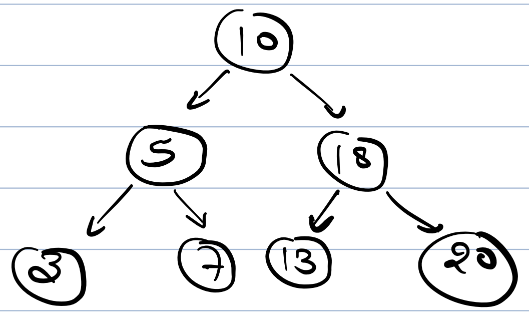
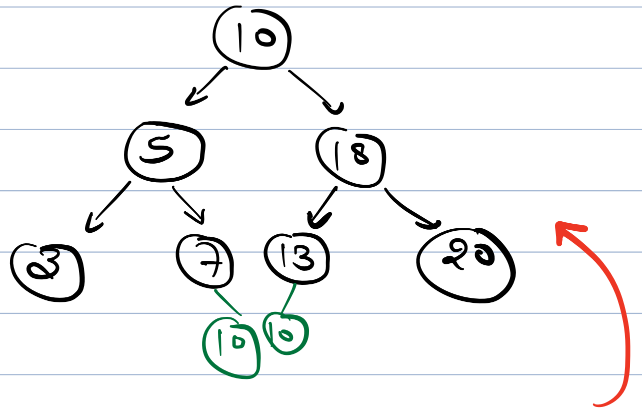
Check if a given tree is BST



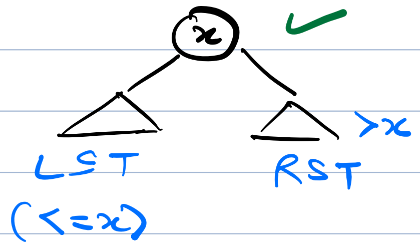
Inorder : 3, 5, 7, 10, 13, 18, 20



BST ??

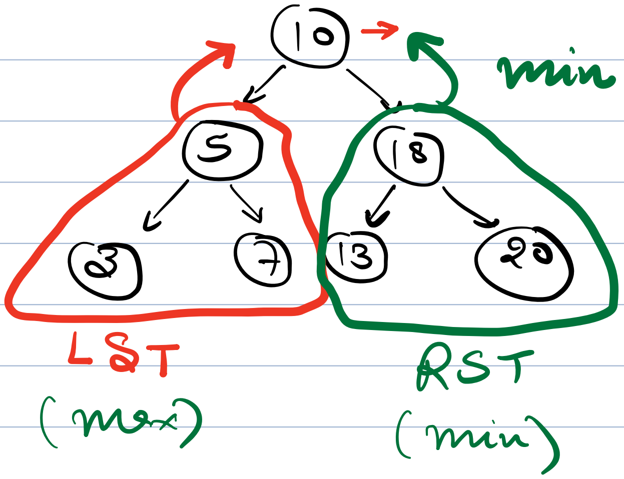
3, 5, 7, 10, 10, 10, 13, 18, 20

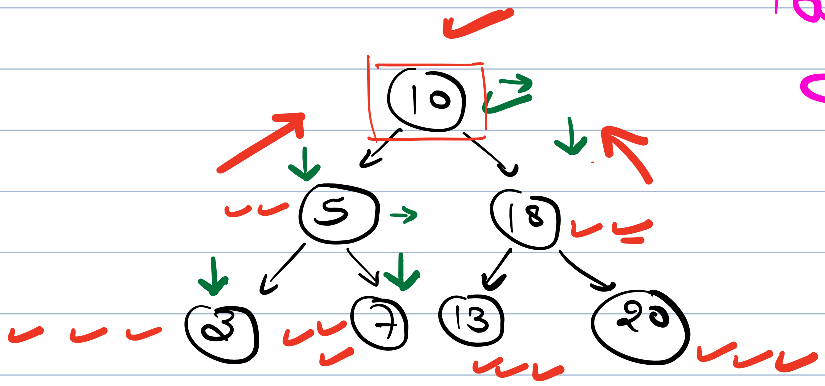
(Sorted)



