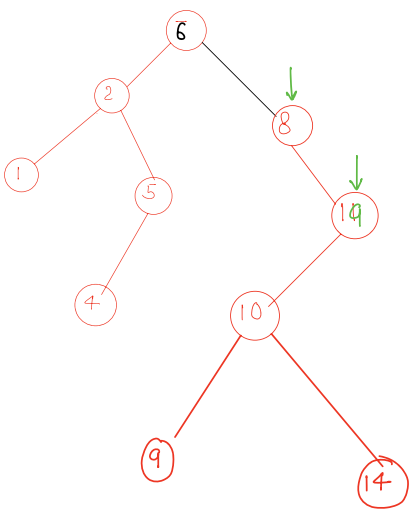
K=19

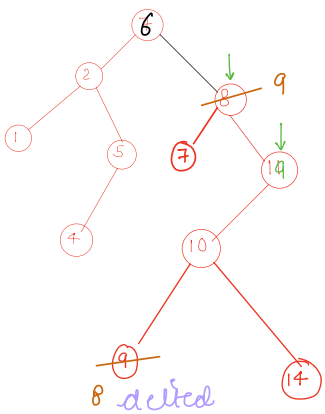
K=19

Case3: Node with two children. [in-order predecessor || successor]









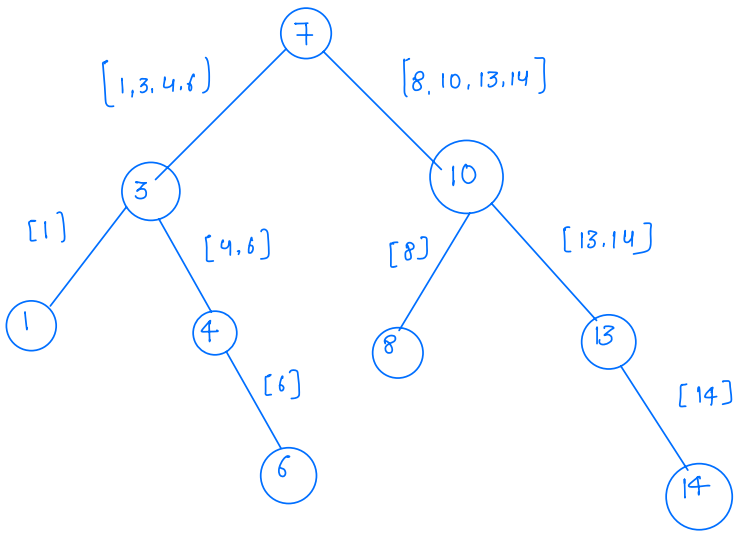
K=8.

1. Look for inorder predecessor / successor. = x
2. swap (given node, x)
3. Delete given node [leaf node]

r inorder predecessor ⟶ Go to left child, try for rightmost child

Qu Construct a binary search tree from sorted array of unique elements. [Medium – hard]

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14



Pseudocode

```
Node sortedArrayToBST(int[] arr, int s, int e) {
    if(s==e) {
        return new TreeNode(arr[s]);
    }
    int mid = (s+e)/2;
    TreeNode root = new Node(arr[mid]);
    root.left  = sortedArrayToBST(arr, s, mid-1);
    root.right = sortedArrayToBST(arr, mid+1, e);
    return root;
}
```

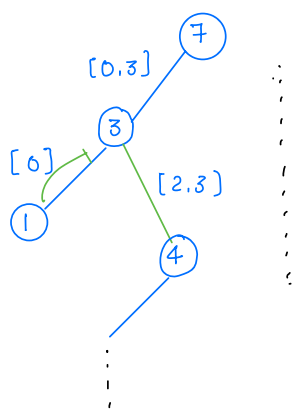
TC: O(n)

SC: O(logn)

└ height of tree

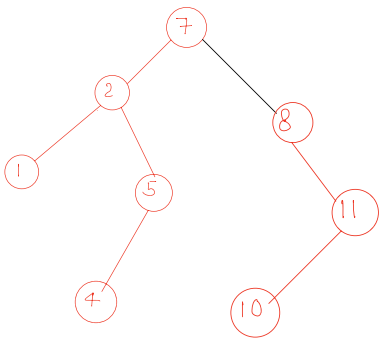
s = 0, e = 8

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14




```
Node sortedArrayToBST(int[] arr, int s, int e) {
    if(s==e) {
        return new TreeNode(arr[s]);
    }
    int mid = (s+e)/2;
    TreeNode root = new Node(arr[mid]);
    root.left  = sortedArrayToBST(arr, s, mid-1);
    root.right = sortedArrayToBST(arr, mid+1, e);
    return root;
}
```

Qu check if binary tree is a binary search tree?



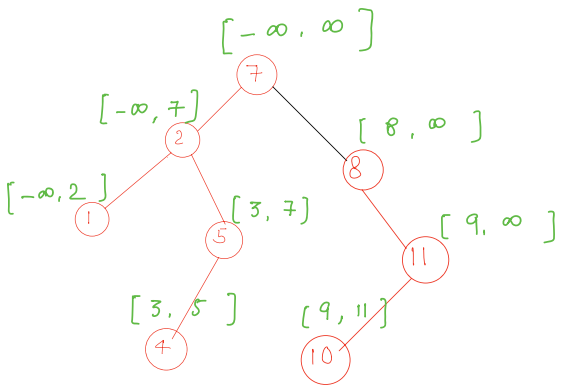
Approach 1:

Inorder traversal. —— non-duplicate

1 2 4 5 7 8 10 11 —— sorted.

```
if ( inorder traversal = sorted) {
        BST.
} else {
        not BST.
}
```

TC: O(n)
SC: O(n)



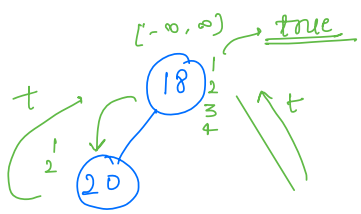
<= left
> = right

< = left
>= right

non-duplicates

```
boolean isBST(root, min, max) {
1 if(root==null) {
    return true;
  }
2 if(root.left==null && root.right==null) {
    return true;
  }
3 if(root.data < min || root.data > max) {
    return false;
  }
4 left = isBST(root.left, min, root.data);
5 right = isBST(root.right, root.data+1, max);
6 return left && right;
}
```

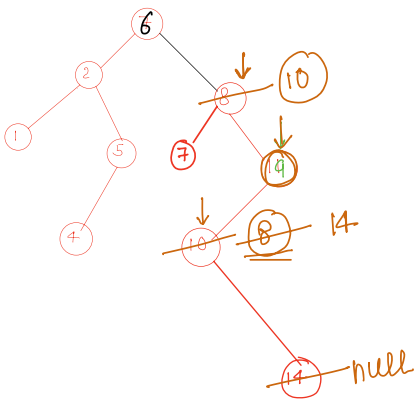
Wrong

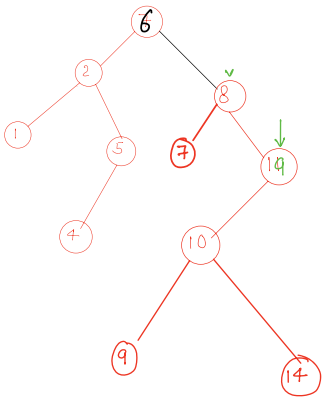


TC: O(n)
SC: O(logn) , O(n)
↑
height of tree

Thankyou :)

Doubt





delete = 8