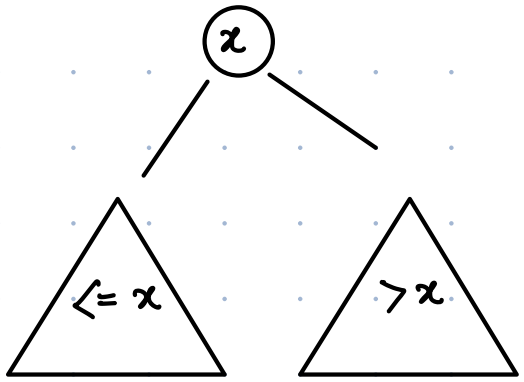
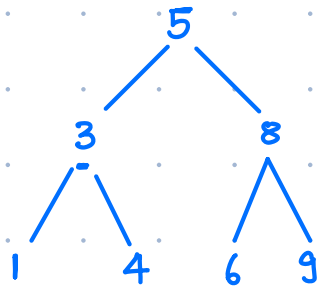
Binary Tree

↓

Binary Search Tree



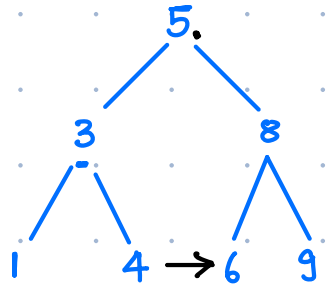


Q. A binary tree where all nodes, x

everything on left have data <= x

and on right > x

- Searching

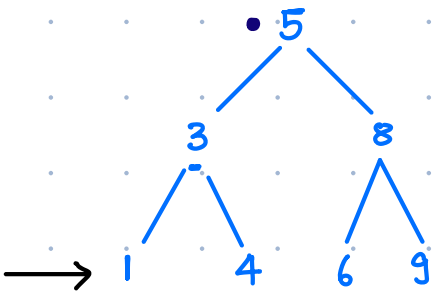


K = 6

1) 5 K = 6

2) 8 K = 6

3) 6 K = 6



K = 1

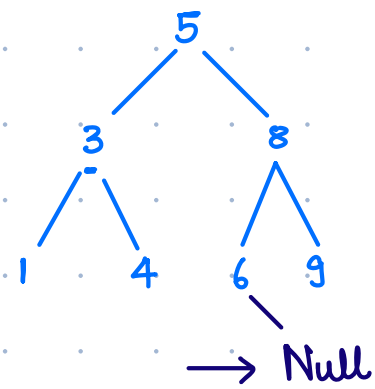
1) 5 K = 1 Go to Left

2) 3 K = 1 Go to Left

3) 1 K = 1 ✓

```
function Node search ( Node root, int K ) {
    if root == NULL
        return NULL
    if root.value == K :
        return root
    else if root.value > K :
        return search (root.left, K)
    else if root.value < K :
        return search (root.right, K)
```

Dry Run



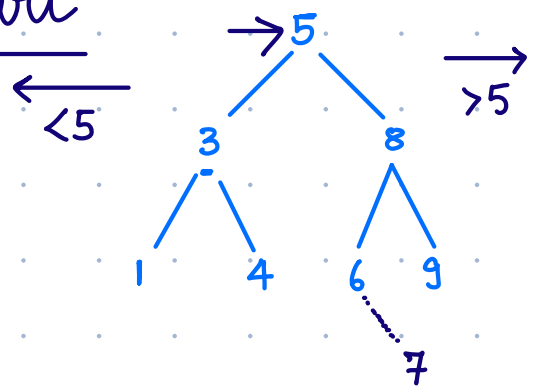
K = 7

5	7	Right
8	7	Left
6	7	Right
Null	7	

max. no of searches = Depth of tree

TC = O(Depth of tree)

