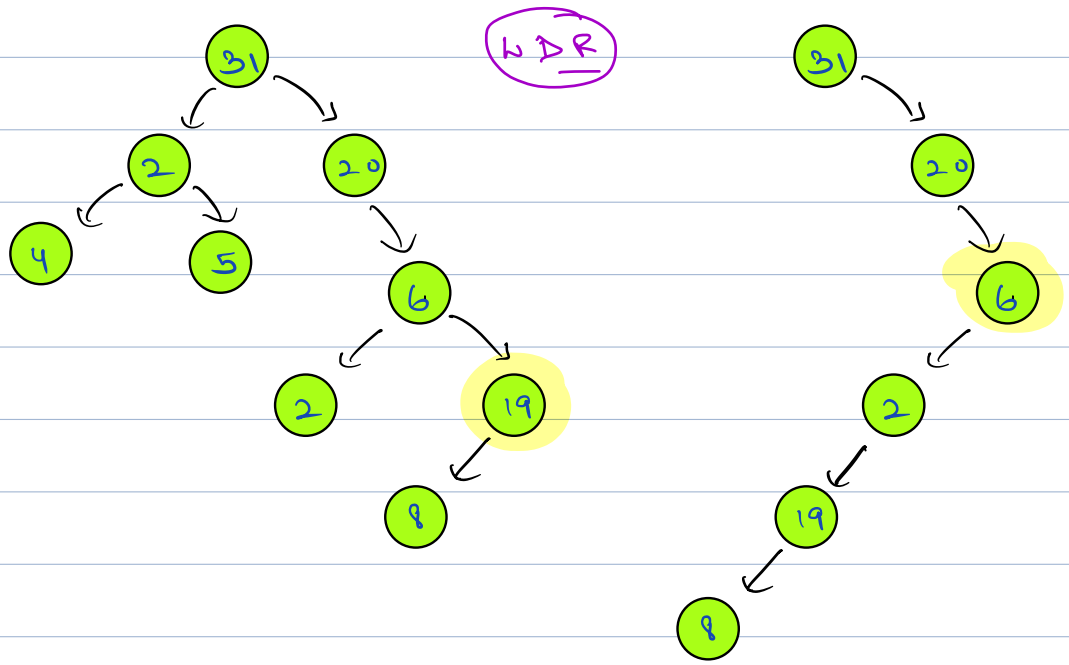
Today's Content :-

- Morris Inorder Traversal
- k^{th} 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.

