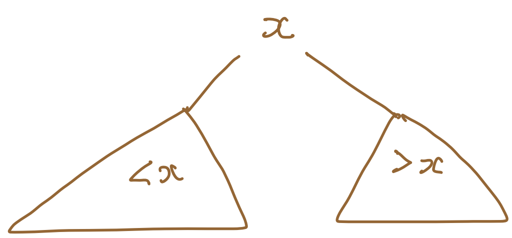
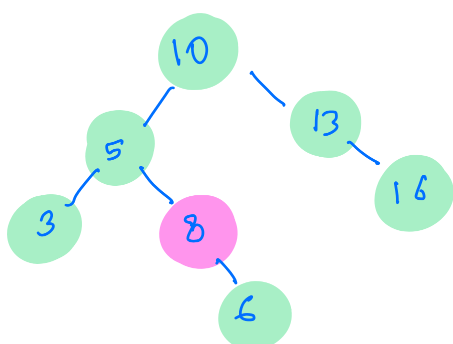
BST (Binary Search Tree)

1. BT

2. ST (Searching in Tree will be efficent)

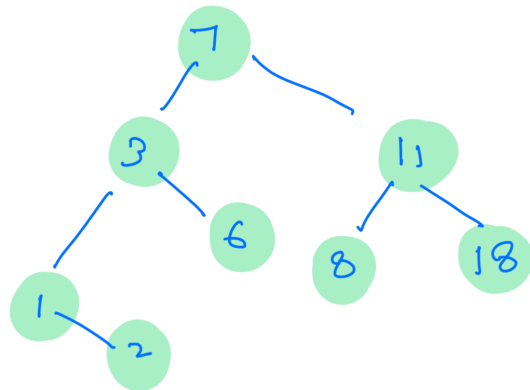
∀ Nodes $\boxed{\text{Left Subtree} \le \text{node.data} < \text{Right Subtree}}$





BST ✗

BST ✓

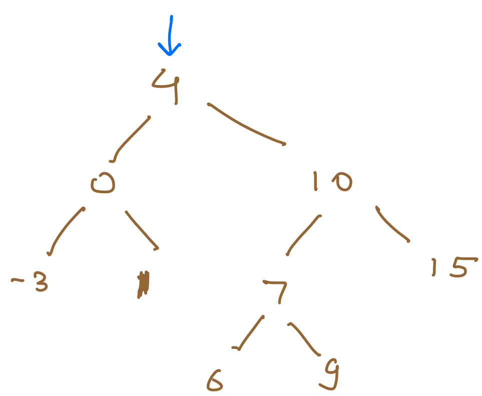


1, 2, 3, 6, 7, 8, 11, 18

Inorder traversal of BST = Sorted

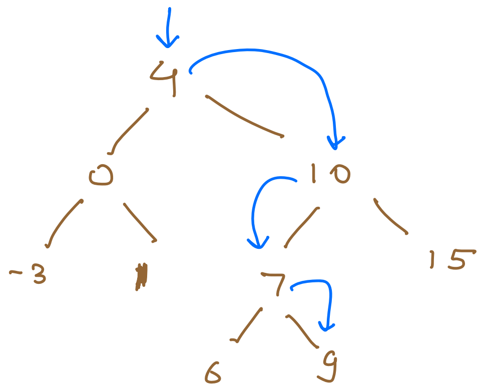
Given a BST, Search if K exists in Tree or not

K = 9



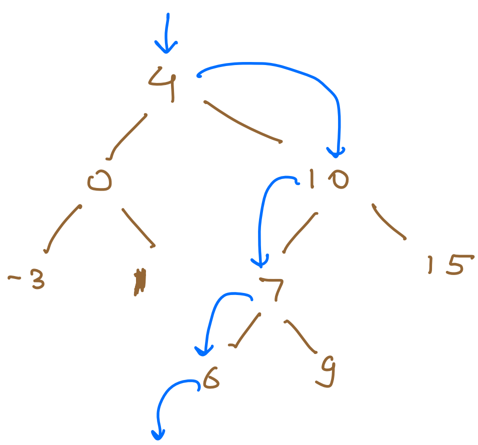
ans = True

K = 9



ans = True

K = 5



ans = False

(Node root, int K)

```
Node tmp = root
while ( tmp != NULL )
{
    if ( tmp.data == K )
        return True
    else if ( K > tmp.data )
        tmp = tmp.right
    else
        tmp = tmp.left
}
return False
```

