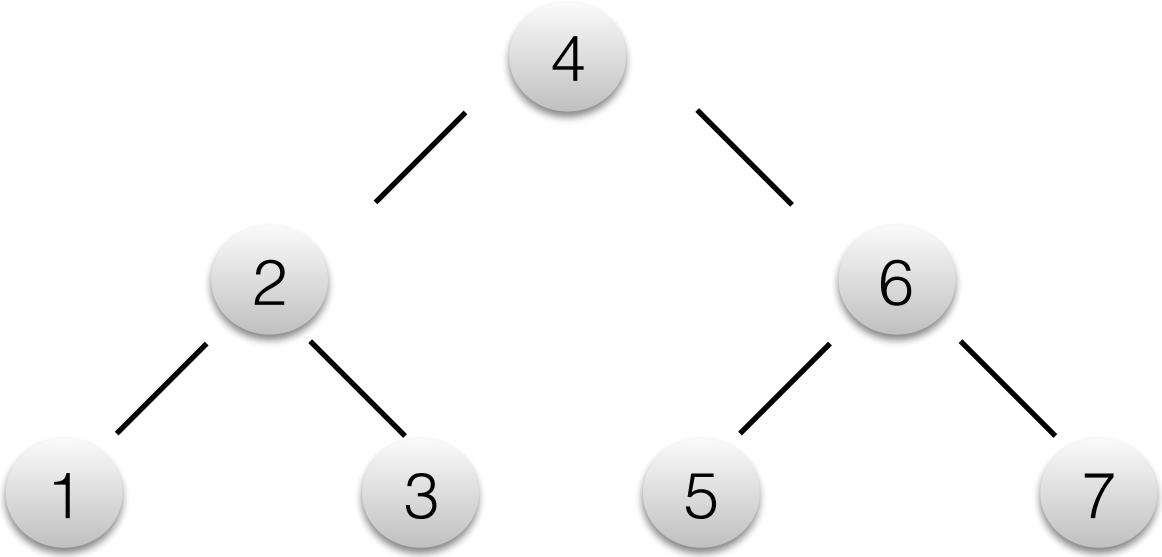
BITTIGER

CLASS_3
BINARY STRUCTURE

Content of Class_3

Binary Search	***Binary Tree***	***Binary Search Tree***
Search in Rotated Sorted Array	*Invert Tree*	*Inorder Successor*
	Symmetric Tree	*Kth Smallest Element in BST*
	Postorder Traversal	

145. Binary Tree Postorder Traversal



inorder : left root right 1 2 3 4 5 6 7

preorder : root left right 4 2 1 3 6 5 7

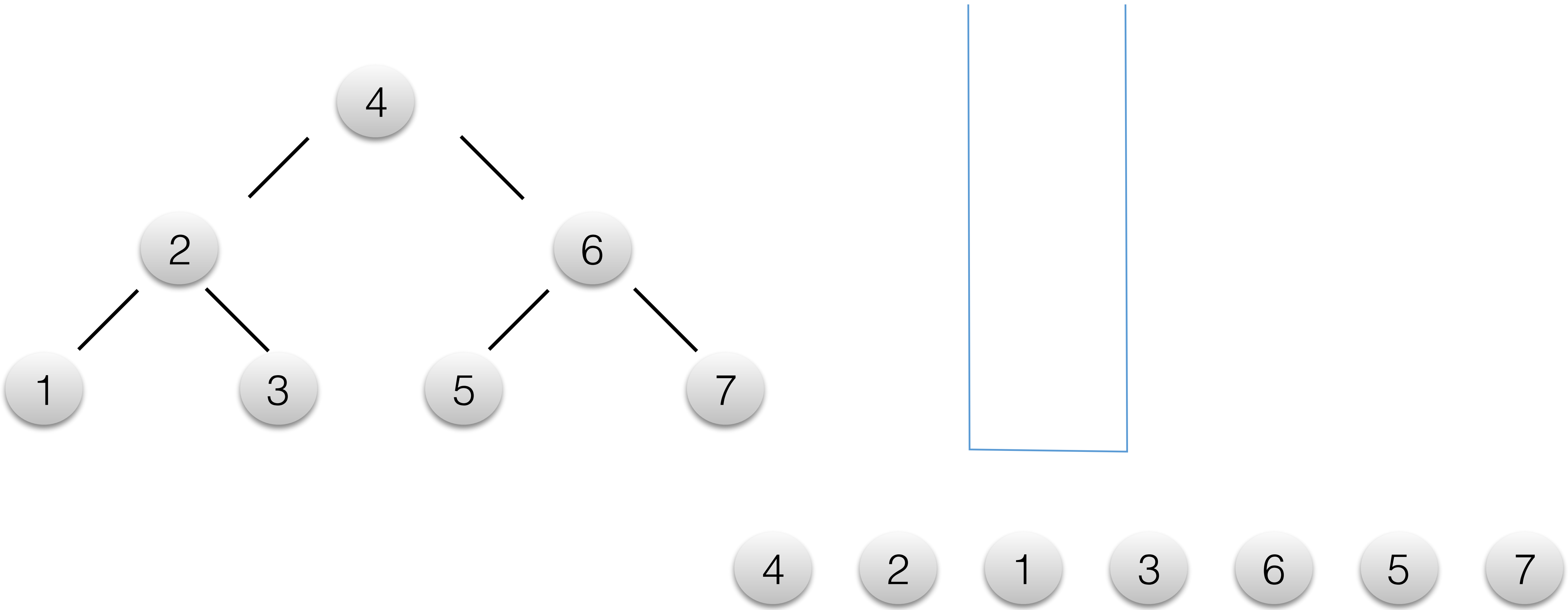
postorder : left right root 1 3 2 5 7 6 4

10 min

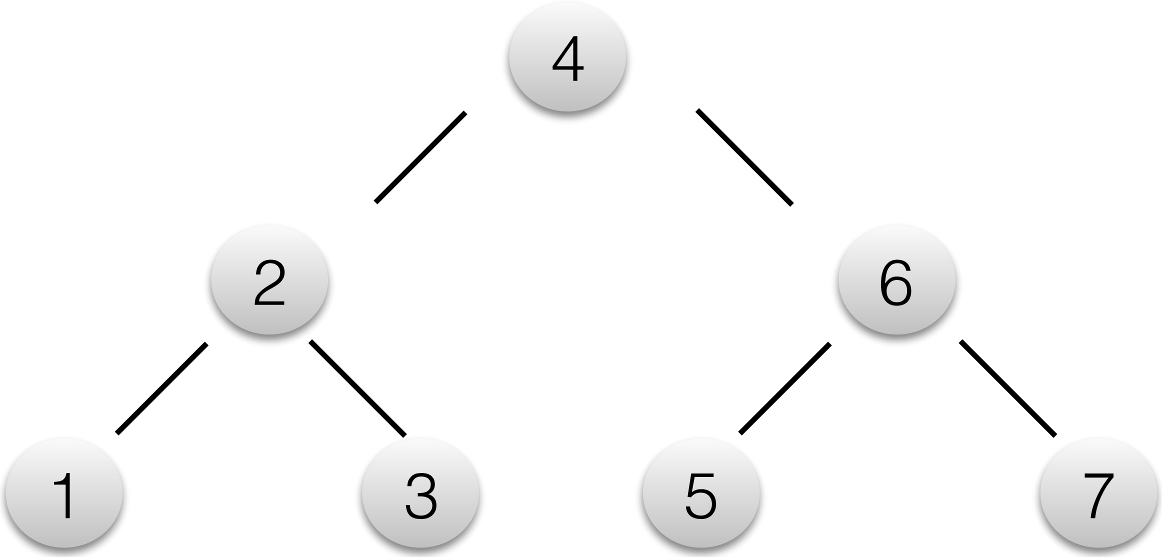
```
public List<Integer> postorderTraversal(TreeNode root){

}
```

Follow Up: One Stack

preorder



How to do it using Iteration ?