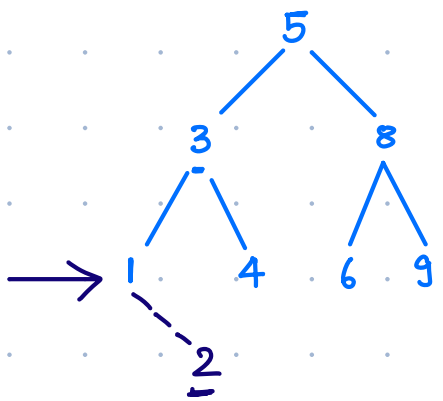
Insert



7

Search the place & then insert

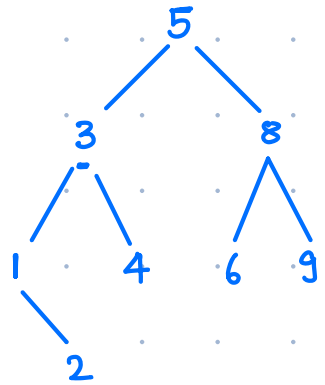
2

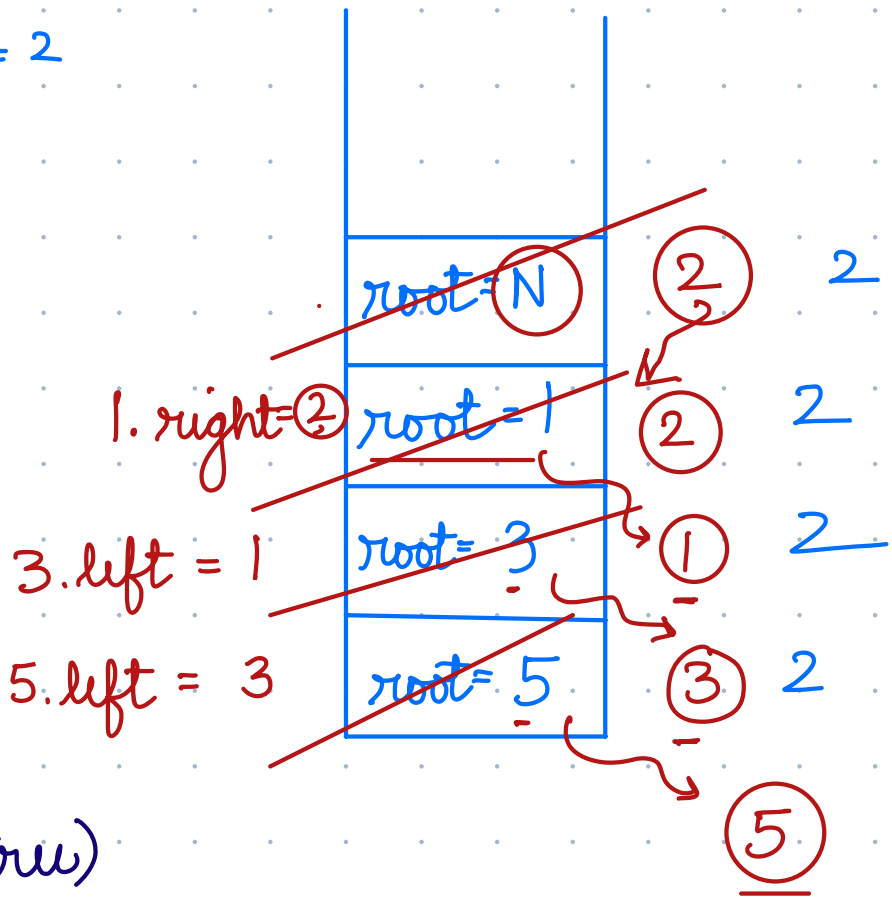


All the new nodes will be inserted as leaf nodes

```
function insert (root, value) {
  if (root == NULL) {
      Node newNode = new Node(value);
      return newNode;
  }
  if (value < root.value) {
      root.left = insert(root.left, value)
  }else if (value > root.value) {
      root.right = insert(root.right, value);
  }
  return root
}
```

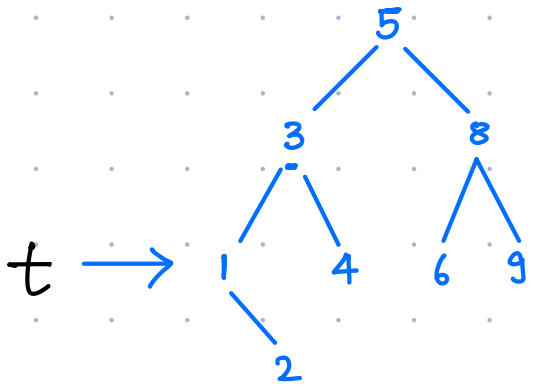
Dry Run → Value = 2





TC = O(Depth of tree)

min of BST



temp = root

while (temp.left != NULL)