LRN

NLR





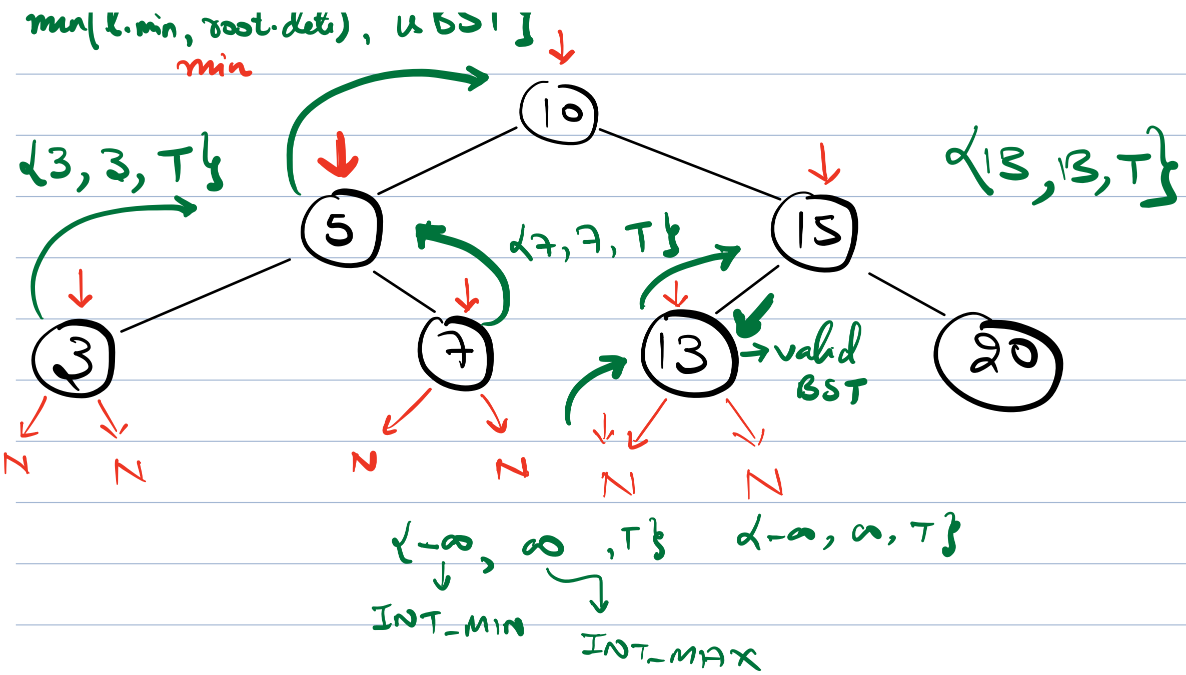
Post Order Traversal

$\underline{\underline{N^2}}$

```
class TreeNode {

        int max;
        int min;

        bool isBST;
}
```

L, R, N

{max(r.max, root.data,
max

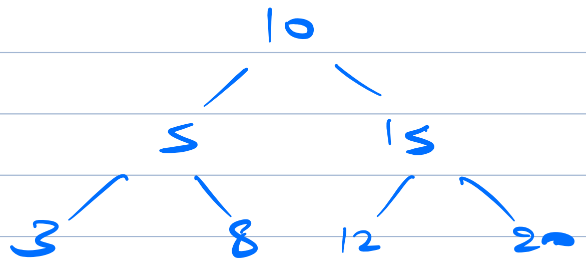


Code

```
TreeNode    checkBST ( Node  root) {

    if (root == NULL) {
          return   new  TreeNode (INT_MIN,
                                   INT_MAX, True);
    }

    TreeNode  l =  checkBST( root.left);
    TreeNode  r =  checkBST (root.right);

    if( l.isBST && r.isBST && (root.data >= l.max) &&
          (root.data < r.min) ) {
            return new TreeNode (
                        max(root.data, r.max),
                        min(root.data, l.min),
                        True);
    }
    return new TreeNode(0, 0, false);
}
```

T.C. = O(N)

Q Given a BST, Return the Kth smallest element.



