Today's Content :-

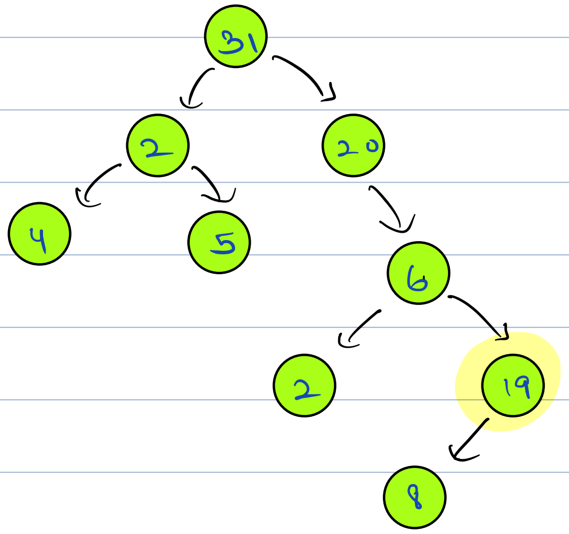
Morris Inorder Traversal

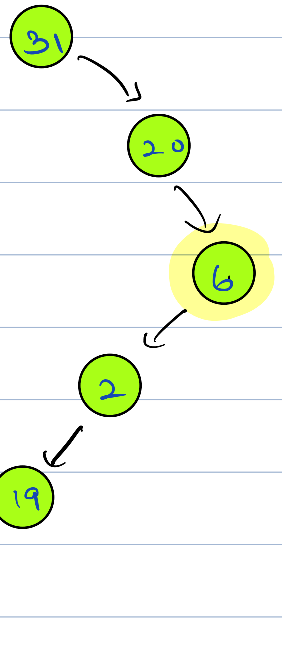
kth Smallest Element in a BST

LCA in BST

LCA in BT.

Ques) with inorder Traversal on a Tree, last Node we print.





claim :- In inorder Traversal of B.T, last node will always be right most node of root.

T.C → O(N) ← Inorder → Recursively → S.C → O(H)

→ Iteratively → S.C → O(H)

→ Morris → S.C → O(1).