Keep moving on left

temp = root

while (temp.left != Null) {

temp = temp.left

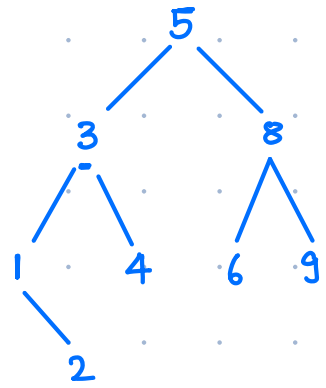
}

return temp

Worst case : TC : O(N)

TC = O(Depth of tree)

SC : O(1)

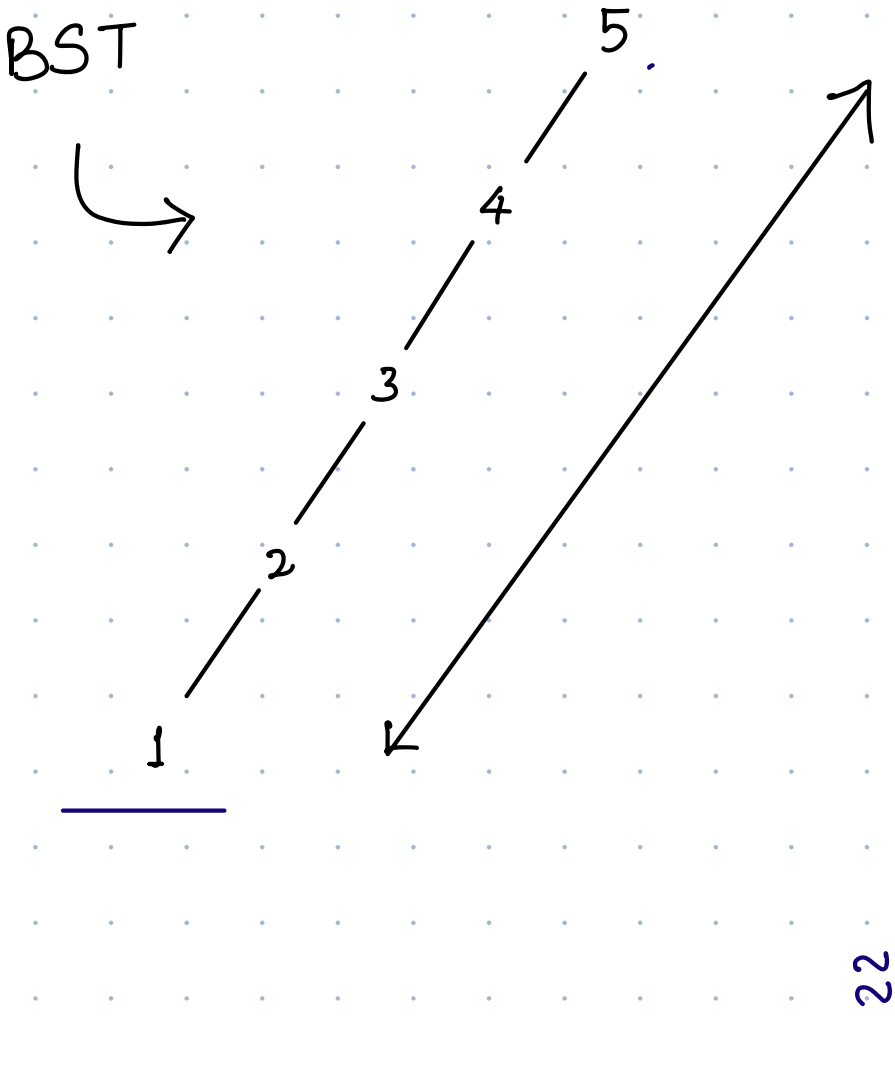


height = 3

↓

comparable to height

BST



O(N)

