Welcome ☺

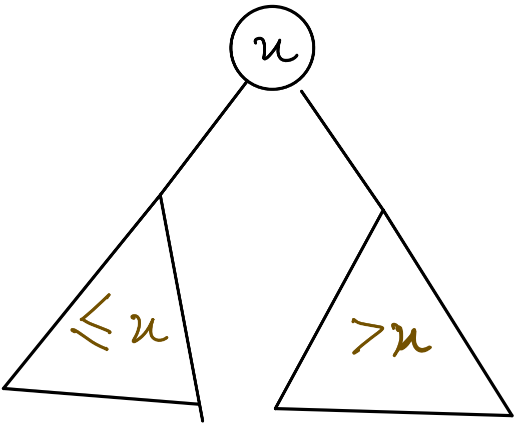
Agenda : Binary Search Tree (BST)

Operations

1-2 ques^{n}s.

Binary Search Tree (BST)

└→ searching data in an organised dataset using divide & conquer.

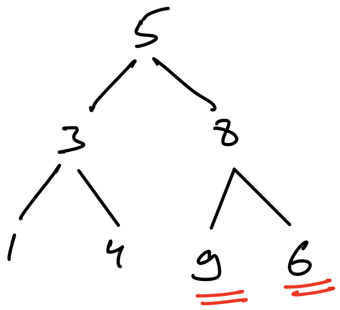


equality on left

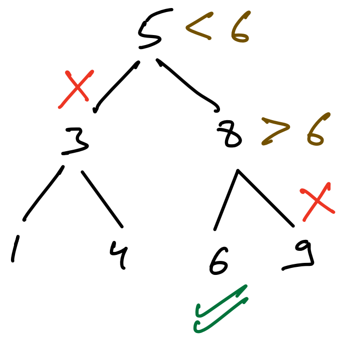
$\forall_{nodes}$ → all nodes in left subtree $\leq x$

→ all nodes in right subtree $> x$

Searching in B.S.T



✗ Not a BST



find 6

find 7 → NULL

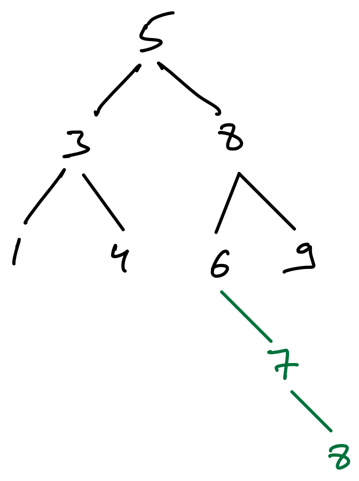
T.C = O(H) → O(log N)

S.C = O(H) → recursion space.

code

```
// Searches first instance of target.
Node search ( root , target)
{
    if ( !root )     return NULL

    if ( root.data == target )
            return root

    // Decide left or right
    if (  target < root.data )
            return search ( root.left , target)
    else
            return search ( root.right, target )
}
```

Insertion in B.S.T



insert (7)

insert (8)

⇒ inserted as leaf node. to avoid complexity. But it is not a compulsion.

Code

```
nn = new Node (x)
if(!root ) return nn
temp = root
while ( temp != NULL)
{
    if( temp. data < x ) // go right
    {
        if( temp. right == NULL) {
            temp. right = nn
            return root
        }
        temp = temp. right
    }
    else
    {
        if( temp. left == NULL) {
            temp. left = nn
            return root
        }
        temp = temp. left
    }
}
```

T.C ⇒ O(H)

S.C ⇒ O(1)

Q Find smallest element in BST

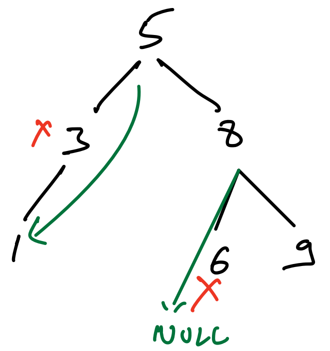
leftmost node will be smallest.

