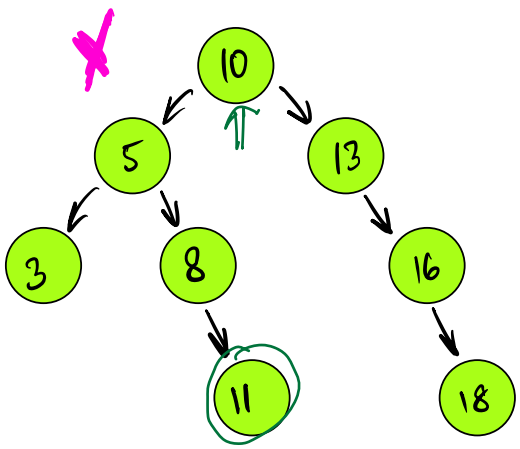
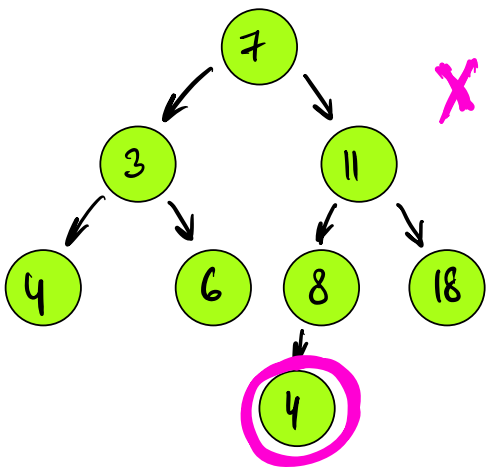
Binary Search Trees is Binary Trees

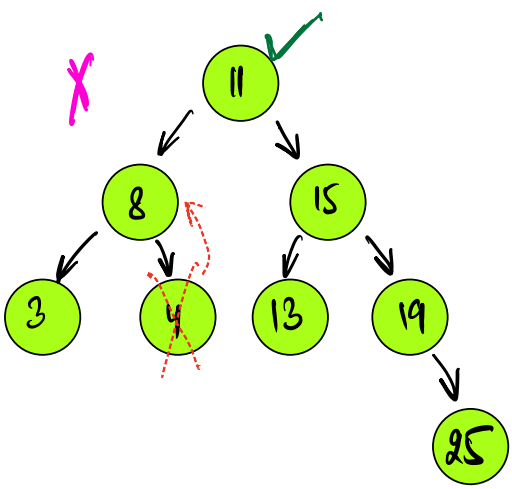
#nodes

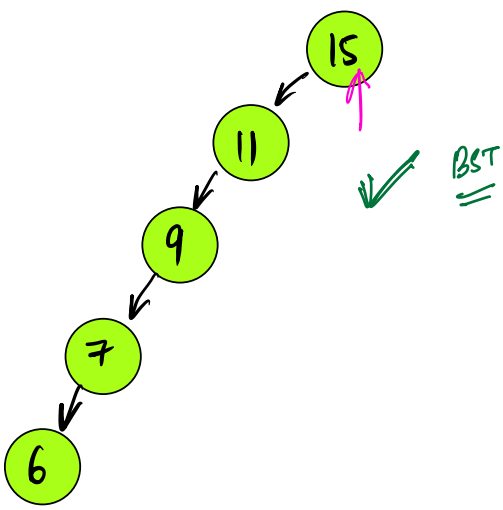
all elmt in LST < node < all elmts in RST