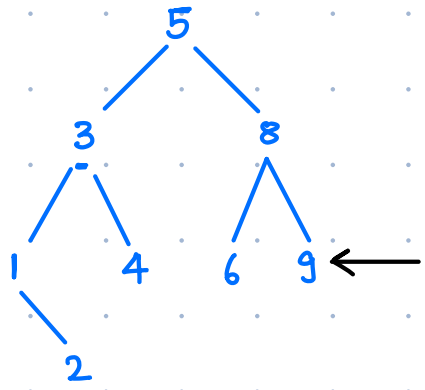
1 → 4 c

-2 → 5 comp

≃ Depth of tree

Max. in BST

- Last Node of Right Subtree
  Or
  Rightmost Node




```
temp = root
while (temp.right != NULL)
    temp = temp.right
return temp.value
```

TC: O(N)
SC: O(1)

TC = O(Depth of tree)

Depth

Root

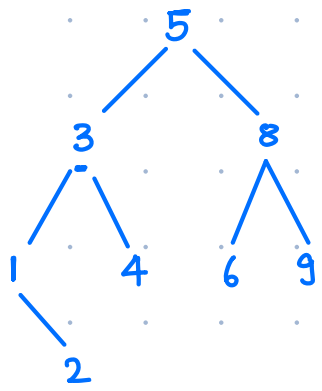
leaf

height

Leaf

root

Delete in a BST



Find that Node

3

[2 children]

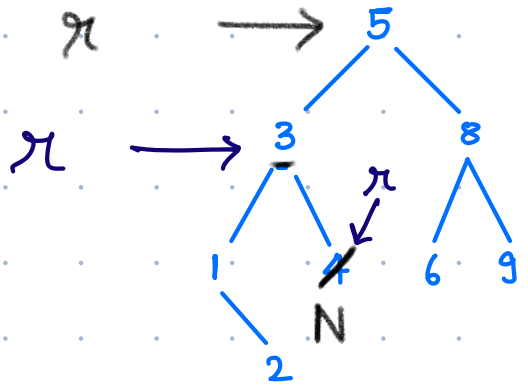
1

[1 child]

4

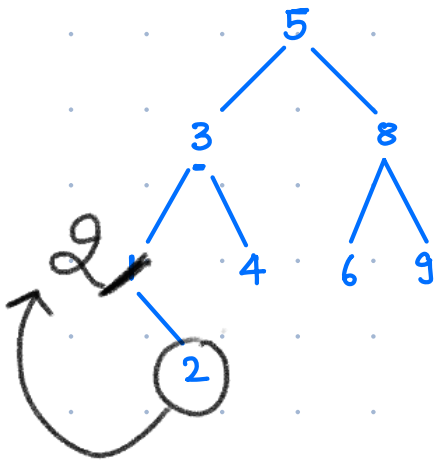
[No Child]

[4]

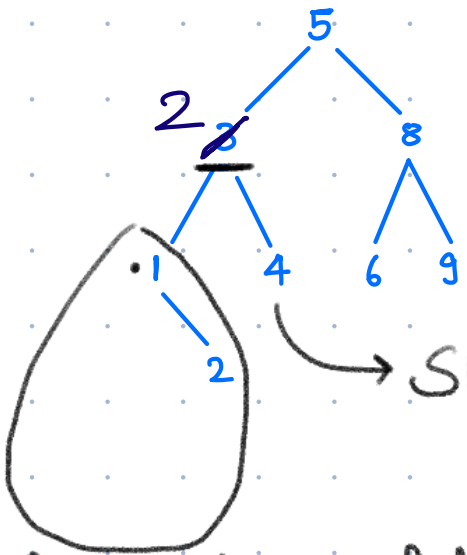


3. right = NULL

[1]



[3]



