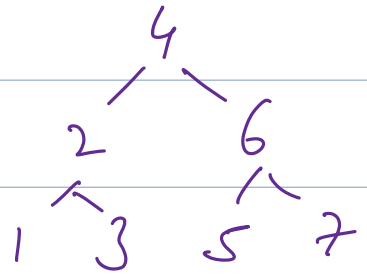
Q k^{th} smallest elem in BST



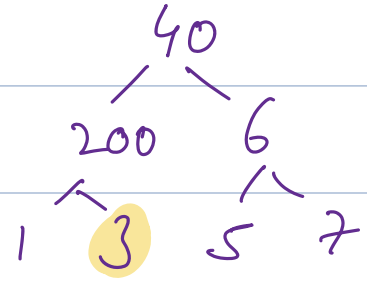
$k = 3$

ans = 3

⇒ Inorder of a BST is sorted

Get inorder & return k^{th} elem

Q Finding an elem in binary tree



$k = 3$

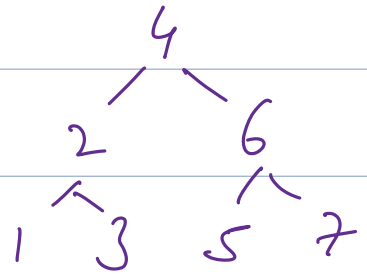
ans = true

Idea: traverse the tree & find the elem

Code

```
bool find (Node node){
    if (node == null)  return false
    if (node.val == k) return true
    x = find (node.left)
    y = find (node.right)
    return x | y          x OR y
}
```

Q Path from root to node



k = 3

ans = 4 → 2 → 3

Idea: Check if elem exist in left or right subtree.

Dry run on above example

4 2 3

Code

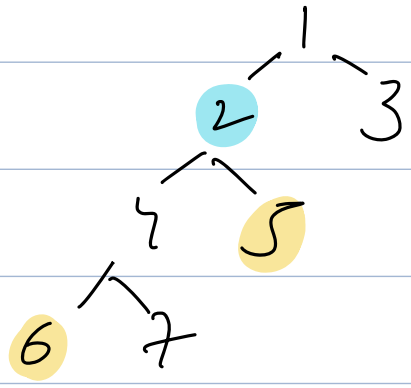
```
list<int> ans

bool findpath (Node node){
    if (node == = null)  return  false
   if (node.val == = target) {
       ans.add(node)
       return true
   }

   result = findpath (root.left) |
            findpath (root.right)

  if (result == true)
     ans.add (node)
  return result
}
```