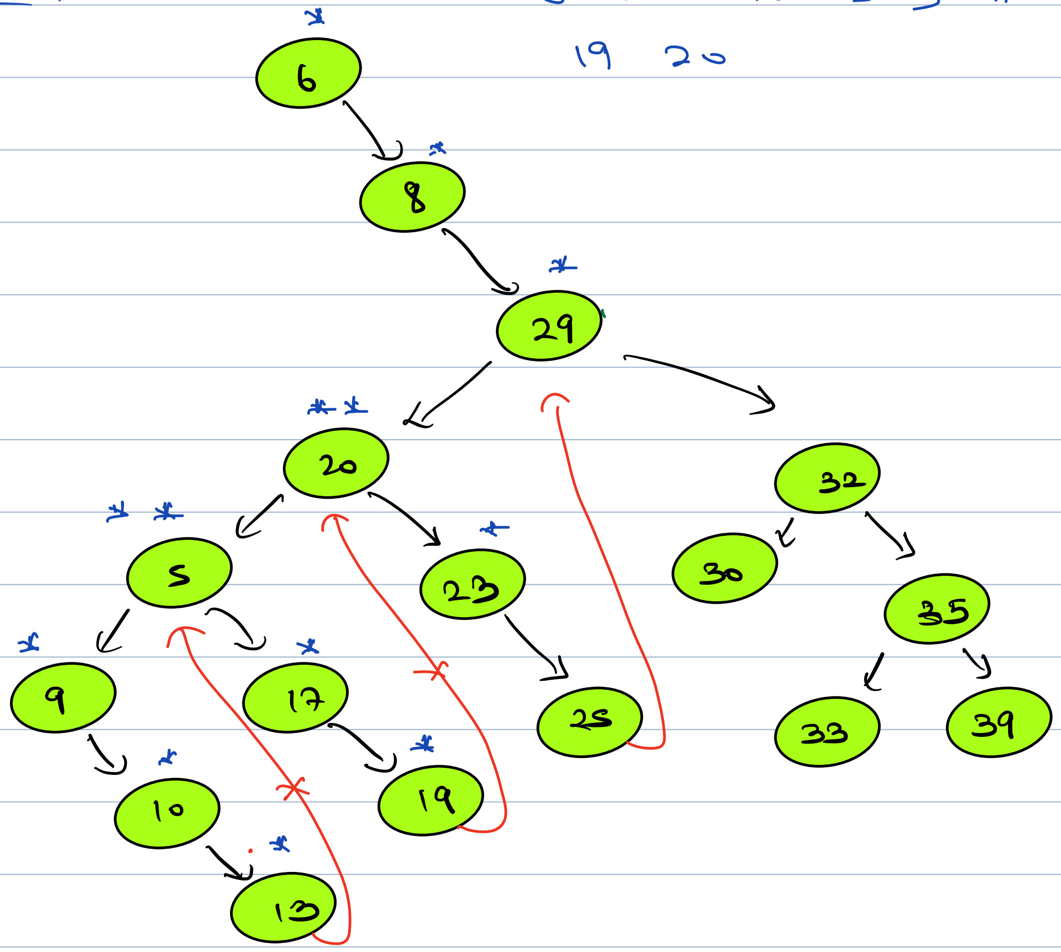
Ques) Morris Inorder Traversal {HARD}

↳ {no extra space}

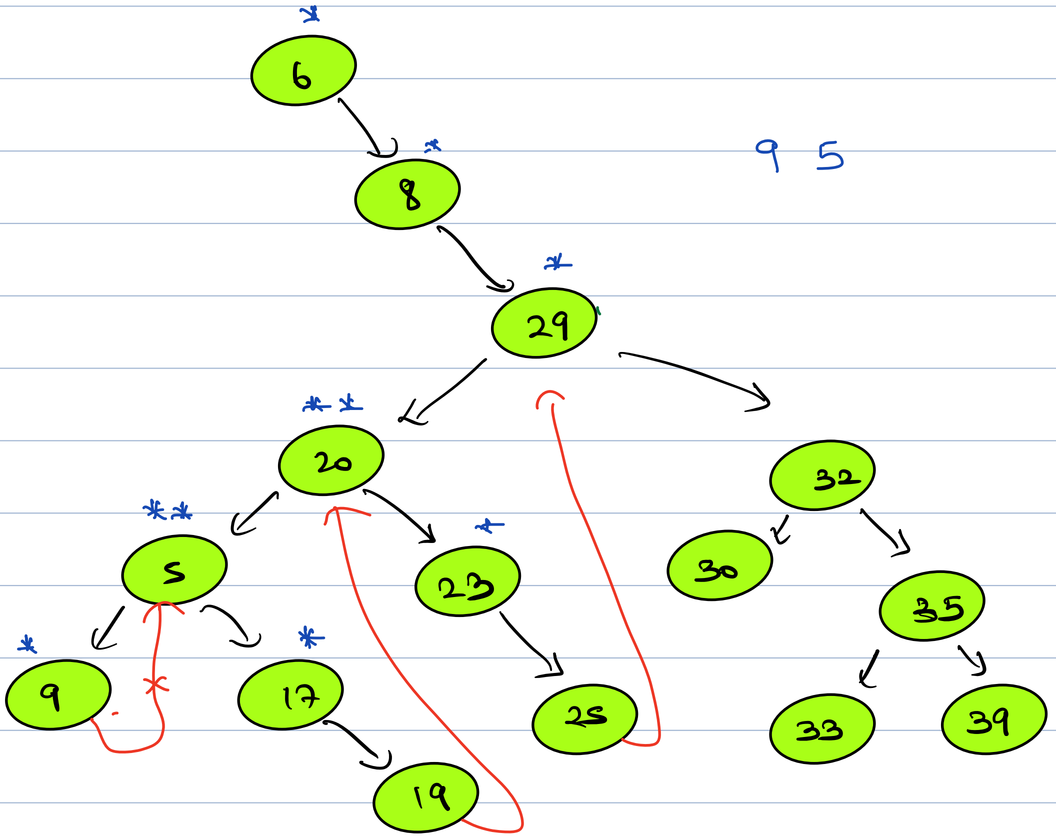
LDR.