smallest - right side

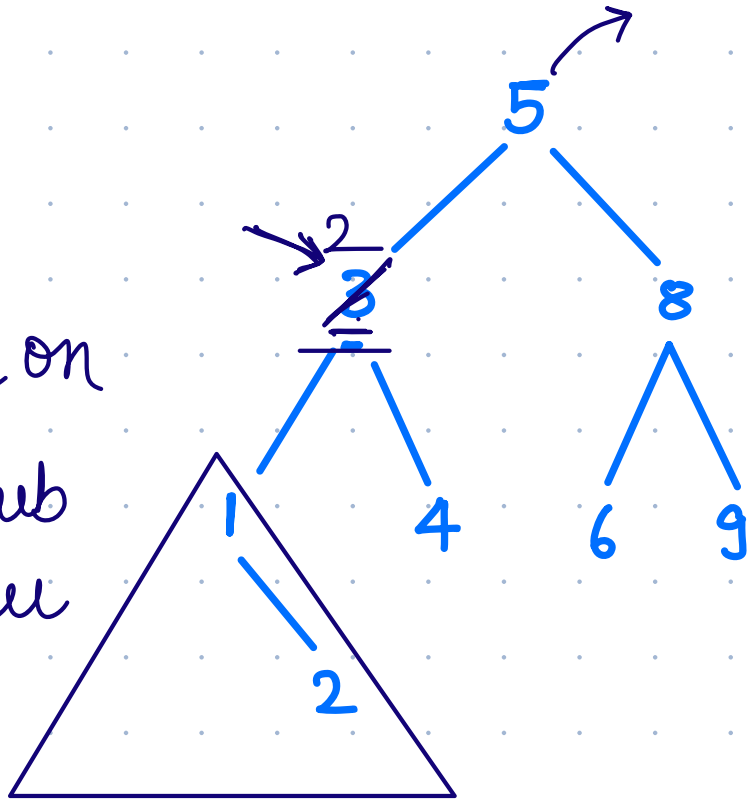
max = 2

delete max in left subtree

largest on left side

Delete 3

↳ largest on Left sub tree



1) max = 2

2) Overwrite 3 with 2

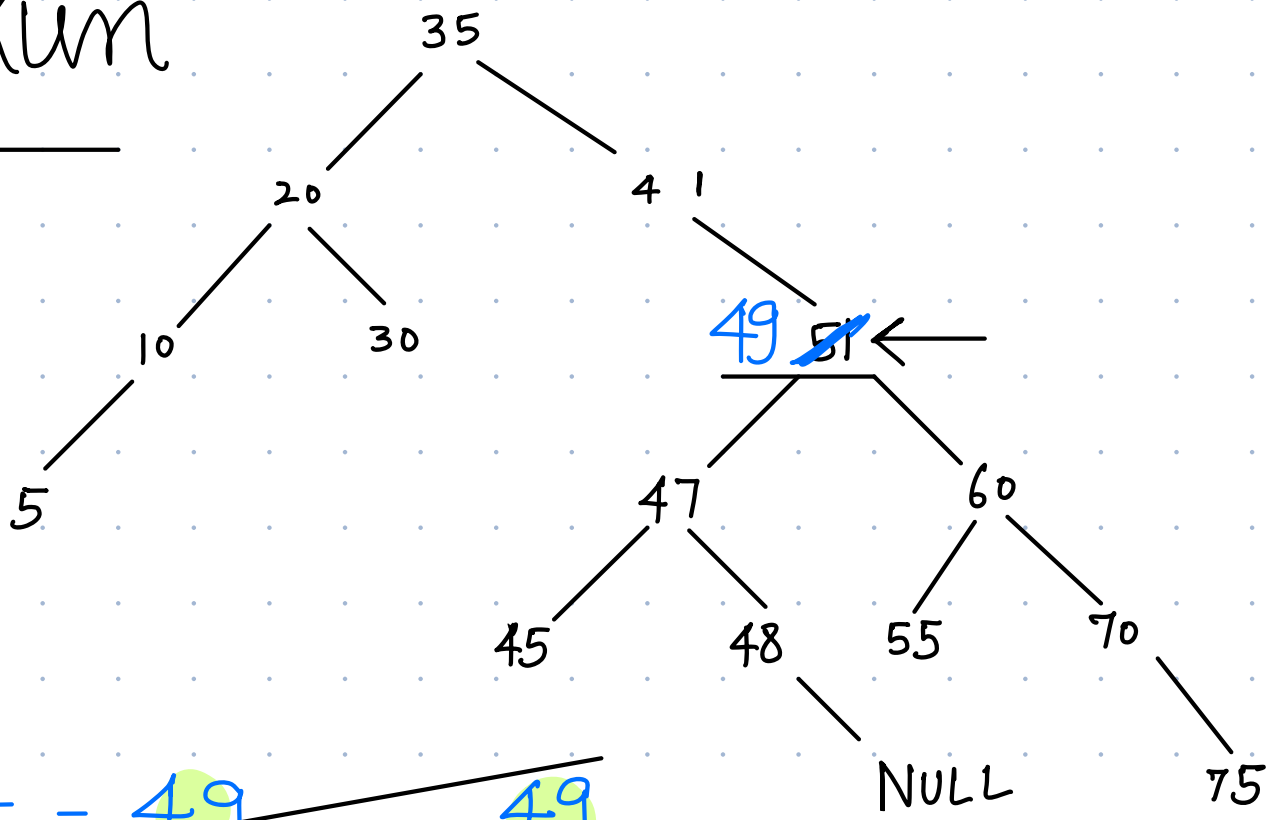
3) in Left subtree delete 2

Pseudocode :

```
Node DeleteBST(root, K){
    if(root == NULL){
        return NULL.
    }
    if(root.value < K){
        root.right = DeleteBST(root.right, K);
    }else if(root.value > K){
        root.left = DeleteBST(root.left, K);
    }
    else{
        if(root.left == NULL && root.right == NULL){
            return NULL
        }
        else if(root.left != null && root.right == null){
```

```
        return root.left
    } else if (root.left == null
        && root.right != null) {
        return root.right
    } else {
        int max = max(root.left);
        root.val = max
        root.left = DeleteBST
            (root.left, max);
    }
}
return root.
}
```

