大厂Java后端面试常考算法题

Facebook面试官 带你刷遍大厂高频题

讲师：令狐冲

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像 一经发现，将被追究法律责任和赔偿经济损失。

大纲

一、拼多多算法类面试真题串讲

LintCode 1252. 根据身高重排队列

LintCode 66. 二叉树的前序遍历

二、携程算法类面试真题串讲

LintCode 1182. 翻转字符串 II

LintCode 88. 最近公共祖先

三、快手算法类面试真题串讲

LintCode 209. 第一个只出现一次的字符

LintCode 98. 链表排序

四、小米算法类面试真题串讲

LintCode 93. 平衡二叉树

LintCode 374. 螺旋矩阵

拼多多算法类面试真题串讲

LintCode 1252. 根据身高重排队列

https://www.lintcode.com/problem/queue-reconstruction-by-height/description

LintCode 1252. 根据身高重排队列

假设你有一个顺序被随机打乱的列表，代表了站成一列的人群。每个人被表示成一个二元组(h, k)，其中h表示他的身高，k表示站在他之前的身高高于或等于h的人数。你需要将这个队列重新排列以恢复其原有的顺序。

样例1

输入：[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

输出：[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]

样例2

输入：[[2,0], [1,1]]

输出：[[2,0], [1,1]]

LintCode 1252. 根据身高重排队列

先对数组按照身高从高到低排序，如果身高一样，则按照第二维大小从小到大排序。遍历数组，根据第二维大小将(h,k)插到顺序队列中。

比如 [[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]] ，排好序结果为：
[[7,0], [7,1], [6,1], [5,0], [5,2], [4,4]]

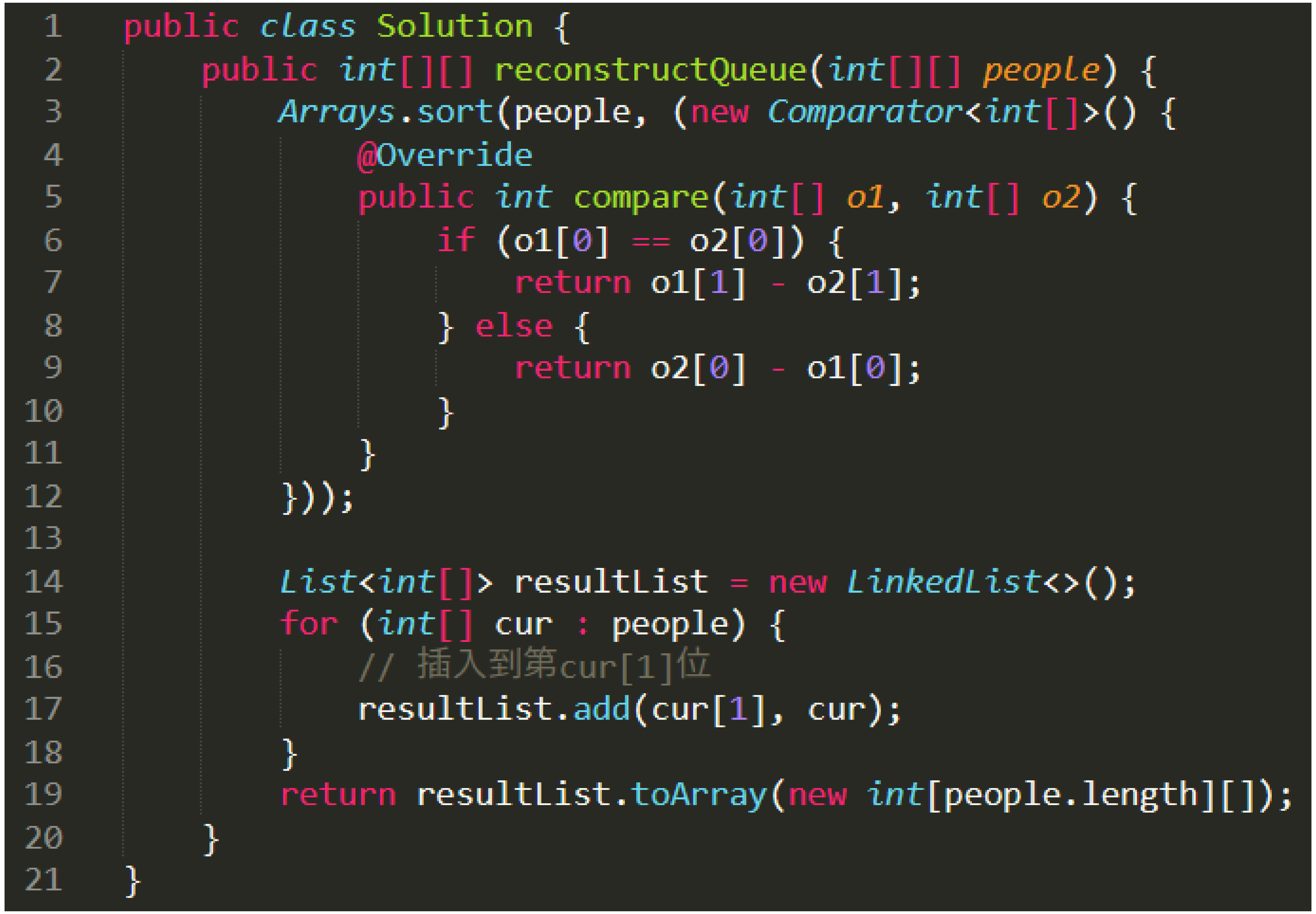
从头至尾遍历整个数组，对于一个二元组(h,k)，插入到当前维护答案序列的下标为k的位置即可。

例如当前答案序列为：[[7,0], [7,1]]。

若要加入：[6,1]，答案序列应变成：[[7,0], [6,1], [7,1]]。

所以上述样例答案为：[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]。

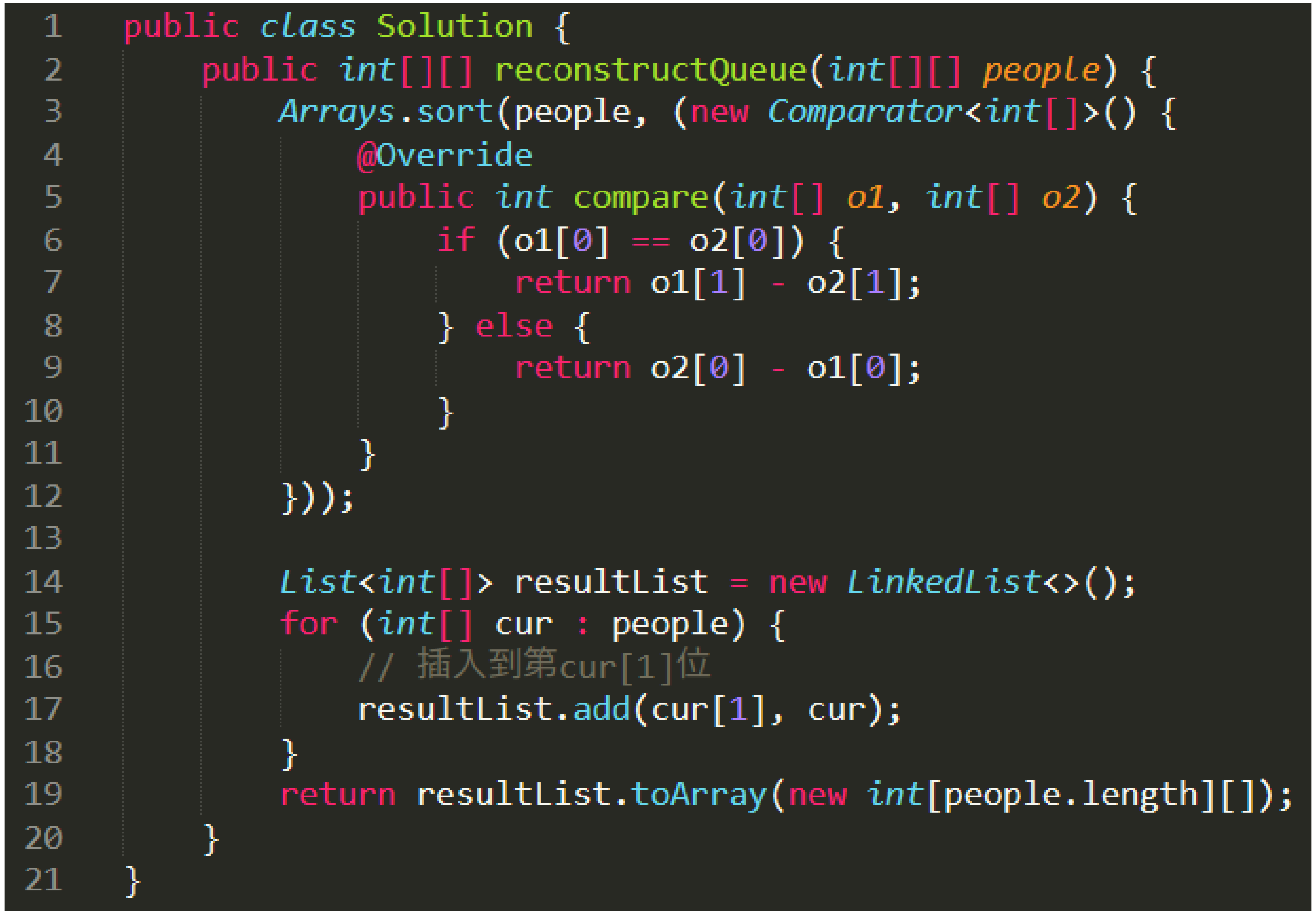
LintCode 1252. 根据身高重排队列



```java
public class Solution {
    public int[][] reconstructQueue(int[][] people) {
        Arrays.sort(people, (new Comparator<int[]>() {
            @Override
            public int compare(int[] o1, int[] o2) {
                if (o1[0] == o2[0]) {
                    return o1[1] - o2[1];
                } else {
                    return o2[0] - o1[0];
                }
            }
        }));

        List<int[]> resultList = new LinkedList<>();
        for (int[] cur : people) {
            // 插入到第cur[1]位
            resultList.add(cur[1], cur);
        }
        return resultList.toArray(new int[people.length][]);
    }
}
```

复杂度多少?

LintCode 1252. 根据身高重排队列



```java
public class Solution {
    public int[][] reconstructQueue(int[][] people) {
        Arrays.sort(people, (new Comparator<int[]>() {
            @Override
            public int compare(int[] o1, int[] o2) {
                if (o1[0] == o2[0]) {
                    return o1[1] - o2[1];
                } else {
                    return o2[0] - o1[0];
                }
            }
        }));

        List<int[]> resultList = new LinkedList<>();
        for (int[] cur : people) {
            // 插入到第cur[1]位
            resultList.add(cur[1], cur);
        }
        return resultList.toArray(new int[people.length][]);
    }
}
```

假设n个人；

时间复杂度：最好O(n*logn)，最坏O(n^2)；

空间复杂度：O(n)。

LintCode 66. 二叉树的前序遍历

https://www.lintcode.com/problem/binary-tree-preorder-traversal/description

LintCode 66. 二叉树的前序遍历

给出一棵二叉树，返回其节点值的前序遍历。

样例 1:

输入：{1,2,3}

输出：[1,2,3]

解释:

```
   1
  / \
 2   3
```

它将被序列化为{1,2,3}

样例 2:

输入：{1,#,2,3}

输出：[1,2,3]

解释:

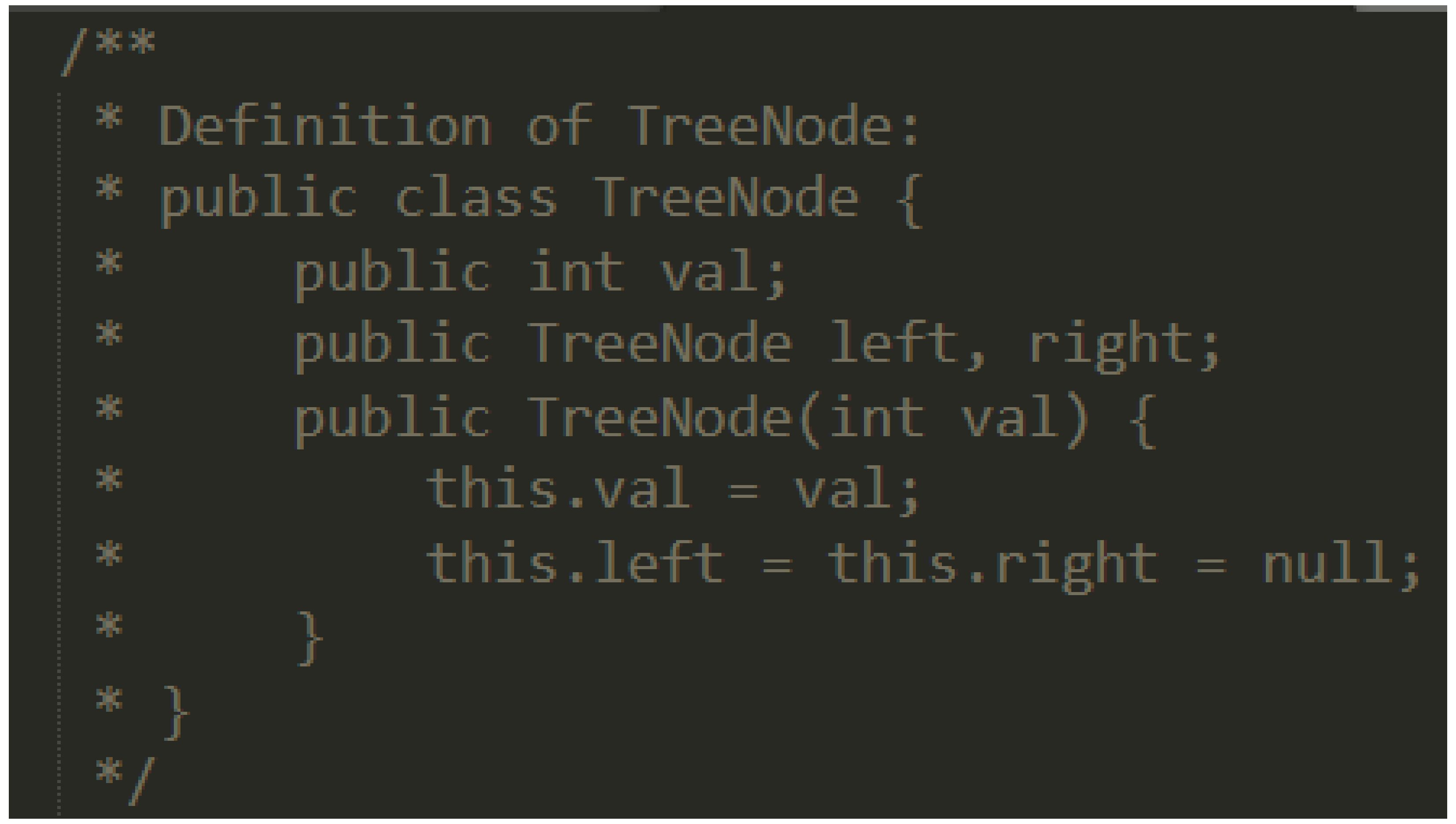
```
1
 \
  2
 /
3
```

它将被序列化为{1,#,2,3}

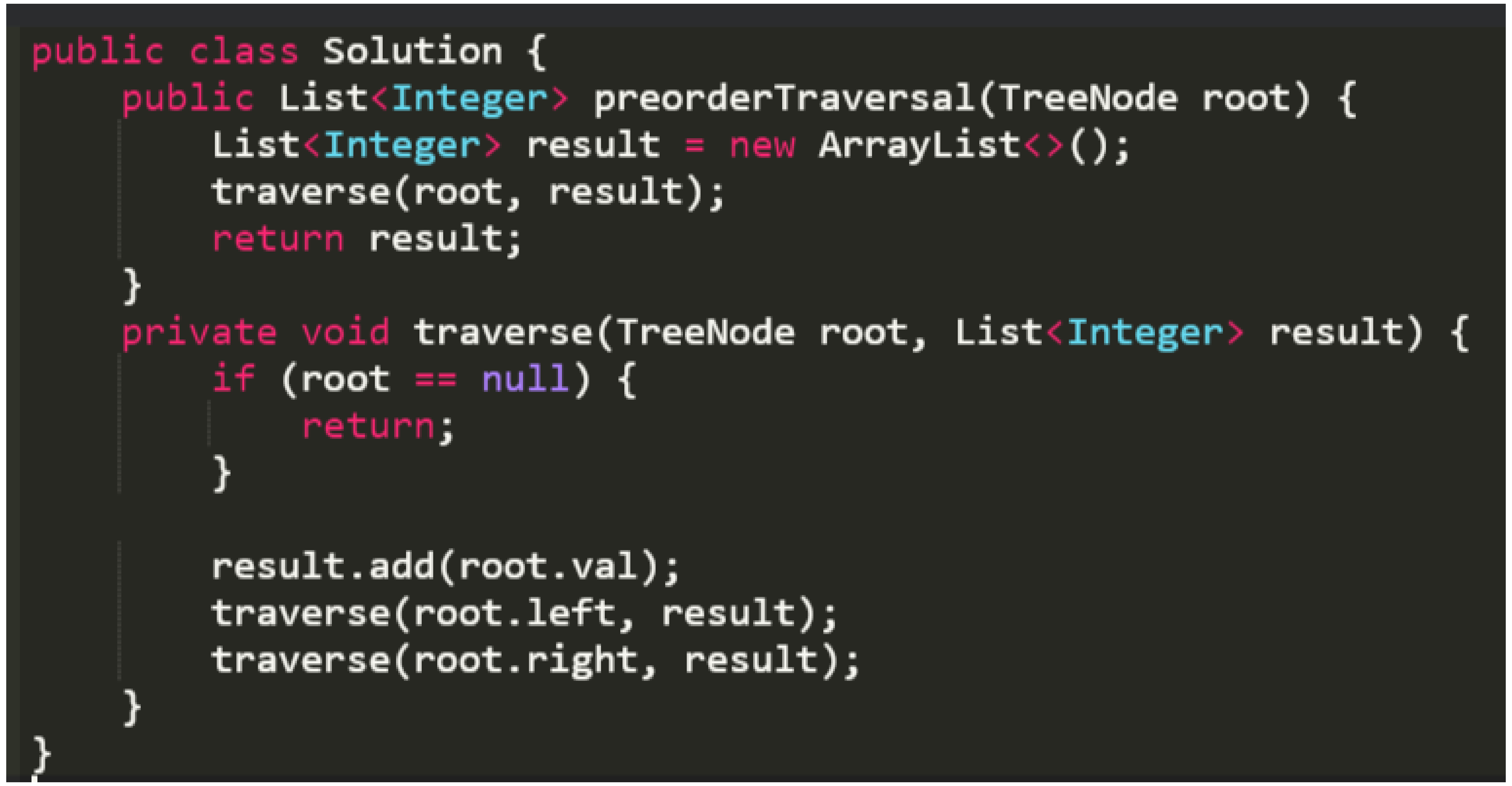
LintCode 66. 二叉树的前序遍历

递归 or 非递归

LintCode 66. 二叉树的前序遍历



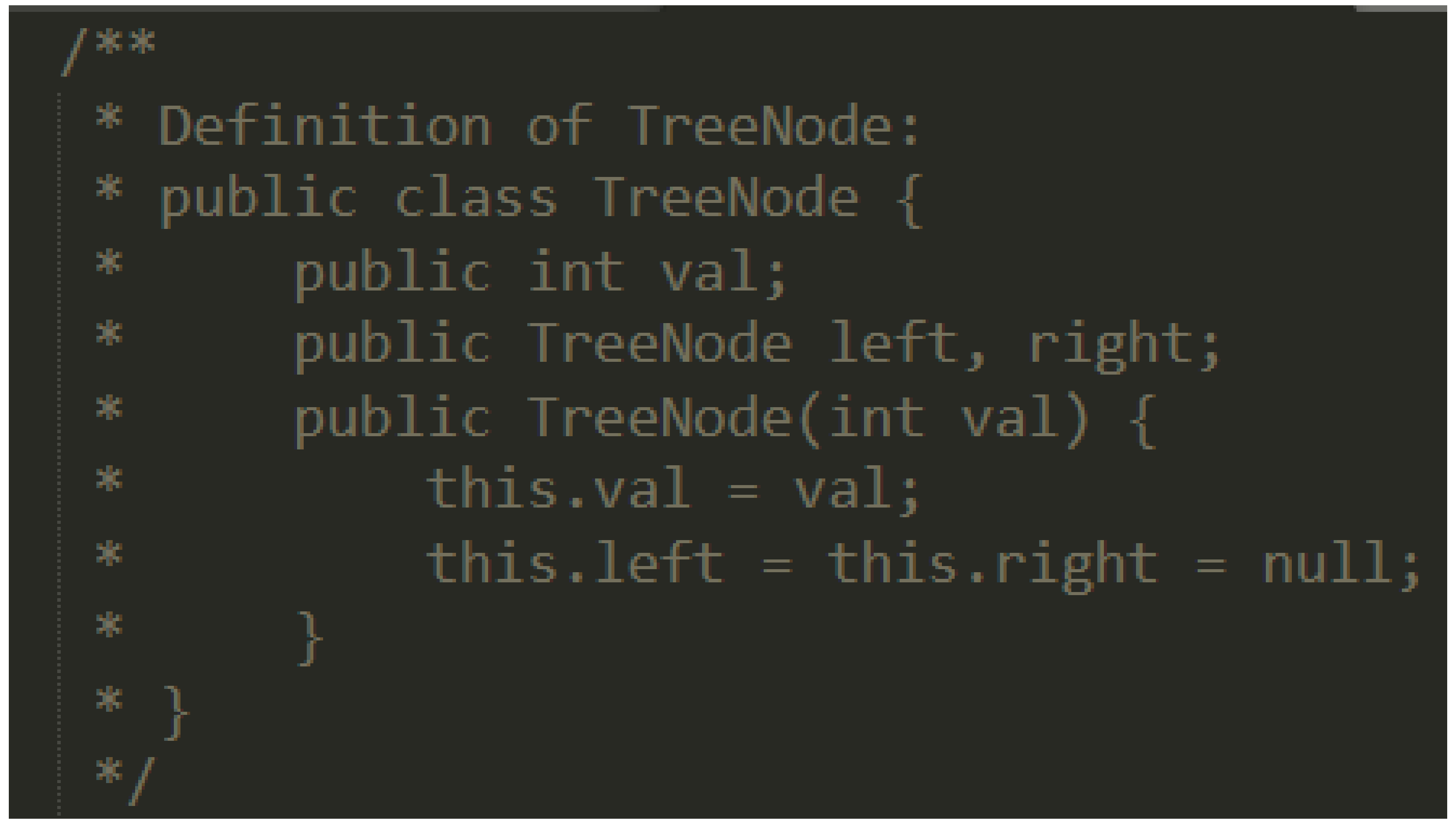
```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
```



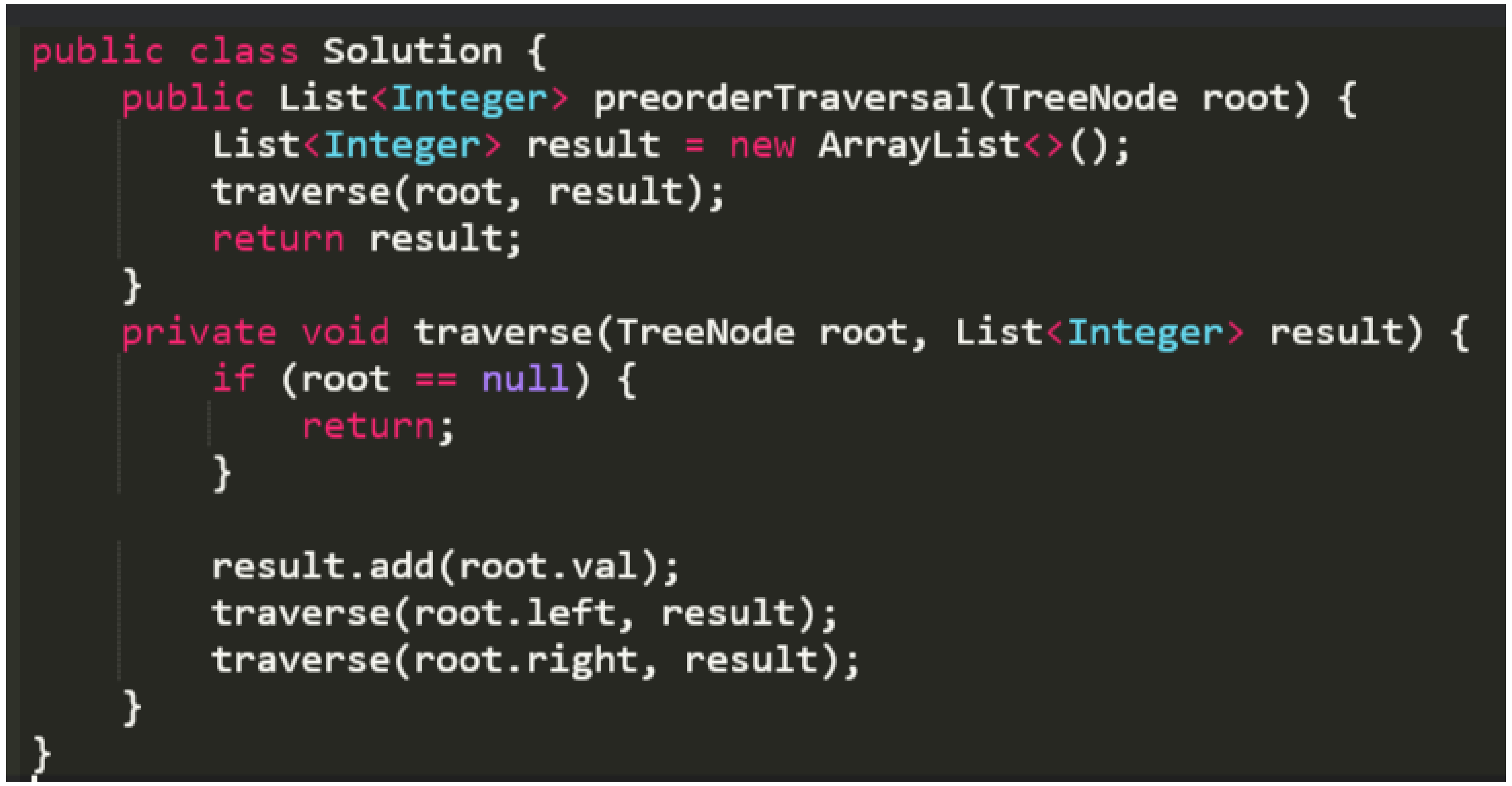
```
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        traverse(root, result);
        return result;
    }
    private void traverse(TreeNode root, List<Integer> result) {
        if (root == null) {
            return;
        }

        result.add(root.val);
        traverse(root.left, result);
        traverse(root.right, result);
    }
}
```

复杂度是多少?

LintCode 66. 二叉树的前序遍历



```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
```



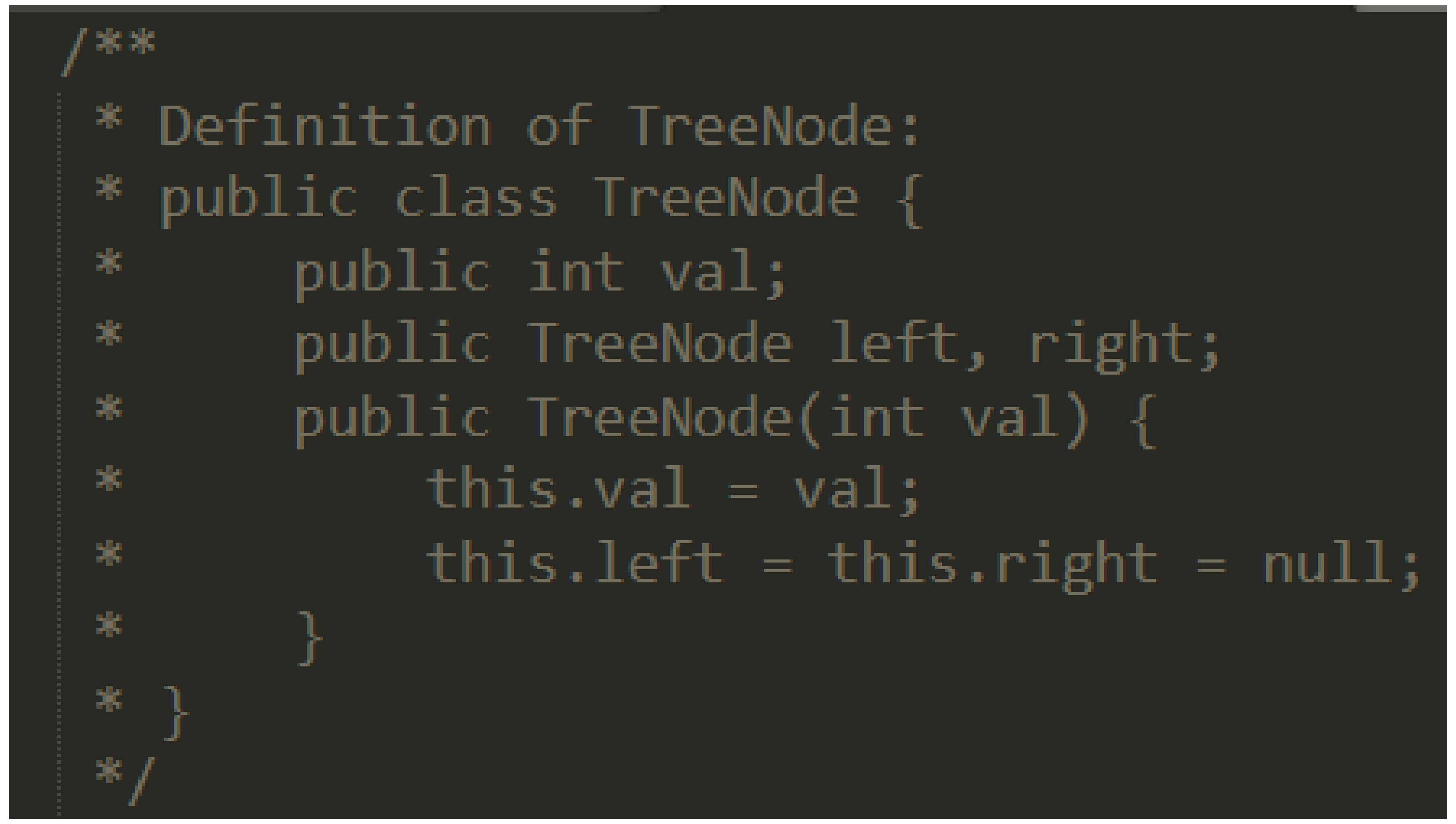
```
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        traverse(root, result);
        return result;
    }
    private void traverse(TreeNode root, List<Integer> result) {
        if (root == null) {
            return;
        }

        result.add(root.val);
        traverse(root.left, result);
        traverse(root.right, result);
    }
}
```