K=3

Ans= 8

⇓

0	1	2	3	4	5	6
3	5	8	10	12	15	20

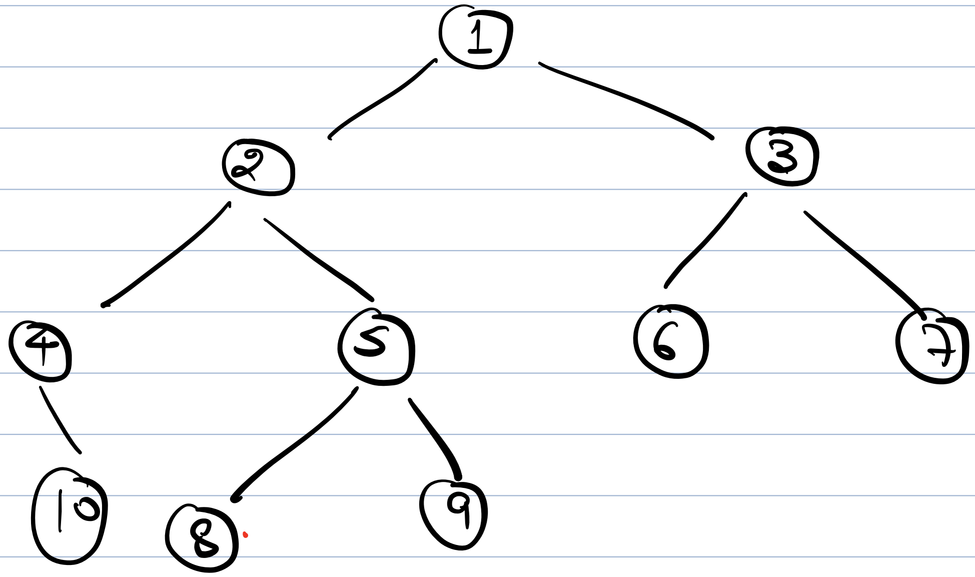
Inorder Traversal.

Code

```
ans = INT_MIN;
count = 0;

void inorder (root) {

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

    count++;

    if (count == k) {
                ans = root.data;
                return;
    }
    inorder (root.right);

}
```

In Order

L, N, R



4, 10, 2, 8, 5, 9, 1, 6, 3, 7,

Code

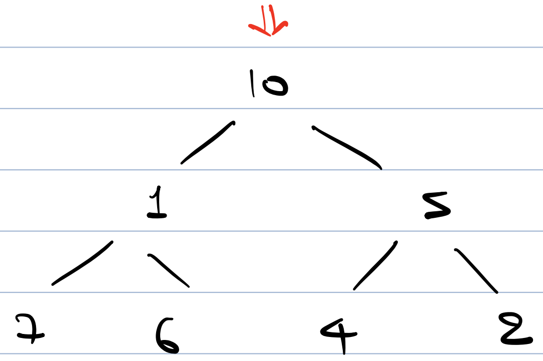
```
void  inOrder (root) {

      Node  curr = root;

      while (curr != NULL) {

            if (curr.left == NULL) {

                        print (curr.data);
                        curr = curr.right;
} else {
    Node temp = curr.left;

    while( temp.right != NULL && temp.right
          != curr) {

          temp = temp.right;
    }
    if (temp.right == NULL) {
             temp.right = curr ;
             curr = curr.left;
    }
    else {
          temp.right = NULL;
          print( curr. data);
          curr= curr. right;
    }
  }
 }
}
```

$T.C. = O(\underline{N})$

Dry

Search in Binary Tree

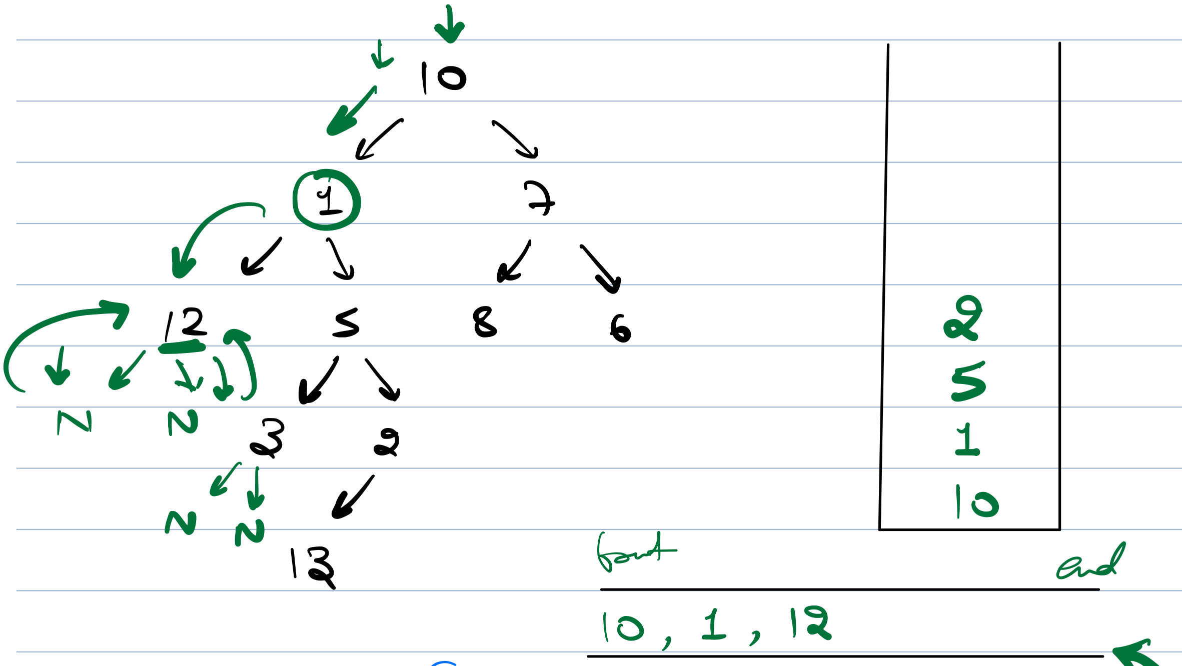
Pre : N, L, R
In: L, N, R
Post : L, R, N



```
bool search ( root, K) {

    if (root == NULL) { return false; }

    if ( root. data == K) {
N               return true;
    }
                L
    if ( search ( root. left, K) || search(
         root. right, K) ) {
            R       return  true;
    }
```