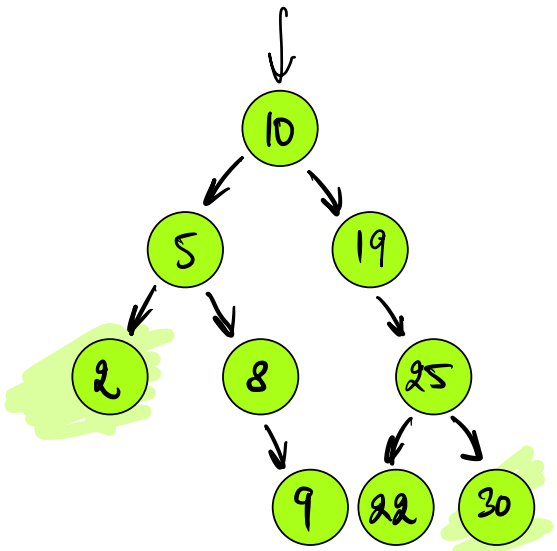
= or =






Yes, it is a BST

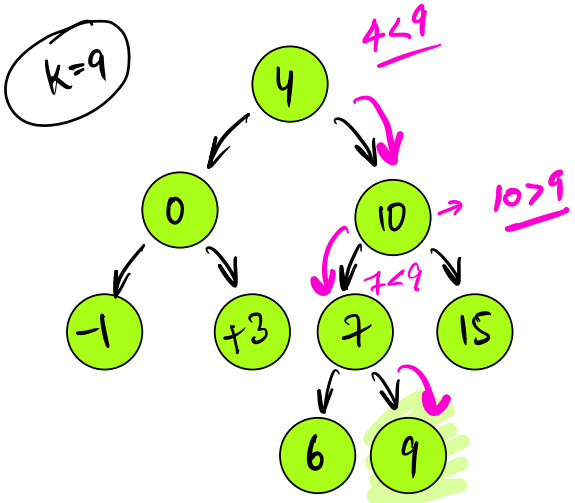
2 5 8 9 10 19 22 25 30

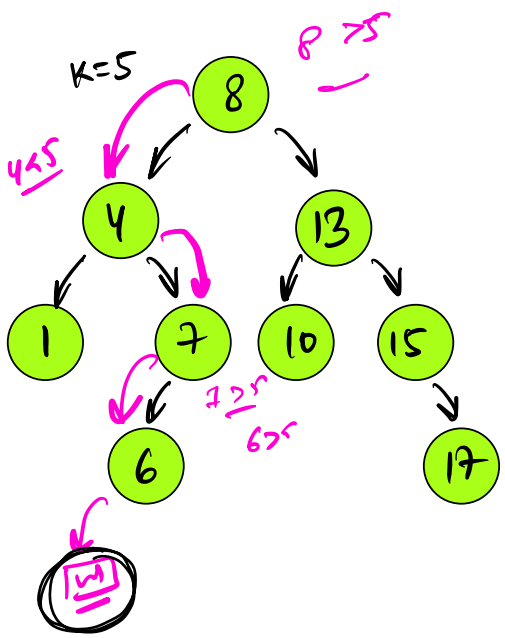
# inorder ( L Root R)

sorted

Left < Root < Right

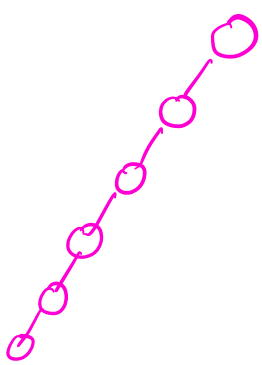
• search K in BST



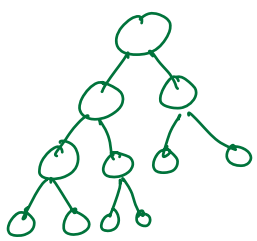
```
bool search ( Node root, int k)
{
        Node temp = root;

        while ( temp != null )
        {   if( temp.data = k) return true;
            else if( temp.data < k) temp = temp.right;
            else                    temp = temp.left;

        }
        return false;
}
```

T.C: O(Height) — log n ↕ N



skewed tree

O(n)