假设n个结点；

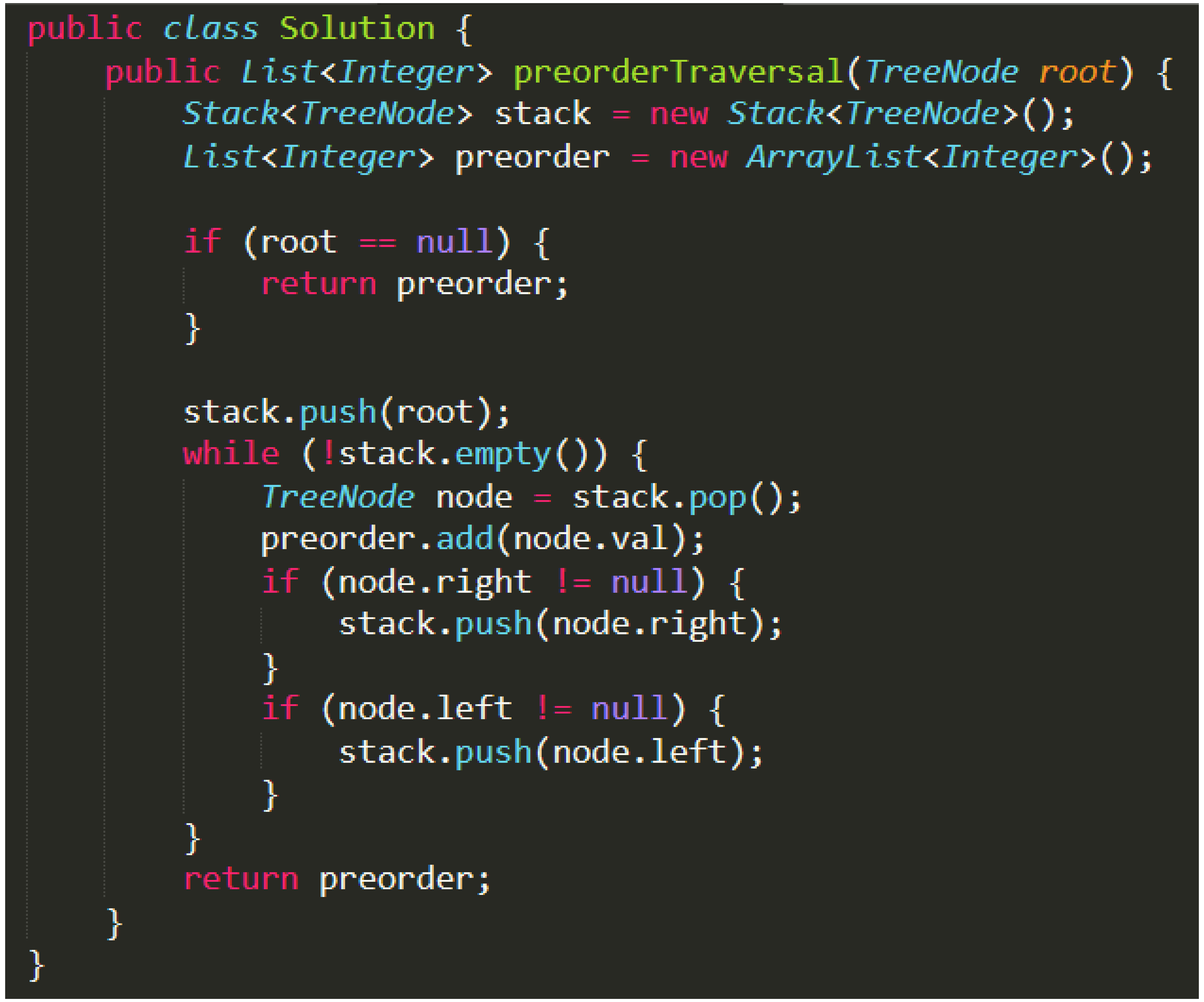
时间复杂度：O(n)；

空间复杂度：O(n)。

LintCode 66. 二叉树的前序遍历



```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
```



```
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        Stack<TreeNode> stack = new Stack<TreeNode>();
        List<Integer> preorder = new ArrayList<Integer>();

        if (root == null) {
            return preorder;
        }

        stack.push(root);
        while (!stack.empty()) {
            TreeNode node = stack.pop();
            preorder.add(node.val);
            if (node.right != null) {
                stack.push(node.right);
            }
            if (node.left != null) {
                stack.push(node.left);
            }
        }
        return preorder;
    }
}
```

假设n个结点;

时间复杂度：O(n);

空间复杂度：O(n)。

携程算法类面试真题串讲

LintCode 1182. 翻转字符串 II

https://www.lintcode.com/problem/reverse-string-ii/description

LintCode 1182. 翻转字符串 II

给定一个字符串和一个整数k，你需要反转从字符串开头算起的每2k个字符的前k个字符。 如果剩下少于k个字符，则反转所有字符。 如果小于2k但大于或等于k个字符，则反转前k个字符并将其他字符保留为原始字符。字符串只包含小写字母，1<=字符串长度，k<=10000。

样例 1:

Input: s = "abcdefg", k = 2

Output: "bacdfeg"

样例 2:

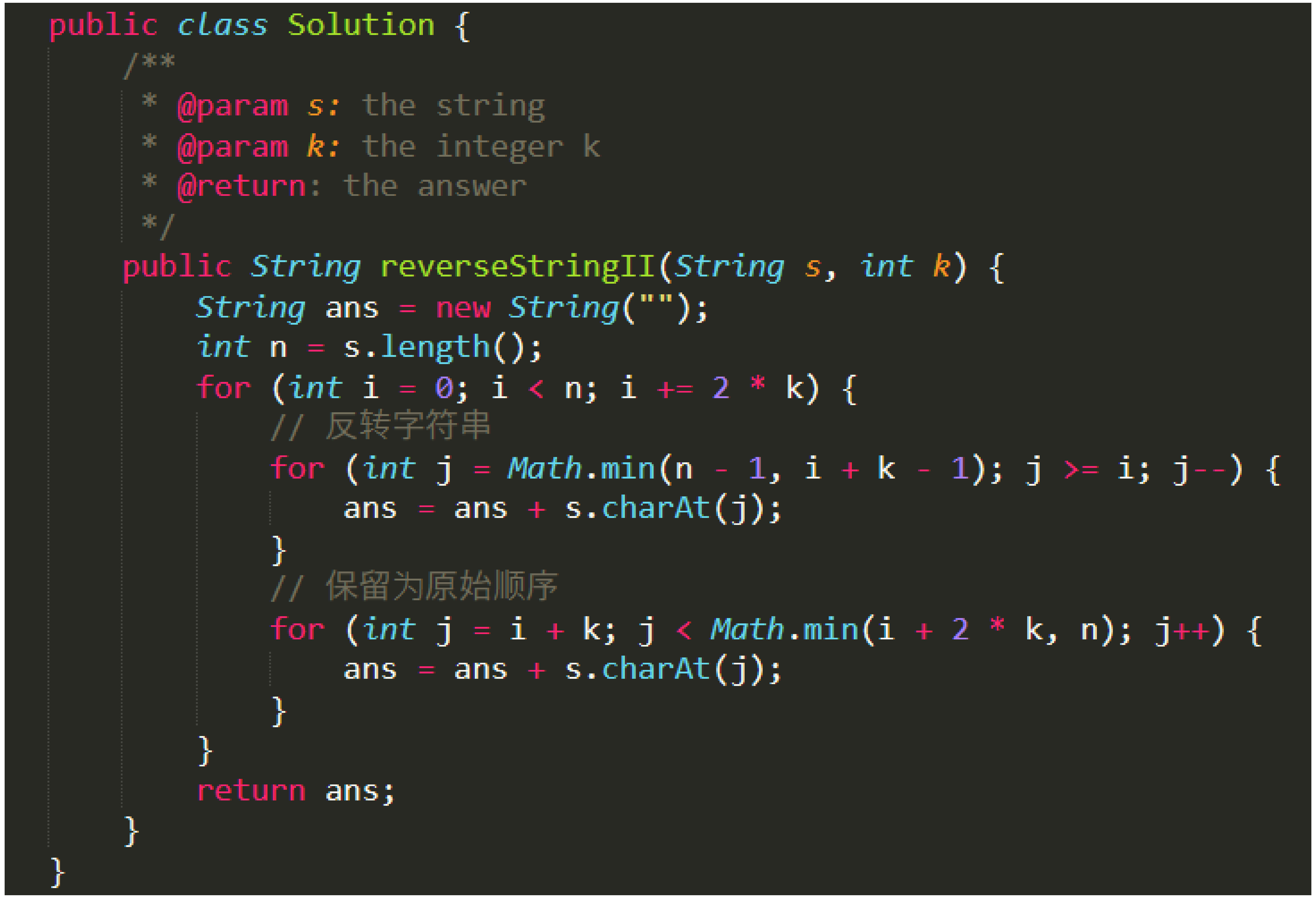
Input: s = "ace", k = 4

Output: "eca"

LintCode 1182. 翻转字符串 II

遍历每一个2*k（可能小于2*k）个长度的字符子串，依次反转前k个字符，保留后k个字符顺序。

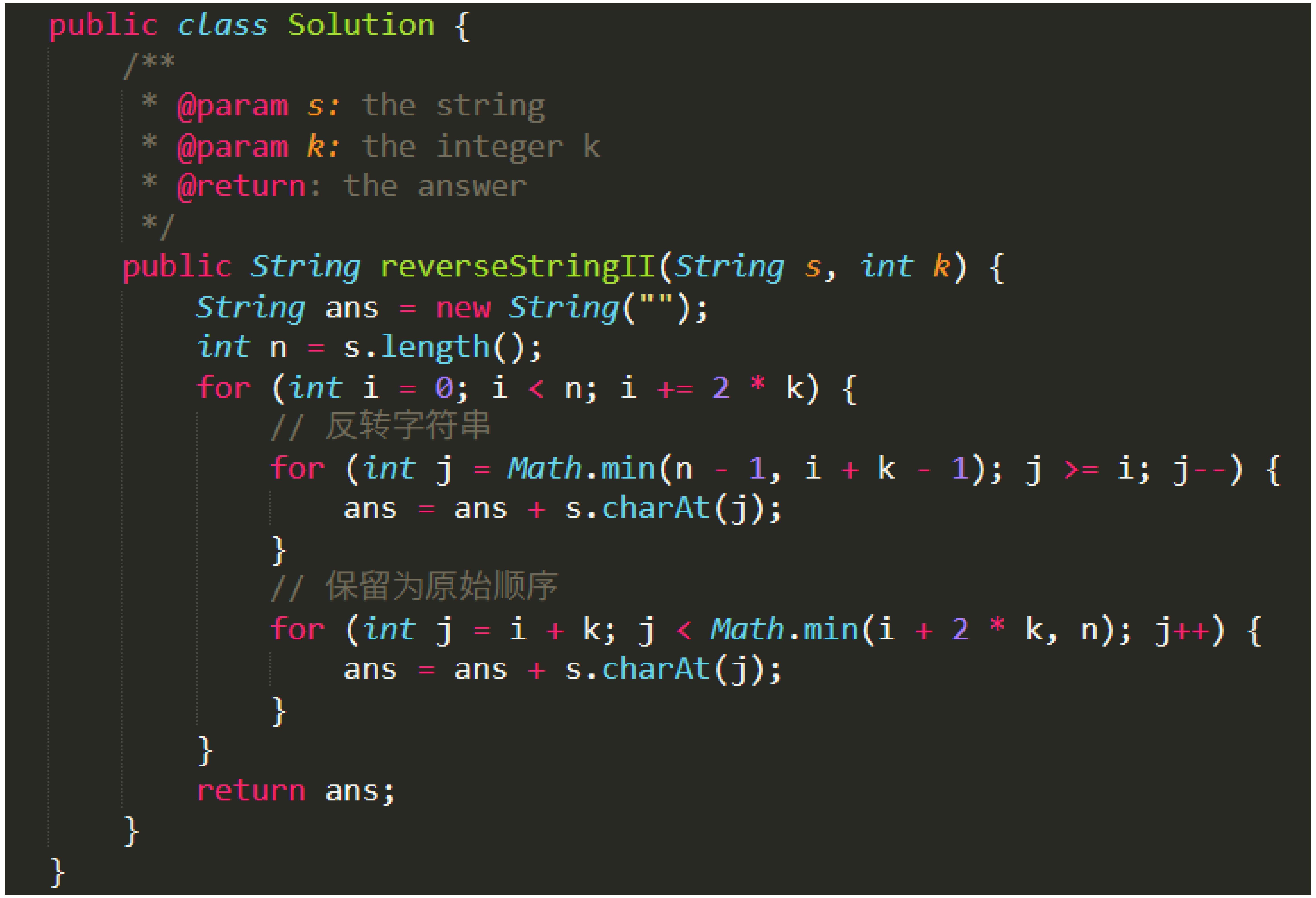
LintCode 1182. 翻转字符串 II



```java
public class Solution {
    /**
     * @param s: the string
     * @param k: the integer k
     * @return: the answer
     */
    public String reverseStringII(String s, int k) {
        String ans = new String("");
        int n = s.length();
        for (int i = 0; i < n; i += 2 * k) {
            // 反转字符串
            for (int j = Math.min(n - 1, i + k - 1); j >= i; j--) {
                ans = ans + s.charAt(j);
            }
            // 保留为原始顺序
            for (int j = i + k; j < Math.min(i + 2 * k, n); j++) {
                ans = ans + s.charAt(j);
            }
        }
        return ans;
    }
}
```

复杂度多少?