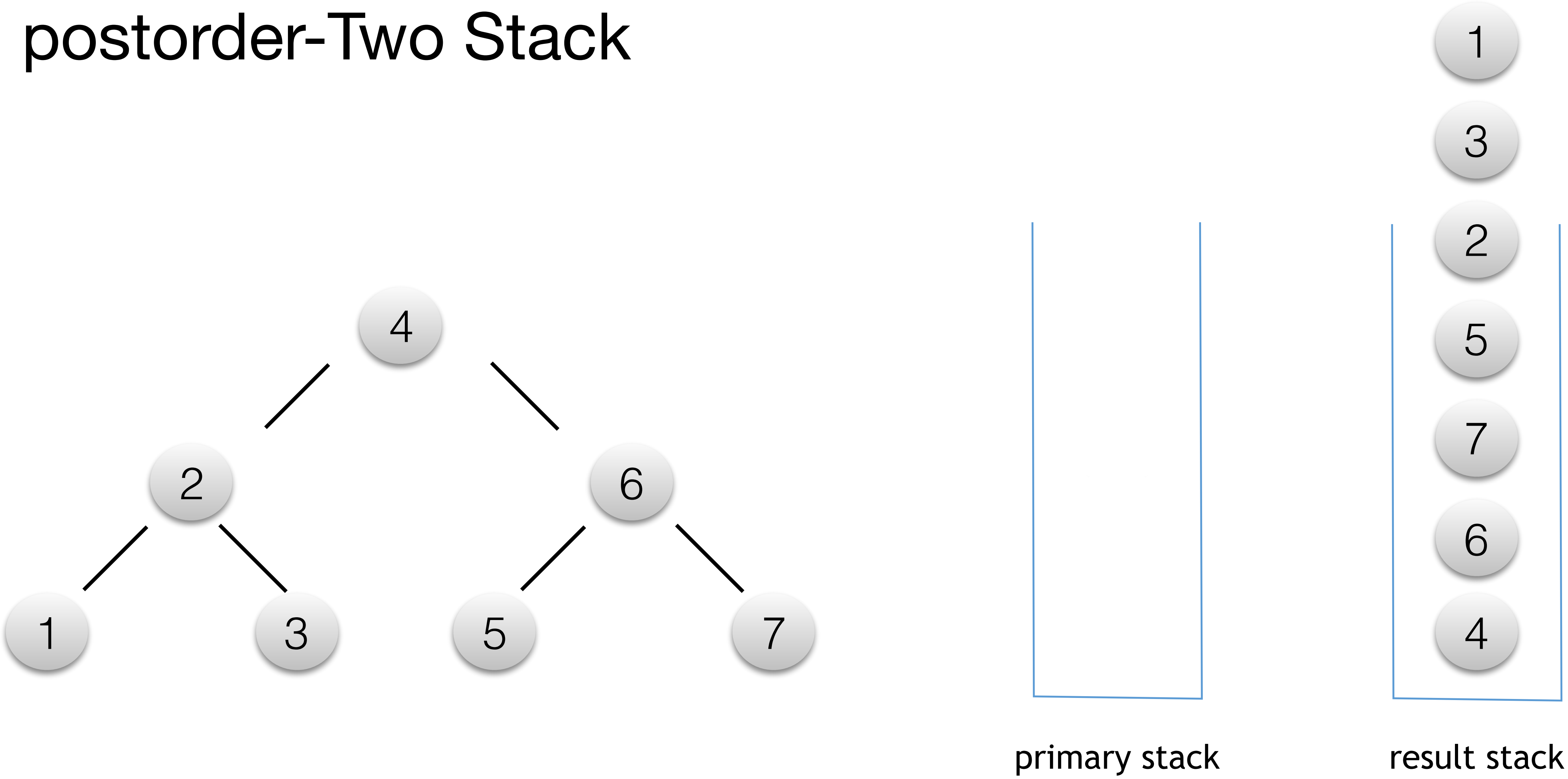


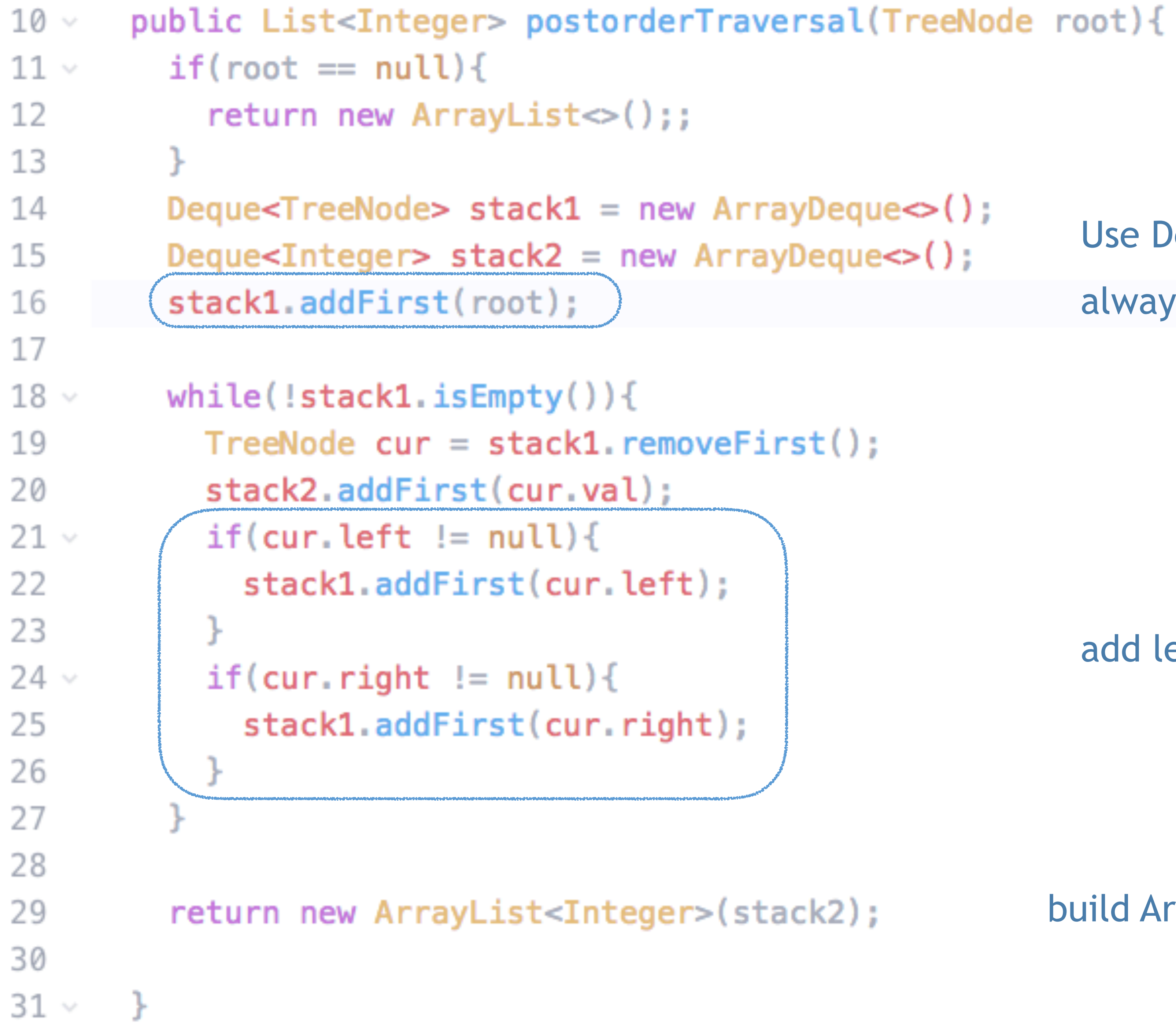
preorder : root left right   4 2 1 3 6 5 7

reverse-postorder: root right left

postorder : left right root 1 3 2 5 7 6 4

postorder-Two Stack





```
public List<Integer> postorderTraversal(TreeNode root){
  if(root == null){
    return new ArrayList<>();;
  }
  Deque<TreeNode> stack1 = new ArrayDeque<>();
  Deque<Integer> stack2 = new ArrayDeque<>();
  stack1.addFirst(root);

  while(!stack1.isEmpty()){
    TreeNode cur = stack1.removeFirst();
    stack2.addFirst(cur.val);
    if(cur.left != null){
      stack1.addFirst(cur.left);
    }
    if(cur.right != null){
      stack1.addFirst(cur.right);
    }
  }

  return new ArrayList<Integer>(stack2);

}
```

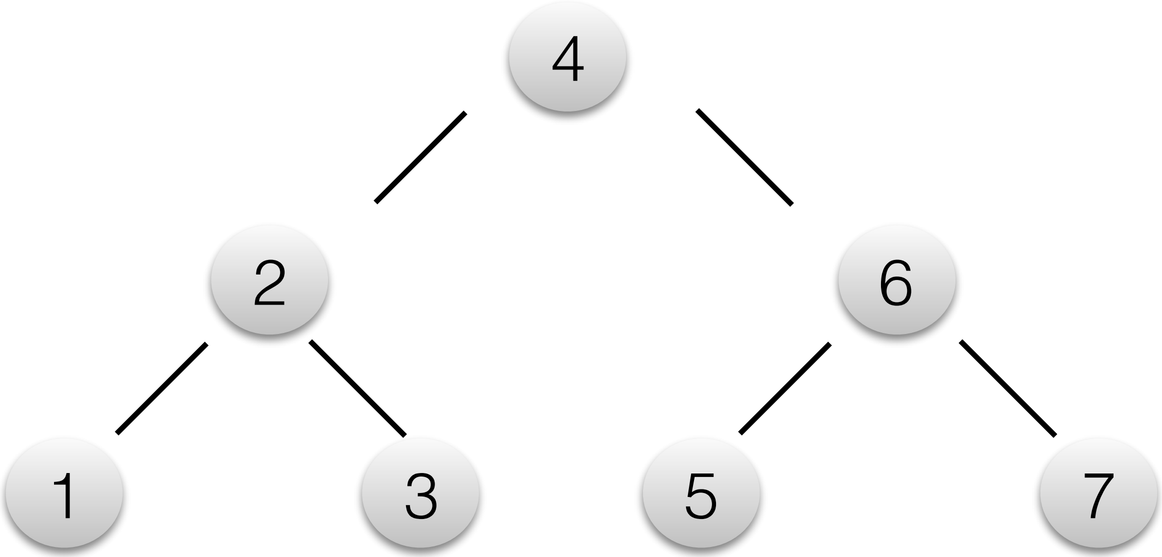
Use Deque instead of Stack

always add root node first

add left then right

build ArrayList using Deque

285. Inorder Successor in BST



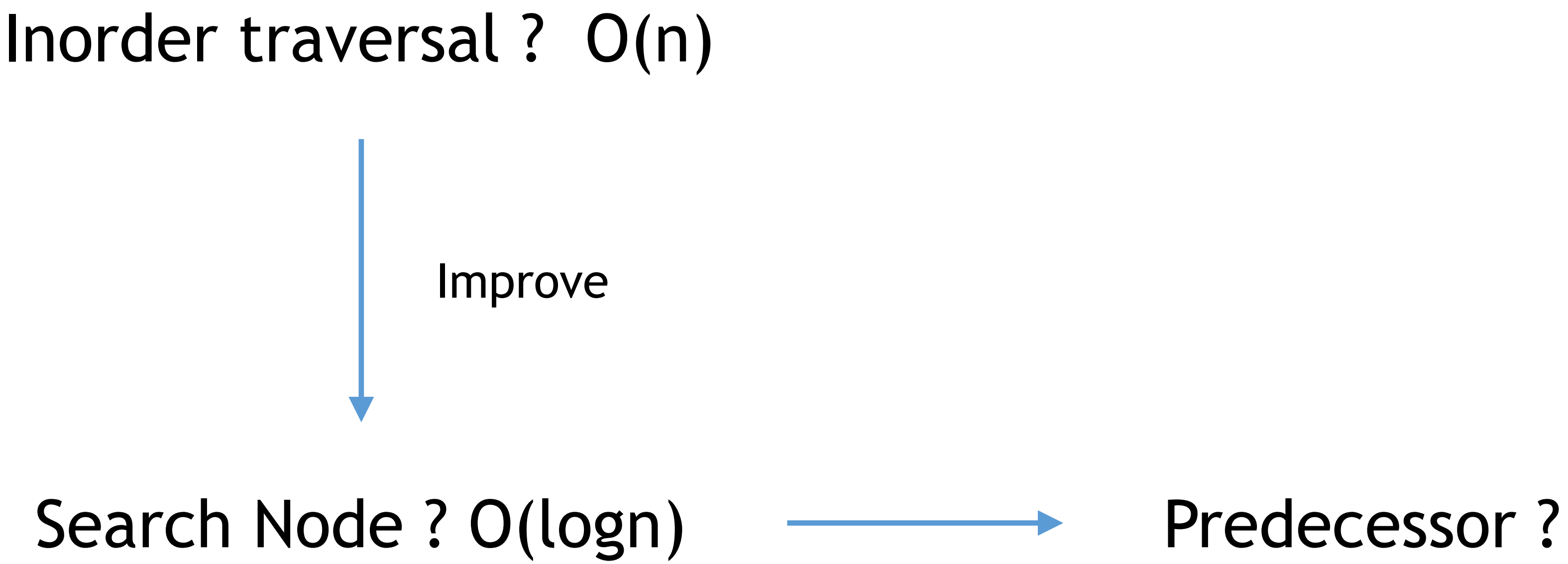
inorder : left root right 1 2 3 4 5 6 7

10 min

```
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {

}
```

Follow Up: Inorder Predecessor

Recursion & Iteration

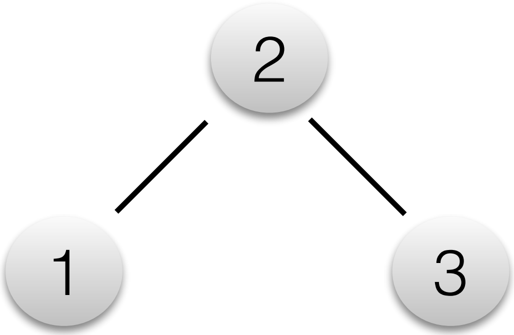


Inorder traversal ? O(n)