$\Rightarrow 2^0$
$\Rightarrow 2^1$
$\Rightarrow 2^2$
$\vdots$
$2^h$

$N = 2^0 + 2^1 + 2^2 \cdots 2^h$

$N = \frac{1(2^{h+1} - 1)}{2-1}$

$N = 2^{h+1} - 1$

$N+1 = 2^{h+1}$

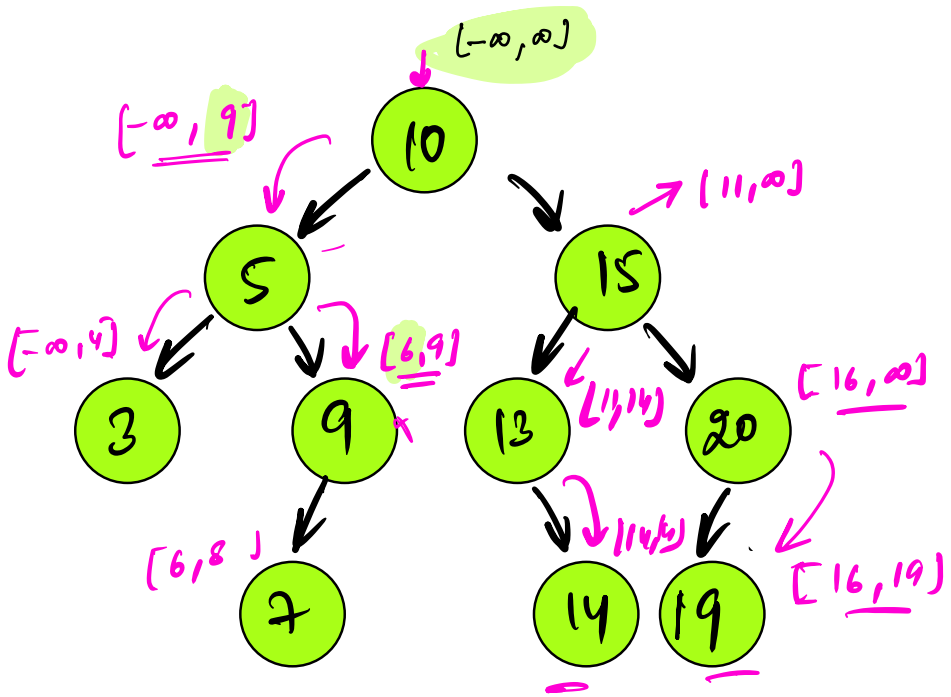
$\log_2 N \simeq h$

```
Node temp = root;
    prev = NULL;
while ( temp != null )  -> prev = temp;
{   if ( temp.data = k) return true;
    else if ( temp.data < k) temp = temp.right;
    else                     temp = temp.left;

}
if ( prev == NULL)  return new Node ( k);
if (  k < prev.data)
        prev.left = new Node ( k);
else
        prev.right = new Nod (k);
```

- check if it given tree is BST ?

Ans1 inorder traversal → sorted — O(N)

Ans2:

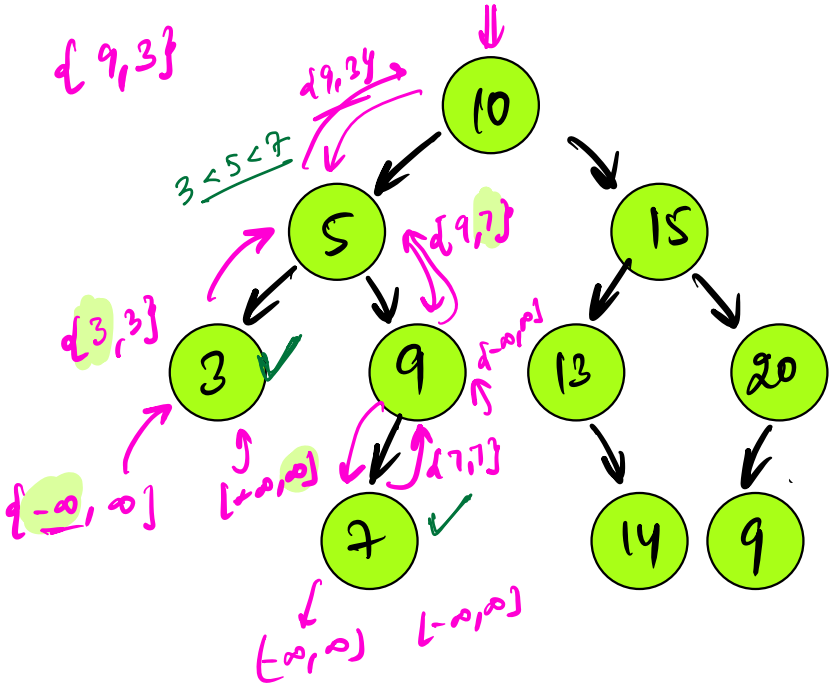



```
                                  ↑ INT_MIN      INT_MAX
bool isBST( root, l, r)
{
        if (root == null) return true;

        if( l <= root.data && root.data <= r)
        {
          bool x = isBST( root.left, l, root.data-1);
          bool y = isBST( root.right, root.data+1, r);
            return x && y;
        }
        return false;
}
```