LintCode 1182. 翻转字符串 II



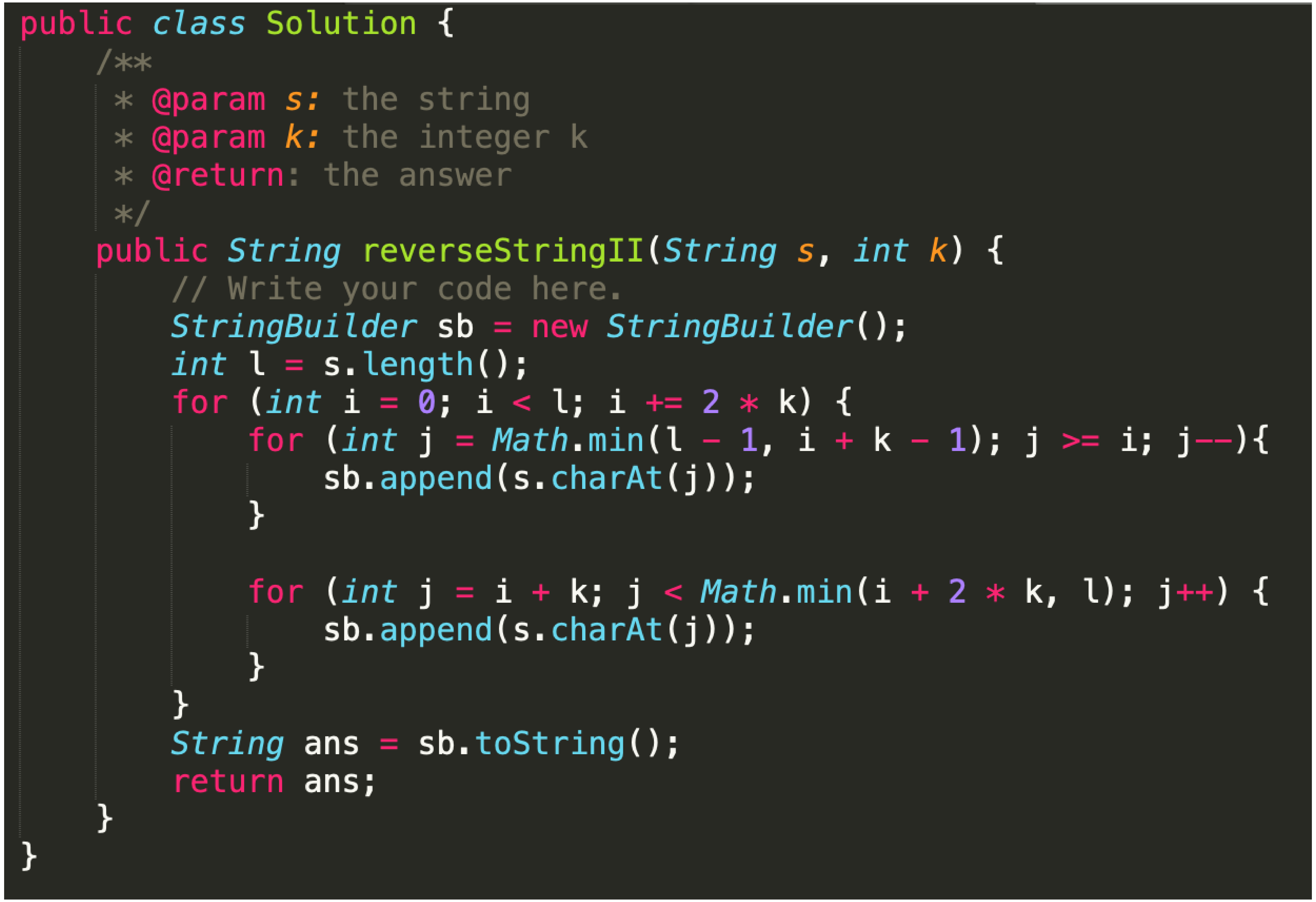
```
public class Solution {
    /**
     * @param s: the string
     * @param k: the integer k
     * @return: the answer
     */
    public String reverseStringII(String s, int k) {
        String ans = new String("");
        int n = s.length();
        for (int i = 0; i < n; i += 2 * k) {
            // 反转字符串
            for (int j = Math.min(n - 1, i + k - 1); j >= i; j--) {
                ans = ans + s.charAt(j);
            }
            // 保留为原始顺序
            for (int j = i + k; j < Math.min(i + 2 * k, n); j++) {
                ans = ans + s.charAt(j);
            }
        }
        return ans;
    }
}
```

假设字符串长度为n;

空间复杂度：O(n)

时间复杂度：O(n) ?

LintCode 1182. 翻转字符串 II



```java
public class Solution {
    /**
     * @param s: the string
     * @param k: the integer k
     * @return: the answer
     */
    public String reverseStringII(String s, int k) {
        // Write your code here.
        StringBuilder sb = new StringBuilder();
        int l = s.length();
        for (int i = 0; i < l; i += 2 * k) {
            for (int j = Math.min(l - 1, i + k - 1); j >= i; j--){
                sb.append(s.charAt(j));
            }

            for (int j = i + k; j < Math.min(i + 2 * k, l); j++) {
                sb.append(s.charAt(j));
            }
        }
        String ans = sb.toString();
        return ans;
    }
}
```

假设字符串长度为n;

空间复杂度：O(n)

时间复杂度：O(n)

LintCode 88. 最近公共祖先

https://www.lintcode.com/problem/lowest-common-ancestor-of-a-binary-tree/description

LintCode 88. 最近公共祖先

给定一棵二叉树，找到两个节点的最近公共父节点(LCA)。最近公共祖先是两个节点的公共的祖先节点且具有最大深度。

样例 1:

输入：{1},1,1

输出： 1

解释：二叉树如下（只有一个节点）

```
  1
LCA(1,1) = 1
```

样例 2:

输入：{4,3,7,#,#,5,6},3,5

输出：4

解释：二叉树如下

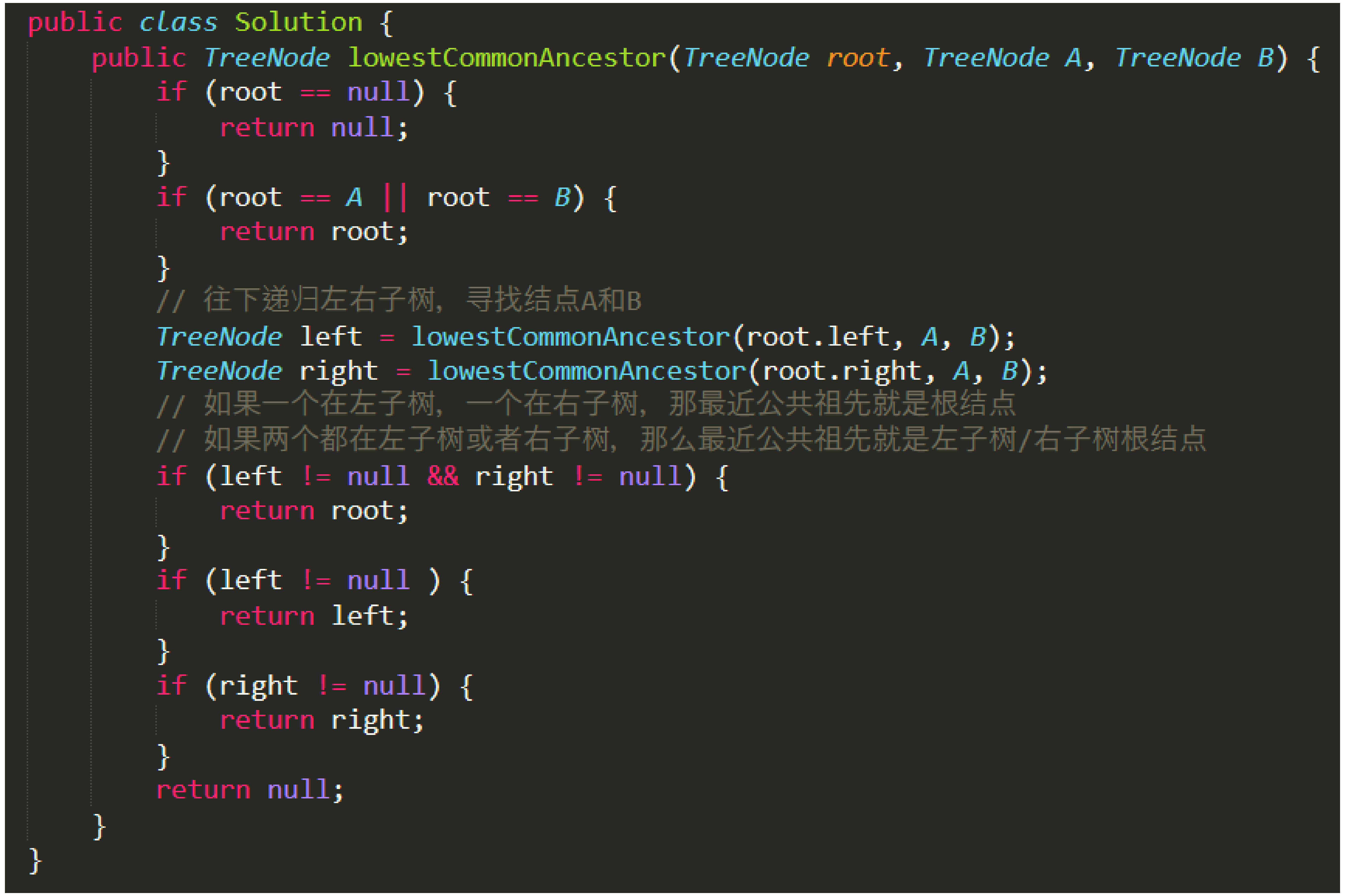
```
   4
  / \
 3   7
    / \
   5   6
LCA(3, 5) = 4
```

LintCode 88. 最近公共祖先

DFS

往下往左右子树搜索两个结点，返回第一个公共祖先结点

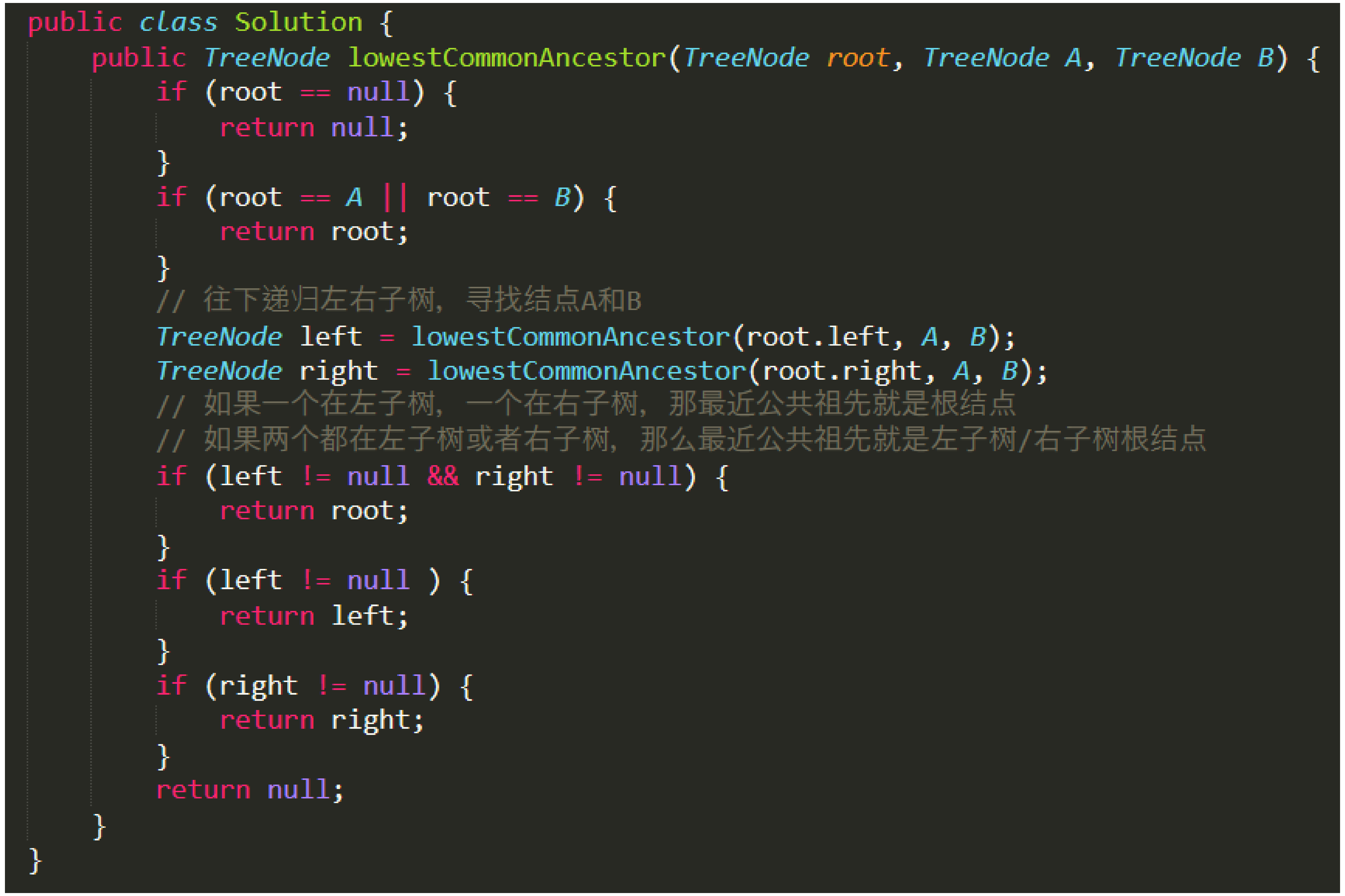
LintCode 88. 最近公共祖先



```java
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode A, TreeNode B) {
        if (root == null) {
            return null;
        }
        if (root == A || root == B) {
            return root;
        }
        // 往下递归左右子树, 寻找结点A和B
        TreeNode left = lowestCommonAncestor(root.left, A, B);
        TreeNode right = lowestCommonAncestor(root.right, A, B);
        // 如果一个在左子树, 一个在右子树, 那最近公共祖先就是根结点
        // 如果两个都在左子树或者右子树, 那么最近公共祖先就是左子树/右子树根结点
        if (left != null && right != null) {
            return root;
        }
        if (left != null ) {
            return left;
        }
        if (right != null) {
            return right;
        }
        return null;
    }
}
```

复杂度是多少?