LCA ⇒ Lowest Common Ancestor



6 → 1 2 4 6

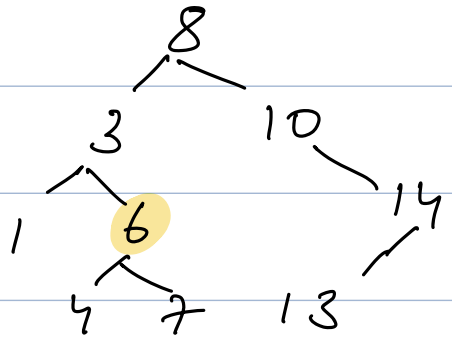
5 → 1 2 5

how to get LCA of 2 nodes

hint: use the paths from root

Last common node is the LCA

LCA in BST

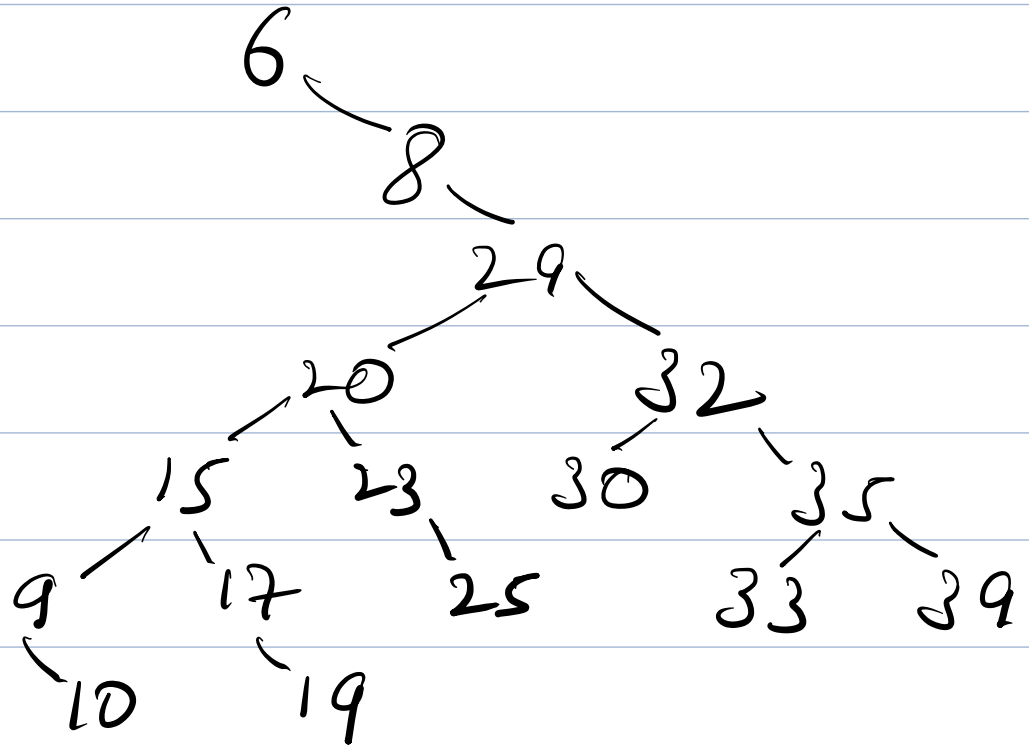


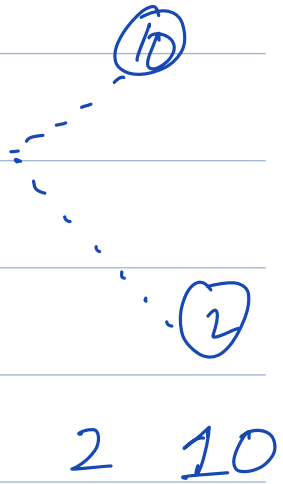
4 & 7

```
while (root != null) {
    if (root.data < val1 && root.data < val2)
        root = root.right
    else if (root.data > val1 && root.data > val2)
        root = root.left
    else    return root
}
```

04 Morris Inorder Traversal.

Print inorder traversal in O(1) SC





6 8 9 10 15 17 19 20
23 25 29 30 32 33 35 39

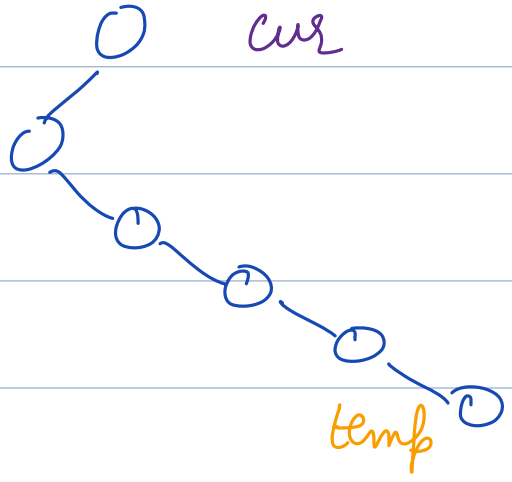
Inorder predecessor is right most of left child

curr

temp = cur.left

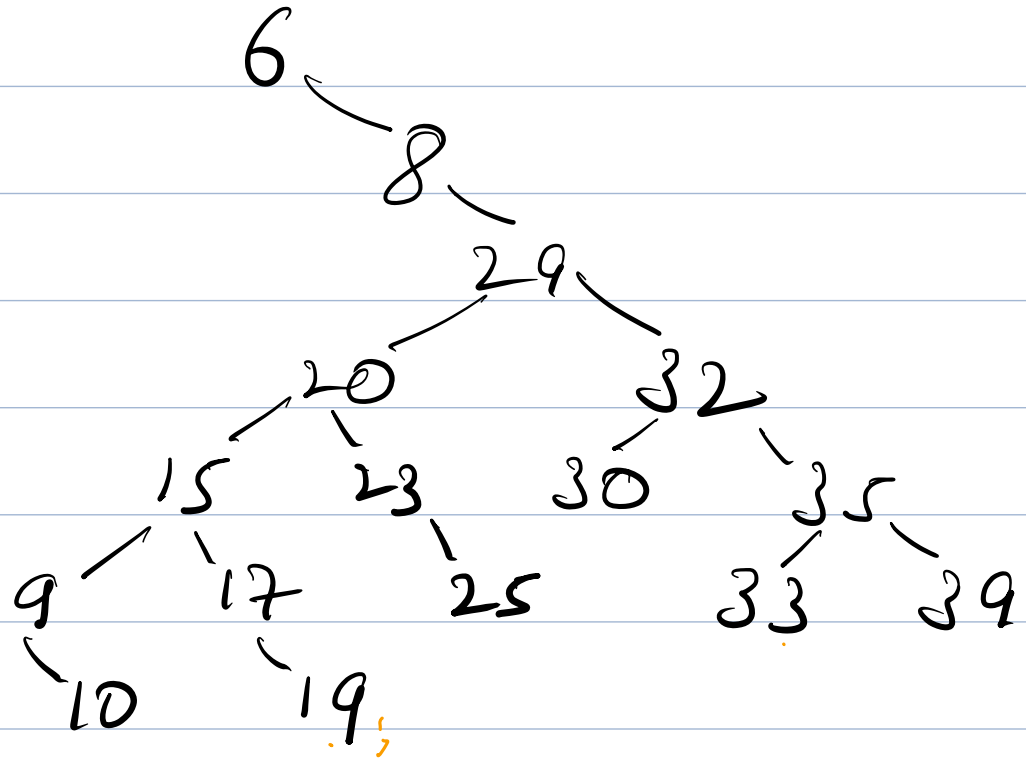
while (temp.right != null)