A33

all nodes on LST < node < all nodes on RST

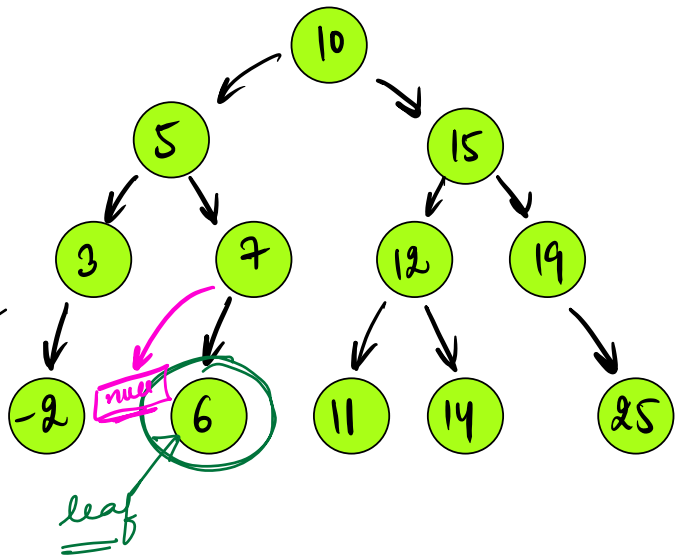
max(LST) < node < min(RST)




every node stored max/min of its subtree =

- deletion

Case I:- when the node to be deleted is leaf node

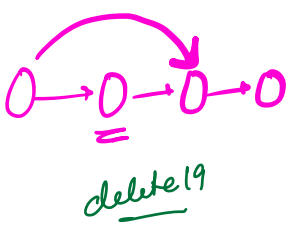


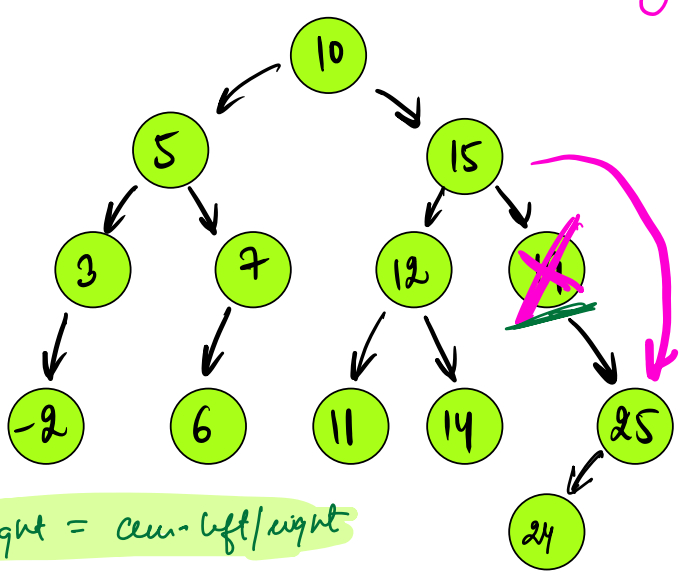

6

part/par

```
if( parent.left == curr)
    parent.left = null;
else
    parent.right = null;
```