Improve

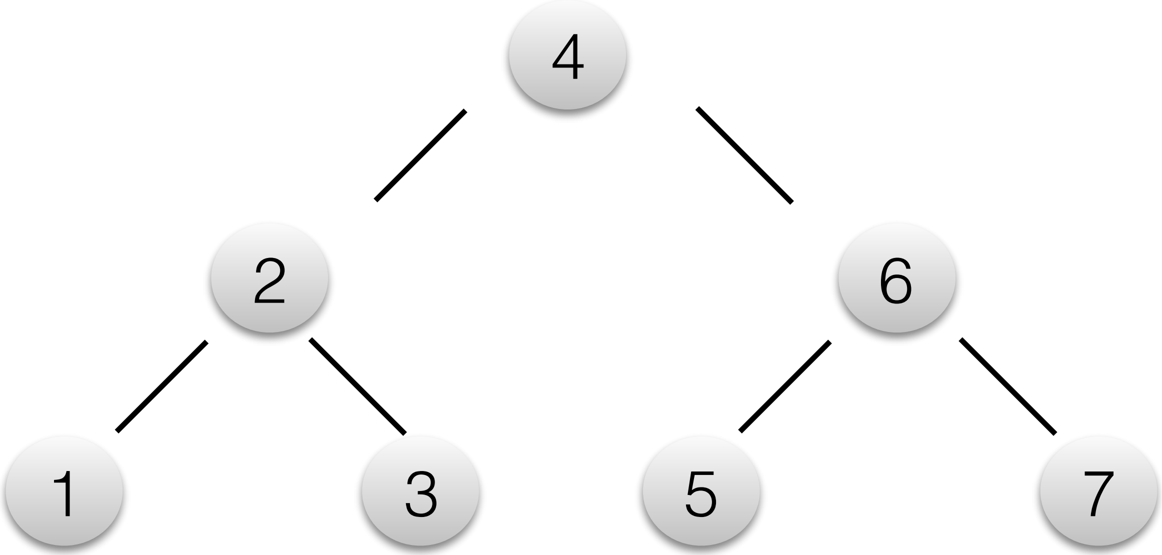
Search Node ? O(logn)

Predecessor ?



left : in left subtree or root
root: in right subtree
right: in right subtree

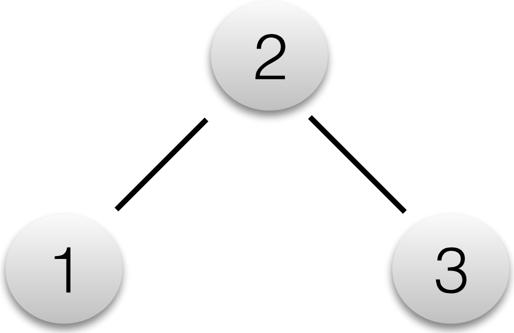
Recursion & Iteration



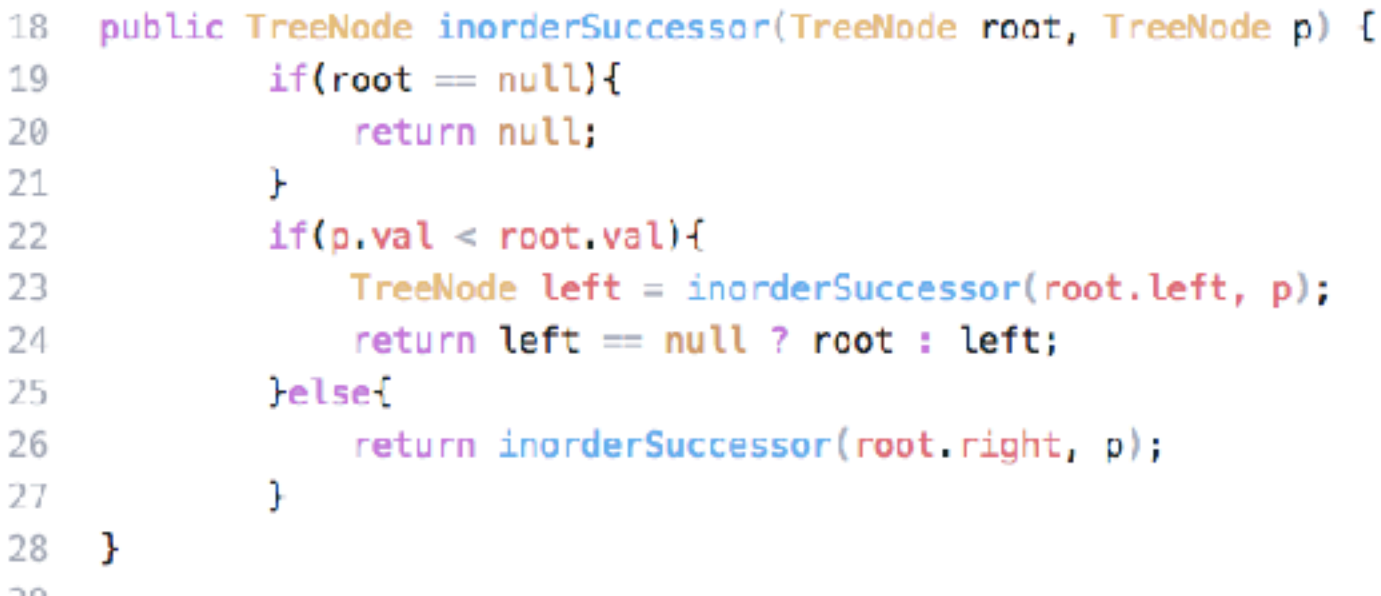
curRoot: 4 2 1

caSucc: 4 2

```java
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        TreeNode suc = null;
        if(root == null || p == null){
            return suc;
        }
        while(root != null){
            if(p.val < root.val){
                suc = root;
                root = root.left;
            }else{
                root = root.right;
            }
        }
        return suc;
}
```



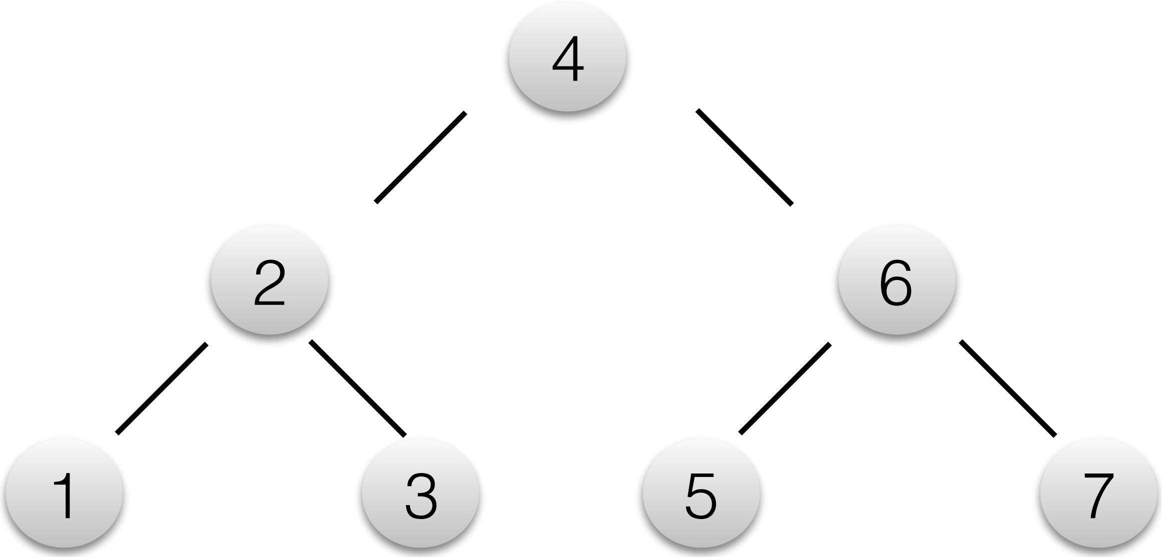
left : in left subtree or root
root: in right subtree
right: in right subtree



```java
public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        if(root == null){
            return null;
        }
        if(p.val < root.val){
            TreeNode left = inorderSuccessor(root.left, p);
            return left == null ? root : left;
        }else{
            return inorderSuccessor(root.right, p);
        }
}
```