LintCode 88. 最近公共祖先



```java
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode A, TreeNode B) {
        if (root == null) {
            return null;
        }
        if (root == A || root == B) {
            return root;
        }
        // 往下递归左右子树，寻找结点A和B
        TreeNode left = lowestCommonAncestor(root.left, A, B);
        TreeNode right = lowestCommonAncestor(root.right, A, B);
        // 如果一个在左子树，一个在右子树，那最近公共祖先就是根结点
        // 如果两个都在左子树或者右子树，那么最近公共祖先就是左子树/右子树根结点
        if (left != null && right != null) {
            return root;
        }
        if (left != null ) {
            return left;
        }
        if (right != null) {
            return right;
        }
        return null;
    }
}
```

假设有n个结点；

时间复杂度：O(n)；

空间复杂度：O(1)。

快手算法类面试真题串讲

LintCode 209. 第一个只出现一次的字符

https://www.lintcode.com/problem/first-unique-character-in-a-string/description

LintCode 209. 第一个只出现一次的字符

给出一个字符串，找出第一个只出现一次的字符。

样例 1:

输入: "abaccdeff"

输出: 'b'

解释:'b' 是第一个出现一次的字符

样例 2:

输入: "aabccd"

输出: 'b'

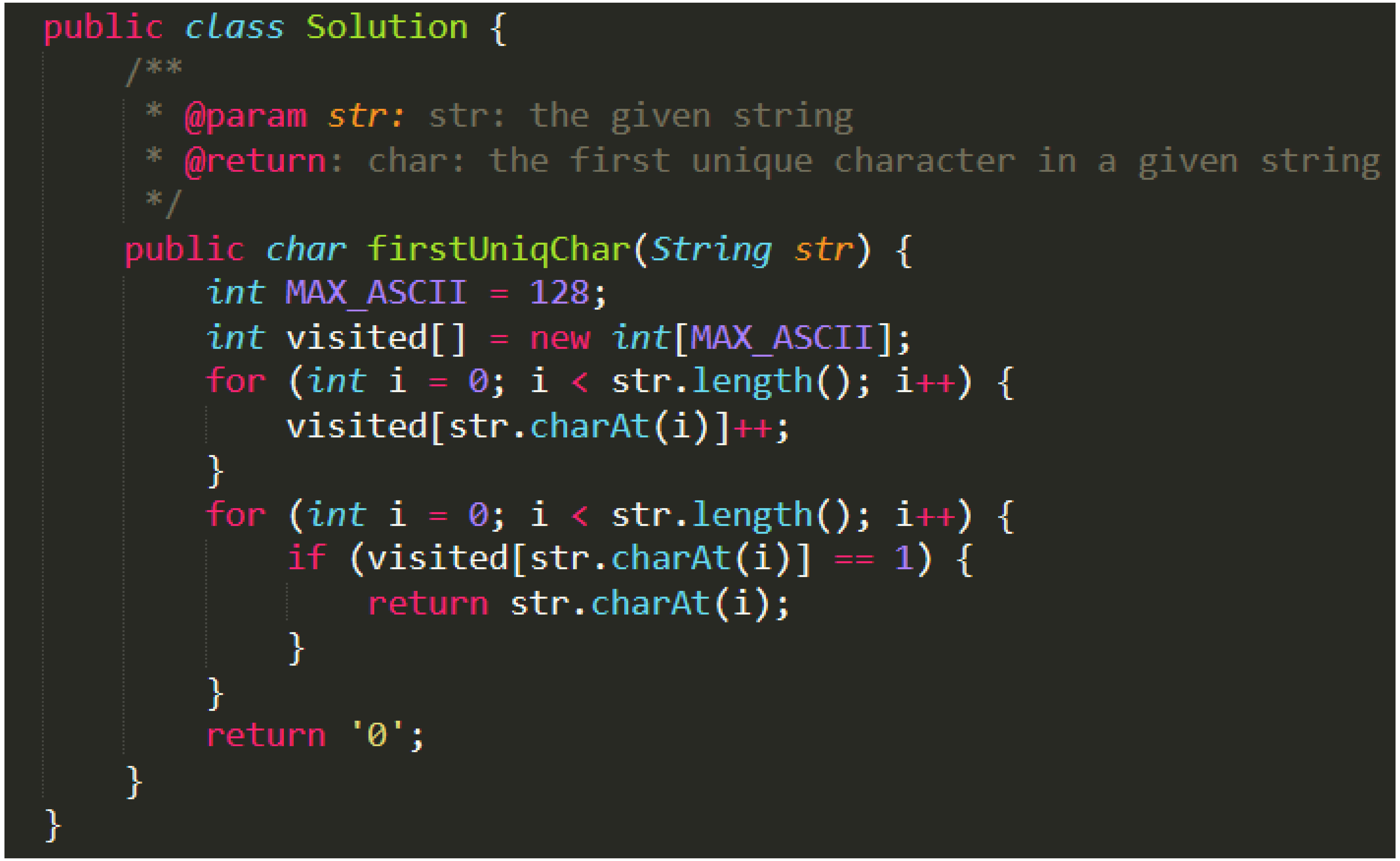
解释:'b' 是第一个出现一次的字符

LintCode 209. 第一个只出现一次的字符

遍历两次

一次记录每个字符出现次数，一次寻找第一个出现一次的字符。

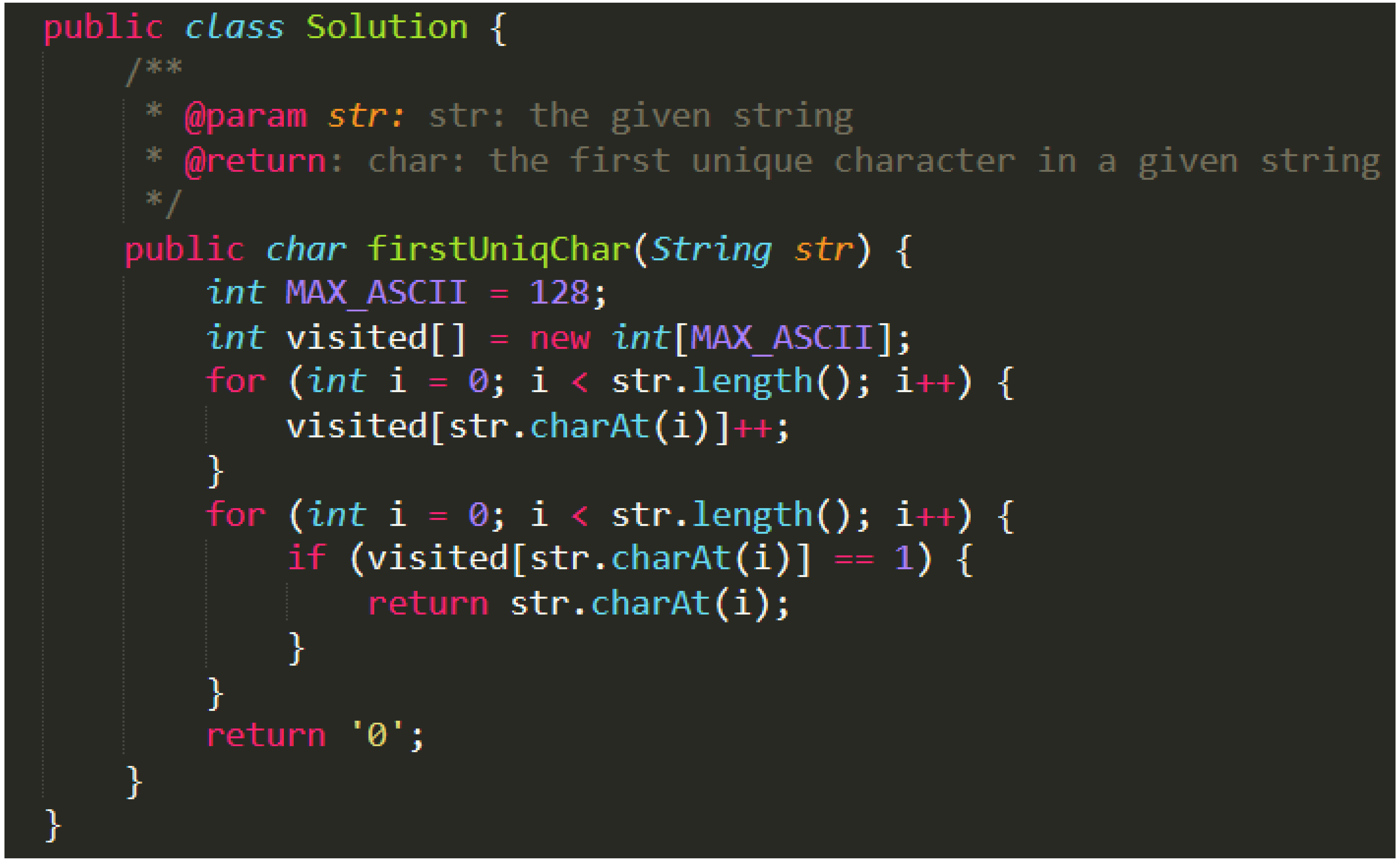
LintCode 209. 第一个只出现一次的字符



```java
public class Solution {
    /**
     * @param str: str: the given string
     * @return: char: the first unique character in a given string
     */
    public char firstUniqChar(String str) {
        int MAX_ASCII = 128;
        int visited[] = new int[MAX_ASCII];
        for (int i = 0; i < str.length(); i++) {
            visited[str.charAt(i)]++;
        }
        for (int i = 0; i < str.length(); i++) {
            if (visited[str.charAt(i)] == 1) {
                return str.charAt(i);
            }
        }
        return '0';
    }
}
```

复杂度是多少?

LintCode 209. 第一个只出现一次的字符



```java
public class Solution {
    /**
     * @param str: str: the given string
     * @return: char: the first unique character in a given string
     */
    public char firstUniqChar(String str) {
        int MAX_ASCII = 128;
        int visited[] = new int[MAX_ASCII];
        for (int i = 0; i < str.length(); i++) {
            visited[str.charAt(i)]++;
        }
        for (int i = 0; i < str.length(); i++) {
            if (visited[str.charAt(i)] == 1) {
                return str.charAt(i);
            }
        }
        return '0';
    }
}
```

