Lecture :- Trees 3

Agenda

- Introduction
- Insert | search | delete
- Is BST?
- Construct BST from sorted array.

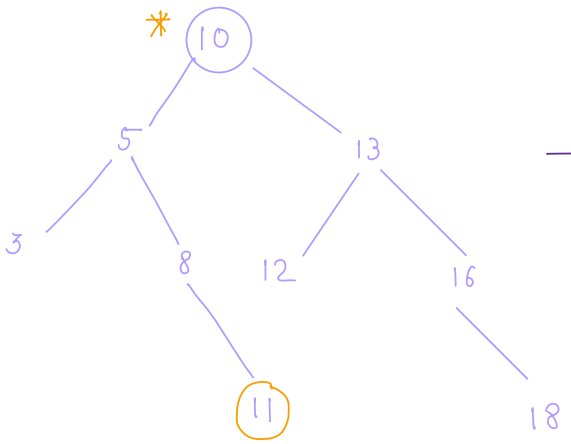
BST Binary search tree

A binary tree with some conditions:

for all nodes:

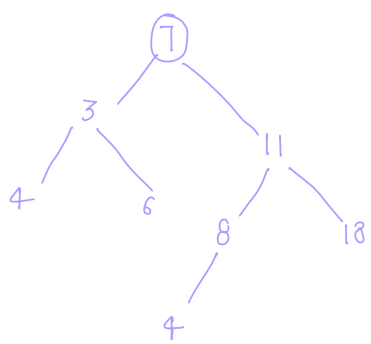
all el in LST < root < all el in RST.

1)



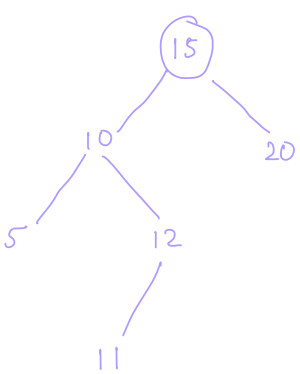
— Invalid BST

2)



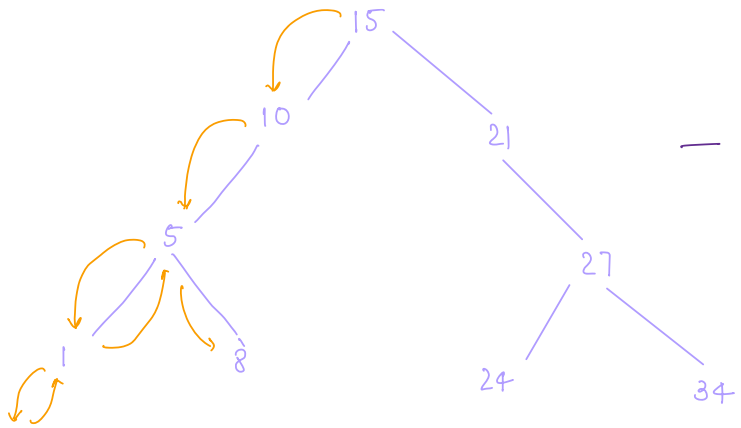
— Invalid BST

3)



— Valid BST.

Interesting property



— Valid BST

Inorder: 1 5 8 10 15 21 24 27 34

L Data Right

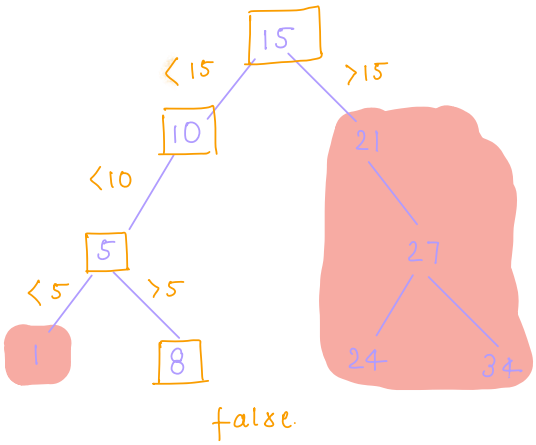
[Sorted]

Reason: L Data Right

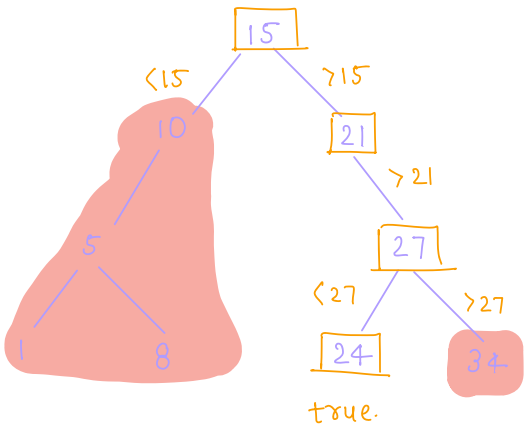
< data > data

Observation Inorder traversal of BST is a sorted array.