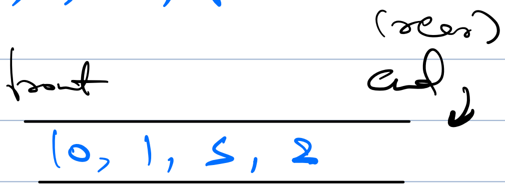
```
        return false;
}
```

Q Given a Binary Tree find the path from root to a given node.



Root to 2 : 10, 1, 5, 2

Code



```
Deque <int> q;
```

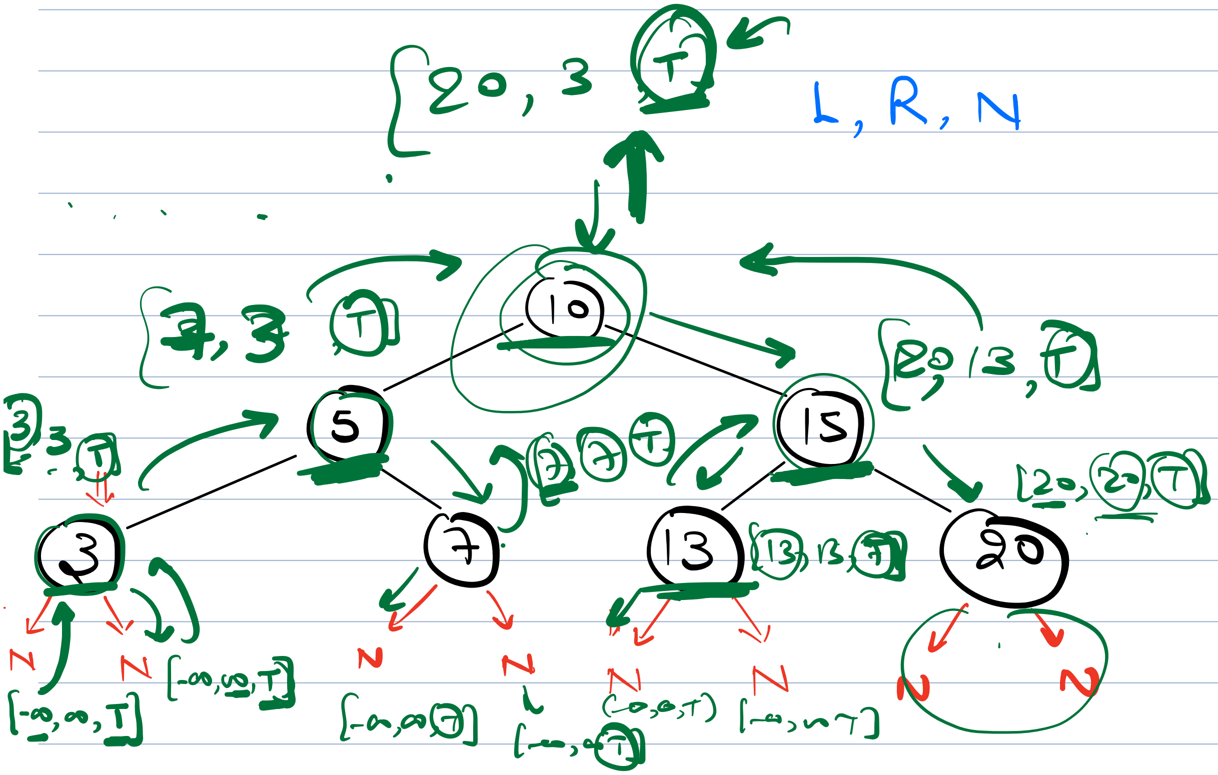
```
bool   createPath ( root , q , target) {
      if (root == NULL) { return   false; }
       q. add_end. ( root. data);
       if ( root. data  ==  target) {
                        return     True;
       } else {
              bool  l  = createPath (root.left, q,
                                          target);
              if ( l== true) {
                         return  true;
              }
              bool  r =  createPath (root. right,
                                       q, target);
              if (r= = true) {
                         return   true;
              }
      }
      q. pop_ end. ();
      return   false;
}
```