Dry Run



root = 49 , 49

↑ right

root = 48 , 49

↑ right

root = 47 , 49

↑ left

max = 49

root = 51

↑ right

root = 41 51

↑ right

root = 35 51

49.right = Null

48.right = NULL

47.right = 48

49.left = 47

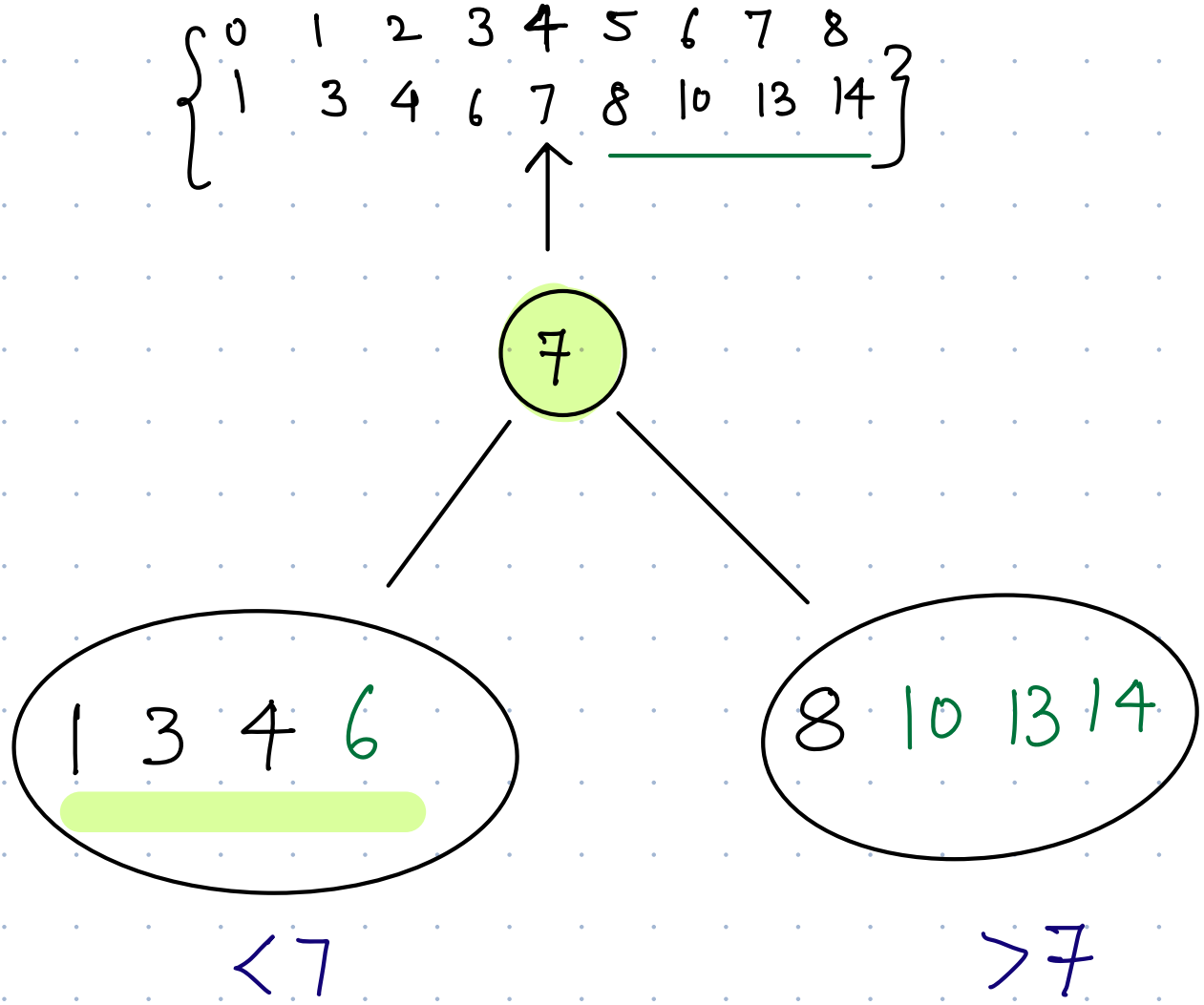
41.right = 49

