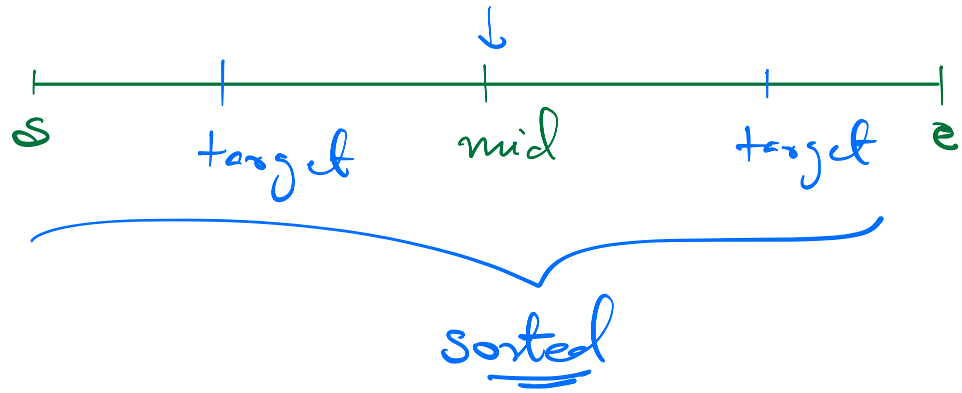
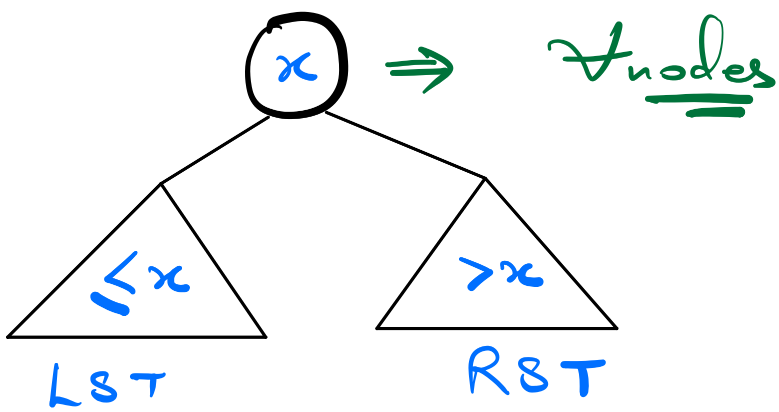
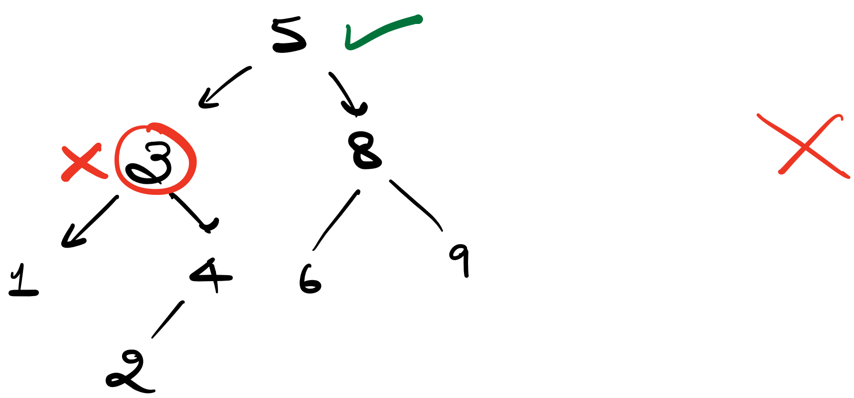
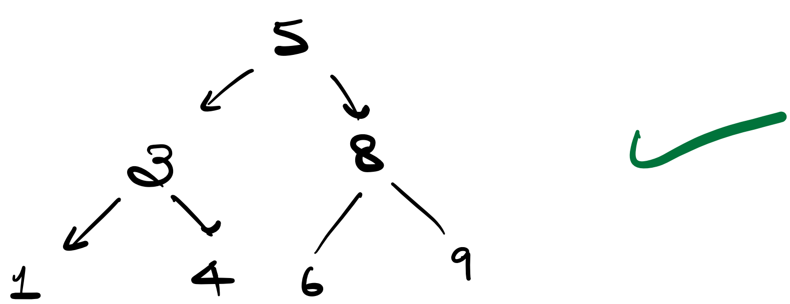
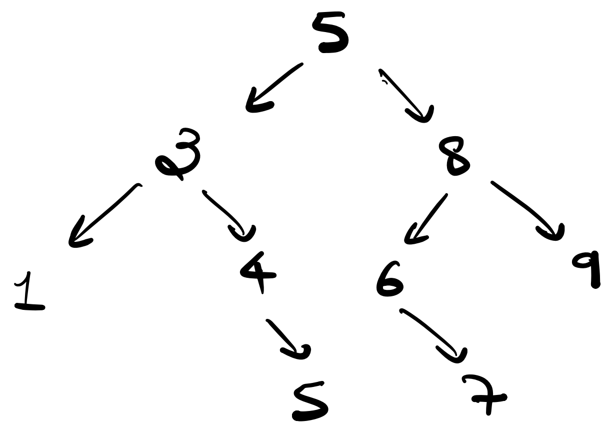
Binary Search Tree