假设字符串长度为n；

时间复杂度：O(n)；

空间复杂度：O(n)。

LintCode 209. 第一个只出现一次的字符

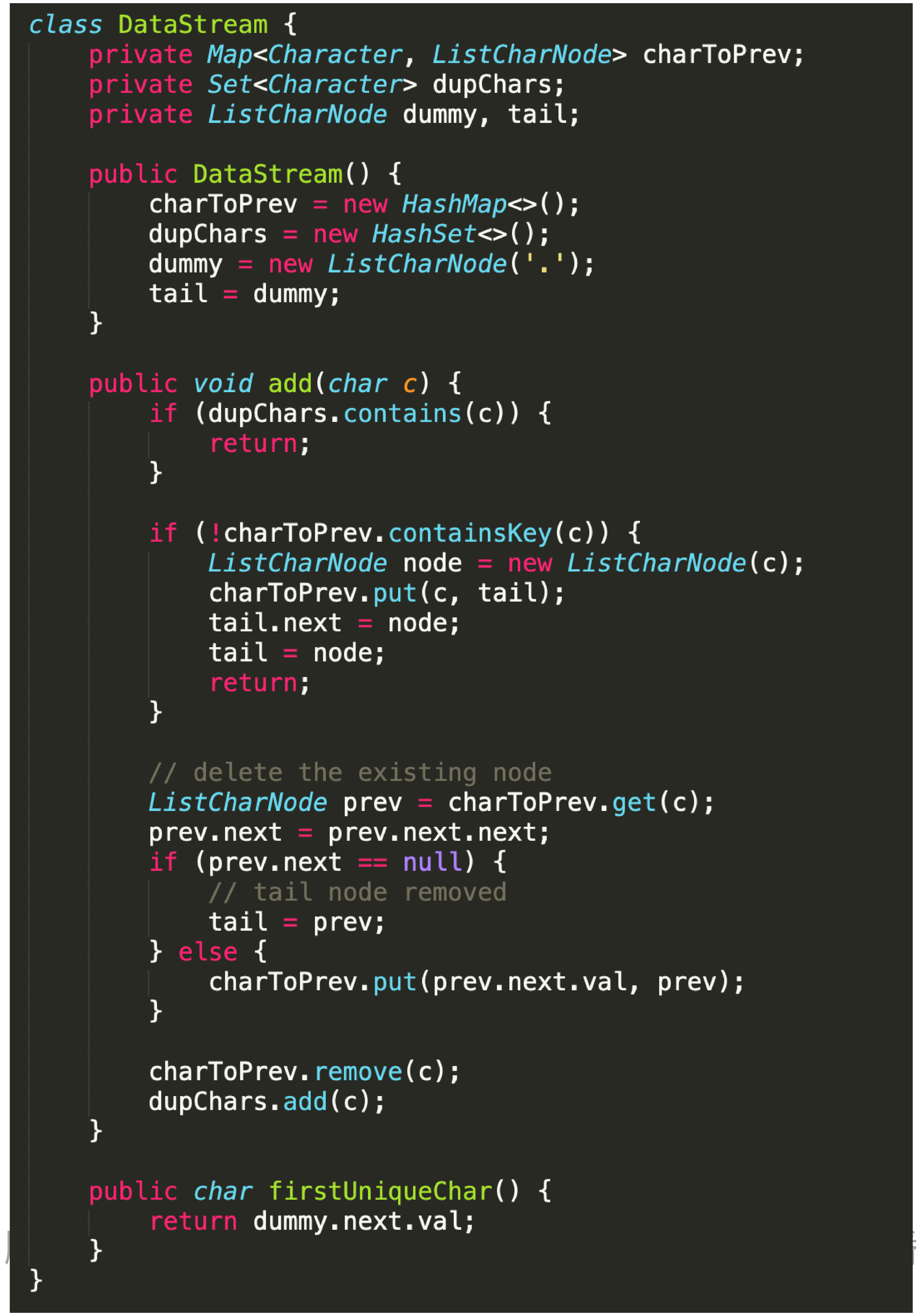
这是面试官想要的吗？

LintCode 209. 第一个只出现一次的字符

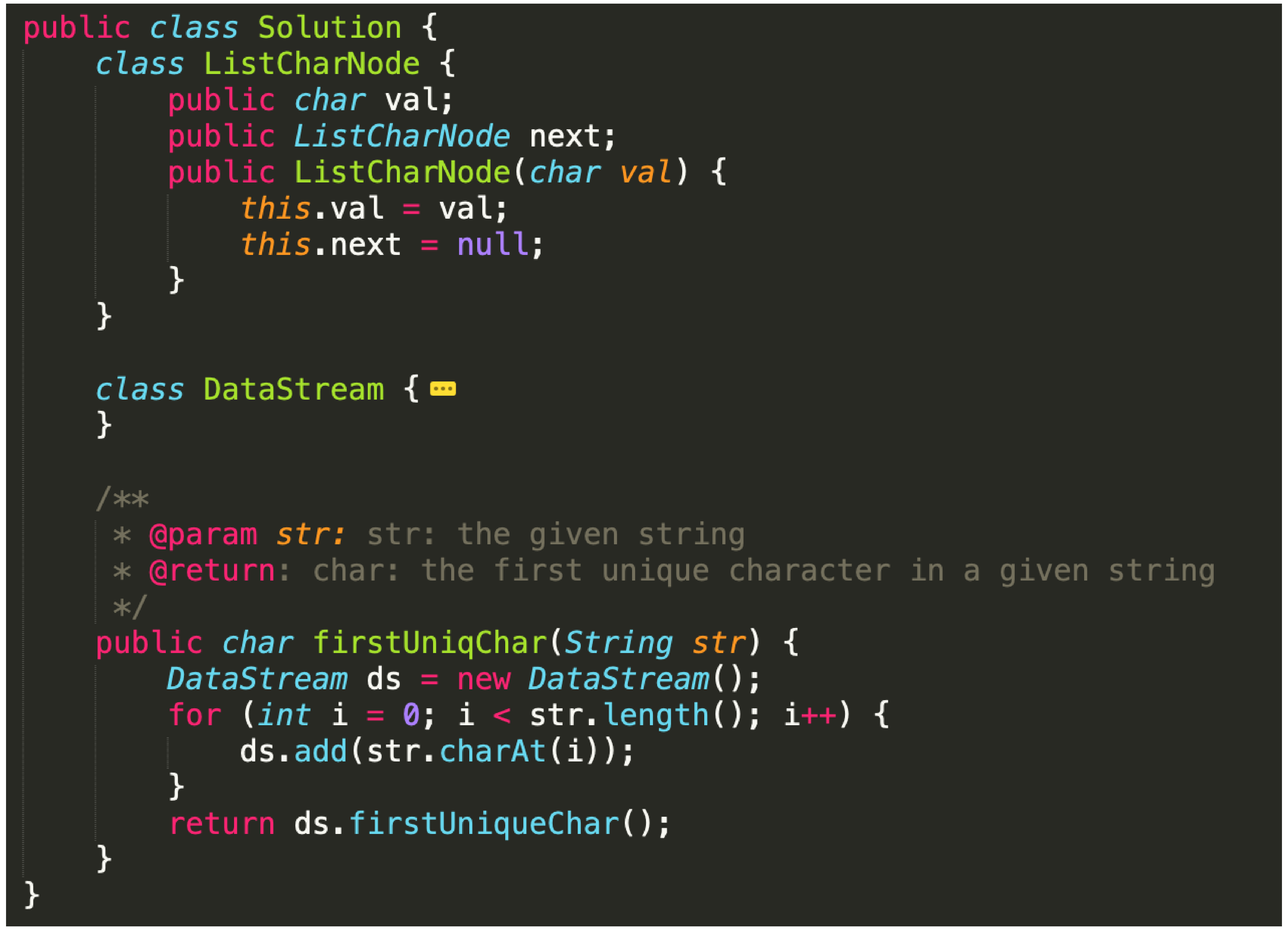
这是面试官想要的吗？

必须将其当作数据流来做

LintCode 209. 第一个只出现一次的字符



```java
class DataStream {
    private Map<Character, ListCharNode> charToPrev;
    private Set<Character> dupChars;
    private ListCharNode dummy, tail;

    public DataStream() {
        charToPrev = new HashMap<>();
        dupChars = new HashSet<>();
        dummy = new ListCharNode('.');
        tail = dummy;
    }

    public void add(char c) {
        if (dupChars.contains(c)) {
            return;
        }

        if (!charToPrev.containsKey(c)) {
            ListCharNode node = new ListCharNode(c);
            charToPrev.put(c, tail);
            tail.next = node;
            tail = node;
            return;
        }

        // delete the existing node
        ListCharNode prev = charToPrev.get(c);
        prev.next = prev.next.next;
        if (prev.next == null) {
            // tail node removed
            tail = prev;
        } else {
            charToPrev.put(prev.next.val, prev);
        }

        charToPrev.remove(c);
        dupChars.add(c);
    }

    public char firstUniqueChar() {
        return dummy.next.val;
    }
}
```



```java
public class Solution {
    class ListCharNode {
        public char val;
        public ListCharNode next;
        public ListCharNode(char val) {
            this.val = val;
            this.next = null;
        }
    }

    class DataStream {…
    }

    /**
     * @param str: str: the given string
     * @return: char: the first unique character in a given string
     */
    public char firstUniqChar(String str) {
        DataStream ds = new DataStream();
        for (int i = 0; i < str.length(); i++) {
            ds.add(str.charAt(i));
        }
        return ds.firstUniqueChar();
    }
}
```

LintCode 98. 链表排序

https://www.lintcode.com/problem/sort-list/description

LintCode 98. 链表排序

在 O(n log n) 时间复杂度和常数级的空间复杂度下给链表排序。

样例 1:

输入: 1->3->2->null

输出: 1->2->3->null

样例 2:

输入: 1->7->2->6->null

输出: 1->2->6->7->null

LintCode 98. 链表排序

归并排序

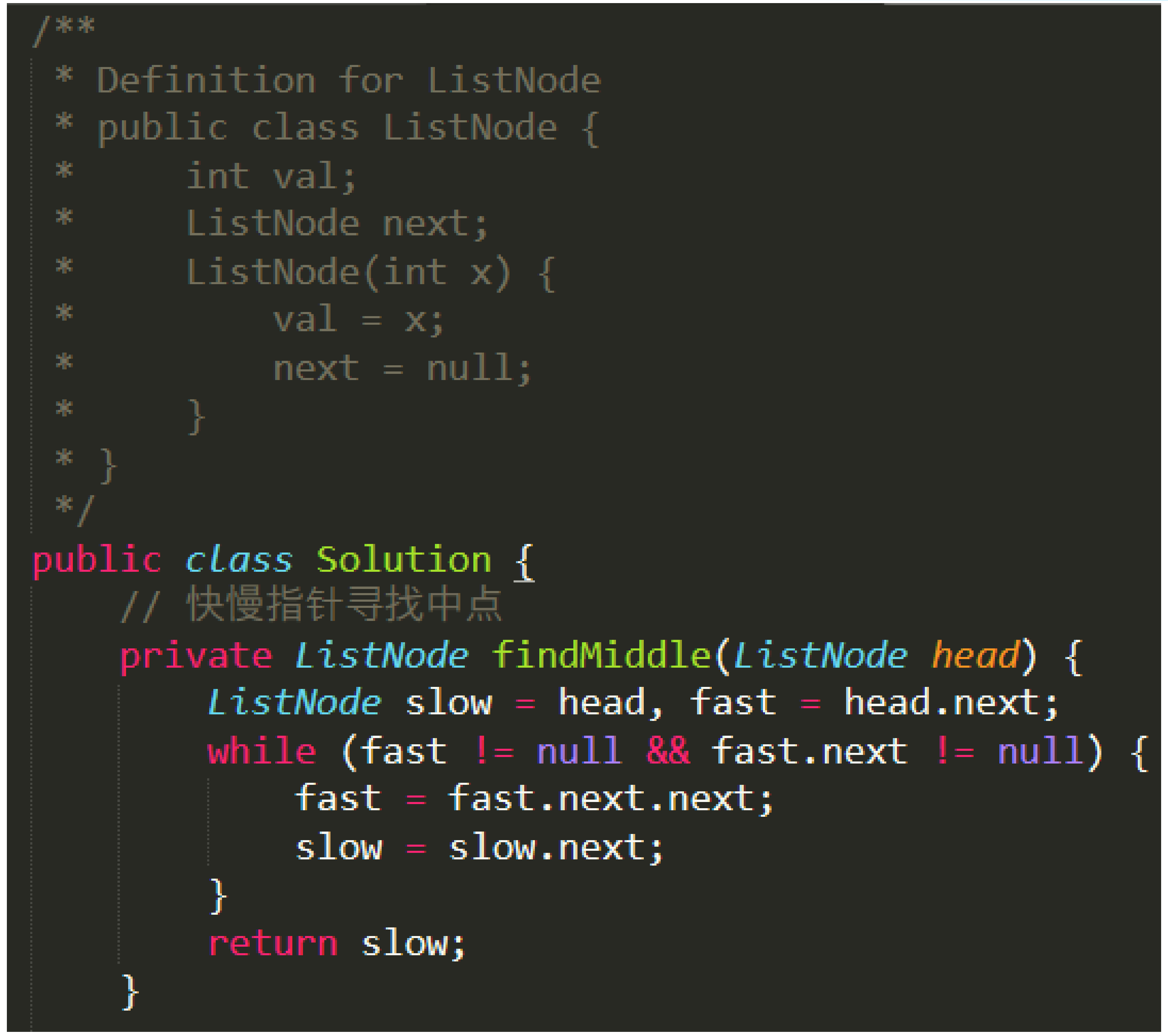
LintCode 98. 链表排序

归并排序是采用分治法的一个典型应用。

首先考虑如何将二个有序数列合并。只要比较两个数列的第一个数，谁小就先取谁，取了后就在对应数列中删除这个数。然后再进行比较，如果其中一个数列为空，那直接将另一个数列的数据依次取出即可。

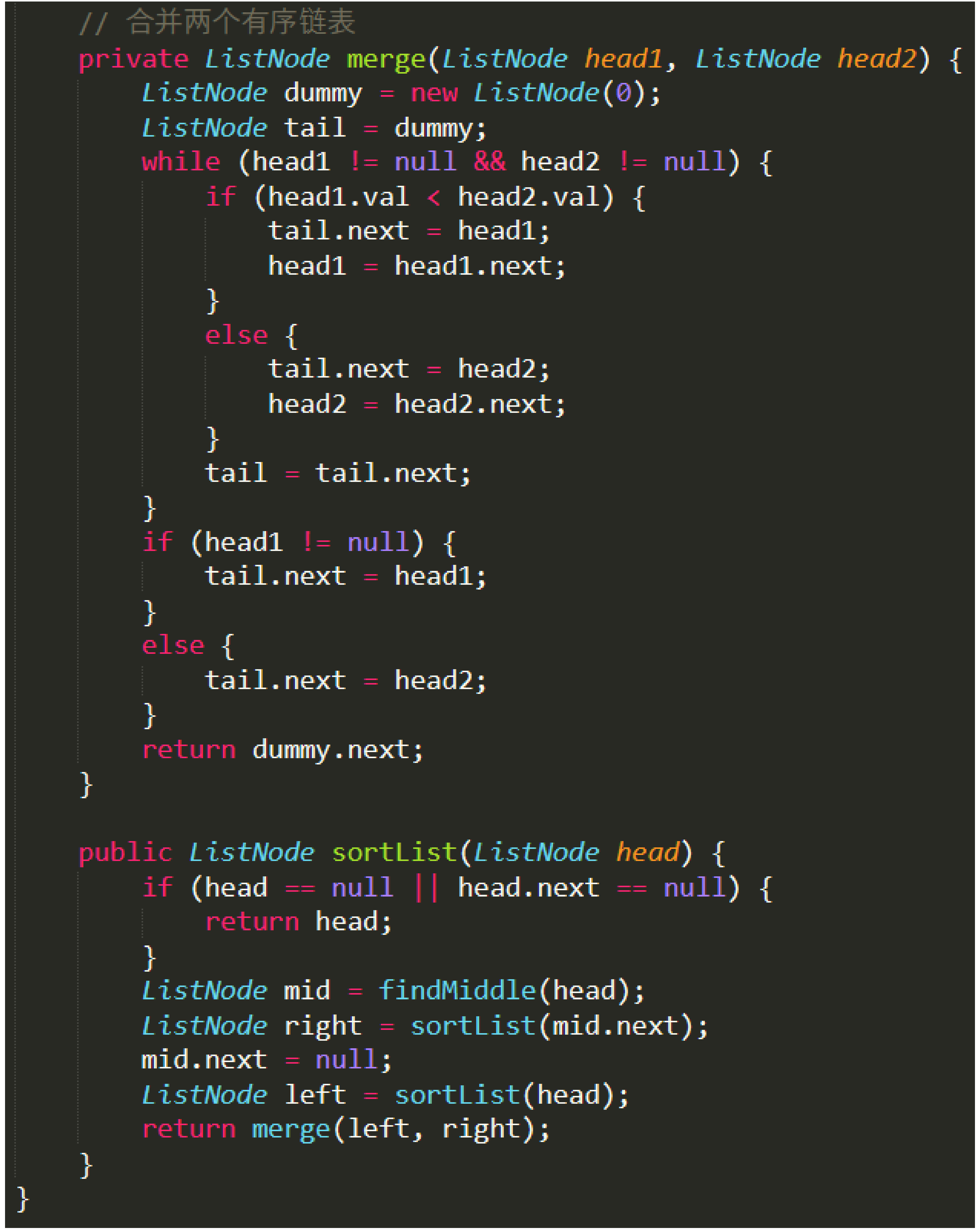
归并排序的基本思路就是将数组依次分成2组A、B，如果这2组组内的数据都是有序的，那么就可以将这2组数据进行如上操作排序。那么如何让这2组数据有序呢?可以将A、B组各自再分成2组。依次类推，当分出来的小组只有一个数据时，可以认为这个小组已经达到了有序，然后再合并相邻的2个小组就可以了。这样通过先递归分解数列，再合并数列就完成了归并排序。

LintCode 98. 链表排序



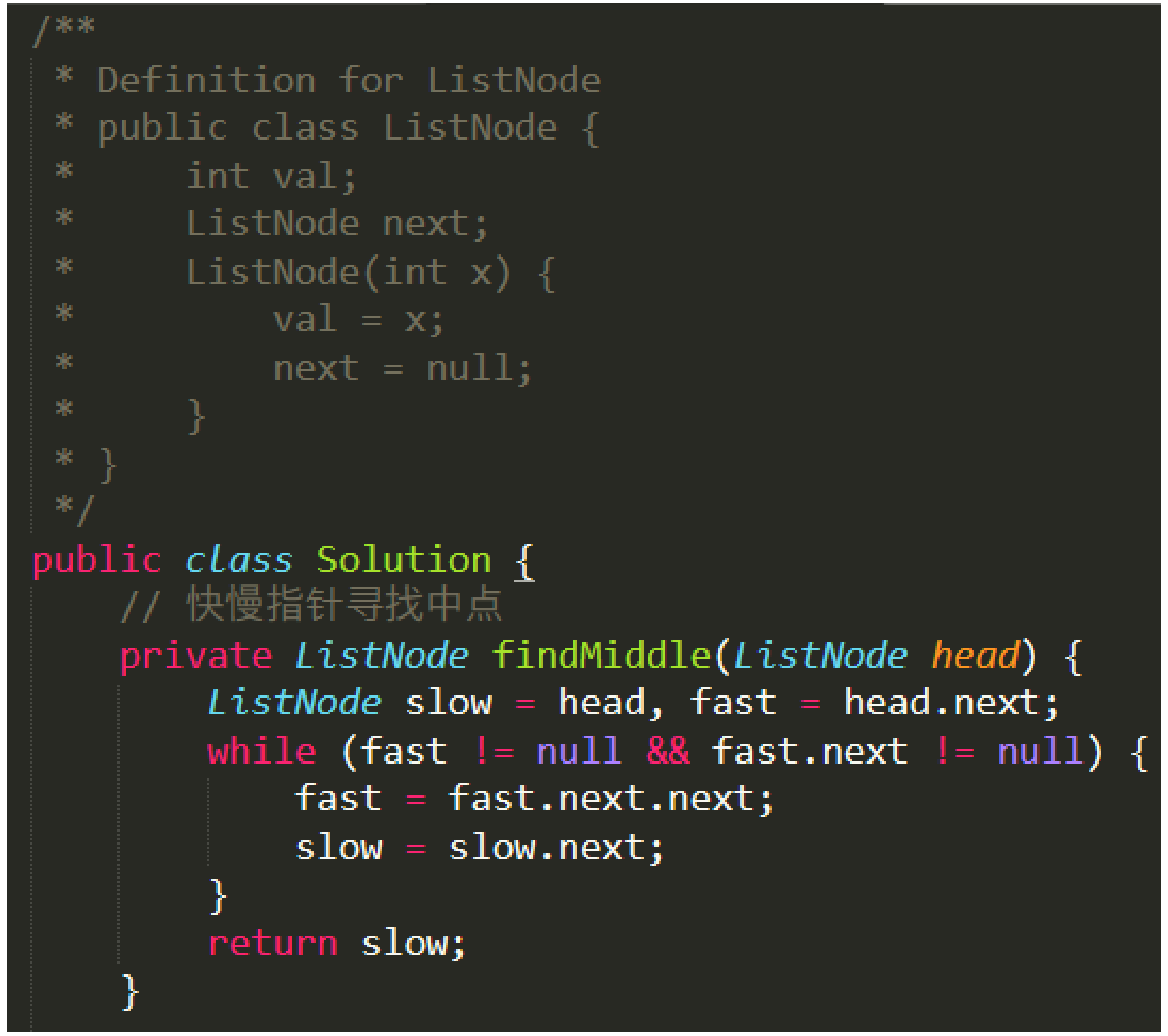
```java
/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    // 快慢指针寻找中点
    private ListNode findMiddle(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow;
    }
```