Qu. Searching in a BST.

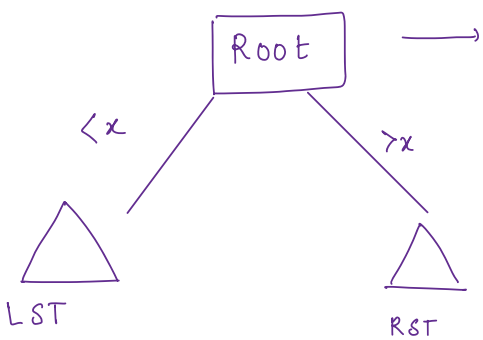


K=9



K=24

Logic:



root · data == x true

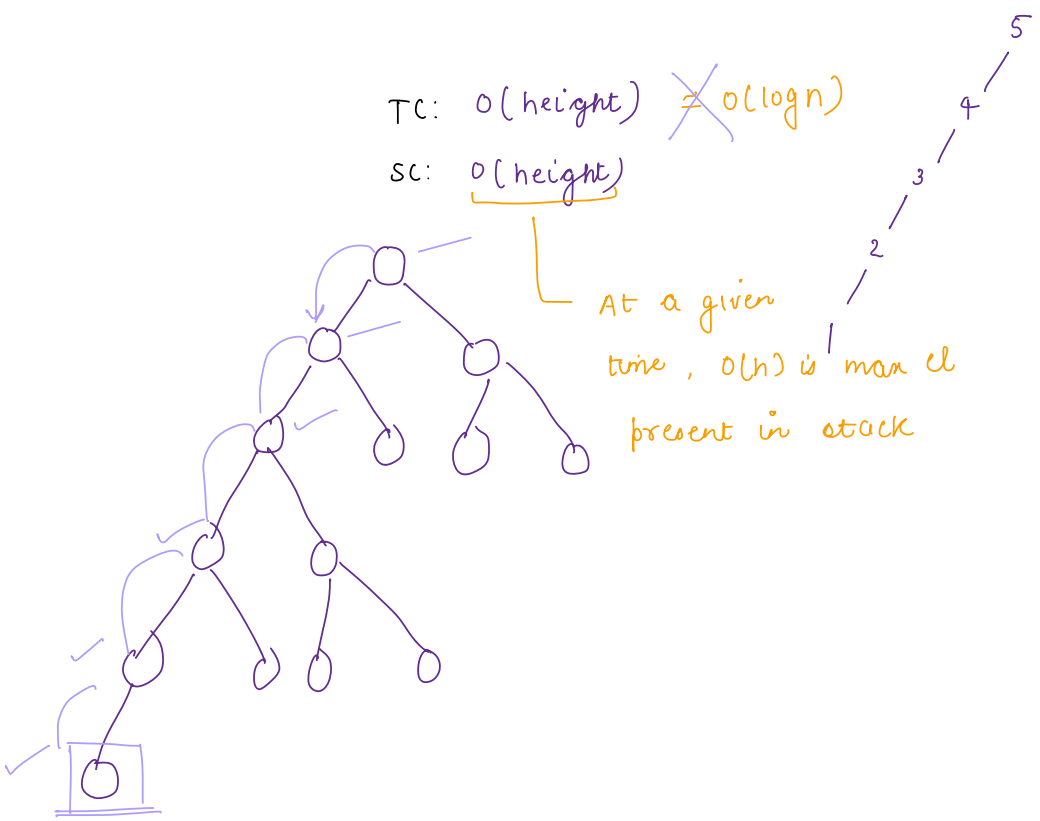
else — do something

Code:

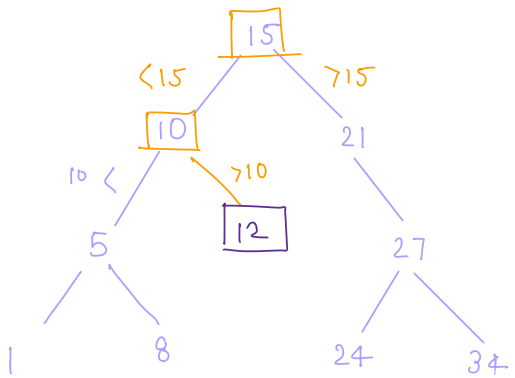
```
boolean search ( Node root , int x) {
        if( root == null) {
            return false;
        }
        if( root.data == x) {
            return true;
        }
        if( root.data > x) {
            return search( root.left , x);
        }
        return search( root.right ,x);
}
```

TC: O(height) ≠ O(log n)

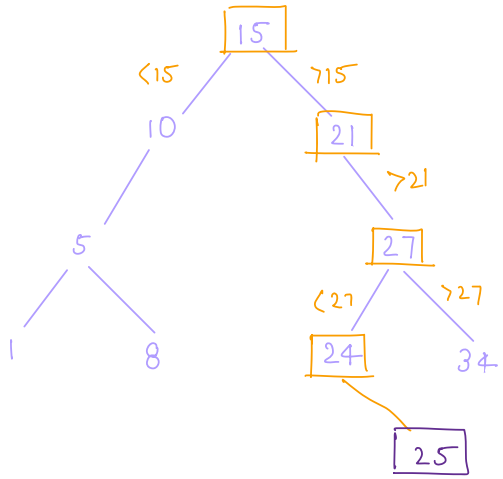
SC: O(height)



Qu Insertion in a BST.

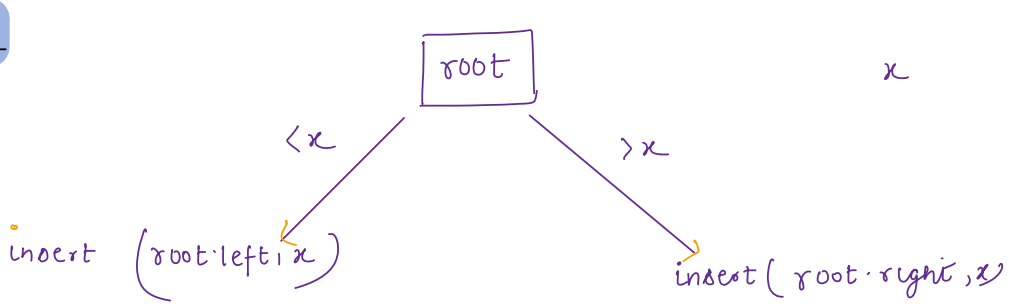


K = 12



K = 25

Logic:

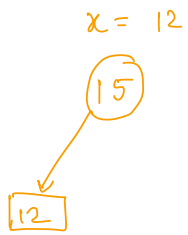


Code:

```
Node insert(Node root, int x) {

    if(root == null) {
        Node xn = new Node(x);
        return xn;
    }

    if(root.data > x) {
        root.left = insert(root.left, x);
    } else {    // root.data < x
        root.right = insert(root.right, x);
    }

    return root;
}
```

Root = null

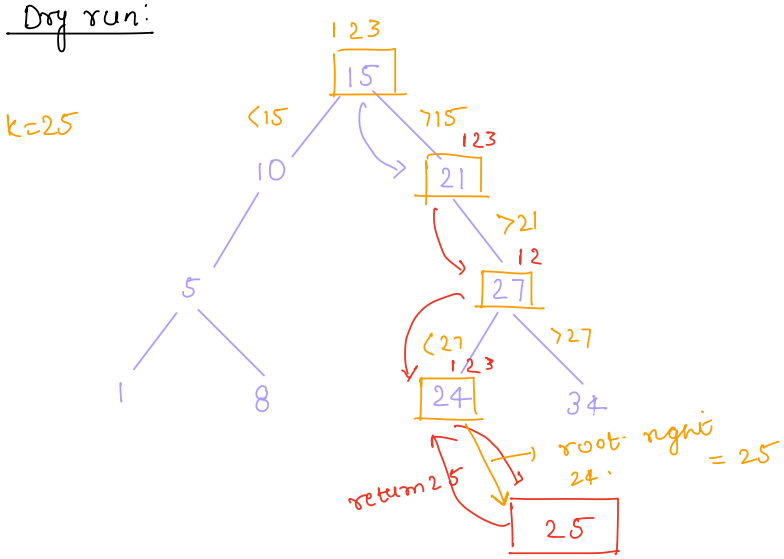
x = 25



TC: O(height)