Divide and Conquer

230. Kth Smallest Element in a BST



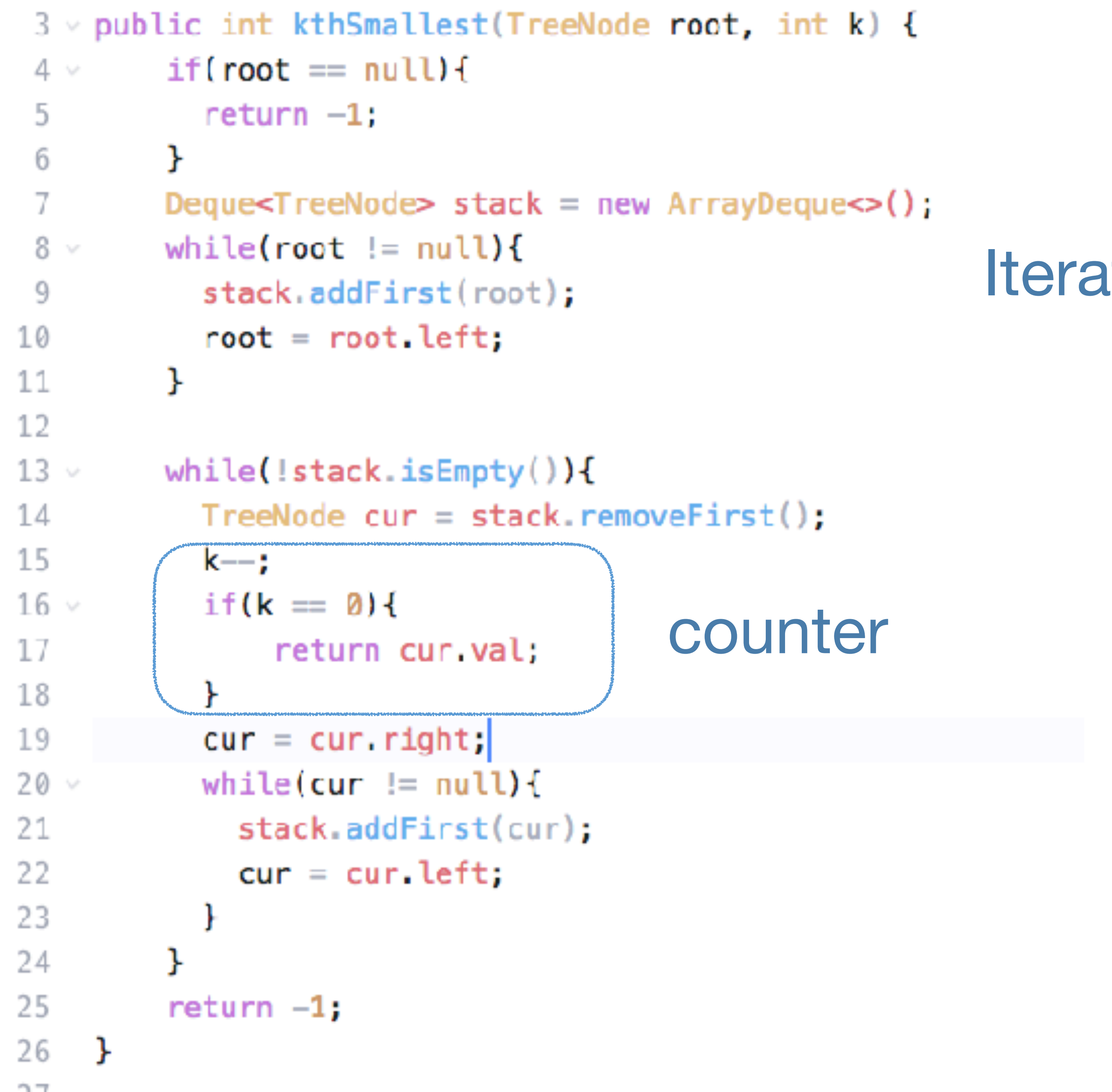
inorder : left root right   1 2 3 4 5 6 7

10 min

```
public int kthSmallest(TreeNode root, int k) {

}
```

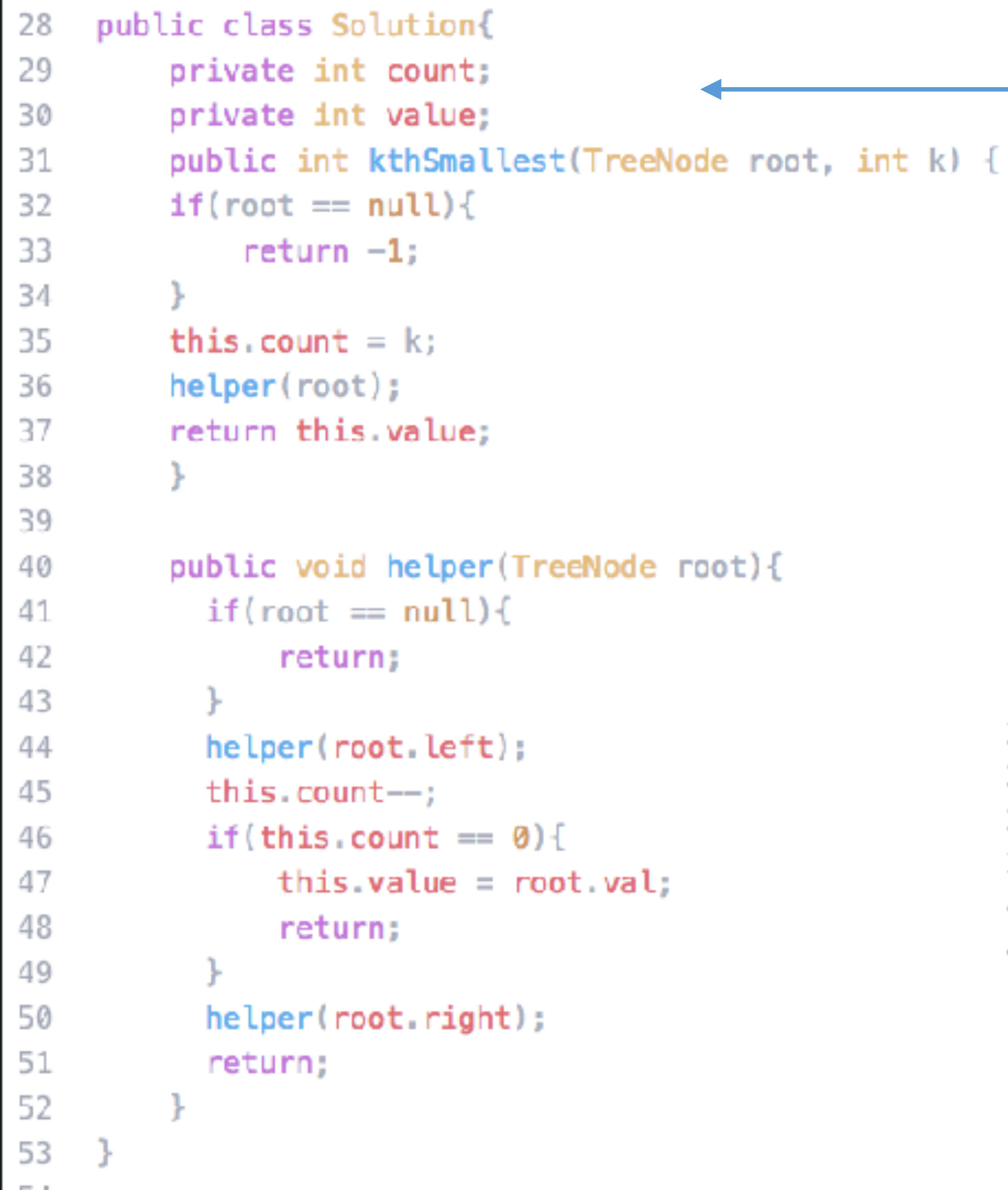
Follow Up: kth Largest

Iteration standard template



```java
public int kthSmallest(TreeNode root, int k) {
    if(root == null){
      return -1;
    }
    Deque<TreeNode> stack = new ArrayDeque<>();
    while(root != null){
      stack.addFirst(root);
      root = root.left;
    }

    while(!stack.isEmpty()){
      TreeNode cur = stack.removeFirst();
      k--;
      if(k == 0){
          return cur.val;
      }
      cur = cur.right;
      while(cur != null){
        stack.addFirst(cur);
        cur = cur.left;
      }
    }
    return -1;
}
```

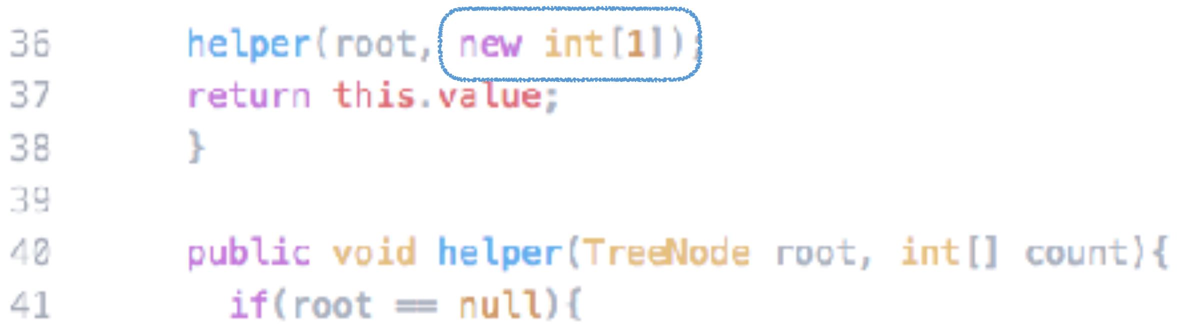
counter



```
public class Solution{
    private int count;
    private int value;
    public int kthSmallest(TreeNode root, int k) {
    if(root == null){
        return -1;
    }
    this.count = k;
    helper(root);
    return this.value;
    }

    public void helper(TreeNode root){
      if(root == null){
          return;
      }
      helper(root.left);
      this.count--;
      if(this.count == 0){
          this.value = root.val;
          return;
      }
      helper(root.right);
      return;
    }
}
```

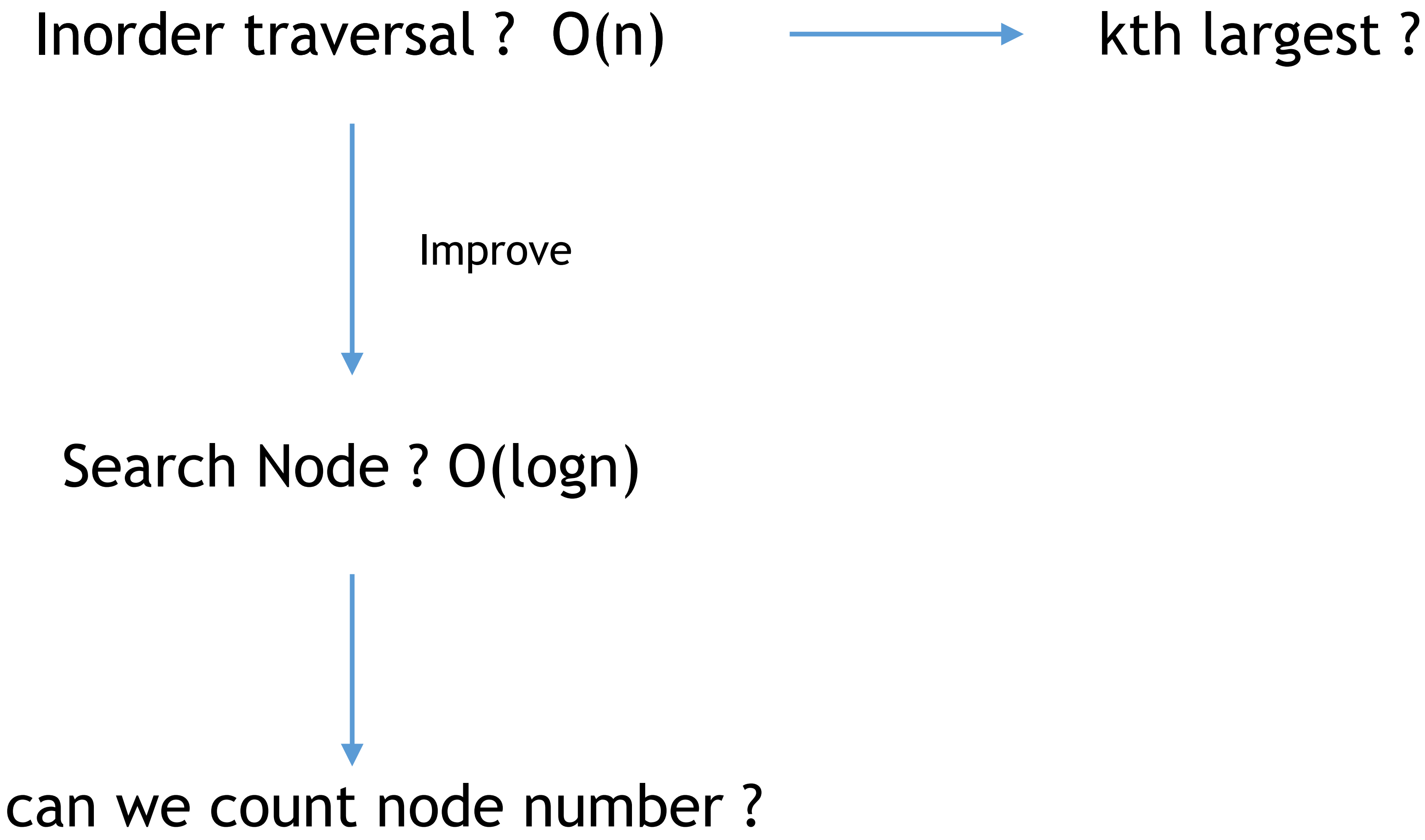
Recursion with class variable

Recursion with int[] input



```
    helper(root, new int[1]);
    return this.value;
    }

    public void helper(TreeNode root, int[] count){
      if(root == null){
```