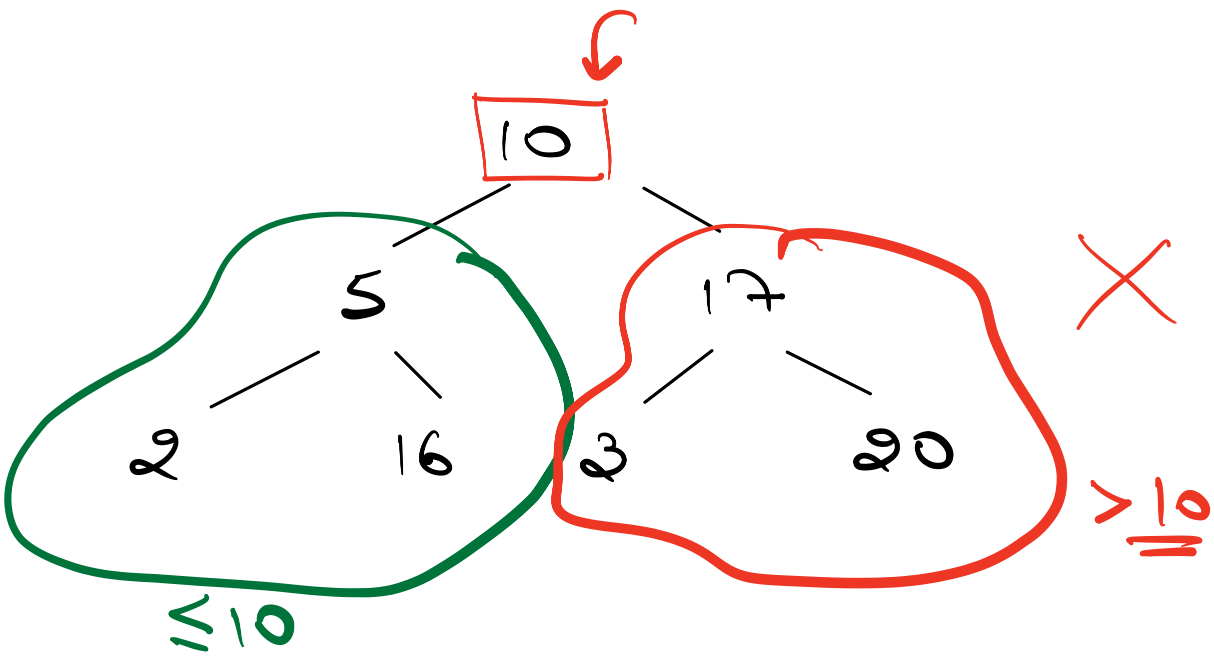




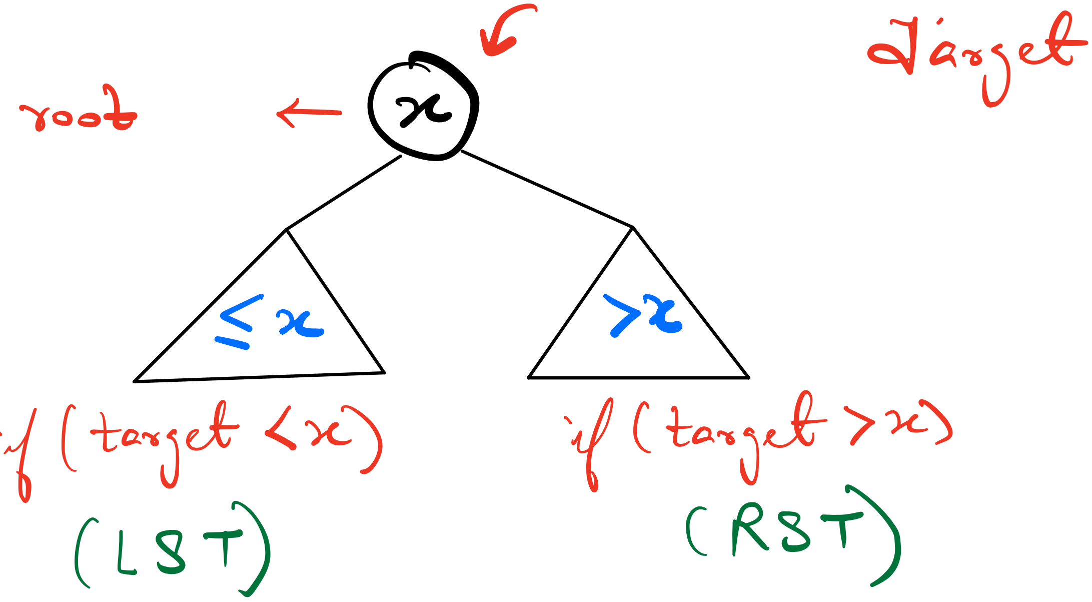
$\forall$ nodes $\Rightarrow$ node.left.data $\leq$ node.data