Case II :- when node has 1 children



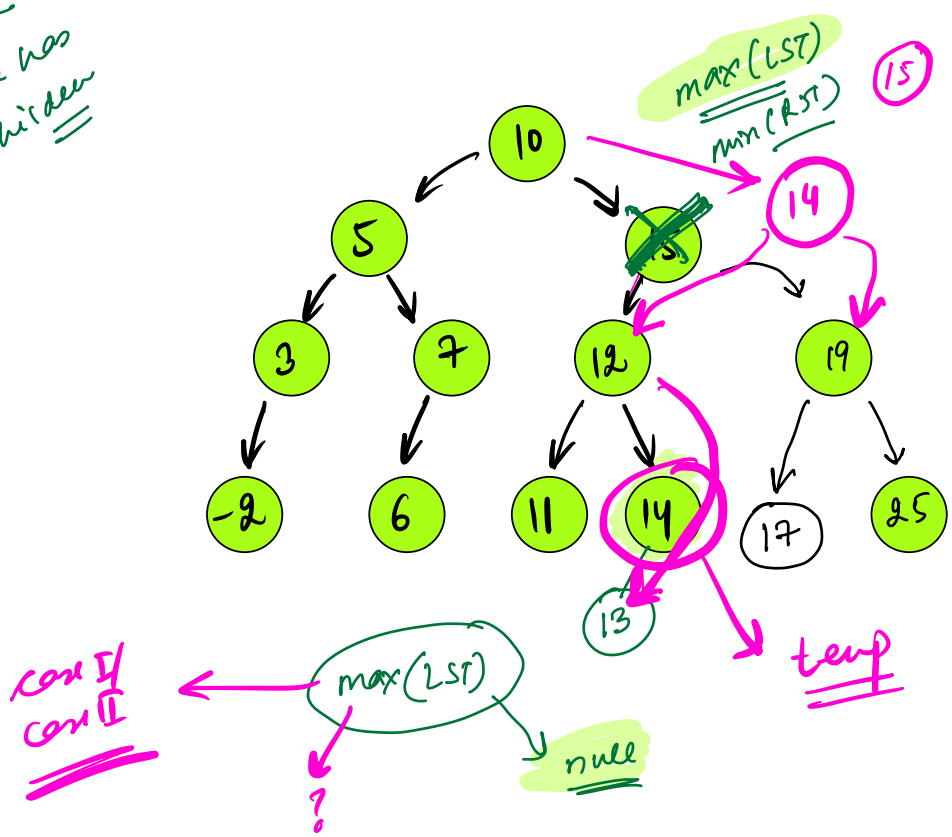



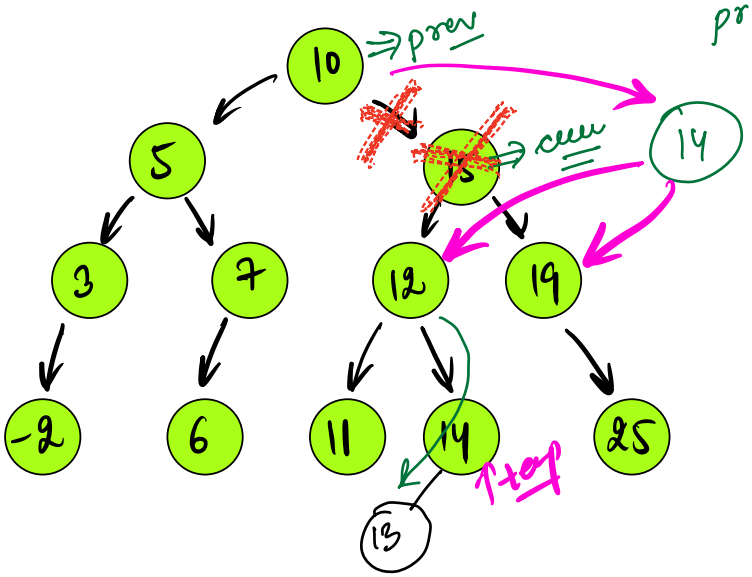

parent.left/right = curr.left/right

```
Node child;
if ( curr.left != null)
        child = curr.left;
else
        child = curr.right;

if ( parent.right == curr)
        parent.right = child;
else
        parent.left = child;
```

case III:

Node has 2 children






prev.left/right = temp;

temp.left = curr.left;

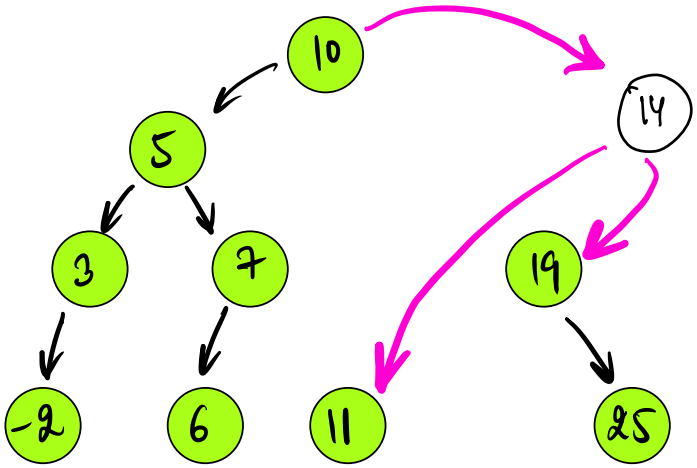
temp.right = curr.right;