Recursion & Iteration



Inorder traversal ? O(n)

kth largest ?

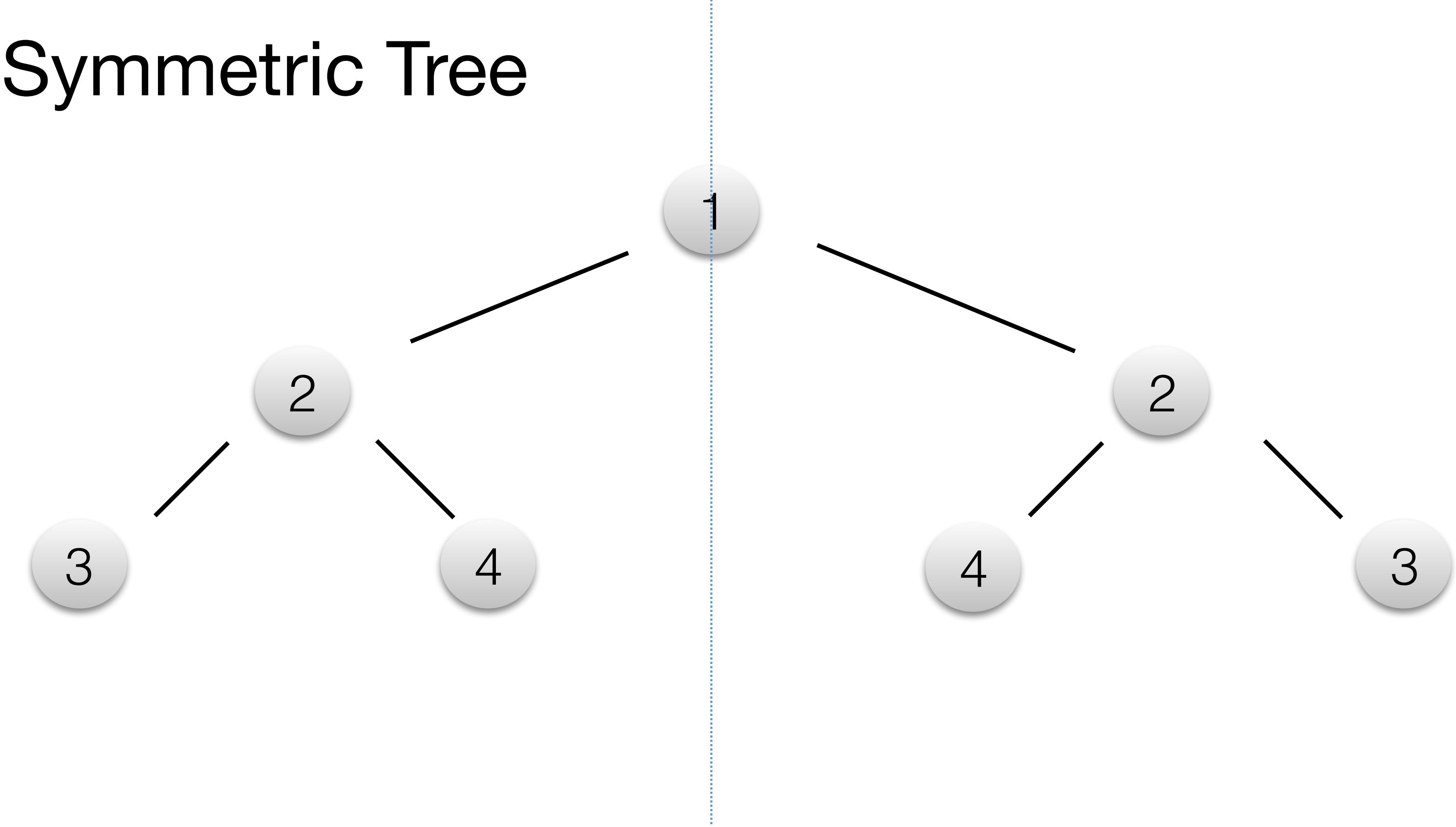
Improve

Search Node ? O(logn)

can we count node number ?

Add ‘count’ data field in tree node.