&

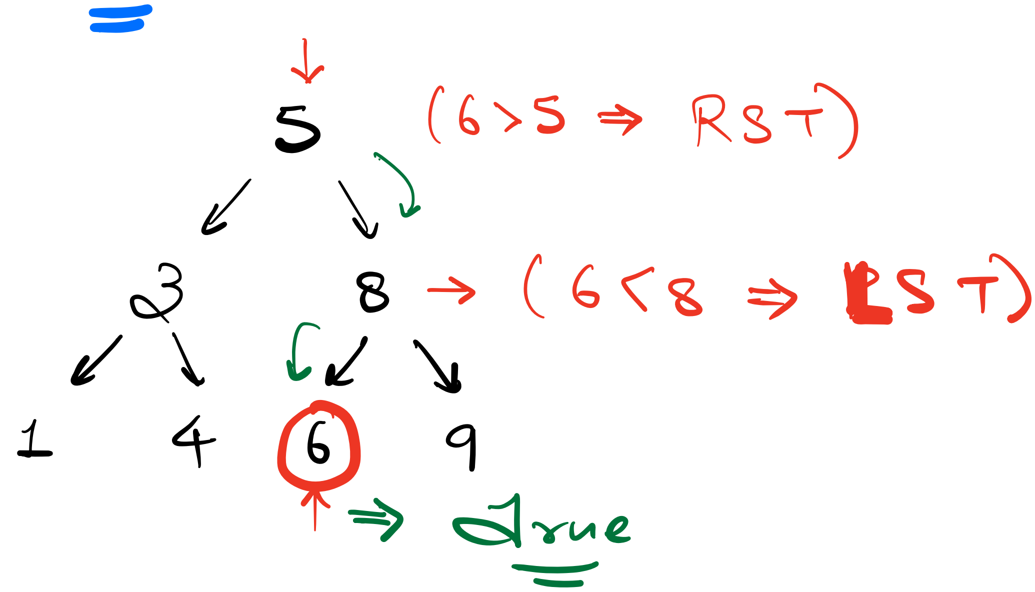
node.right.data $>$ node.data



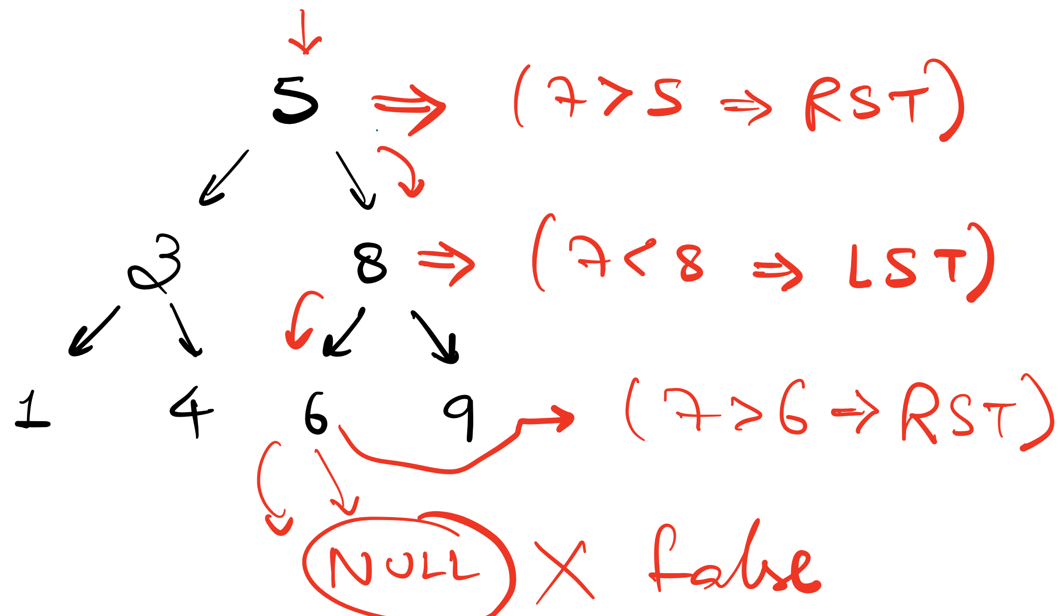
Q Searching in a BST



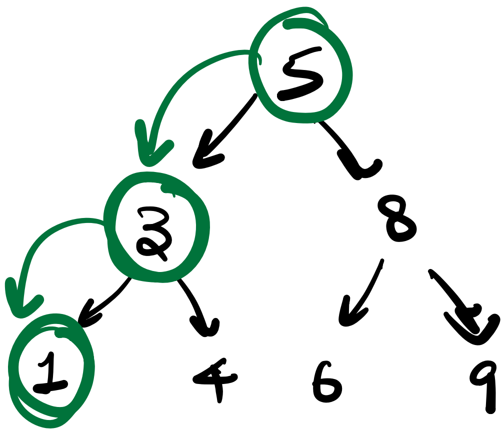
Target = 6



Target = 7 ⇒ false



Quiz



Ans = 3

Code

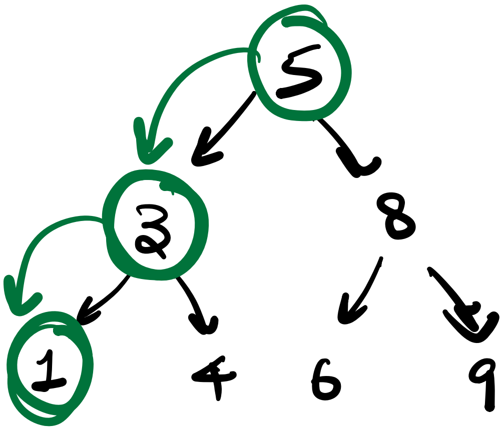
```
boolean search (Node root, int target) {
    if (root == NULL) {
            return false;
    }

    if (root.data == target) {
            return true;
    } else if (root.data > target) {
        return search (root.left, target);
    } else {
        return search (root.right, target);
    }
}
```

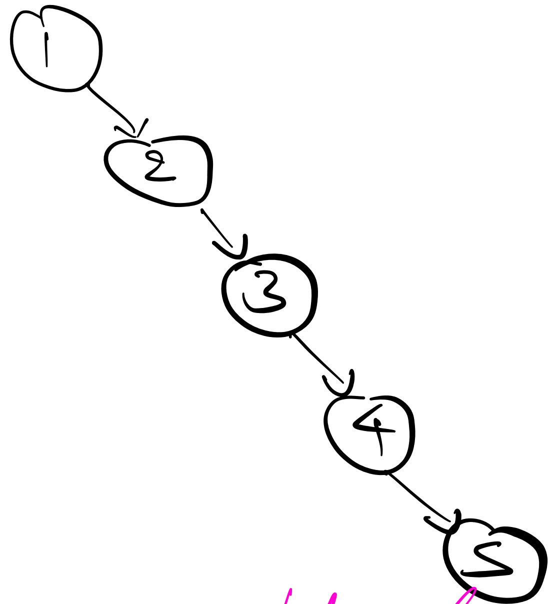
