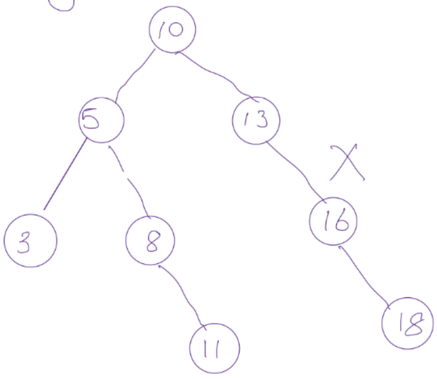
Trees 3

Intro To BST

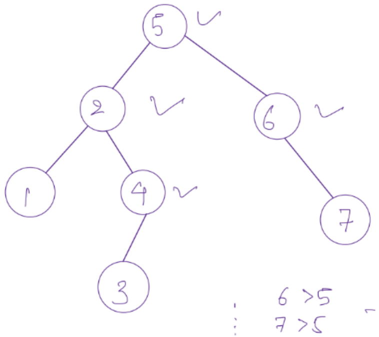
* A Binary Tree is a BST if :-

∀ Nodes, All elements in LST <= Node < All elements in RST

Eg:



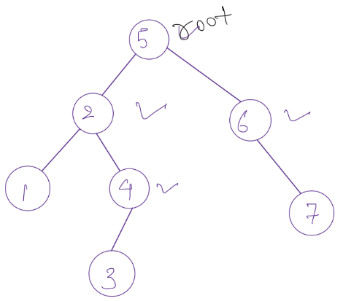
Eg 2:



6 > 5
7 > 5 - - -

BST

Searching In BST



K = 4

4<5 → Go left

4>2 → Go right

4==4 → return.

If NULL not present

Pseudo Code :

```
bool Search (TreeNode root, int K)
{   if (root == NULL)
        return false;

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

Iterative,

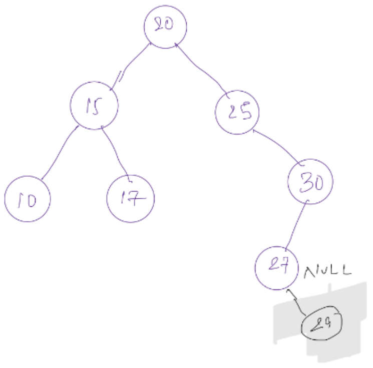
root = root.left

root = root.right

T.C : O(Height of Tree)

S.C : O(1)

Insertion In BST



K = 29

20 < 29, right
25 < 29, right
30 > 29, left
27 < 29, right.

Pseudo Code :-

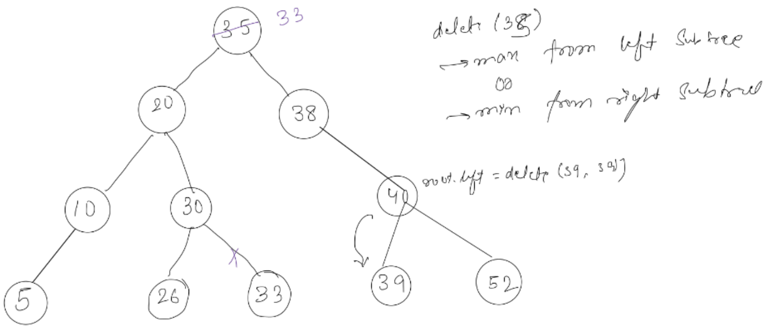
```
Node insert (Node root, int k)
{  if (root == NULLptr)
   { return new Node (k);

   if (root . data < k)
   { root.right = insert ( root → left, k);
   }else{
      root.left = insert (root → right, K);
    }
}
```

Q Given a BST return Smallest node / Largest Node

```
Node temp = root
while (temp != nullptr) {
     temp = temp → left;
}
std::cout << temp → data << std::endl.
```

```
Node temp = root
while (temp != nullptr)
     temp = temp → right;
}
std::cout << temp → data << std::endl.
```

Deletion In BST



delete (38)
→ max from left Subtree
or
→ min from right Subtree

```
Node  delete ( Node  root,  int  K)
{   if (root. data = = k)
    {   // found the element to be deleted.
      //Case 1 : Leaf Node
      if (root. left = = NULL & root. right == NULL)
            return NULLptr;
      //Case 2 : Only one children
      if (root. left == NULLptr)
            return root. right;
      if (root. right == nullptr)
            return root. left;
      if (root. left != nullptr || root. right != nullptr)
      {     int x = maxvalue ( root. left) :
            root = x. data;
            root. left = delete( root. left , x. data);   // keep deleting the x. data.

            return root
      }

    } else if (root. data > k) {
          root. left = delete (root . left , k);
    } else {
          root. right = delete (root. right, k);
    }
```

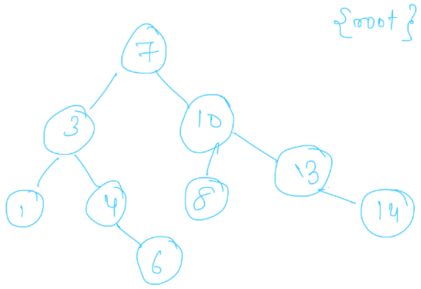
TC: O(H)
S.C : O (1)

Construct A BST from Sorted Array

Q6 Construct a BST from a sorted array of unique elements.

Eg:- arr[] : {1 ,3, 4, 6, 7, 8, 10, 13, 14};

mid
{root}



Pseudocode:-

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

T : O(N)
S : O(1)

Check If given tree is a BST

⇒ For BST :-
Inorder Traversal of a BST should be Sorted

Approach 1
Find inorder Traversal & Check Sorted or not.
T.C: O(n)
S.C: O(1)

Approach 2
For ∀ node
root.left < root
root.right > root.

Check Again !!

Pseudo code :-

```
bool  isBST(Node root, int l, int r)
{    if(root == nullptr)
         return true;  //empty tree
     if(root.data >= l && root.data <= r)
     {    return((isBST(root->left, l, root.data-1))
                     &&
                 (isBST(root->right, r, rot.data +1))
           )
     }
     else{
          return false;
      }

}
```