101. Symmetric Tree

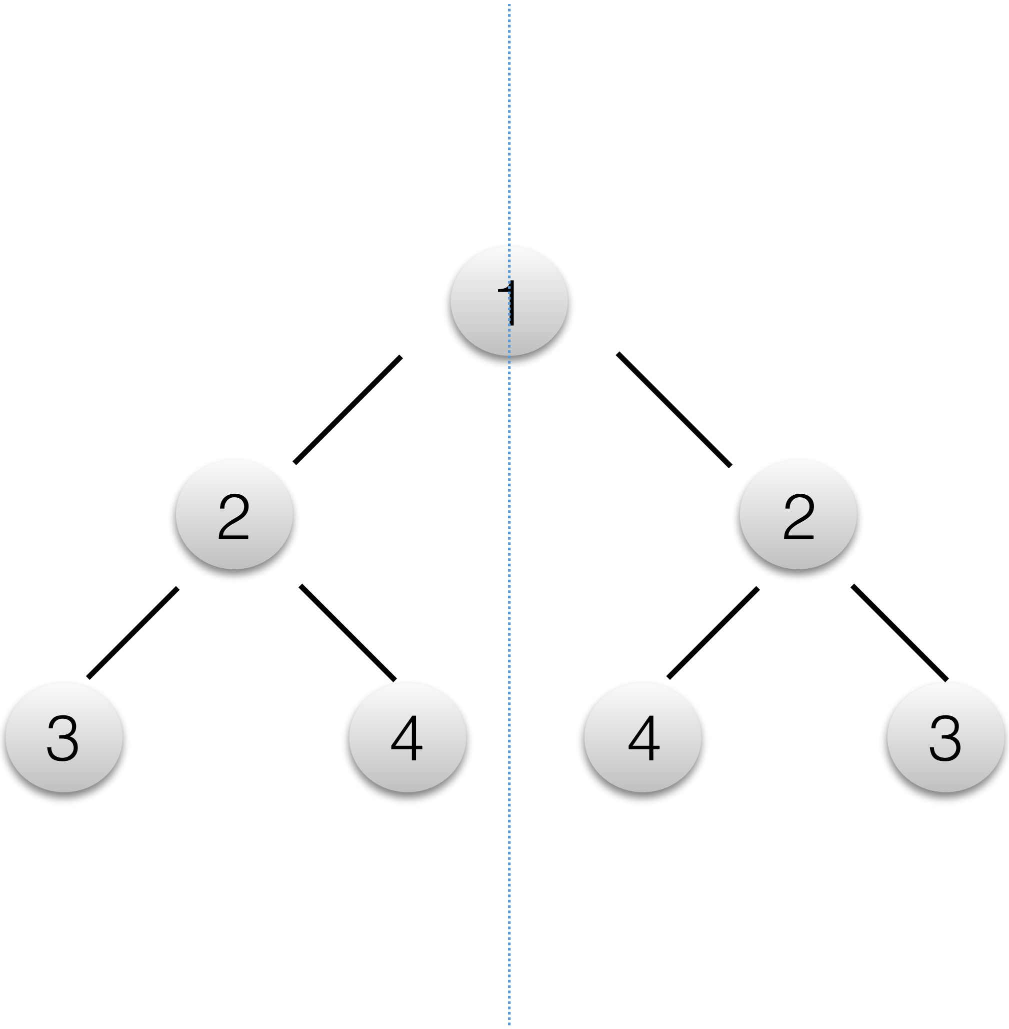


轴对称

10 min

```
public boolean isSymmetric(TreeNode root) {

}
```

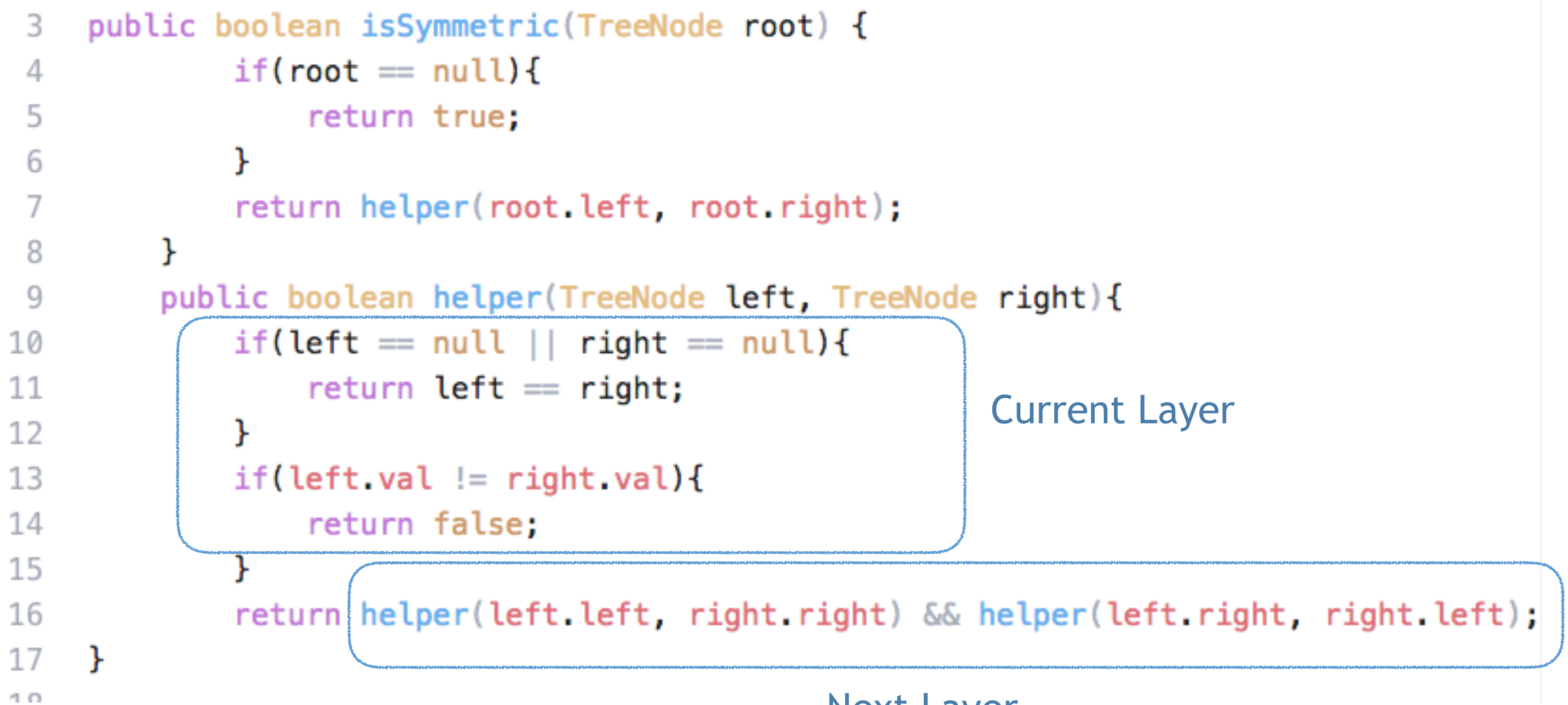
Recursion & Iteration



compare two nodes per time

currently compare left and right

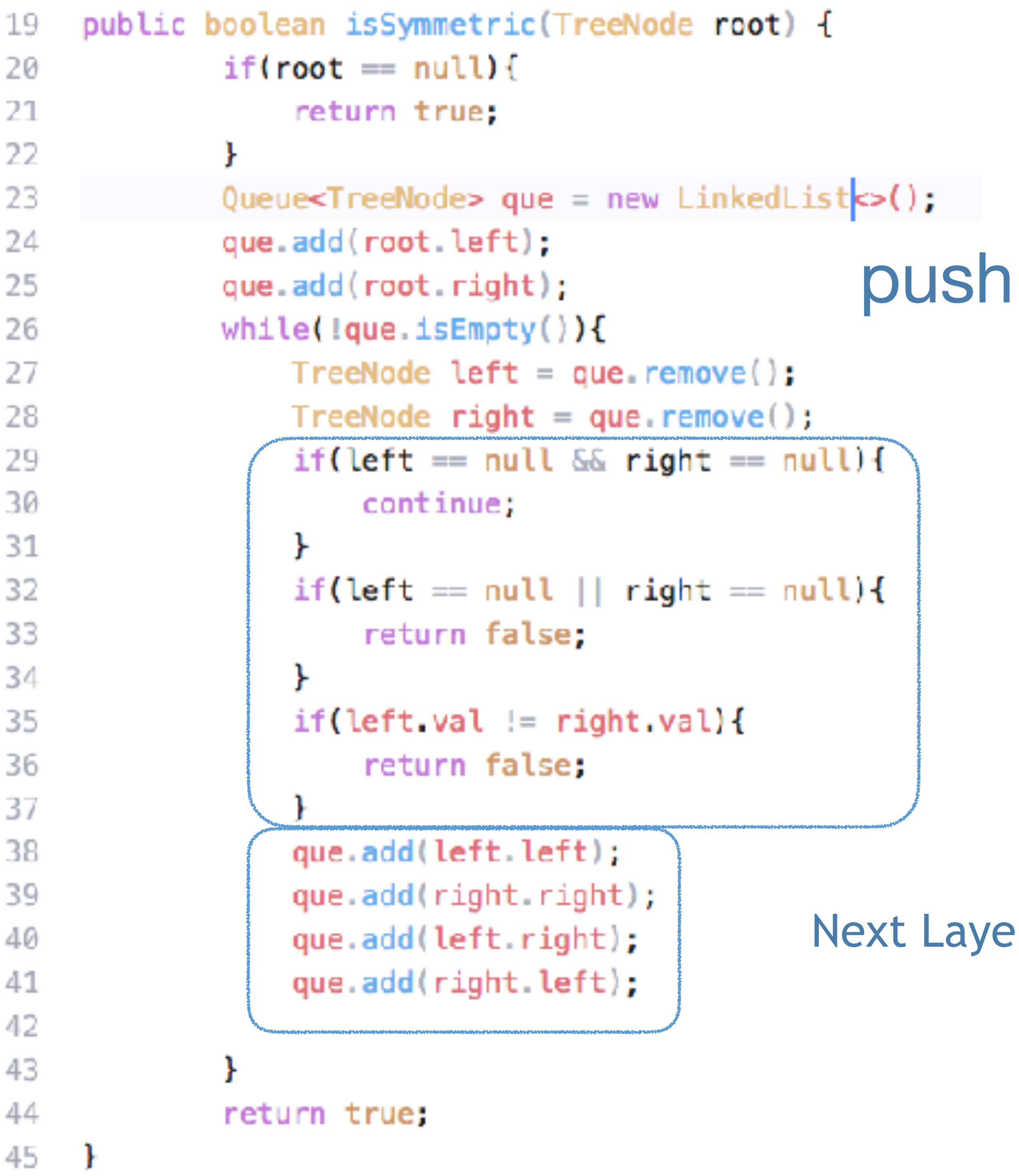
next compare left.left with right.right
AND left.right with right.left



```
public boolean isSymmetric(TreeNode root) {
        if(root == null){
            return true;
        }
        return helper(root.left, root.right);
    }
    public boolean helper(TreeNode left, TreeNode right){
        if(left == null || right == null){
            return left == right;
        }
        if(left.val != right.val){
            return false;
        }
        return helper(left.left, right.right) && helper(left.right, right.left);
}
```

Current Layer

Next Layer



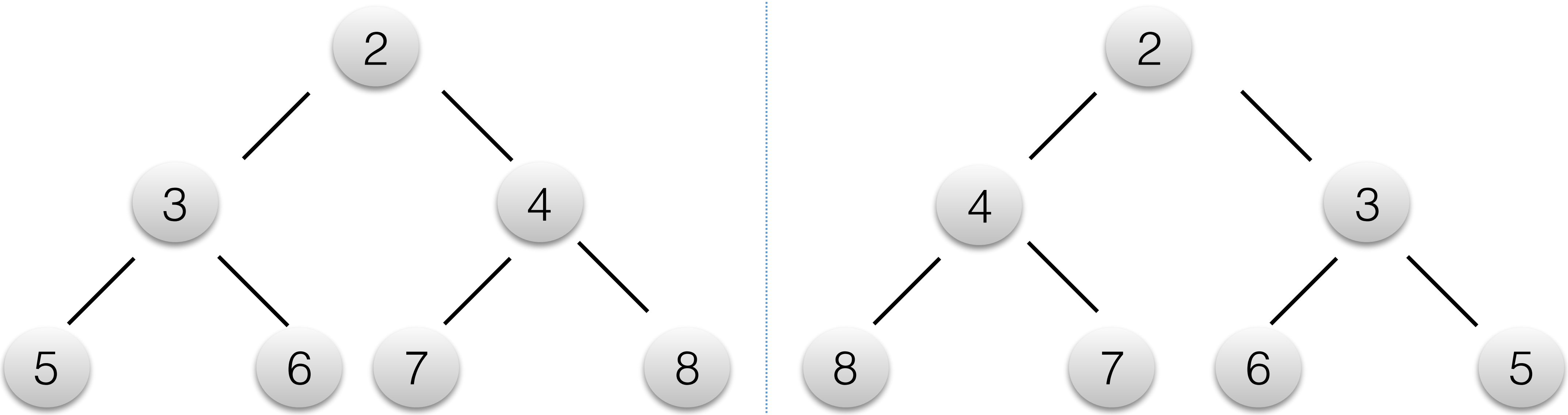
```
public boolean isSymmetric(TreeNode root) {
        if(root == null){
            return true;
        }
        Queue<TreeNode> que = new LinkedList<>();
        que.add(root.left);
        que.add(root.right);
        while(!que.isEmpty()){
            TreeNode left = que.remove();
            TreeNode right = que.remove();
            if(left == null && right == null){
                continue;
            }
            if(left == null || right == null){
                return false;
            }
            if(left.val != right.val){
                return false;
            }
            que.add(left.left);
            que.add(right.right);
            que.add(left.right);
            que.add(right.left);

        }
        return true;
}
```