temp = temp.right



How does morris traversal work

↳ name of scientist



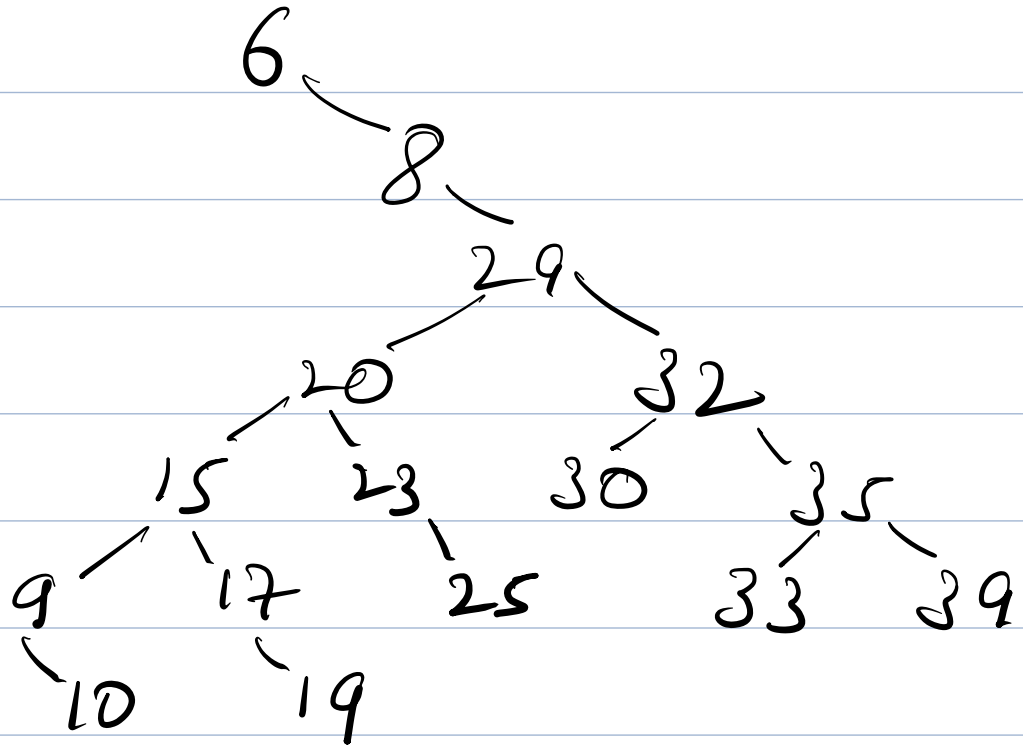
6 8 9 10 15 17 19 20 23 25

29 30 32 33 35 39

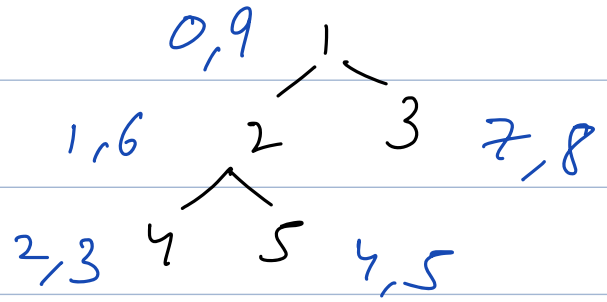
```
curr = root
while (curr != null) {
  if (curr.left == null) {
    print (cur.data)
    cur = cur.right
  }
  else {          // if you have left child
    pred = cur.left
    while(pred.right != null &&
          pred.right != curr) {
      pred = pred.right
    }
    if (pred.right == null) {
      pred.right = cur
      cur = cur.left
    }
    else {
      pred.right = null
      print (cur.data)
      cur = cur.right
```

TC: O(N)

SC: O(1)



In time - out time

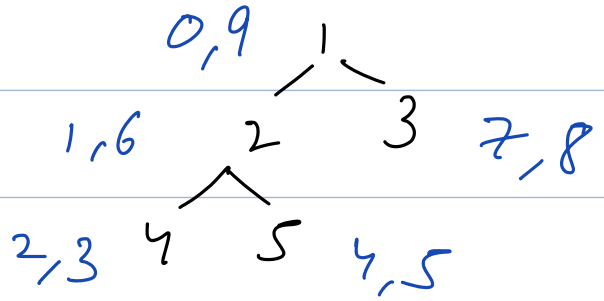


time = 0 1 2 3 4 5 6 7 8 9 10

Code intime / outtime is array

```
time = 0
void traverse (Node node) {
    intime[node] = time
    time ++
    if (root.left != null)
        traverse (root.left)
    if (root.right != null)
        traverse (root.right)
    outtime[node] = time
    time ++
}
```

LCA & intime outtime



ancestor relationship

anc

in_1 out_1

child

in_2 out_2

$in_1 < in_2$ &&

$out_1 > out_2$

x y

in_1 out_1 in_2 out_2

$in_1 < in_2$

$out_1 > out_2$ $LCA(x, y) = x$

path (x) → 1 x

path (y) → 1 . . . . x . . . . . y

{done}