SC: O(height)

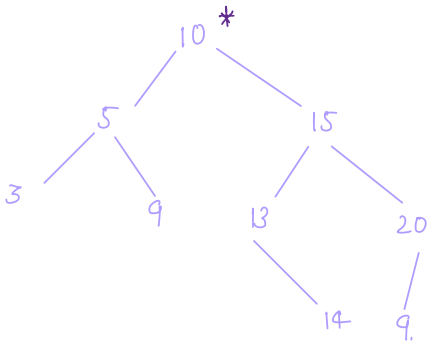
Dry run:




```
Node insert(Node root, int x) {
1.  if(root == null) {
        Node xn = new Node(x);
        return xn;
    }
2.  if(root.data > x) {
        root.left = insert(root.left, x);
3.  } else {    // root.data < x
        root.right = insert(root.right, x);
    }
4   return root;
}
```

Qu Given a Binary tree, check if it is a BST?

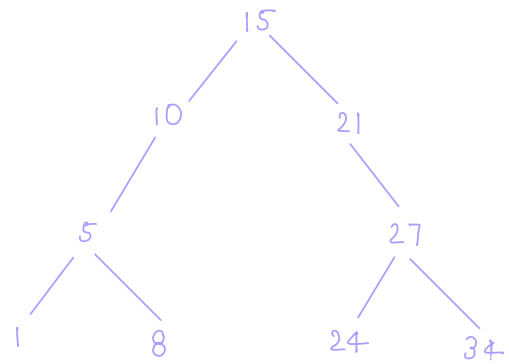
1.>



—— NO

Inorder: 3 5 9 10 13 14 15 9 20 [Unsorted]

2.>



⟶ Yes

Inorder: 1 5 8 10 15 21 24 27 34 [sorted]

Ideal: calculate inorder traversal of BT.