push two initialized nodes

Current Layer

Next Layer

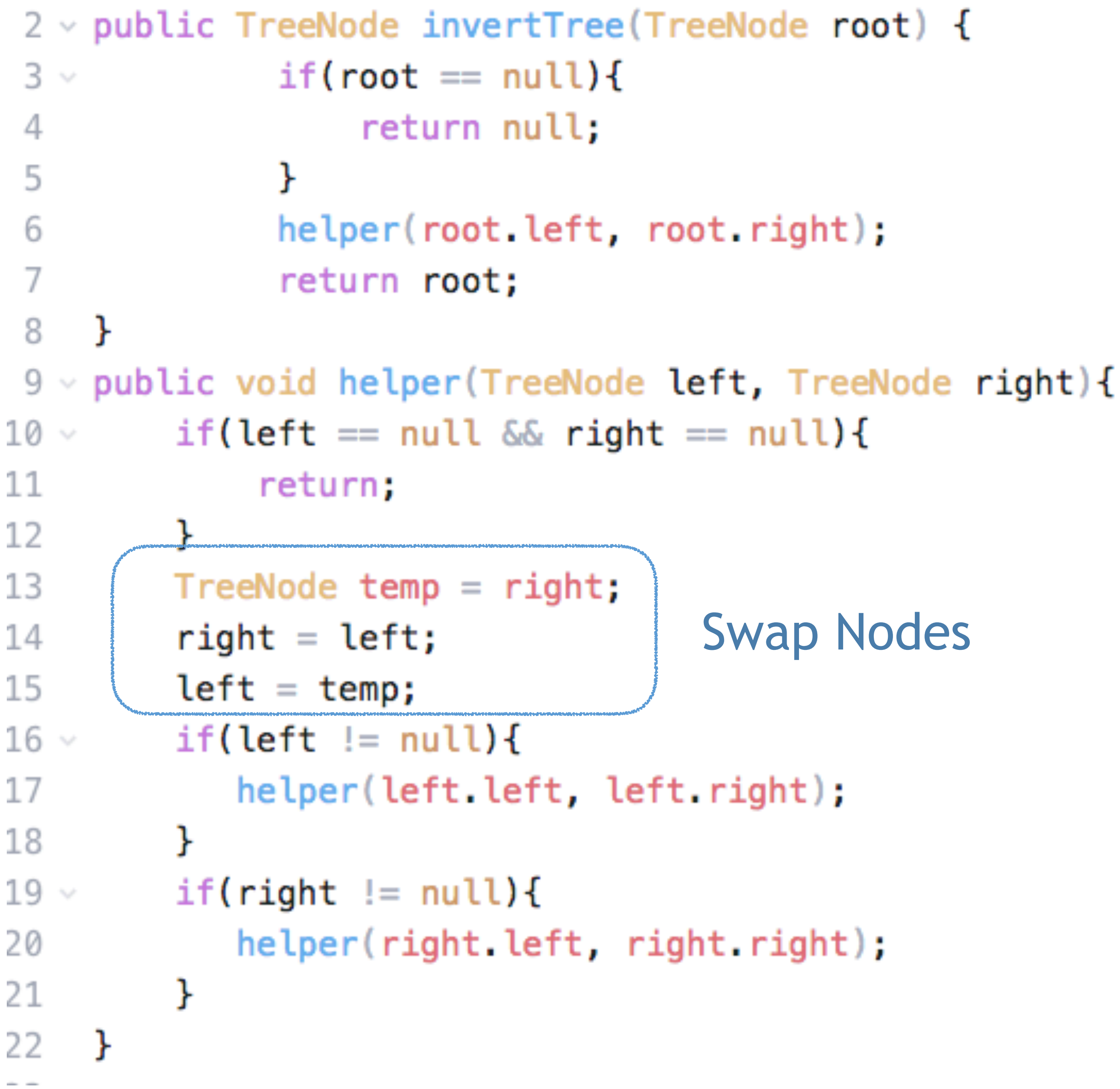
226. Invert Binary Tree



轴对称翻转（与上一题类似）

10 min

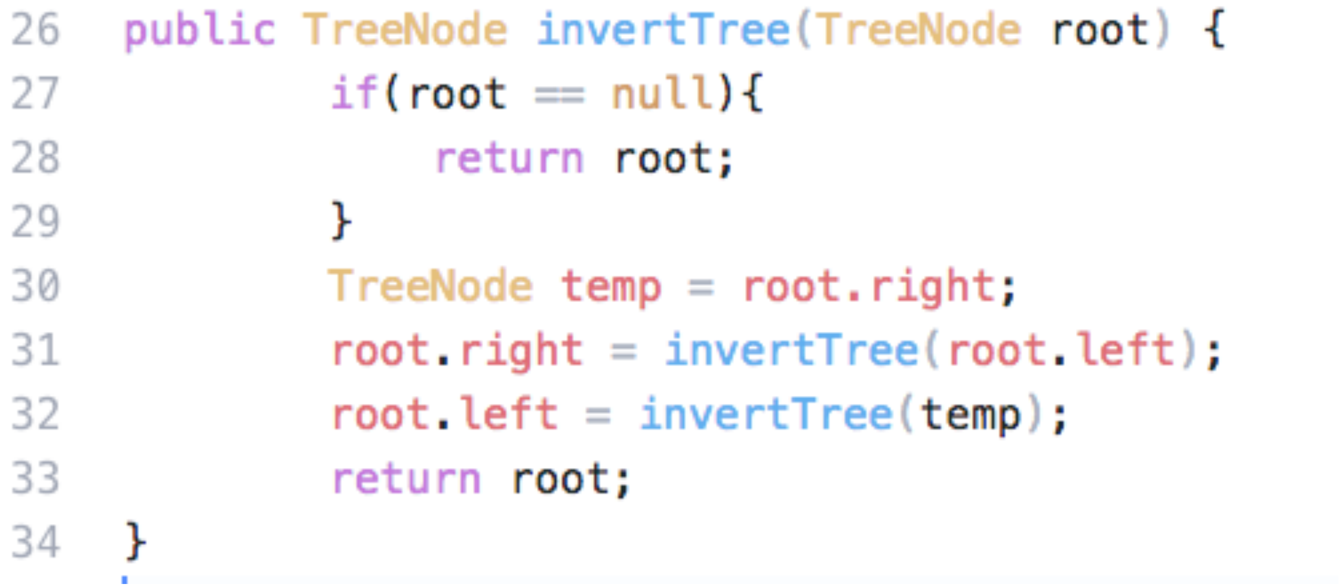
```
public TreeNode invertTree(TreeNode root) {

}
```



```
public TreeNode invertTree(TreeNode root) {
         if(root == null){
             return null;
         }
         helper(root.left, root.right);
         return root;
}
public void helper(TreeNode left, TreeNode right){
    if(left == null && right == null){
        return;
    }
    TreeNode temp = right;
    right = left;
    left = temp;
    if(left != null){
       helper(left.left, left.right);
    }
    if(right != null){
       helper(right.left, right.right);
    }
}
```

Swap Nodes

WRONG !!!

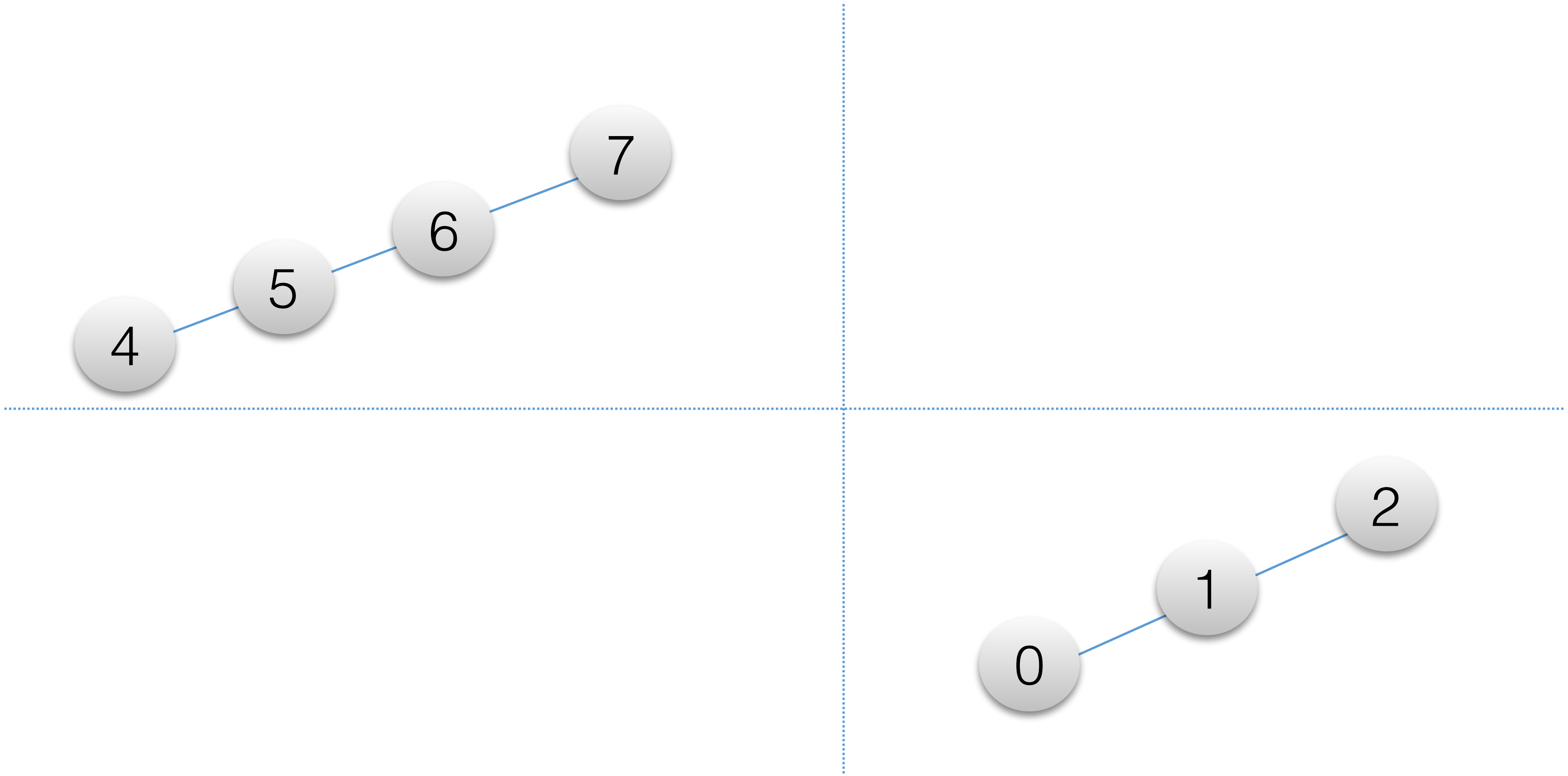


```java
public TreeNode invertTree(TreeNode root) {
        if(root == null){
            return root;
        }
        TreeNode temp = root.right;
        root.right = invertTree(root.left);
        root.left = invertTree(temp);
        return root;
}
```