35.right = 41

35

Q. From sorted array of unique elements

Create a BST



Node createBST (int[] arr, int l, int r) {

Base Case {

Dry Run

$$m = \frac{l+r}{2}$$

Node newnode = new Node(arr[m])

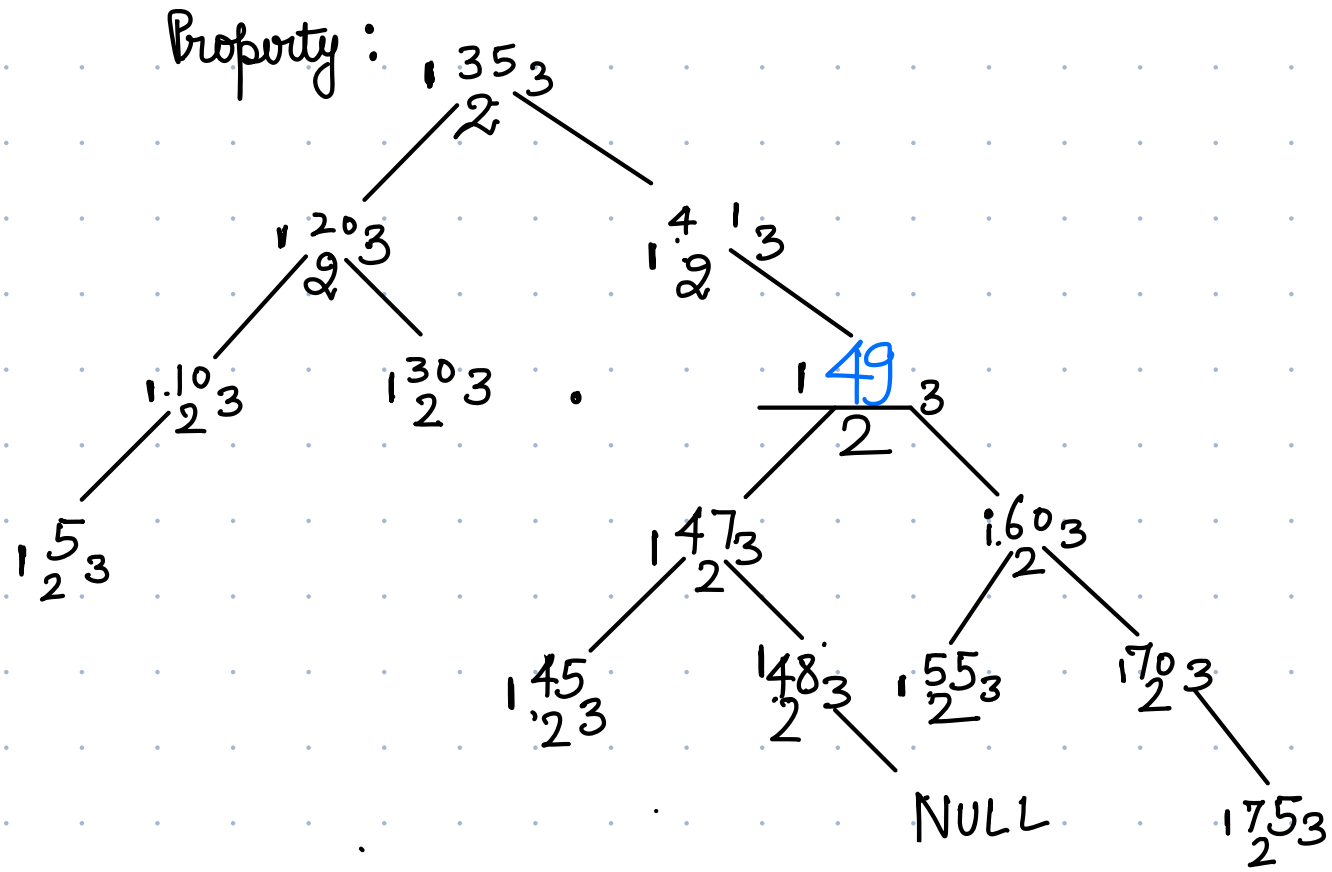
newnode.left = createBST(arr, l, m-1);