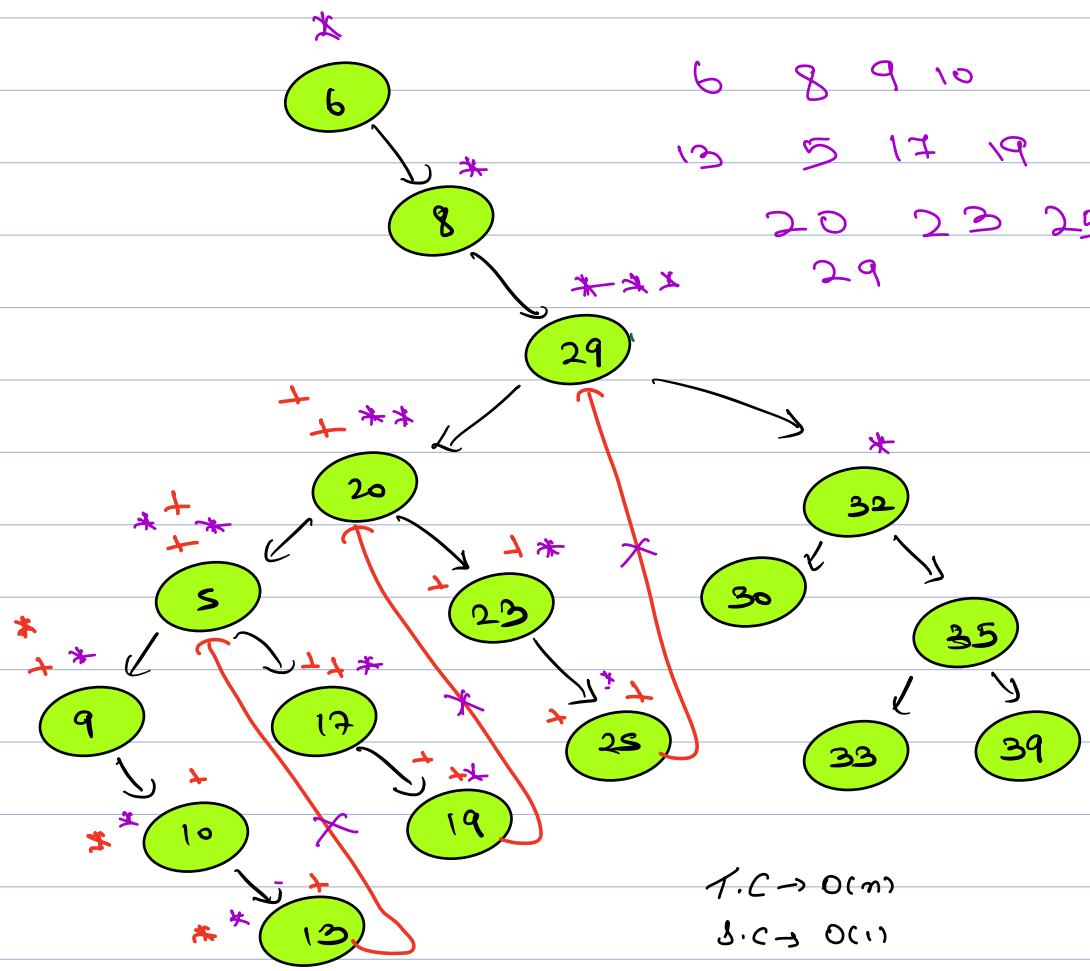
T.C → O(N) ← Inorder → Recursively → S.C → O(H)
→ Iteratively → S.C → O(H)
→ Morris → S.C → O(1).

Ques) Morris Inorder Traversal {HARD}

↳ {no extra space}

L D R

6 8 9 10 13



6 8 9 10
13 5 17 19
20 23 25
29

T.C → O(n)
S.C → O(1)

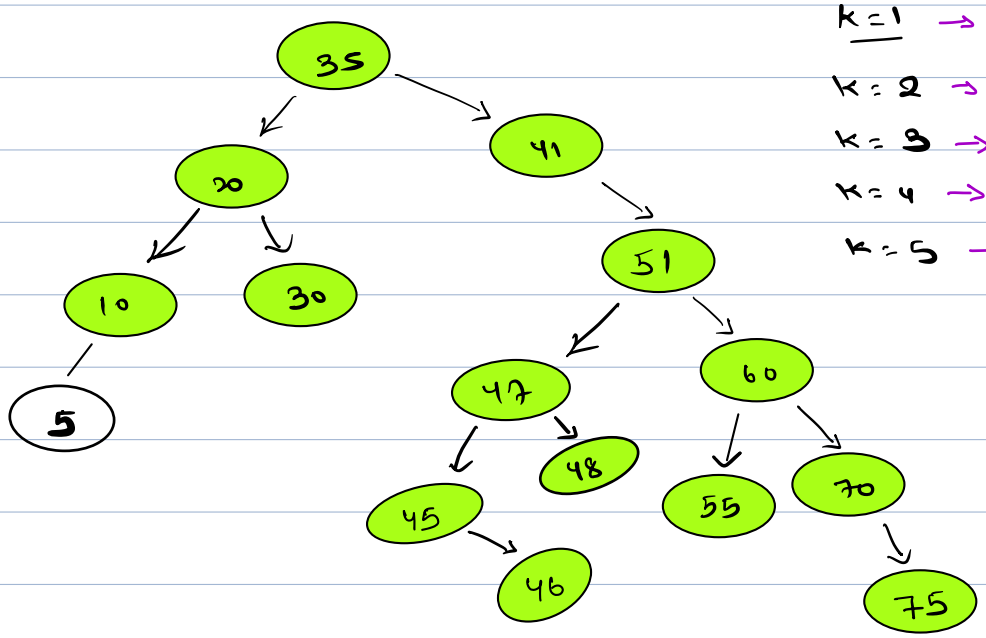
```
inorder (Node root) {

    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;
            }
        }
    }
}
```