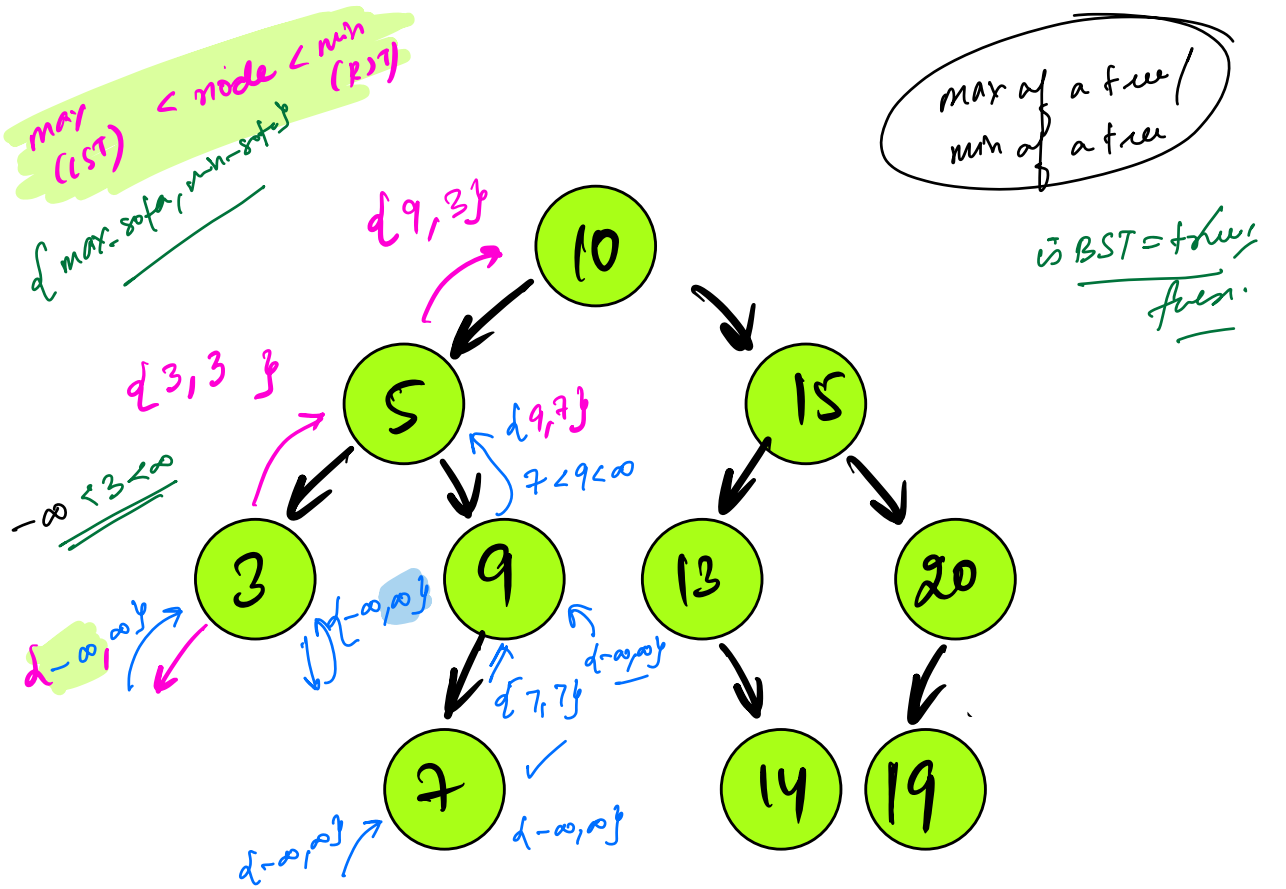
p = curr;

```
temp = curr.left;   p = null;
while( temp.right != null)
{
        p = temp;
        temp = temp.right;
}
```

p with child of temp

```
if( p == curr)
{
        p.left = temp.left;
}
else
{       p.right = temp.left;
}
```


10
5
14
3
7
19
-2
6
11
25




max of a tree = max( root, max(LST), max(RST))

min of a tree = min( root, min(LST), min(RST))

#