$$T.C. = O(\text{Height})$$

$$\Downarrow$$

$$O(N) \Rightarrow \text{Skewed Tree}$$

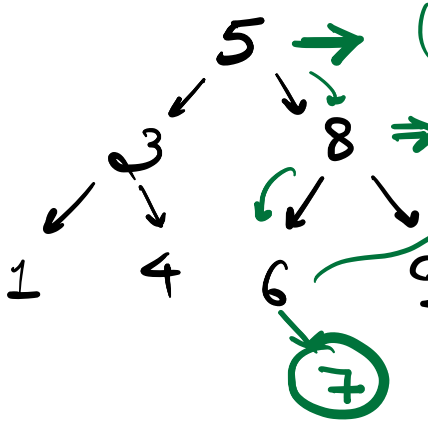


Height = $\log N$



Height = $\underline{\underline{N}}$

Insertion in a BST



$(7 > 5 \Rightarrow RST)$

$(7 < 8 \Rightarrow LST)$

$(7 > 6 \Rightarrow RST)$

Insert a Node with a value $\underline{\underline{7}}$

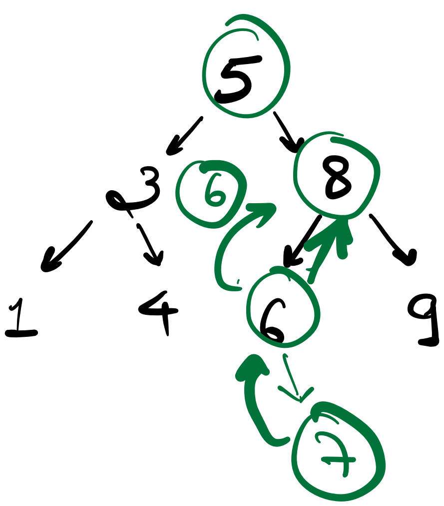
Code

```
Node insert(root, value) {

    if (root == NULL) {
        return new Node(value);
    }

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

    return root;
}
```



6.right = insert(NULL, 7)