6 8 9 10 13 5 17

19 20

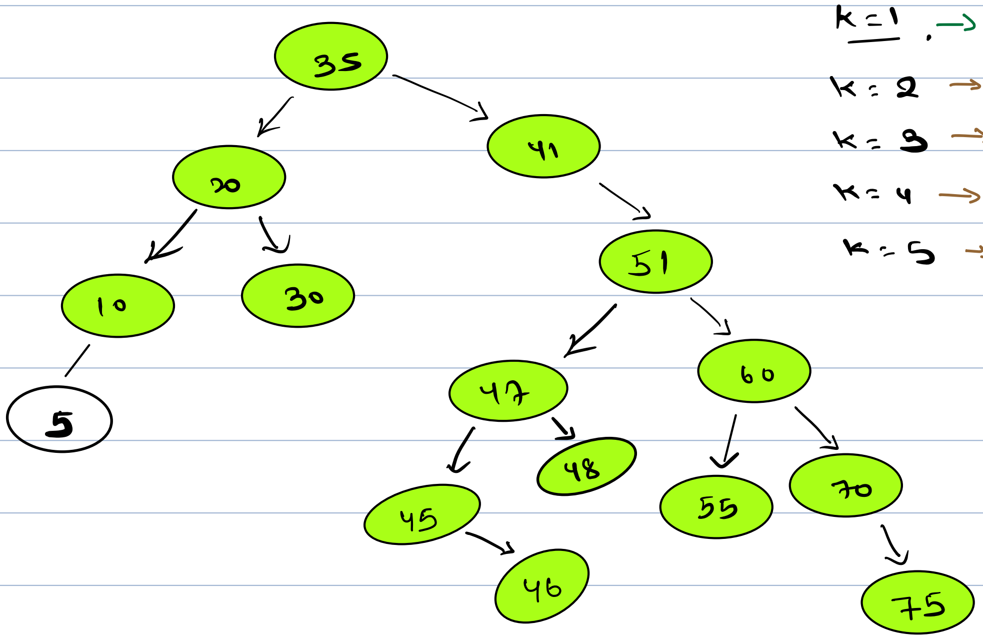


9 5



```
inorder (Node 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) {              1st time
                temp.right = curr;
                curr = curr.left
            }
            else if (temp.right == curr) {         2nd time
                temp.right = null;
                print (curr.data);
                curr = curr.right;
            }
        }
    }
                        T.C -> O(n)
                        S.C -> O(1)
}
```

Ques   Kth Smallest element in a BST.