1st time

2nd time

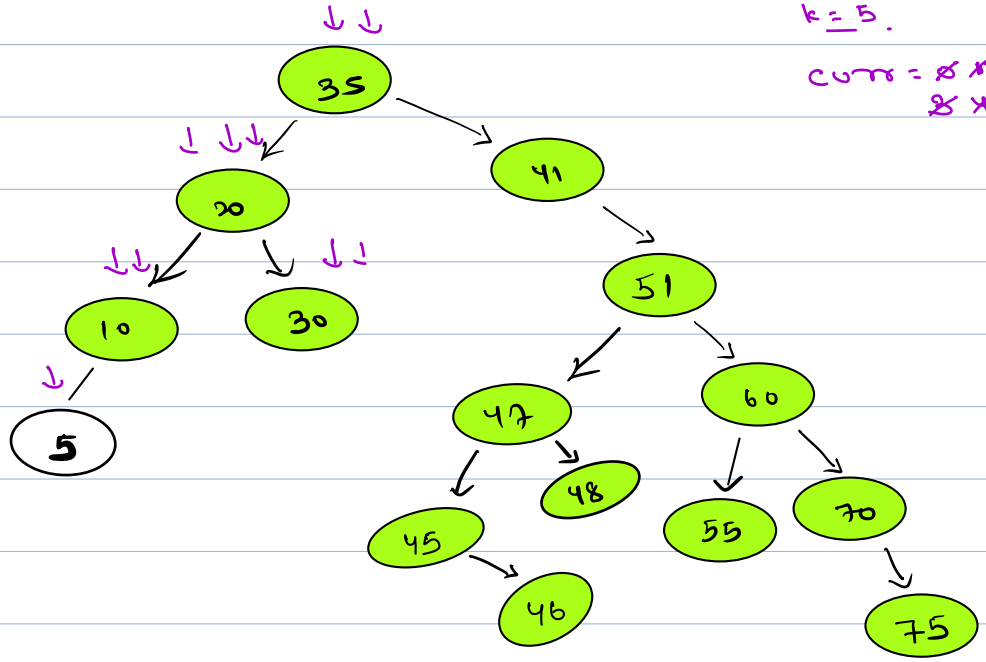
Break 9:48pm - 9:59pm

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.