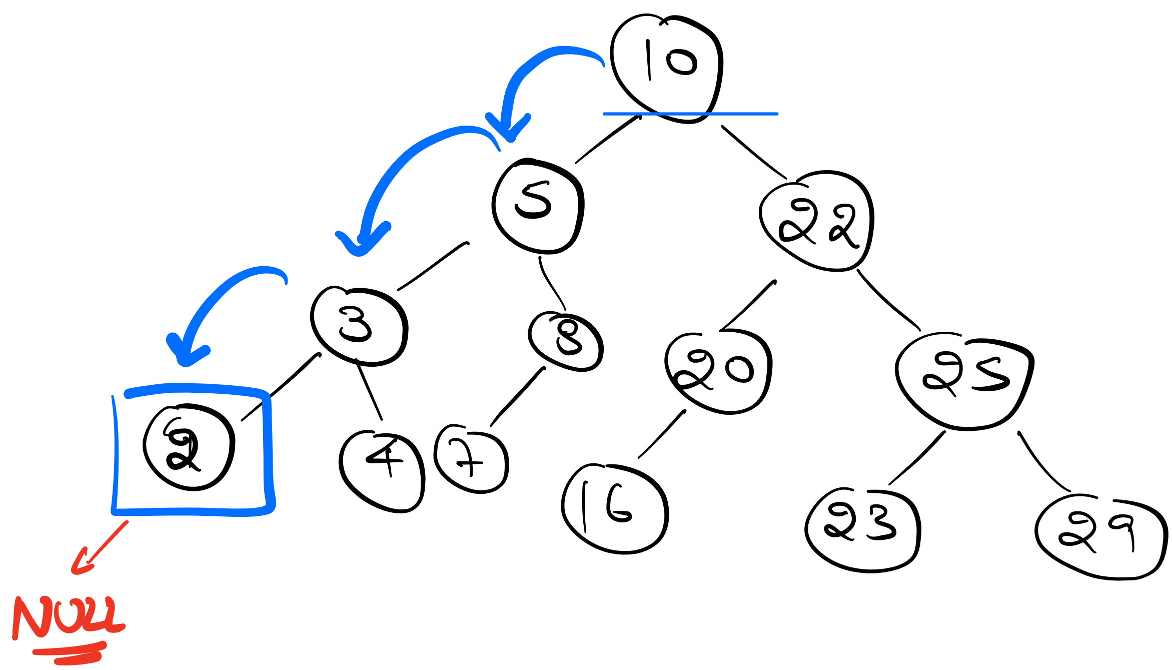
8.left = insert(6, 7) 6

5.right = insert(8, 7)

insert(5, 7)

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

S.C. = O(Height) = O(N)



Code

```
Node smallest (root) {
    if (root == NULL) { return NULL; }
    temp = root;
    while (temp.left != NULL) {
```

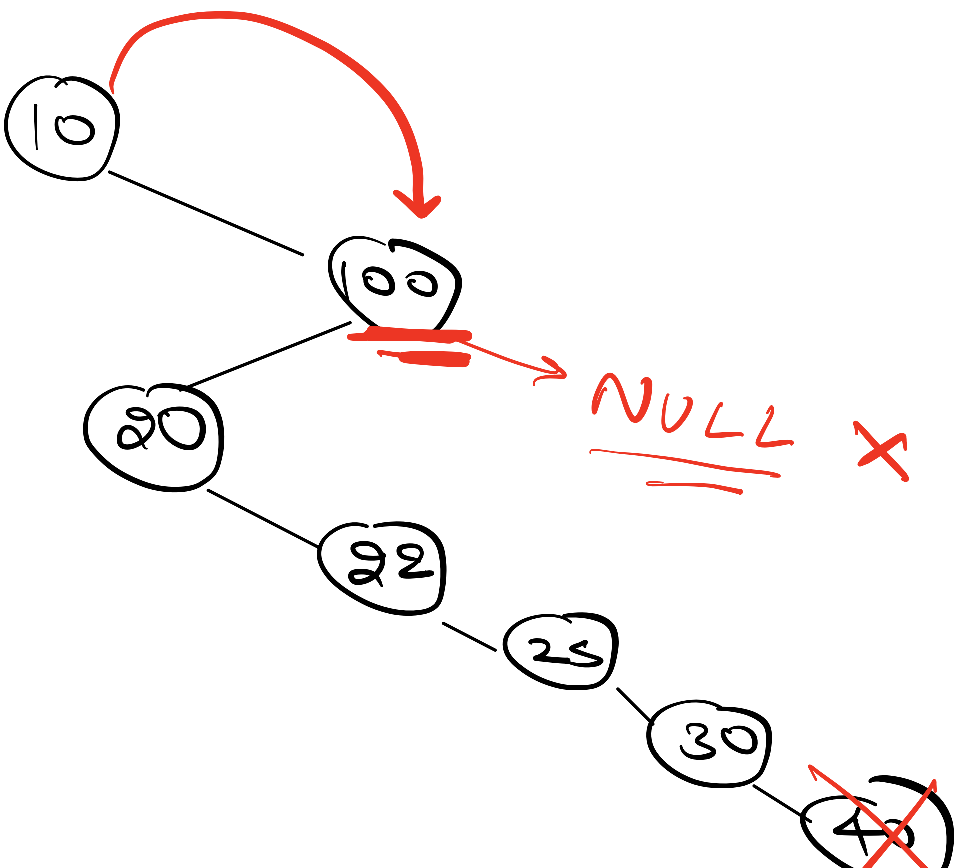
```
        temp = temp.left;
    }

    return temp;
}
```