newnode.right = createBST(arr, m+1, r);

return newnode;

}

Q. Check if given Binary tree is BST or not.



Inorder : 5 10 20 30 35 41 45 47 48 49 55 60 70 75

→ Inorder traversal of a BST is a sorted array

2 Nodes

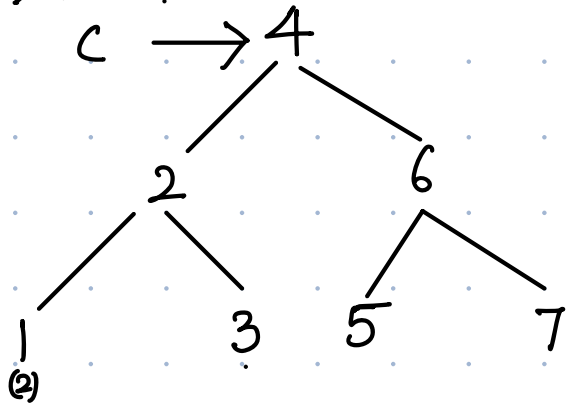
curr Node

prev

prev.value < curr.value Node

---

prvNode = ~~-∞ 1 2 3 4 5 6 7~~



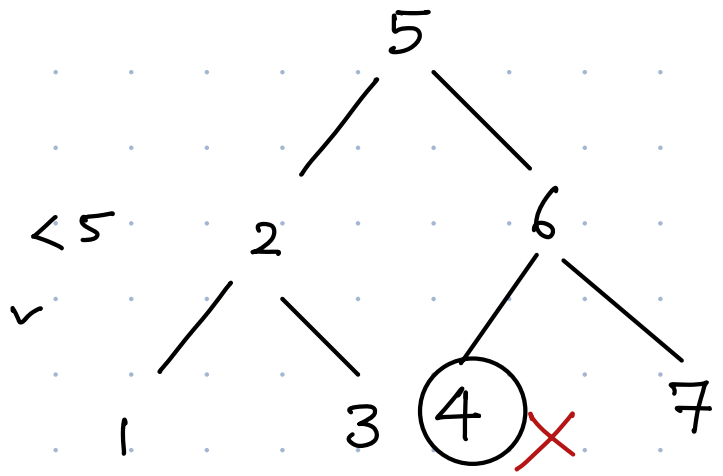
everywhere → condition ⇒ curr > prv

1 > -∞ ✓ was correct

c > p ✓

p = curr

- c <= p → return false



Not BST

prevValue = $-\infty$

```
function inOrder( curr ) {
  if ( curr == NULL) { return true }

  boolean ans = inOrder ( curr.left)
  if ( ans == false) { return false }
  if ( curr.value <= prevValue) {
      return false
  } else if ( curr.value > prevValue) {
      prevValue = curr.value
  }
```

```
        return inOrder(curr.right)
    }
```