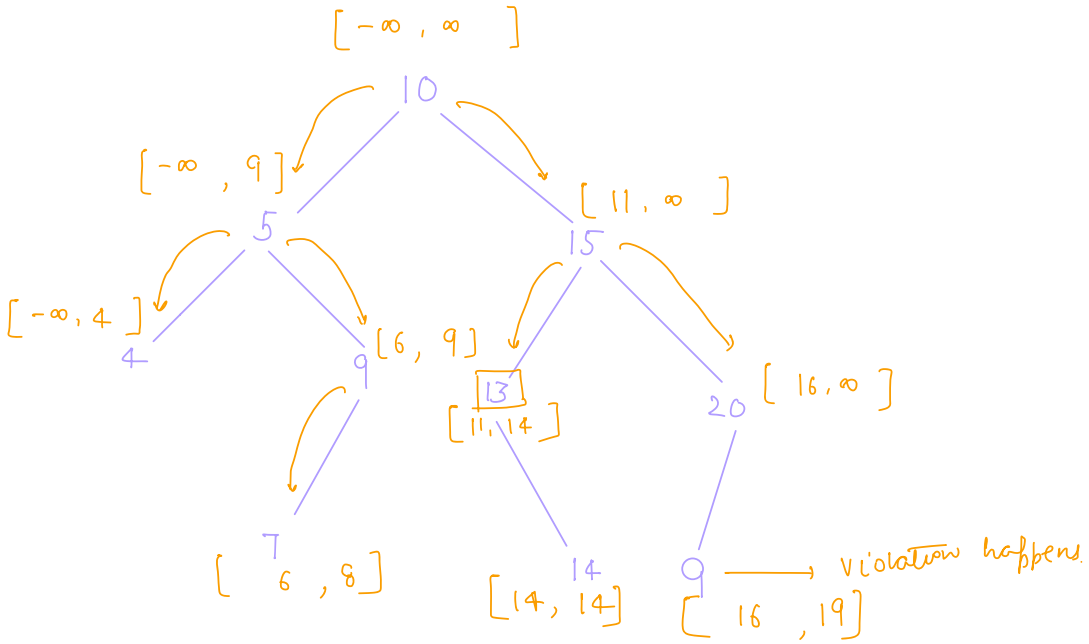
if inorder == sorted ⟶ BST

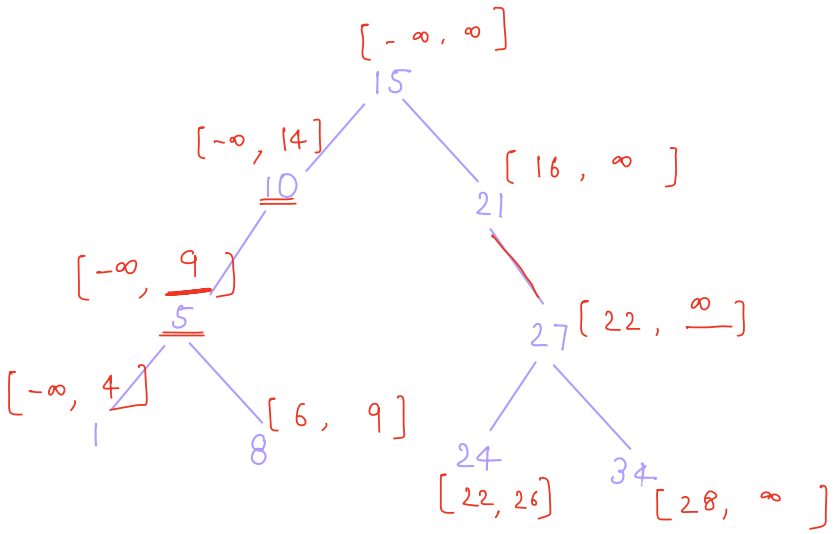
else ⟶ Not a BST.

TC: O(n)

SC: O(n)

Idea2: Space optimisation





Code:

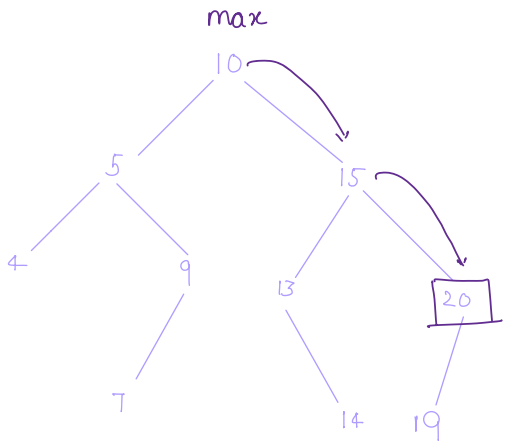
```
boolean isBST(Node root, s, e) {
1.  if(root == null) {
        return true;
    }
Violation — 2  if(root.data < s || root.data > e) {
        return false;
    }
3   boolean left = isBST(root.left, s, root.data-1);
    if(left == false) {
        return false;
    }
4   boolean right = isBST(root.right, root.data+1, e);
    if(right == false) {
        return false;
    }
5.  return true;
}
```

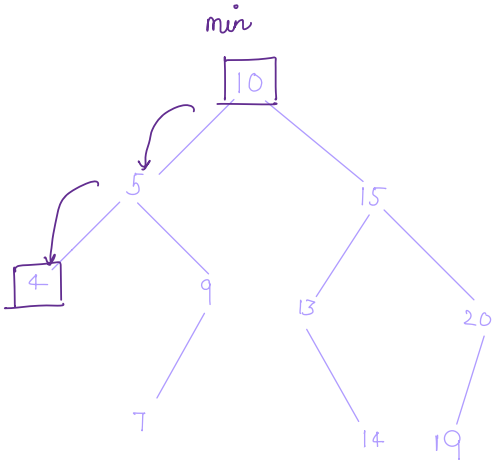
TC: O(n)

SC: O(height)

Break: 8:41 AM

Qu: Max | min a BST






```
Node findMax(root) {
    curr = root;
    while(curr.right != null) {
        curr = curr.right;
    }
    return curr;
}
```

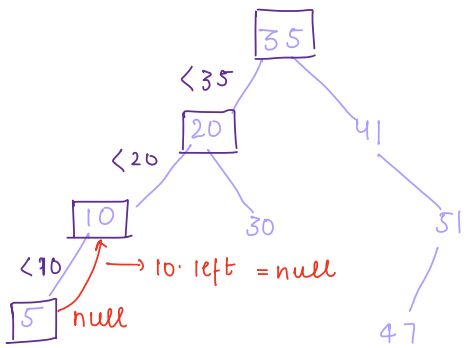
```
Node findMin(root) {
    curr = root;
    while(curr.left != null) {
        curr = curr.left
    }
    return curr;
}
```

Qu: Delete a node in a BST.