k=5.
curr = ~~0~~ ~~1~~ 2
~~3~~ ~~4~~ 5

```
ans = 0;
count = 0;

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

idea :- Do morris inorder traversal.

T.C → O(n).

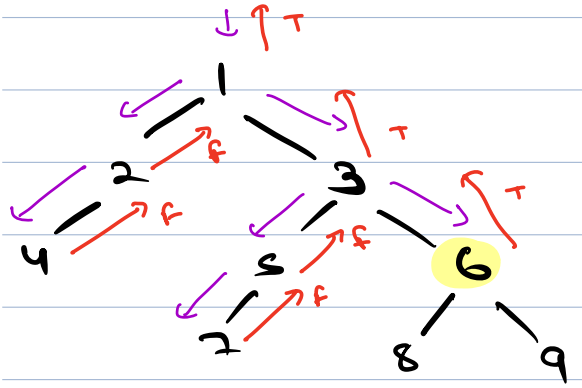
S.C → O(1)

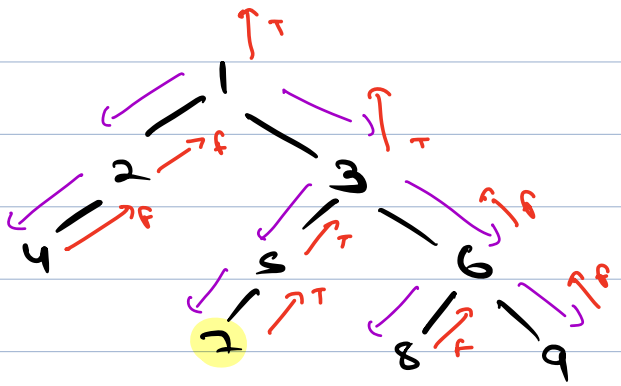
Root to Node Path

k = 6 (1, 3, 6).

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

Search

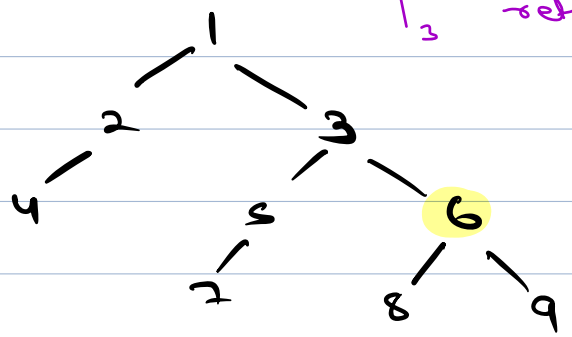


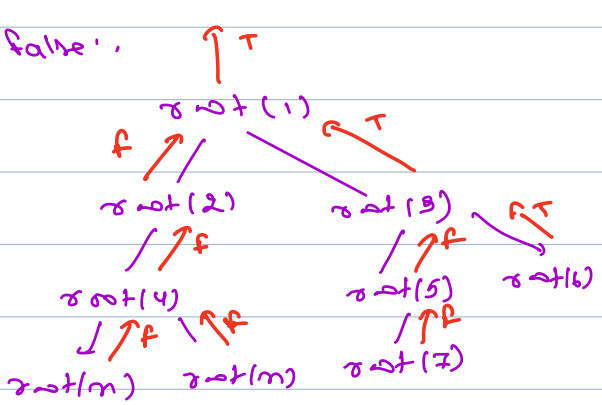


```
search (root, k, List<Int> ans) {
    if (root == null) {return false}

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

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

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

    return false;
}
```

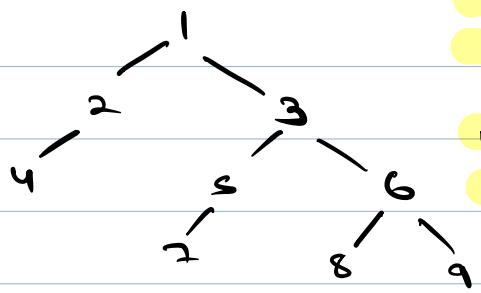
T.C → O(n)

S.C → O(H).





Ques Lowest Common Ancestor of a B.T.



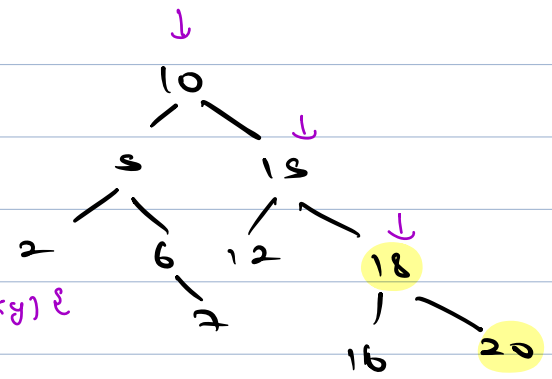
LCA(7,8) → 7 → 1 3 5 7
8 → 1 3 6 8
LCA(8,9) → 6 → 8 → 1 3 6 8
9 → 1 3 6 9
LCA(7,6) → 3 → 7 → 1 3 5 7
6 → 1 3 6
LCA(3,7) → 3 → 3 → 1 3
7 → 1 3 5 7
LCA(6,9) → 6 → 6 → 1 3 6
9 → 1 3 6 9

LCA(x,y) → find root to node paths of x & y, find last common value in both arrays.

10:33pm - 10:38pm Break

* LCA in BST.

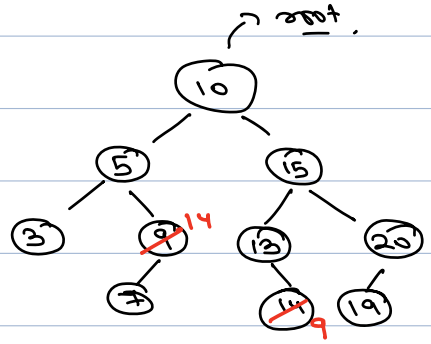
LCA(12,16)



T.C → O(n)
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;
    }
}
```