k=1 → 5

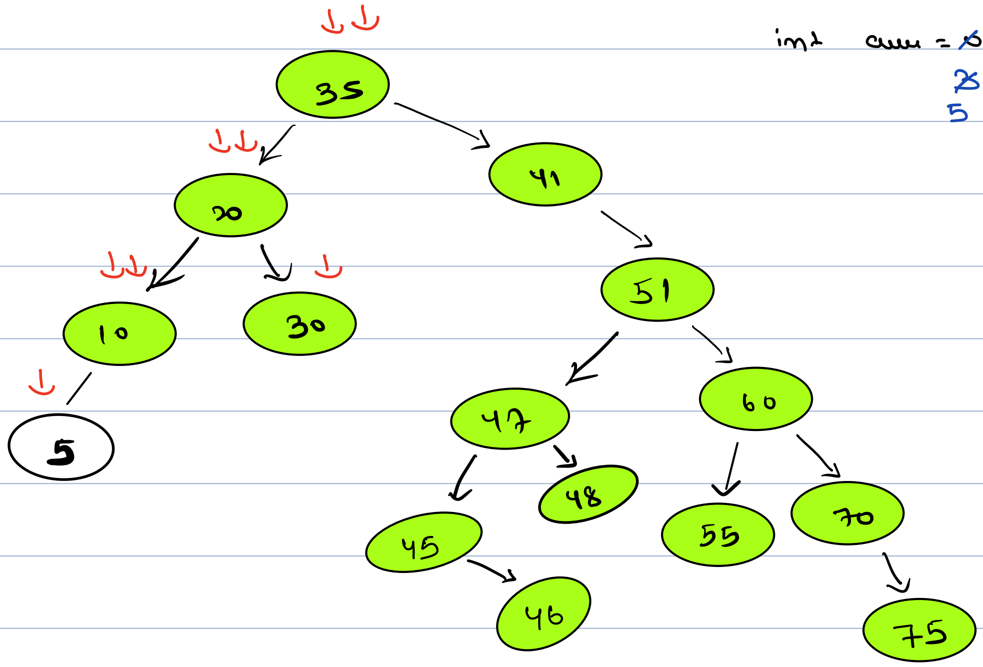
k=2 → 10

k=3 → 20

k=4 → 30

k=5 → 35

idea :- Do inorder Traversal, & store elements in list, return arr[k-1] element.



int ans = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

int k = 5;

```
        int    count = 0;
        int    ans  =  — ;

public  void   inorder  ( Node root , k)
 |
 |        if (root == null) { return }
 |
 |        inorder (root.left, k);
 |           count++;
 |            if (count == k) {  ans = root.data , return }
 |          inorder (root. right , k);
 |
  }
```

T.C → O(n), S.C → O(H)

idea 3 :- use Morris Traversal.

↳ T.C → O(n)

S.C → O(1)

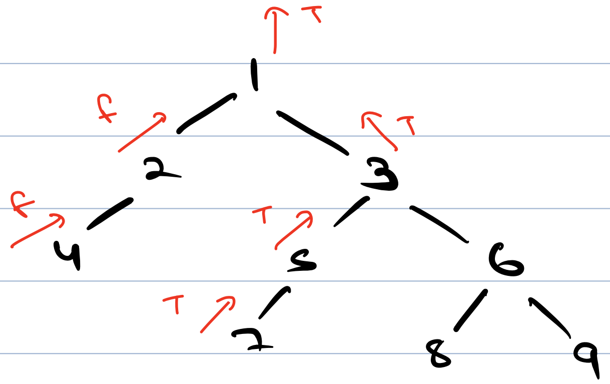
Break 8:13 am - 8:23 am

Root to Node Path

k = 6 (1, 3, 6).

search

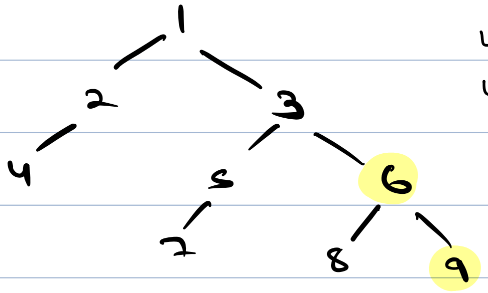
k = 7 (1, 3, 5, 7)



T.C → O(N)
S.C → O(H)

```
List <Node> ans;

boolean search ( Node root, int k){

        if (root == null){ return false }

      if (root.data == k) {
               ans.add(root);
               return True
        }

        if (search (root.left, k)) {
                  ans.add(root);
                 return True;
         }
        if (search (root.right, k)) {
                  ans.add(root);
                 return True;
         }
        return false
}
```

Lowest Common Ancestor



LCA(7,8) = 3 → {1,3,5,7} {1,3,6,8}

LCA(8,9) = 6 → {1,3,6,8} {1,3,6,9}

LCA(7,6) = 3 → {1,3,5,7} {1,3,6}

LCA(3,7) = 3

LCA(6,9) = 6 → {1,3,6} {1,3,6,9}

LCA(x,y)

find root to node path of x & y, find last common node in both arrays.