复杂度是多少?



```java
    // 合并两个有序链表
    private ListNode merge(ListNode head1, ListNode head2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (head1 != null && head2 != null) {
            if (head1.val < head2.val) {
                tail.next = head1;
                head1 = head1.next;
            }
            else {
                tail.next = head2;
                head2 = head2.next;
            }
            tail = tail.next;
        }
        if (head1 != null) {
            tail.next = head1;
        }
        else {
            tail.next = head2;
        }
        return dummy.next;
    }

    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode mid = findMiddle(head);
        ListNode right = sortList(mid.next);
        mid.next = null;
        ListNode left = sortList(head);
        return merge(left, right);
    }
}
```

LintCode 98. 链表排序

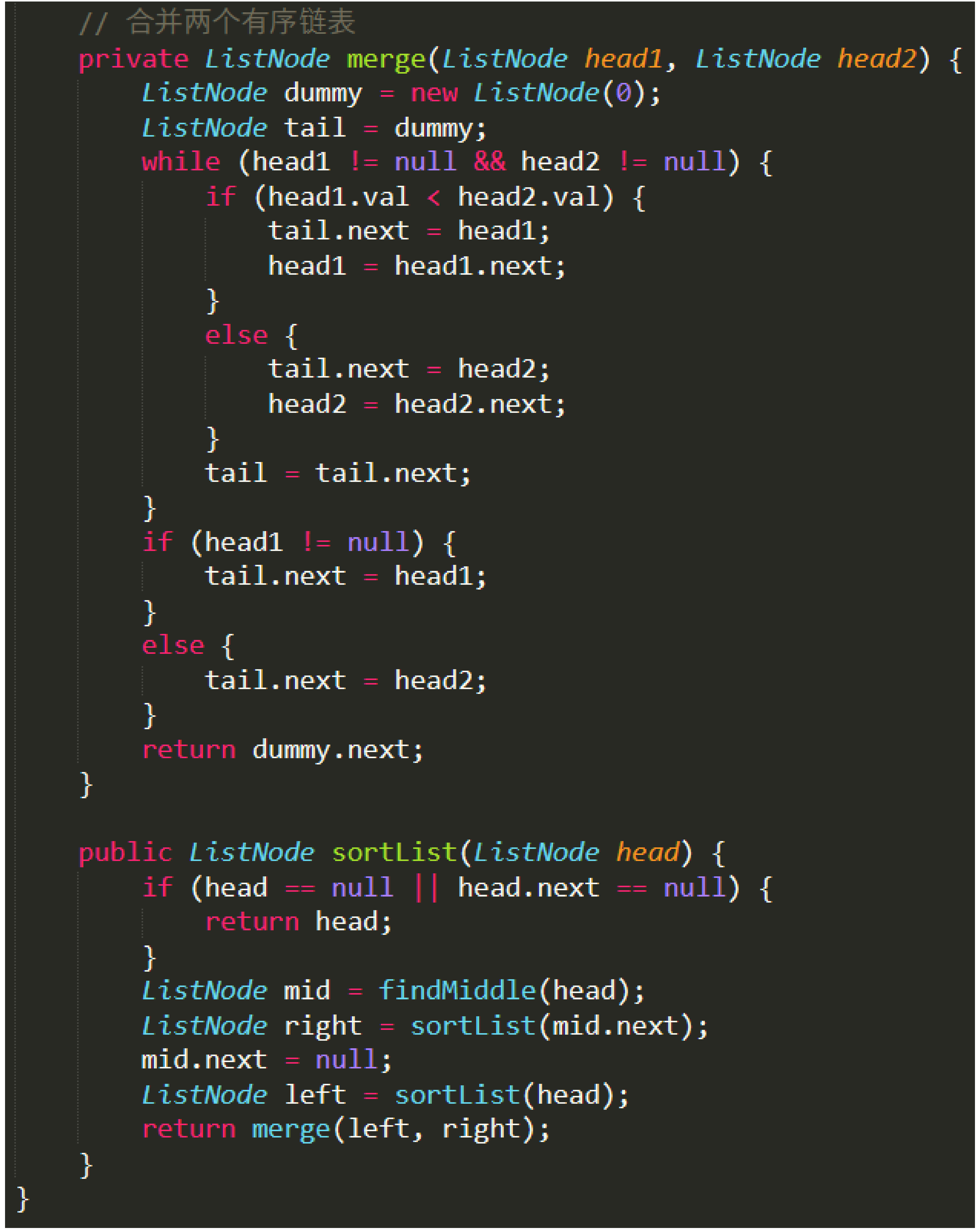


```
/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    // 快慢指针寻找中点
    private ListNode findMiddle(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }
        return slow;
    }
```

假设链表长度为n；

时间复杂度：O(nlogn)；

空间复杂度：O(1)。



```
    // 合并两个有序链表
    private ListNode merge(ListNode head1, ListNode head2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (head1 != null && head2 != null) {
            if (head1.val < head2.val) {
                tail.next = head1;
                head1 = head1.next;
            }
            else {
                tail.next = head2;
                head2 = head2.next;
            }
            tail = tail.next;
        }
        if (head1 != null) {
            tail.next = head1;
        }
        else {
            tail.next = head2;
        }
        return dummy.next;
    }

    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode mid = findMiddle(head);
        ListNode right = sortList(mid.next);
        mid.next = null;
        ListNode left = sortList(head);
        return merge(left, right);
    }
}
```

快速排序

LintCode 98. 链表排序

快速排序分为数组划分和递归排序两个步骤。

1.数组划分

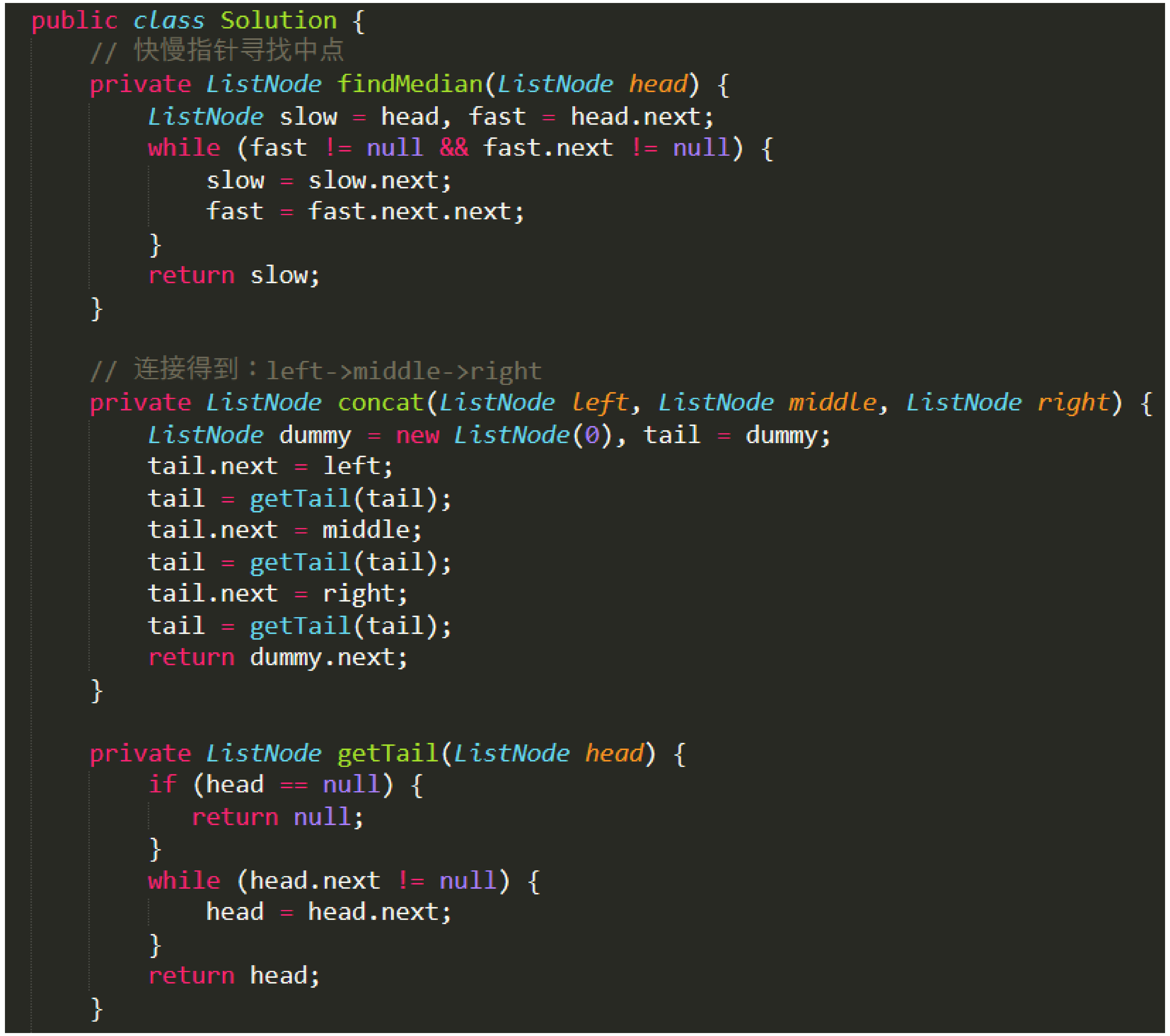
选取一个基值，将数组分为大于基值以及小于基值两部分，并返回基值所在位置以利用于递归划分。

2.递归排序

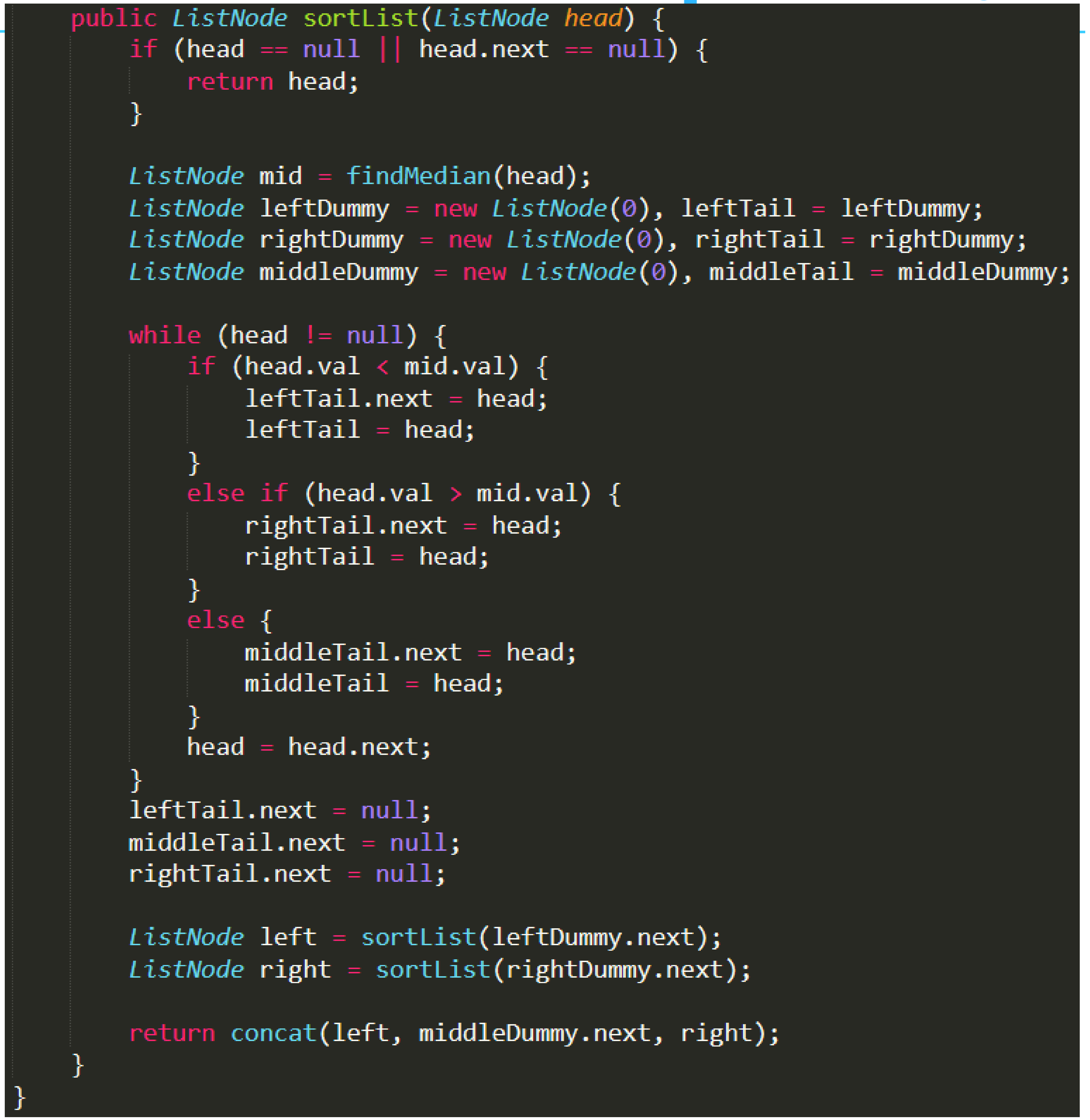
在对整个数组进行了划分后，我们将数组分成了两部分，一部分比基值小，一部分比基值大，并且我们知道了基值所在的位置，因此只需对划分出来的两部分进行递归，对划分出的两个子数组排序即可。