Ques Recover BST

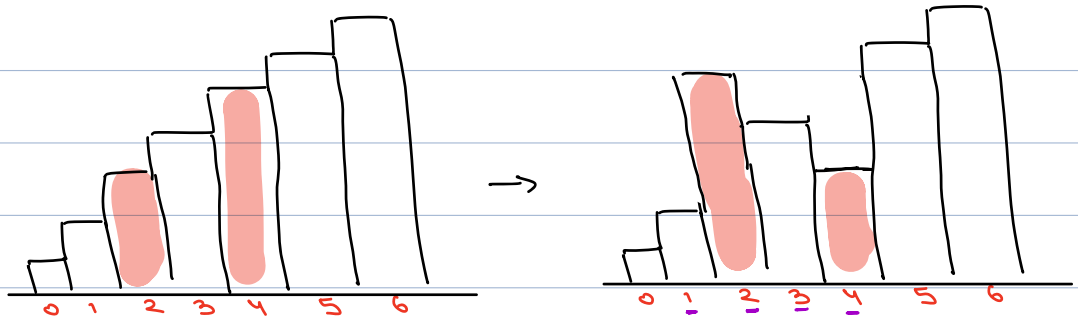


Two nodes of a BST are swapped, find the two nodes.

Soln :- Inorder of BST is sorted :-

3 5 7 14 10 19 9 15 19 20

Case -1



arr[i] < arr[i-1]

first time

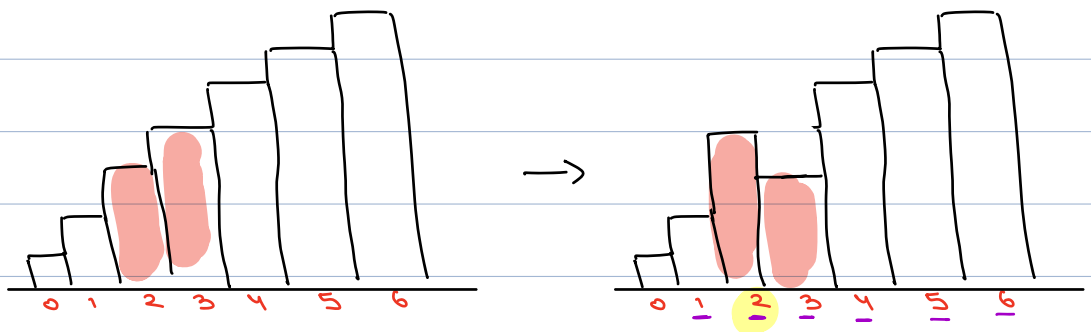
prob1 = arr[i-1];

second time,

prob2 = arr[i];

3 5 7 14 10 13 9 15 19 20

case - 2



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

first time

prob1 = arr[i-1];

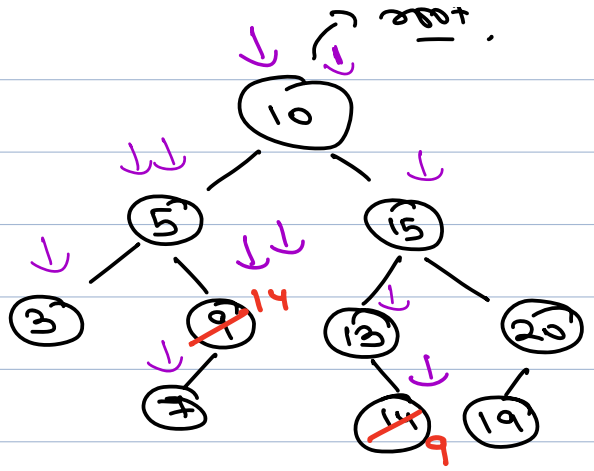
prob2 = arr[i];

second time,

prob2 = arr[i];

T.C → O(n)
S.C → O(h)

y = null 14
s = null 9
p = null 5 13




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

(8) we move