T.C → O(n)

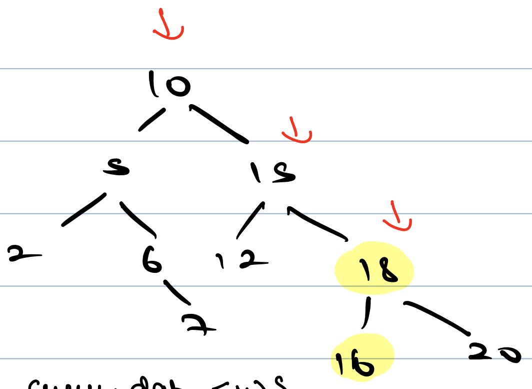
S.C → O(n)

* LCA in BST .

LCA(12,16)

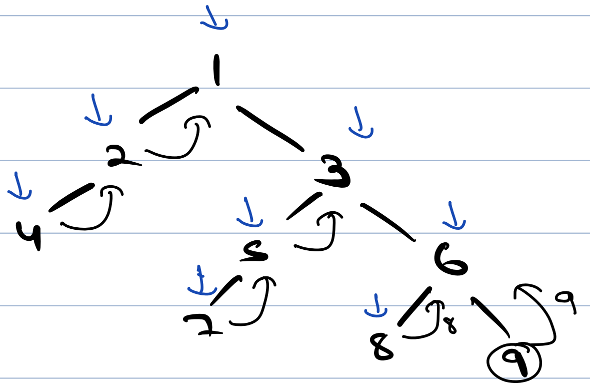
T.C → O(H)

S.C → O(1)

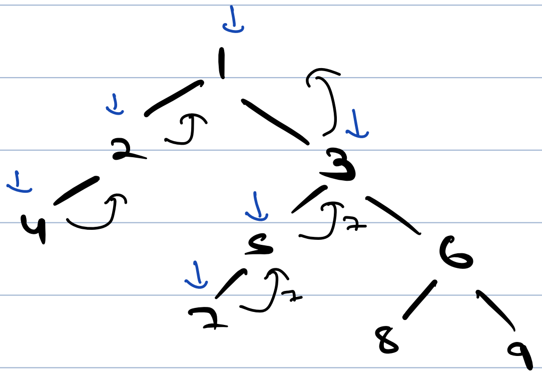


```
curr = root;
while (curr != null){
    if (curr.data < x && curr.data < y){
        curr = curr.right;
    }
    else if (curr.data > x && curr.data > y){
        curr = curr.left;
    }
    else{
        return curr;
    }
}
```

L.C.A in B.T



LCA(8,9)

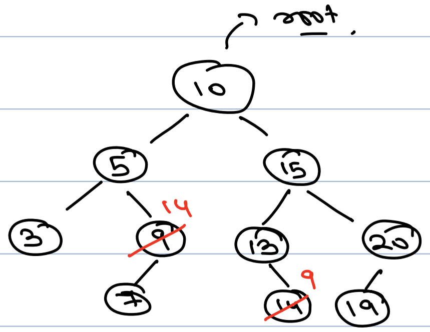


LCA(7,9) = 3

LCA(3,7)

Code → Todo.

Ques Recover BST

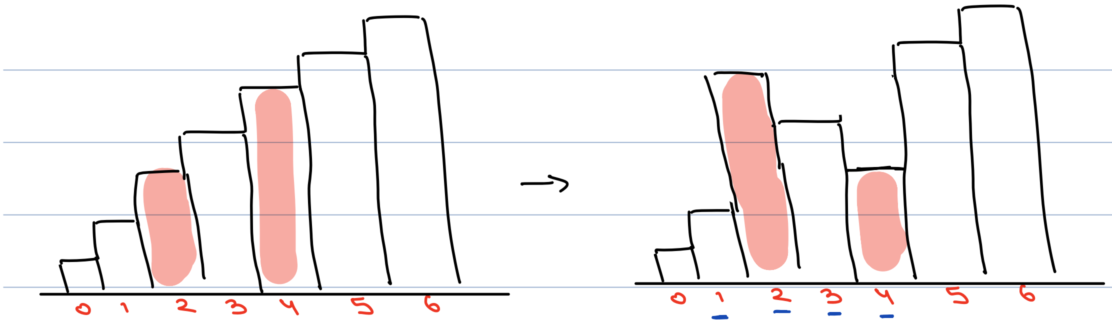


Two nodes of a BST are swapped, find the two nodes.