$T.C. = O(Height) = O(N)$

$S.C. = O(1)$

Q find the largest Node of BST



```
Node largest (root) {
    if (root == NULL) { return NULL; }
    temp = root;
    while (temp.right != NULL) {
            temp = temp.right;
    }
    return temp;
}
```

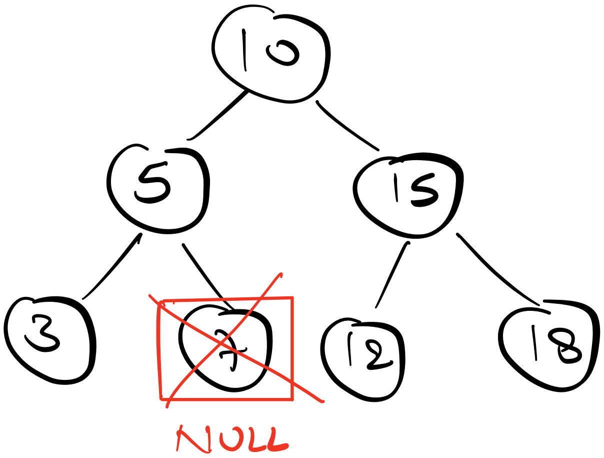
$T.C. = O(Height) = O(N)$

$S.C. = O(1)$

Delete in a Binary Search Tree

Case I: Node with No Children (Leaf Node)

Delete the Node

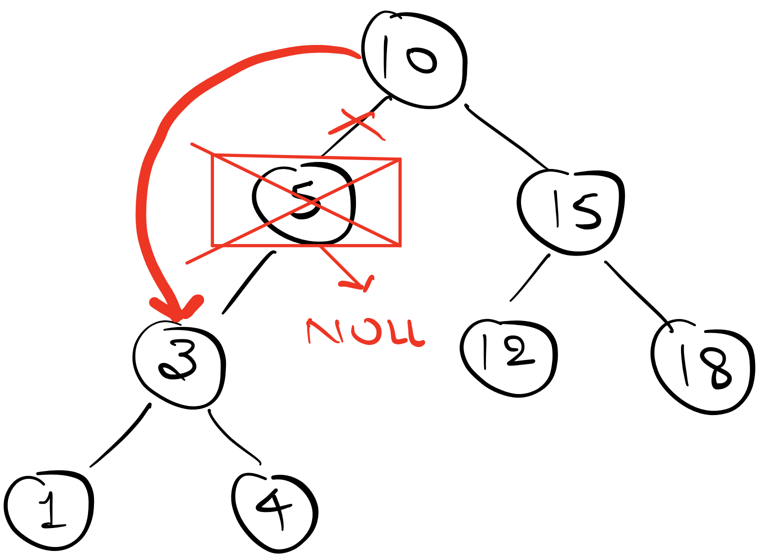


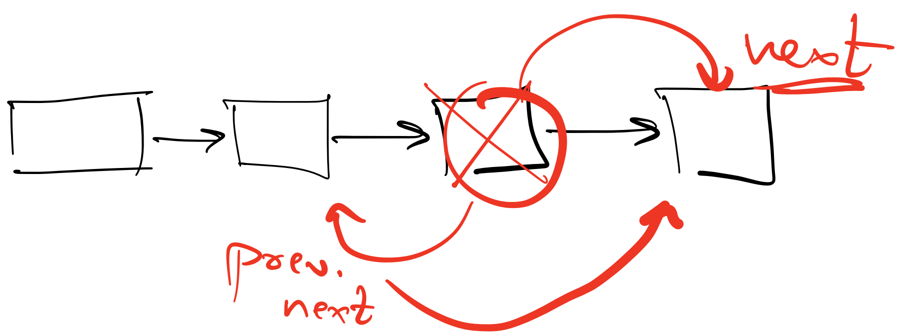
Target = 7

Case II Node with One Child.

Replace with the non NULL Child.