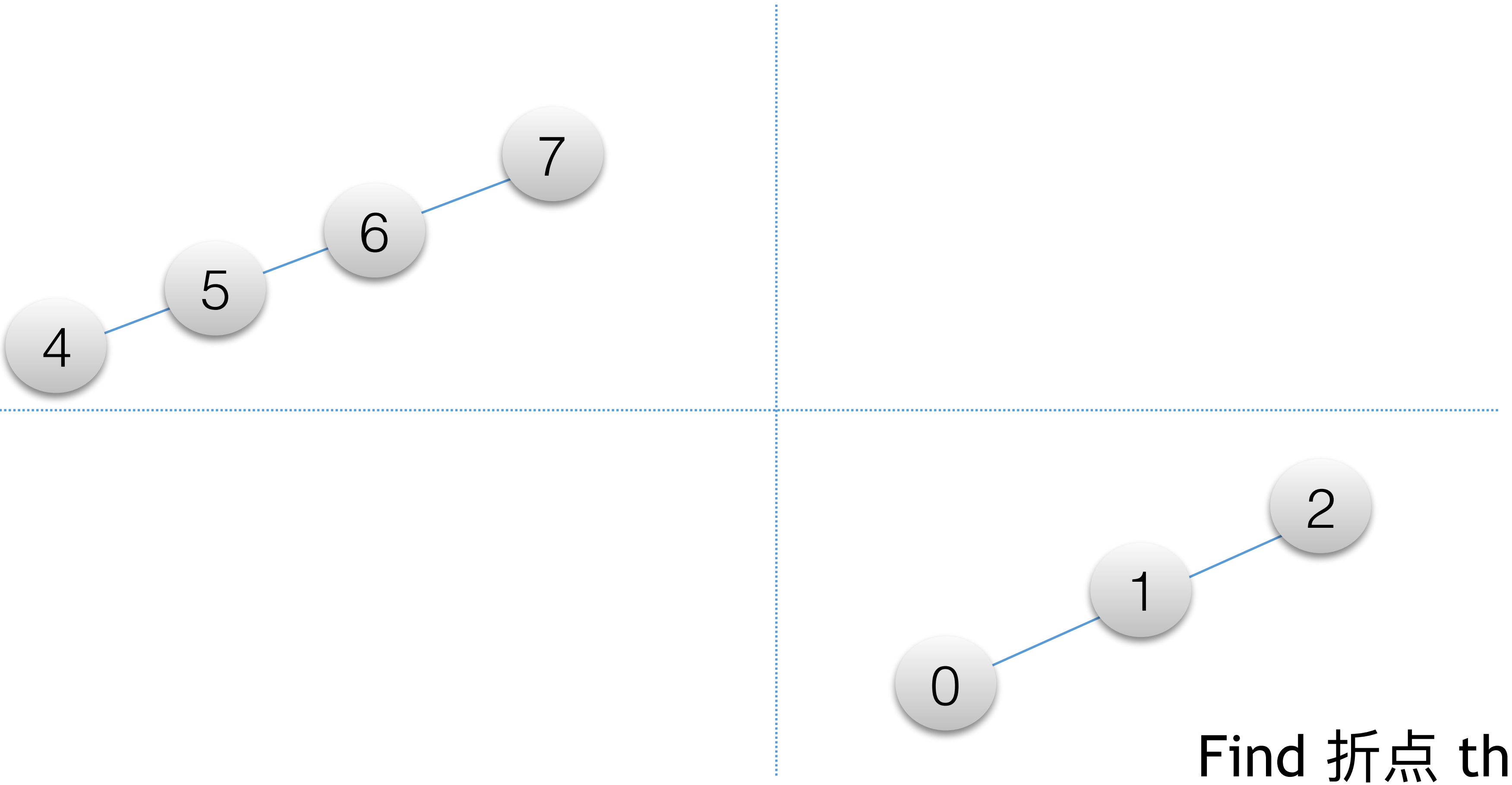
RIGHT !!!

33. Search in Rotated Sorted Array

4 5 6 7 0 1 2



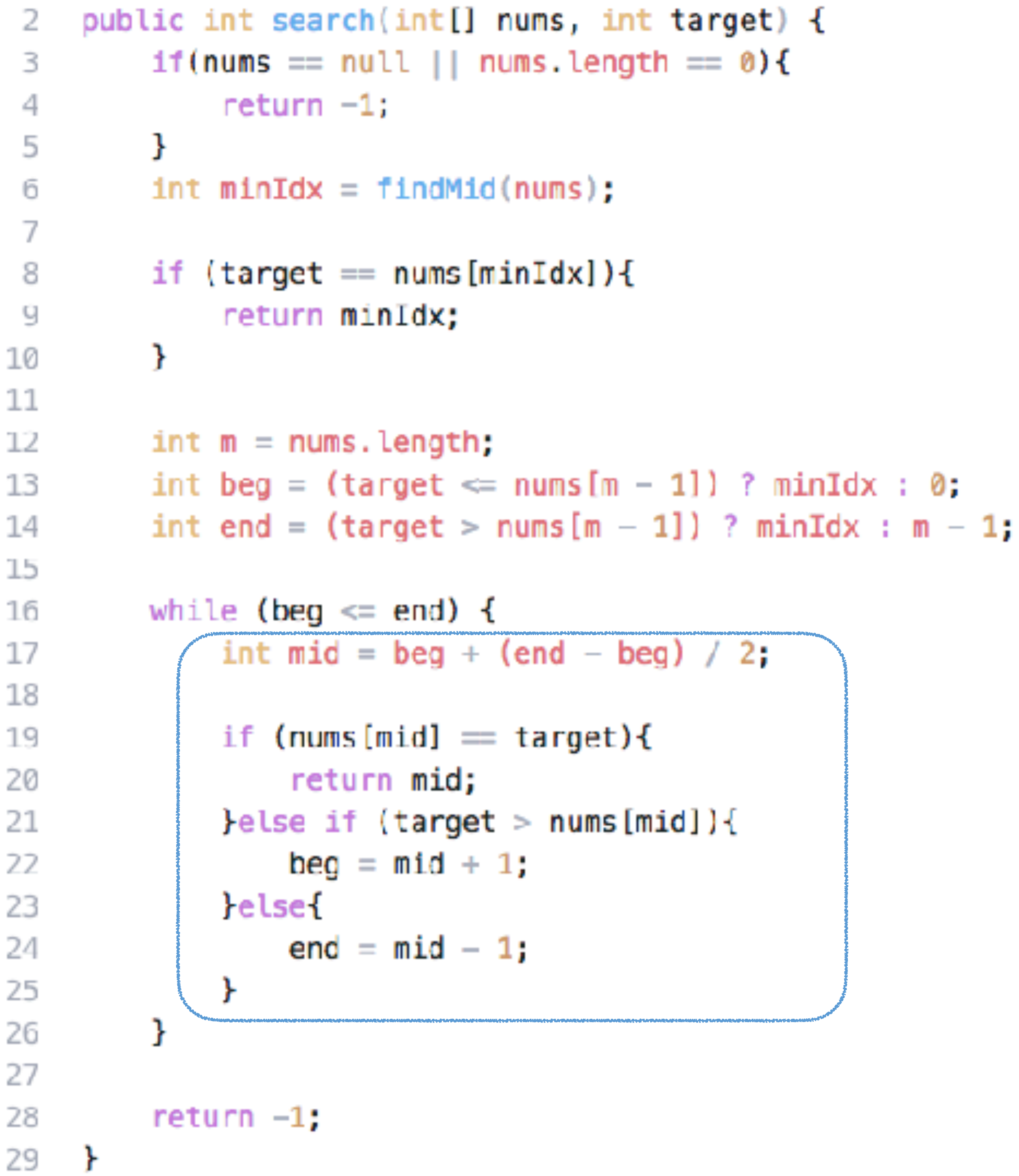
Binary Search



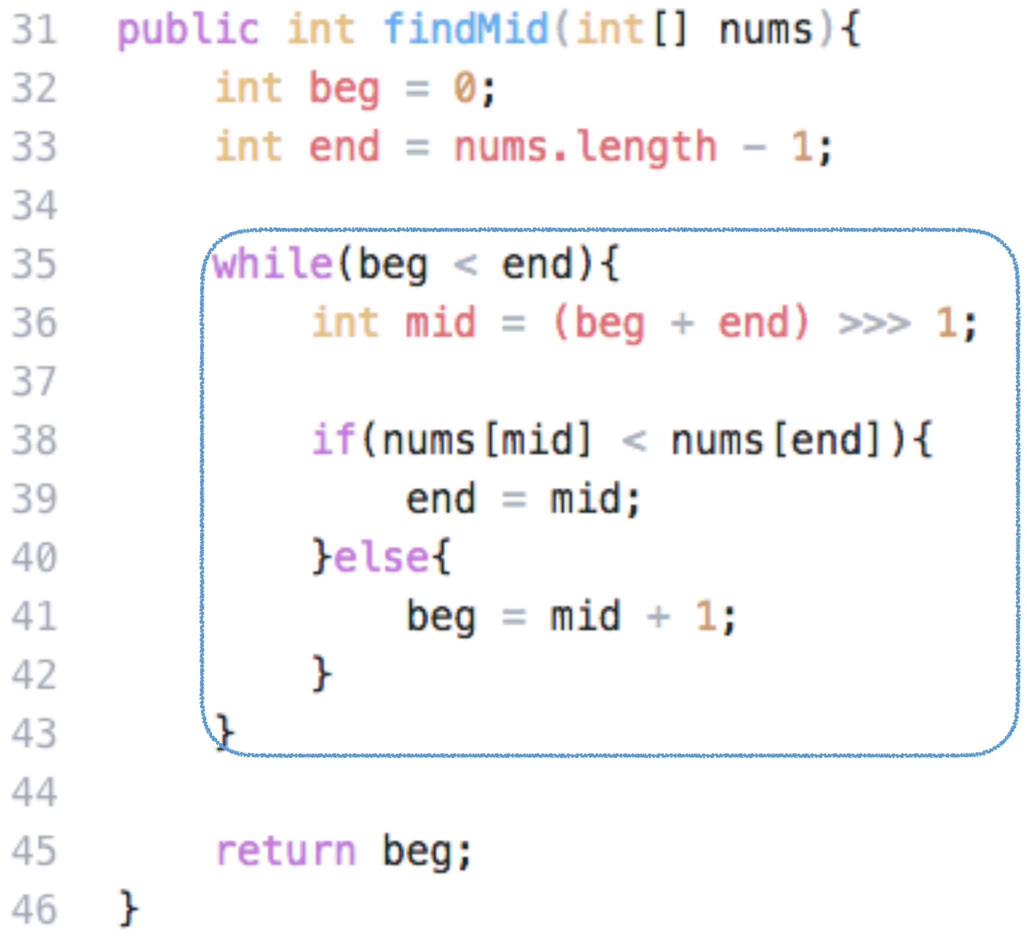
Sorted Array

Binary Search ?

Find 折点 then Binary Search



```
public int search(int[] nums, int target) {
    if(nums == null || nums.length == 0){
        return -1;
    }
    int minIdx = findMid(nums);

    if (target == nums[minIdx]){
        return minIdx;
    }

    int m = nums.length;
    int beg = (target <= nums[m - 1]) ? minIdx : 0;
    int end = (target > nums[m - 1]) ? minIdx : m - 1;

    while (beg <= end) {
        int mid = beg + (end - beg) / 2;

        if (nums[mid] == target){
            return mid;
        }else if (target > nums[mid]){
            beg = mid + 1;
        }else{
            end = mid - 1;
        }
    }

    return -1;
}
```



```
public int findMid(int[] nums){
    int beg = 0;
    int end = nums.length - 1;

    while(beg < end){
        int mid = (beg + end) >>> 1;

        if(nums[mid] < nums[end]){
            end = mid;
        }else{
            beg = mid + 1;
        }
    }

    return beg;
}
```

Binary Search

Homework

Binary Search	*Binary Tree*	*Binary Search Tree*
Search in Rotated Sorted Array 2	*balanced binary tree*	*delete node in BST*
	path sum 1 2	
	max depth of binary tree	
	binary tree level order traversal	

Q & A

Thank you