Soln :- Inorder of BST is sorted :-

3 5 7 14 10 13 9 15 19 20

case -1



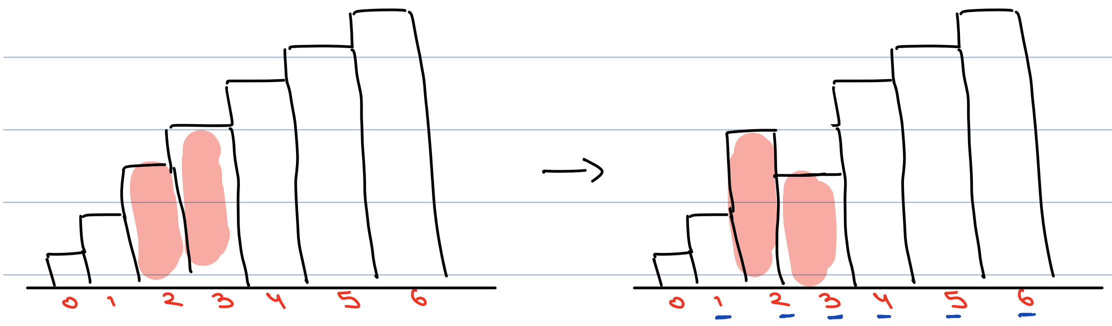
arr[i] ~~>~~ arr[i-1]

first time
prob1 = arr[i-1];

second time
prob2 = arr[i];

3 5 7 14 10 13 9 15 19 20

case - 2



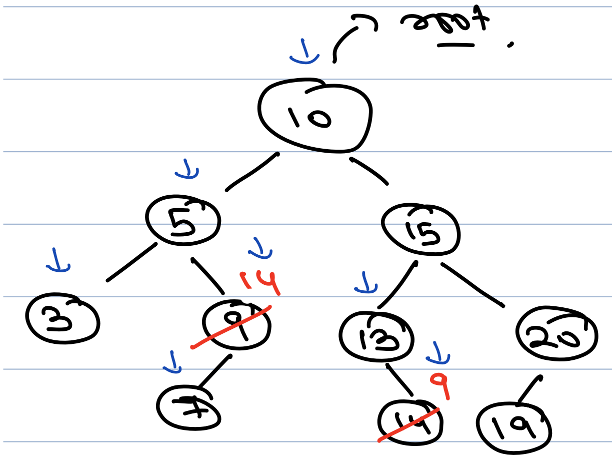
arr[i] $\not>$ arr[i-1]

first time

```
    prob1 = arr[i-1];
    prob2 = arr[i];
```

second time

```
    prob2 = arr[i];
```



arr[i] ~~>~~ arr[i-1]

first time

prob1 = arr[i-1];
prob2 = arr[i];

second time

prob2 = arr[i];

prev = ~~null~~ ~~3~~ ~~5~~ ~~7~~ ~~14~~ 10 13
first = ~~null~~ 14
second = ~~null~~ ~~10~~ 9

```
Node f = s = p = null;
```

T.C -> O(N)
S.C -> O(H)

```
inorder (Node root ) {
            if (root == null) { return }
          inorder (root. left) ;
            if (p != null && root.data < p.data) {
                if (f == null) {
                           f = p;
                           s = root;
                }
                else {
                           s = root;
                }
            }
            p = root;
             inorder (root. right);
}
```

idea 3 :- use Morris Traversal ,

T. C → O(n)

S. C → O(1) ,