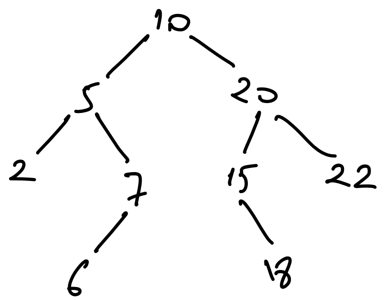
```
if ( !root ) return NULL
temp = root
while ( temp.left != NULL)
{
    temp = temp.left
}
return temp.data.
```

T.C = O(H)
S.C = O(1)

Q find largest element in BST. H.W

Deletion in BST





1) Search the node that you want to delete

2) a) Node to be deleted is a leaf node.

→ parent points to NULL.

b) If node to be deleted has 1 child.

→ parent points to single child.

c) If node to be deleted has 2 child.

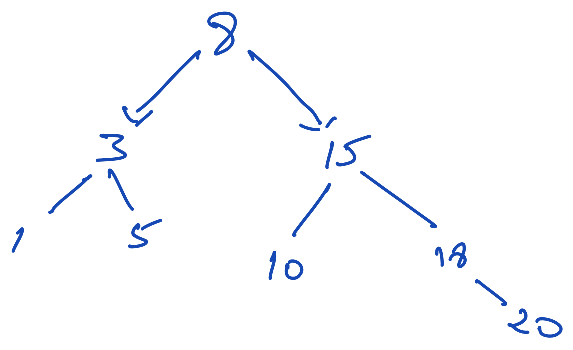
→ find greatest ele. on left subtree to be replaced with deleted node.

Code

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

    if ( root.data == K )
    {
        if (!root.left && !root.right) // no child
            return NULL
        if ( !root.left || !root.right) { // 1 child
            if ( !root.left) return root.right
            else return root.left
        }
        temp = root.left
        while (!temp.right)
            temp = temp.right
        root.data = temp.data // replace value.
        root.left = delete (root.left , temp.data)
        return root
    }
    else if ( root.data > K)
        root.left = delete ( root.left, K)
    else
        root.right = delete ( root.right, K)
}
```

Q Construct BST from sorted array of unique elements.

1 3 5 8 10 15 18 20



Code

```
                          0        N-1
Node build ( A[], L , R )
{
    if ( L > R ) return NULL.
    mid = $\frac{L+R}{2}$
    root = newNode ( A[mid] )
    root.left = build ( A[], L , mid-1 )
    root.right = build ( A[], mid+1 , R )
    return root
}
```

T.C ⇒ O(N)

S.C ⇒ O(log N)
↳ recursion

Q Check if a binary tree is a binary search tree ?

$\forall_{nodes}$ $X \geq$ left_max.

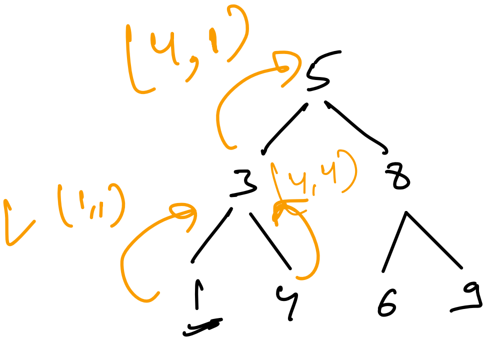
$<$ right_min

Code

```
bool isBST ( root )
{
    if ( !root )   return 1
    int maxL = maxValue ( root. left )
    int minR = minValue ( root. right )

    if ( maxL > root. data )
        return 0

    if ( minR <= root. data )
        return 0

    if ( ! isBST ( root. left ) || ! isBST ( root. right ) )
        return 0

    return 1
}
```

T.C => ~~O(N)~~ O(N*N)

S.C => O(logN)



```
{ max , min }          isBST = True.

pair<> Max_Min (root)
{
    if ( !root ) return { INT_MIN , INT_MAX }
    L = max_min ( root. left )
    R = max_min ( root. right )

    if ( L.max > root. data || R.min ≤ root. data )
            isBST = False

    return { max( root. data , L.max , R.max ),
             min ( root. data , L.min , R.min )
           }
}
                                        T.C => O(N)
return isBST
```