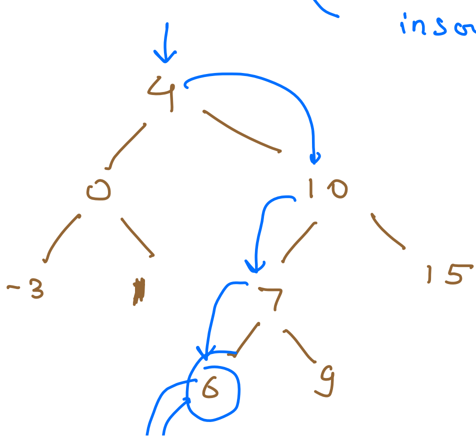
height

TC : $O(H)$

SC : $O(1)$

Q. Given BST, Insert K (in correct position, so after insertion. It is still BST)

K = 5



N5

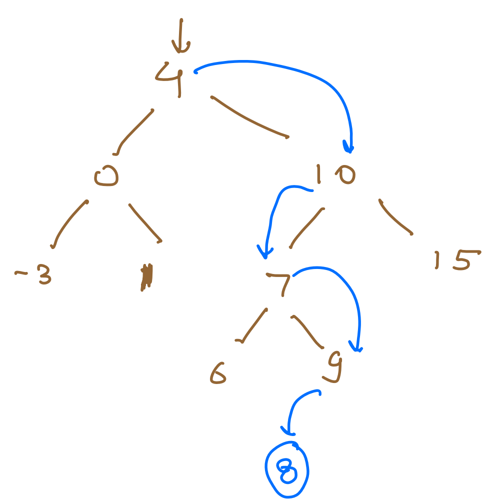
insert K, and return new root of the tree

TC: O(H)

SC: O(H)
↓
recursion stack

```
Node Insert ( Node root, int K )
{
    if ( root == NULL)
    {
        return new Node(K)
    }

    if ( K > root. data)
        root.right = insert( root.right, K)
    else if( K < root.data )
        root.left = insert( root.left, K )
    else
        return root
}
```

K = 8



Search ✓

Insert ✓

delete

heavy

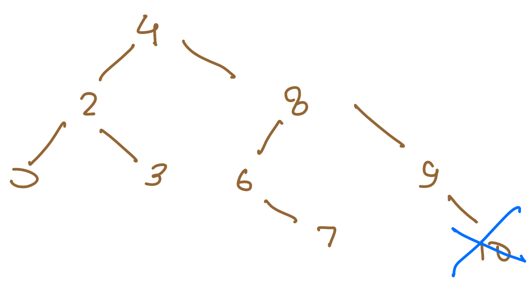
Q. Delete given number from BST (Still BST)

(i) Delete leaf (No child)

(ii) Node with one child

(iii) Node with two children.

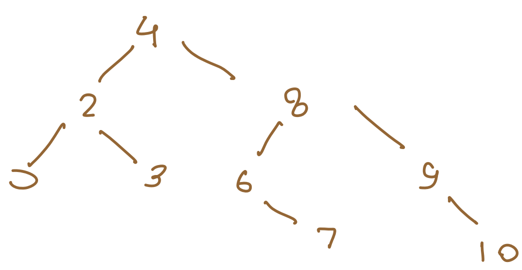
(I)



K = 10

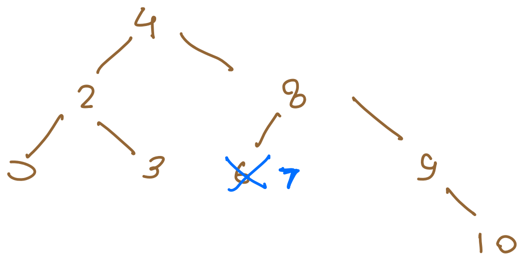
(II)