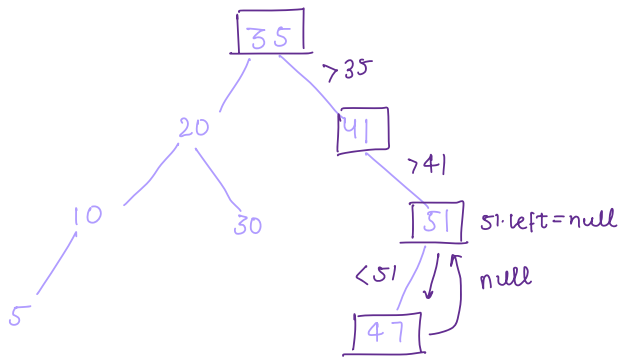
Cases:

1. Node: 0 children
2. Node: 1 "
3. Node: 2 "
4. Node not present.

case1: Node having 0 child

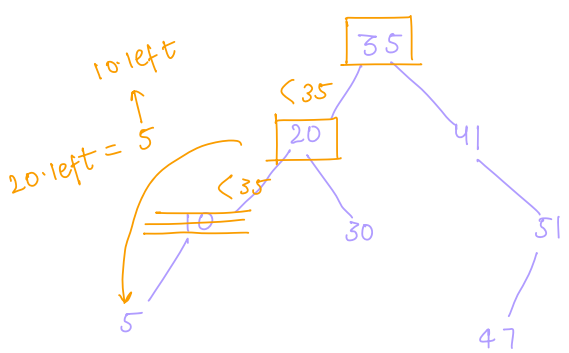


K = 5

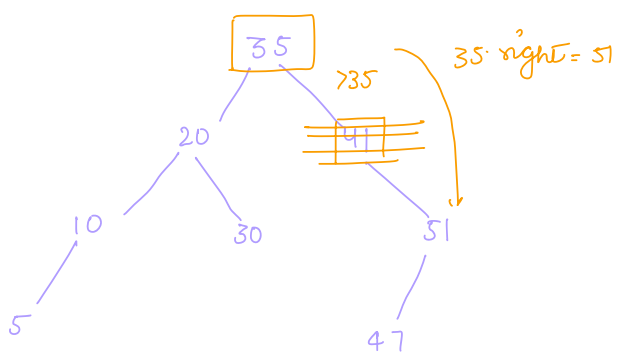


K = 47.

case2: Node having 1 child.