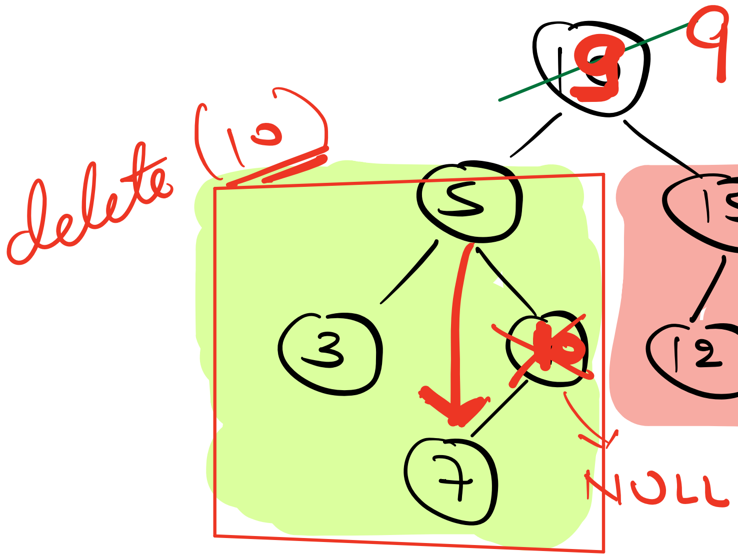
Target = 5





Case III

Node with Two Children



Target = 10

InOrder Predecessor ✓ — Largest Node of LST

InOrder Successor — Smallest Node of RST

Root

Inorder Traversal = 3, 5, 7, 9, 10, 12, 15, 18

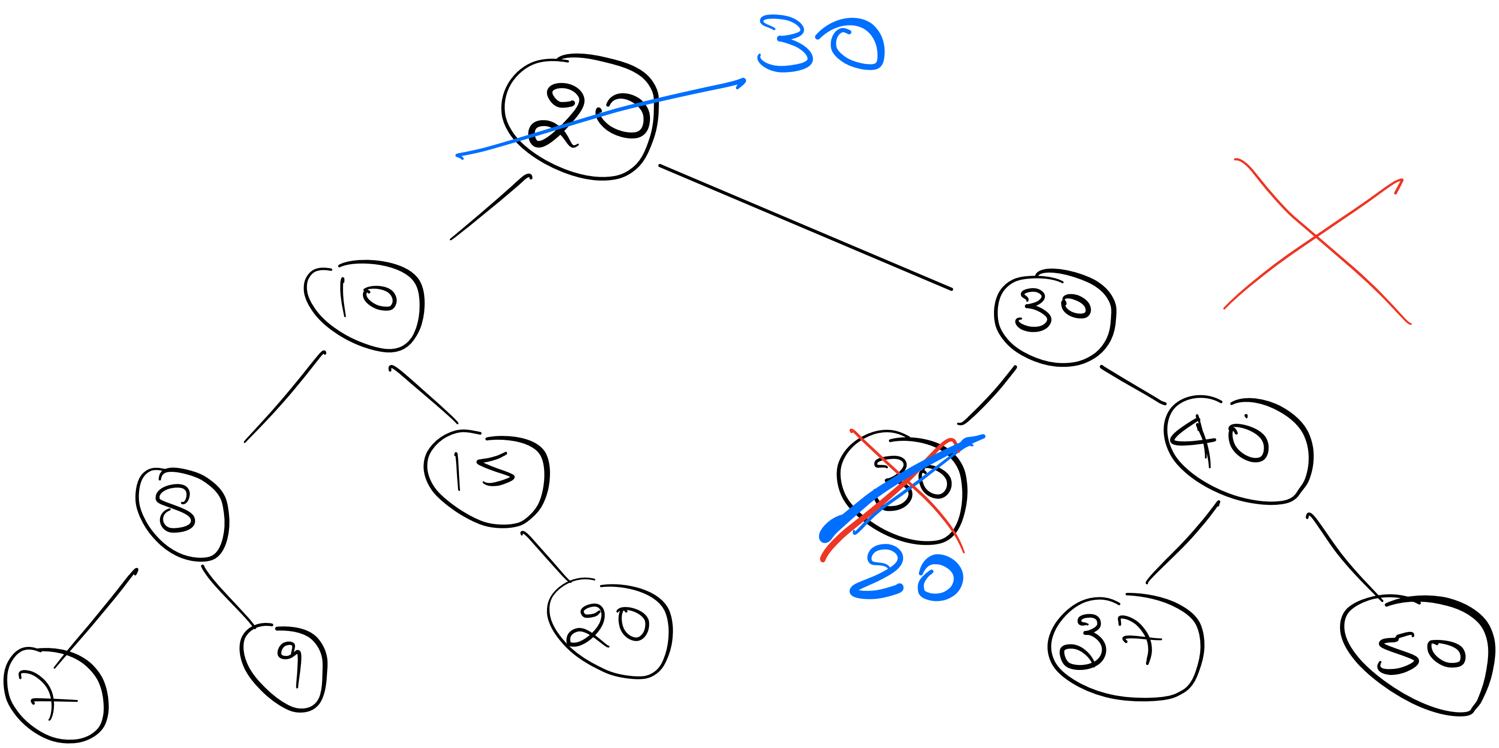
IP: 9 | IS: 12

Sorted

L, N, R

L ≤ N < R

H.W. Think if anything can go wrong in case of replacing with inorder successor if equality is in the left.



Code

```
Node delete(root, value) {
    if (root == NULL) { return NULL; }
    if (root.data == value) {
        if (root.left == NULL && root.right == NULL) {
            return NULL;
        }

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

        Node temp = root.left;
        while (temp.right != NULL) {
            temp = temp.right;
        }

        swap(root, temp);   // Swap values.
        root.left = delete(root.left, value);
```

```
} else {

    if (value < root.data) {
        root.left = delete(root.left, value);
    } else {
        root.right = delete(root.right, value);
    }
}
return root

}
```

Create a BST

Given a sorted array

↳ Create a BST.