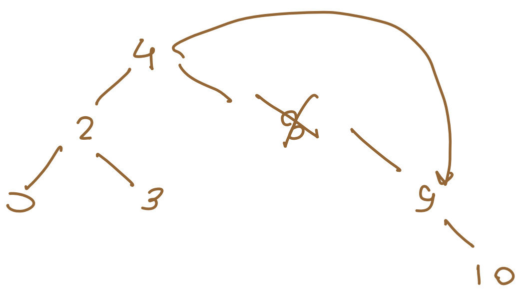
one child



K = 6

⇓

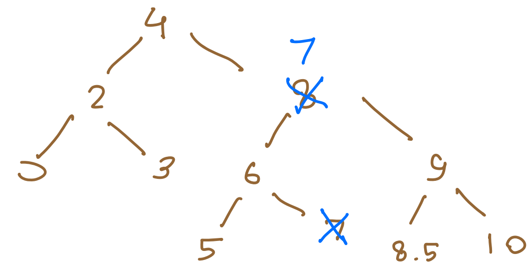




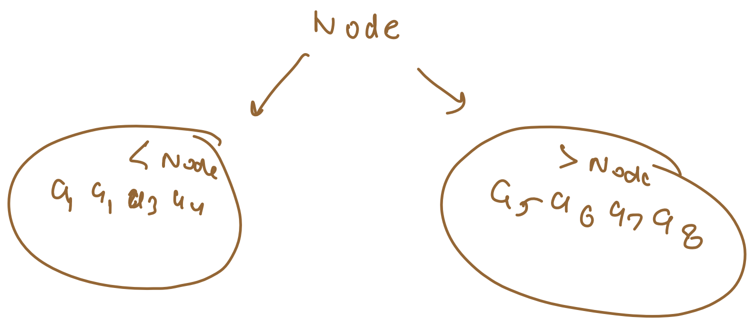
K = 8

III

(2 children)



K = 8



(Both works)

1. max of left ~~child~~ subtree
2. min of right subtree

4

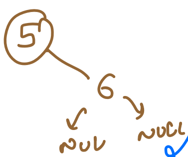
2 1 3

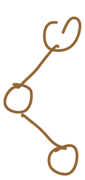
6 7 8 9

new root return after delete by K

```
Node delete (Node root, int K)
{
    if ( root == NULL )  return NULL

    if ( K > root.data )
        root.right = delete ( root.right , K )
    else if ( K < root.data )
        root.left = delete ( root.left , K )
    else
    {
        // K == root.data
        if ( root.left == NULL && root.right == NULL )      I
        {  return NULL

        else if ( root.left == NULL || root.right == NULL )
        {
            if ( root.left != NULL )                        II
                return root.left
            else
                return root.right
        }
        else
```

K = 6





III

// Standing at root, delete root

1) Replace root.data with rightmost in left child (x)

2. Delete (x) from left subtree

```
Node tmp = root.left
while( tmp.right != NULL)
{
    tmp = tmp.right
}
root.data = tmp.data
root.left = delete( root.left, tmp.data)
```

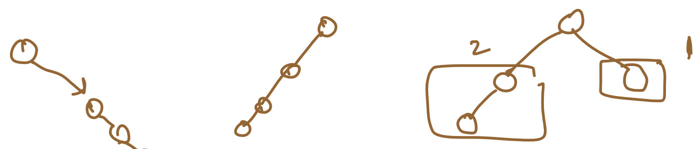
TC: O(H) — worst case ⇒ O(N)

SC: O(H)

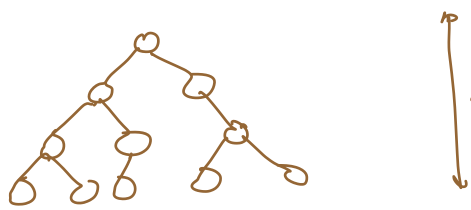
recursion stack.