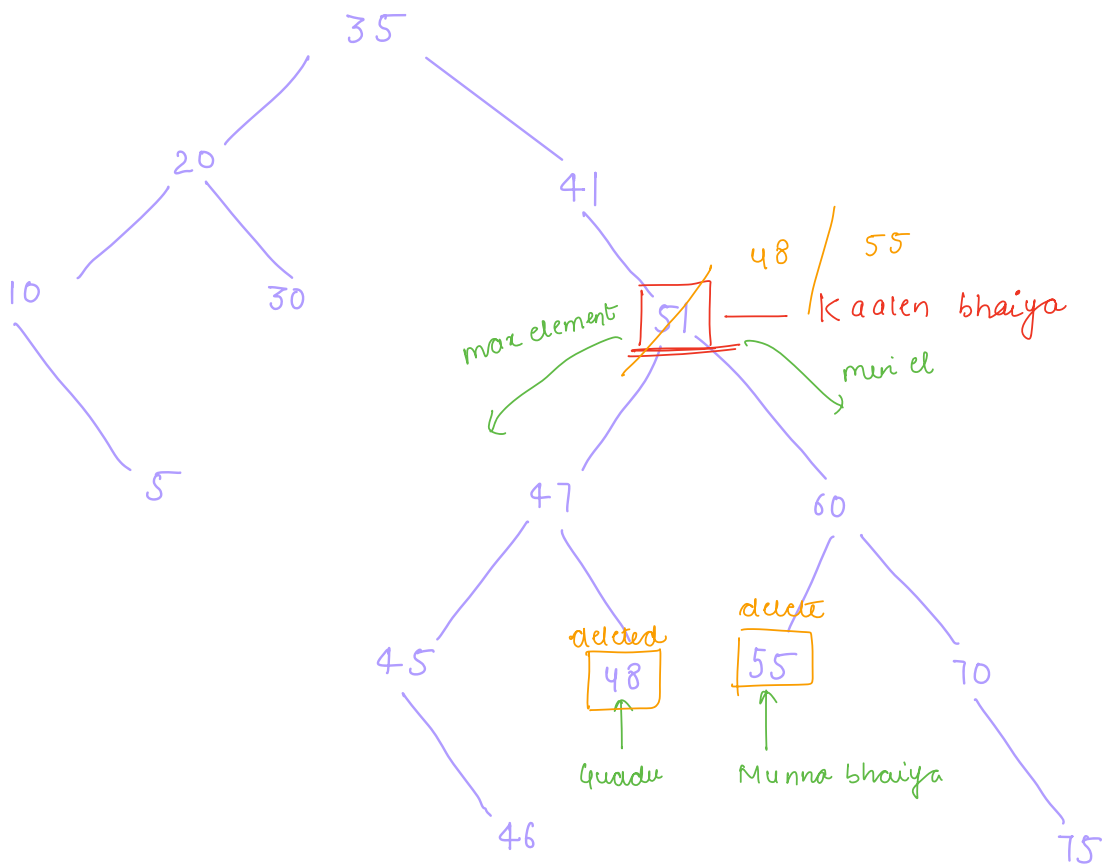


K = 10



K = 41

Case3: Node: 2 children



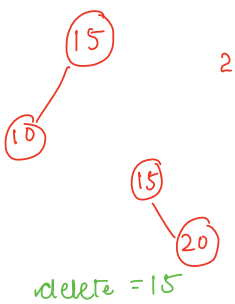
K = 51

Code:

```
Node delete (Node root, int x) {
    if (root.data > x) {
        root.left = delete (root.left, x);
    } else if (root.data < x) {
        root.right = delete (root.right, x);
    } else {
        1.  if (root.left == null && root.right == null) {
                return null;
            }
        2.  if (root.left == null) {
                return root.right;
            }
            if (root.right == null) {
                return root.left;
            } else {
                int max = findMax (root.left);
                root.data = max;
                root.left = delete (root.left, max);
            }
        return root;
    }
```

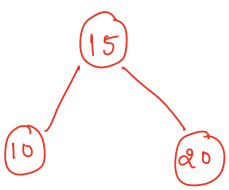
x = 15

0 child



delete = 15

1 child



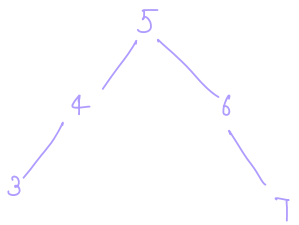
delete = 15

TC:

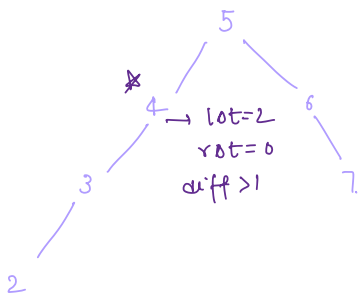
SC: O(height)

Qu: Construct a balanced BST using sorted array.

Balanced binary search tree → diff of height of LST & rst <=1



→ Yes

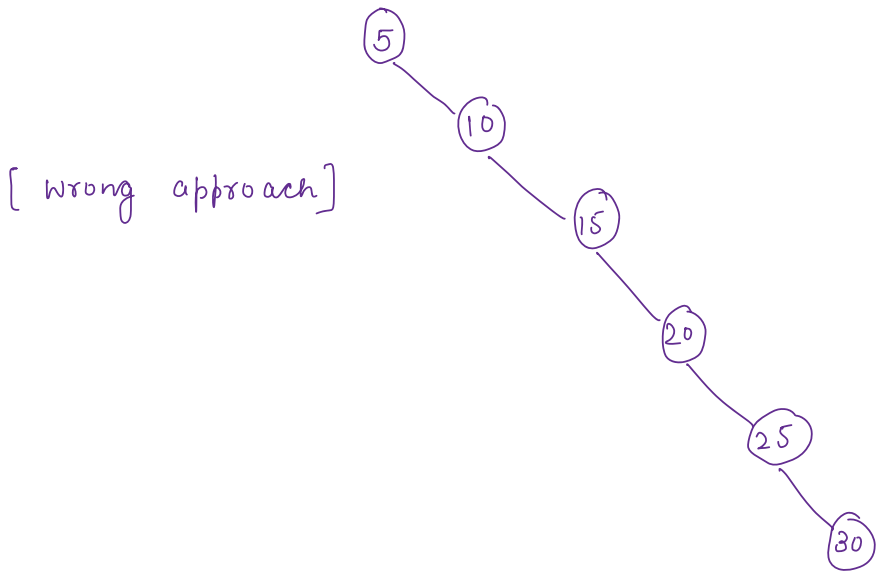


→ NO

Ideal:

Qu Given a sorted array, **construct balanced bst.**

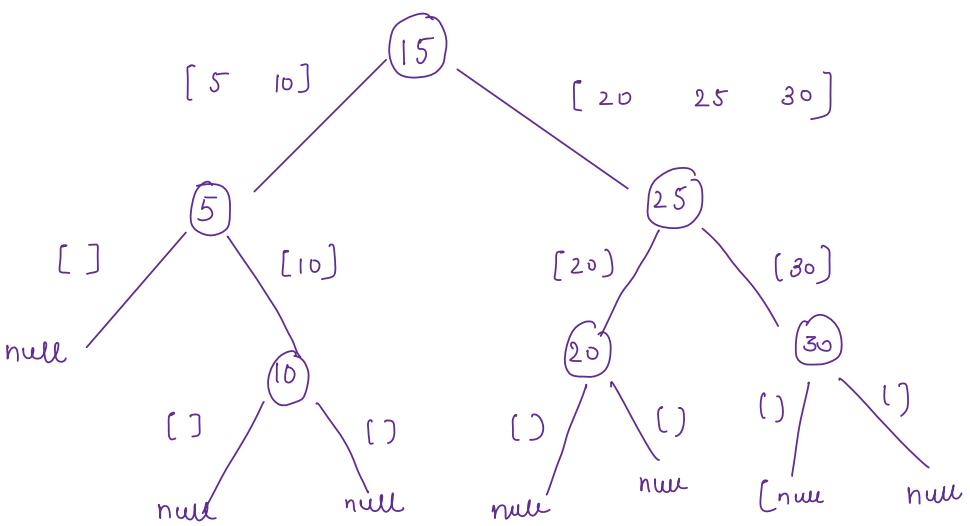
Sol arr[] = [5 10 15 20 25 30]



[wrong approach]

Idea2:

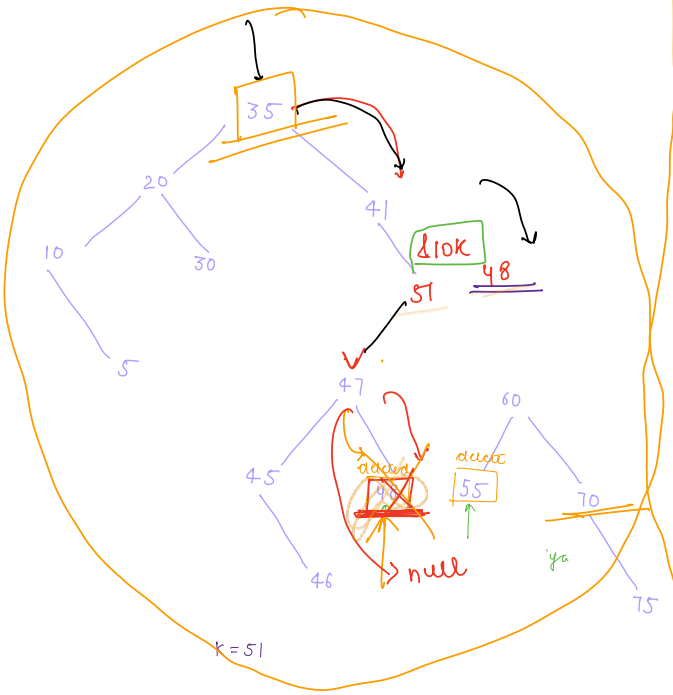
0	1	2	3	4	5
5	10	15	20	25	30



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

Thank you

Doubt:




```
Node delete (Node root, int x) {
1. if (root.data > x) {
        root.left = delete (root.left, x);
2. } else if (root.data < x) {
        root.right = delete (root.right, x);
  } else {
3   if (root.left == null && root.right == null) {        } 0 child
        return null;
    }
4   if (root.left == null) {                              } 1 child
        return root.right;
    }
5   if (root.right == null) {
        return root.left;
6   } else {
        int max = findMax (root.left);
        root.data = max;
        root.left = delete (root.left, max);
    }
7   return root;
}
```