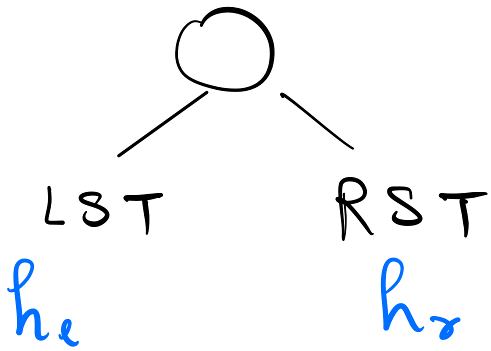
	0	1	2	3	4	5	6	7	8
A:	1,	3,	4,	6,	7	8,	10,	13,	14.

$\underbrace{1,\ 3,\ 4,\ 6}_{LST}$ ⇓ BST

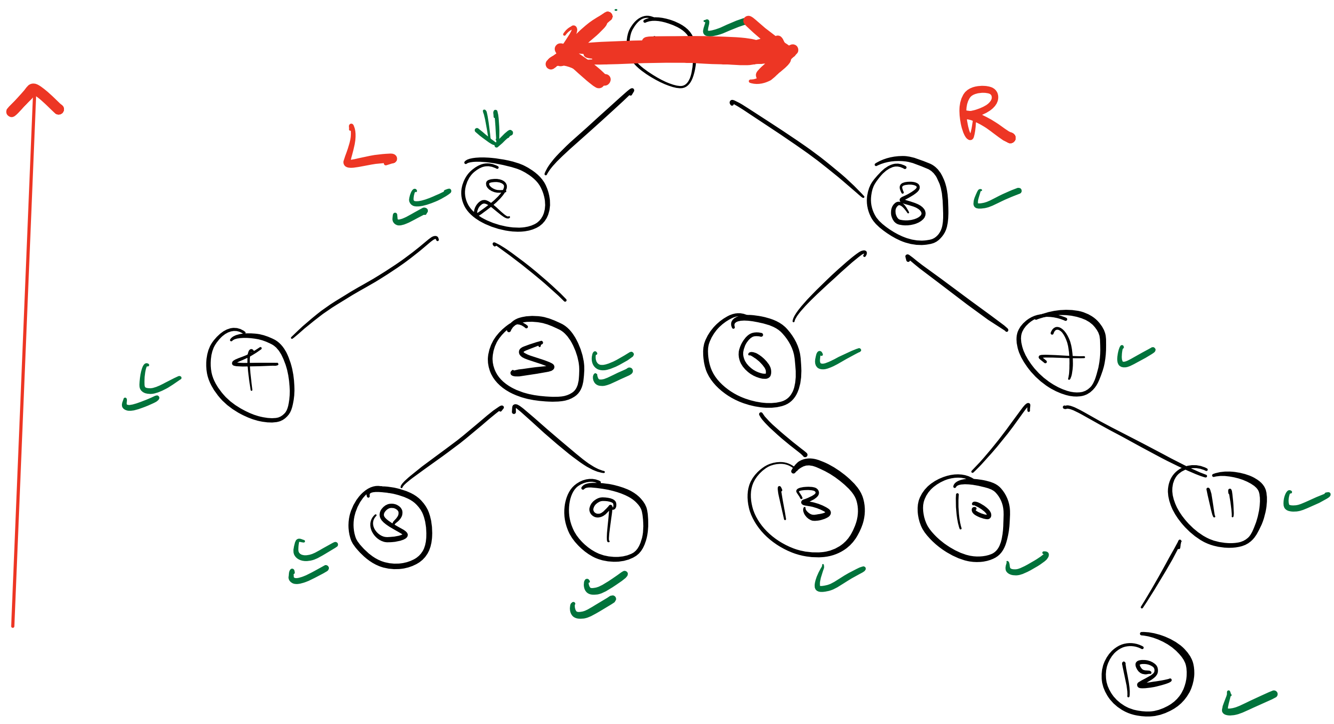
7 ⇓ root

$\underbrace{8,\ 10,\ 13,\ 14}_{RST}$ ⇓ BST

H.W. Code



$$|h_l - h_r| <= 1$$



L, R, N $\Rightarrow$ Post

Given Inorder & Postorder

⇓

Construct the Binary Tree

			LST		RST
In :	L, N, R	:	2, 13, 6	10	8, 5, 7
Post :	L, R, N	:	2, 6, 13	8, 7, 5	10
			LST	RST	Root

10

LST RST

In : 2, 13, 6

Post 2, 6, 13

L R

13

2 6

2 6

LST RST

In : 8, 5, 7

Post : 8, 7, 5

L R

5

8 7

8 7

2 6 8 7