Balanced Binary Tree

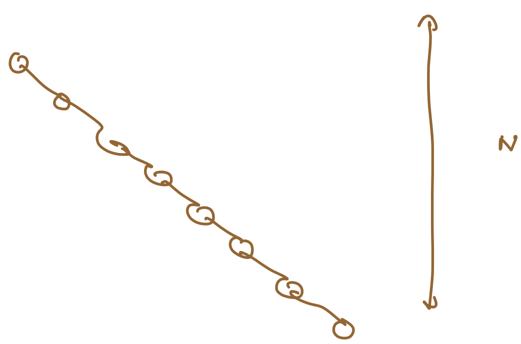
$\forall$ nodes (abs(height of children) ≤ 1)



Unbalanced   Unbalanced   balanced



$H = \log N$

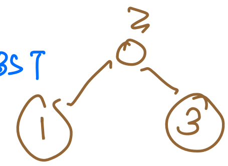


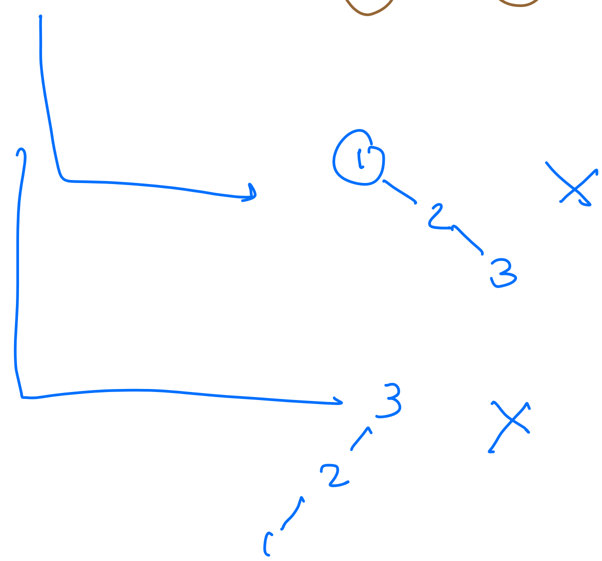
$\log N \le$ height $\le N$

BBT

Q Given sorted arry. Construct BBST.

arr [] = {1, 2, 3} → BBST





arr[] = [1, 2, 4, 5, 6, 7, 8]

$\boxed{5}$

Node BBST (arr[], l, r) → 0, n-1

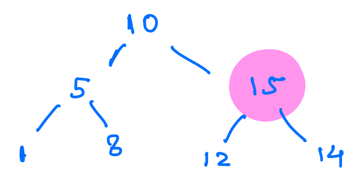
```
{
    if (l > r)
    {   return NULL

        int mid = (l+r)/2

    Node root = new Node ( arr[mid])
```

root.left = BBST (arr, l, mid-1)

root.right = BBST (arr, mid+1, r)

return root

Q. Given BT, check if BST or not.



ans = False

1. Inorder $\longrightarrow$ if sorted ✓

if not sorted ✗

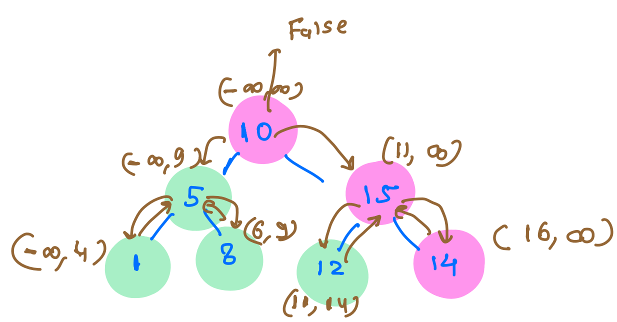
TC: O(N)

SC: O(N + H) $\longrightarrow$ O(H)

array, recursion stack

recursion stack

2. Taking range for each node.



$(-\infty, \infty, root)$

```
bool IsBST( Node root, int l, int r)
{
    if (root == NULL) return True

    if ( root.data < l || root.data > r)
        return False

    bool left = IsBST( root.left, l, root.data-1)
    bool right = IsBST( root.right, root.data+1, r)
    return left & right
}
```

TC: $O(N)$

SC: $O(H)$

Doubts