$T.C. = O(N)$

$S.C. = O(H) = \underline{O(N)}$

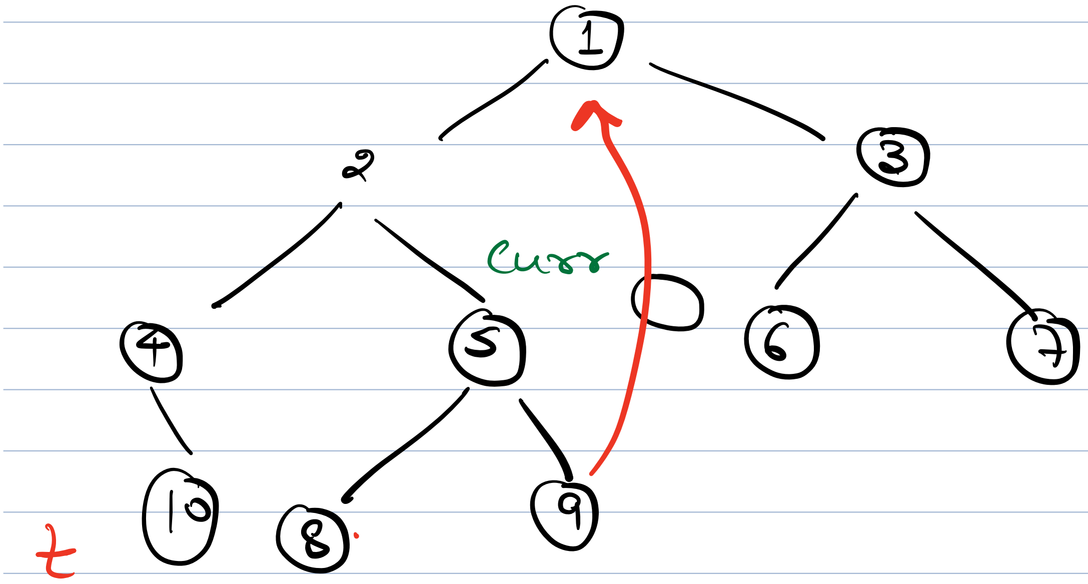
1 Extra session

Monday ⇒ Trees followUp



Code

```
TreeNode  checkBST ( Node  root) {

    if (root == NULL) {
        return  new  TreeNode(INT_MIN,
                              INT.MAX, True);
    }

    TreeNode  l =  checkBST( root. left);
    TreeNode  r =  checkBST (root. right);

    if( l.isBST && r.isBST && (root.data >= l.max) &&
        (root.data < r.min) ) {

        return  new  TreeNode (
                max(root.data ,  r.max),
                min (root.data , l. min),
                True);
    }
    return  new  TreeNode ( 0, 0,  false);
}
```



```
void  inOrder (root) {
      Node  curr = root;
      while (curr != NULL) {
          if ( curr. left = = NULL) {
                  print (curr. data);
                  curr = curr. right;
          } else {
              Node  temp  =  curr. left;

              while ( temp. right != NULL  &&  temp. right
                      != curr) {
```

```
                temp = temp.right;
        }
        if (temp.right == NULL){
                temp.right = curr ;
                curr = curr.left;
        }
        else {
              temp.right = NULL;
              print( curr. data);
              curr = curr.right;
        }
    }
  }
}
```