LintCode 98. 链表排序

复杂度是多少?



```java
public class Solution {
    // 快慢指针寻找中点
    private ListNode findMedian(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    // 连接得到：left->middle->right
    private ListNode concat(ListNode left, ListNode middle, ListNode right) {
        ListNode dummy = new ListNode(0), tail = dummy;
        tail.next = left;
        tail = getTail(tail);
        tail.next = middle;
        tail = getTail(tail);
        tail.next = right;
        tail = getTail(tail);
        return dummy.next;
    }

    private ListNode getTail(ListNode head) {
        if (head == null) {
           return null;
        }
        while (head.next != null) {
            head = head.next;
        }
        return head;
    }
```

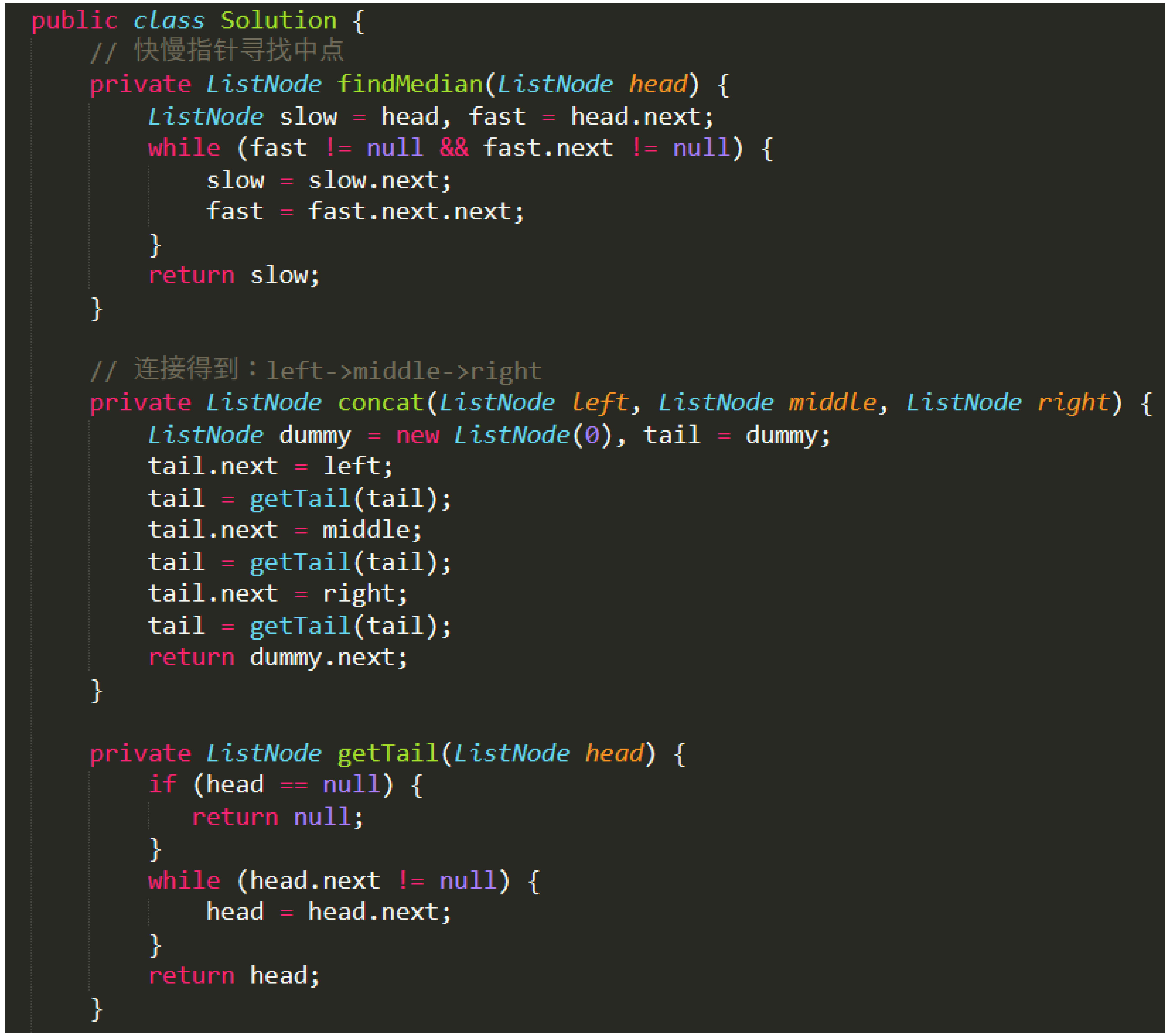


```java
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode mid = findMedian(head);
        ListNode leftDummy = new ListNode(0), leftTail = leftDummy;
        ListNode rightDummy = new ListNode(0), rightTail = rightDummy;
        ListNode middleDummy = new ListNode(0), middleTail = middleDummy;

        while (head != null) {
            if (head.val < mid.val) {
                leftTail.next = head;
                leftTail = head;
            }
            else if (head.val > mid.val) {
                rightTail.next = head;
                rightTail = head;
            }
            else {
                middleTail.next = head;
                middleTail = head;
            }
            head = head.next;
        }
        leftTail.next = null;
        middleTail.next = null;
        rightTail.next = null;

        ListNode left = sortList(leftDummy.next);
        ListNode right = sortList(rightDummy.next);

        return concat(left, middleDummy.next, right);
    }
}
```

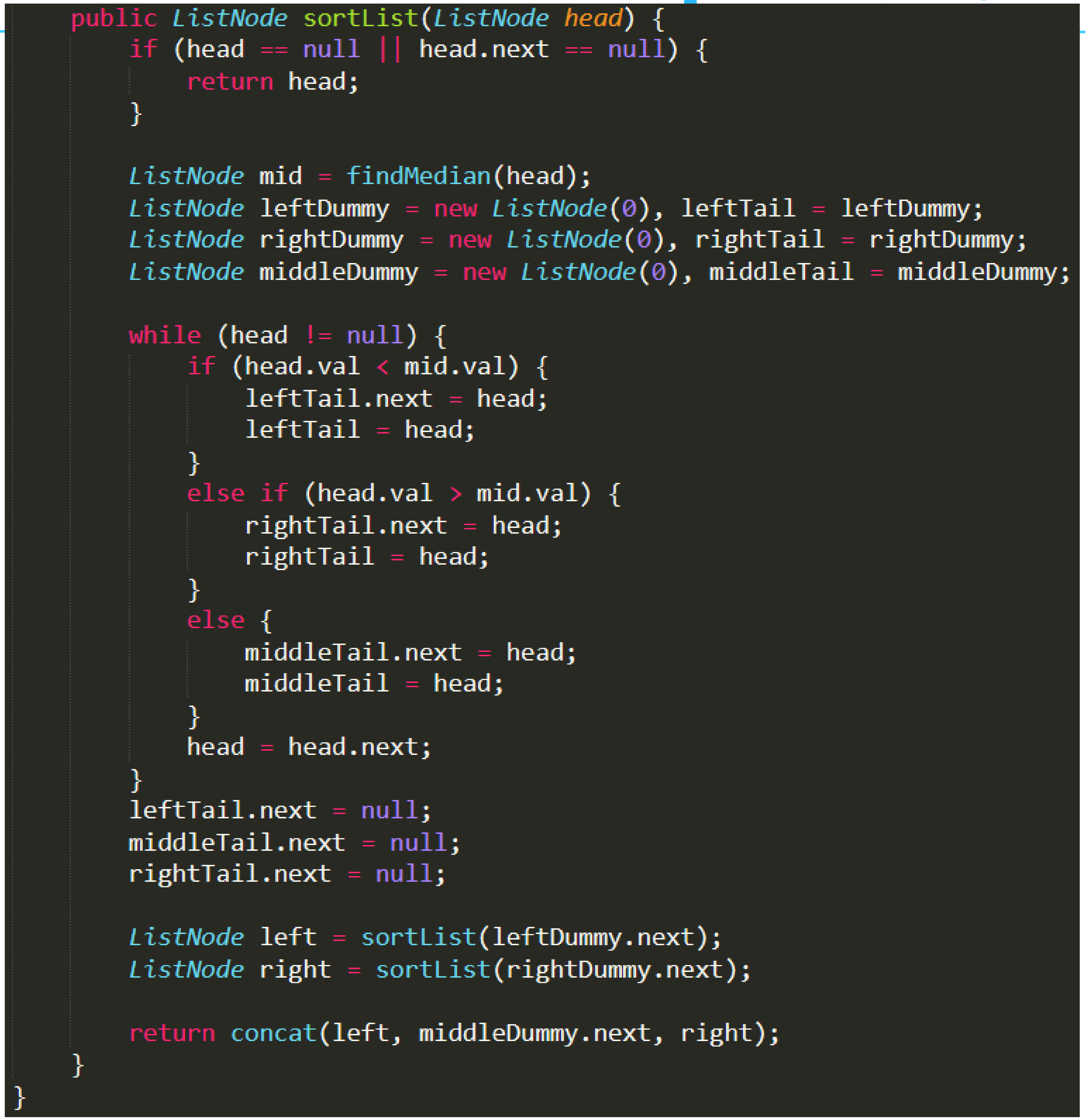
LintCode 98. 链表排序

时间复杂度：O(nlogn)；

空间复杂度：O(1)。



```java
public class Solution {
    // 快慢指针寻找中点
    private ListNode findMedian(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    // 连接得到：left->middle->right
    private ListNode concat(ListNode left, ListNode middle, ListNode right) {
        ListNode dummy = new ListNode(0), tail = dummy;
        tail.next = left;
        tail = getTail(tail);
        tail.next = middle;
        tail = getTail(tail);
        tail.next = right;
        tail = getTail(tail);
        return dummy.next;
    }

    private ListNode getTail(ListNode head) {
        if (head == null) {
            return null;
        }
        while (head.next != null) {
            head = head.next;
        }
        return head;
    }
```



```java
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode mid = findMedian(head);
        ListNode leftDummy = new ListNode(0), leftTail = leftDummy;
        ListNode rightDummy = new ListNode(0), rightTail = rightDummy;
        ListNode middleDummy = new ListNode(0), middleTail = middleDummy;

        while (head != null) {
            if (head.val < mid.val) {
                leftTail.next = head;
                leftTail = head;
            }
            else if (head.val > mid.val) {
                rightTail.next = head;
                rightTail = head;
            }
            else {
                middleTail.next = head;
                middleTail = head;
            }
            head = head.next;
        }
        leftTail.next = null;
        middleTail.next = null;
        rightTail.next = null;

        ListNode left = sortList(leftDummy.next);
        ListNode right = sortList(rightDummy.next);

        return concat(left, middleDummy.next, right);
    }
}
```

小米算法类面试真题串讲

LintCode 93. 平衡二叉树

https://www.lintcode.com/problem/balanced-binary-tree/description

LintCode 93. 平衡二叉树

给定一个二叉树，确定它是高度平衡的。一棵高度平衡的二叉树的定义是：一棵二叉树中每个节点的两个子树的深度相差不会超过1。

样例 1:

输入: tree = {1,2,3}

输出: true

样例解释: 如下，是一个平衡的二叉树。

```
  1
 / \
2   3
```

样例 2:

输入: tree = {1,#,2,3,4}

输出: false

样例解释: 如下，是一个不平衡的二叉树。1的左右子树高度差2

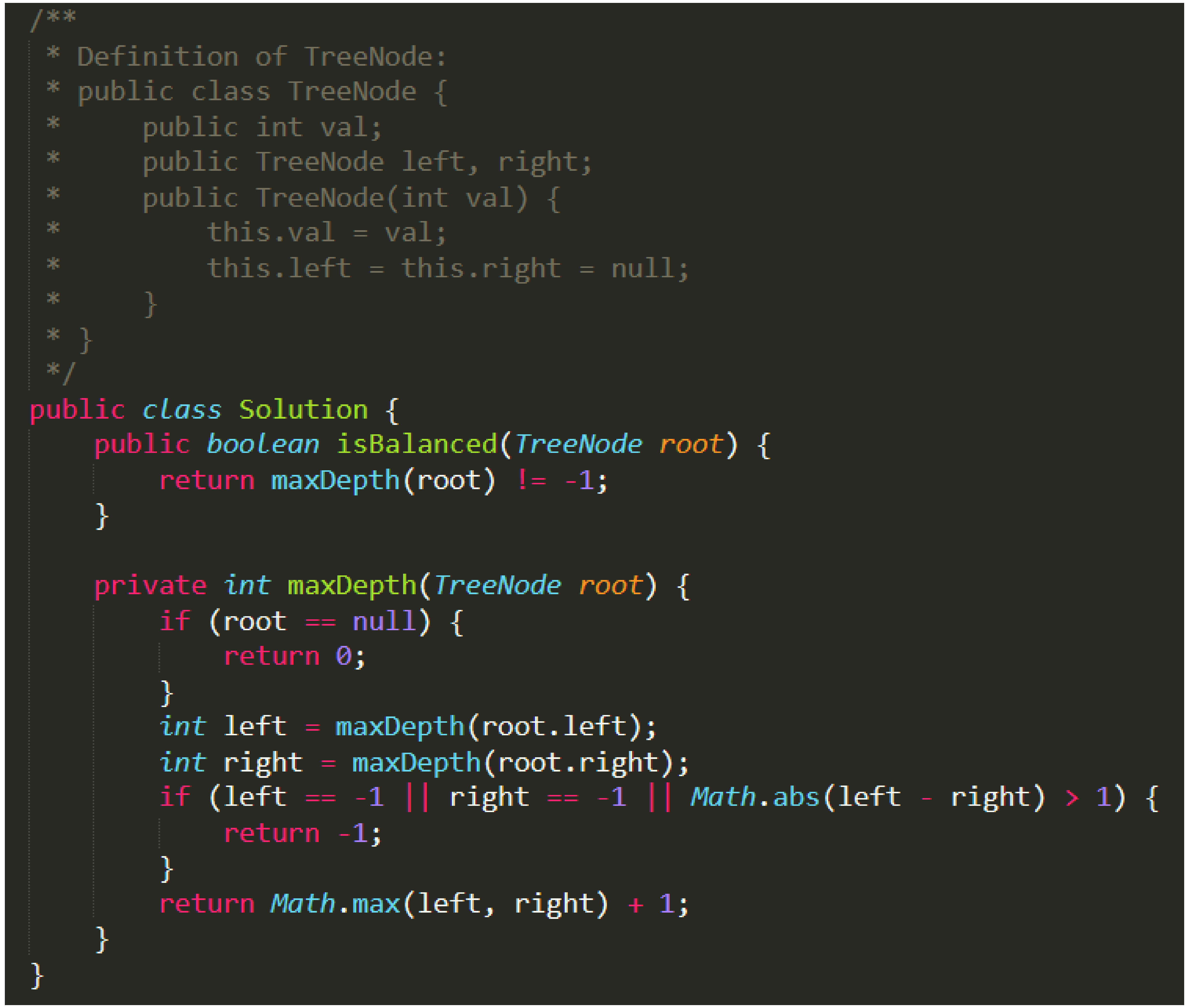
```
1
 \
  2
 / \
3   4
```

LintCode 93. 平衡二叉树

DFS

递归判断高度差是否大于1

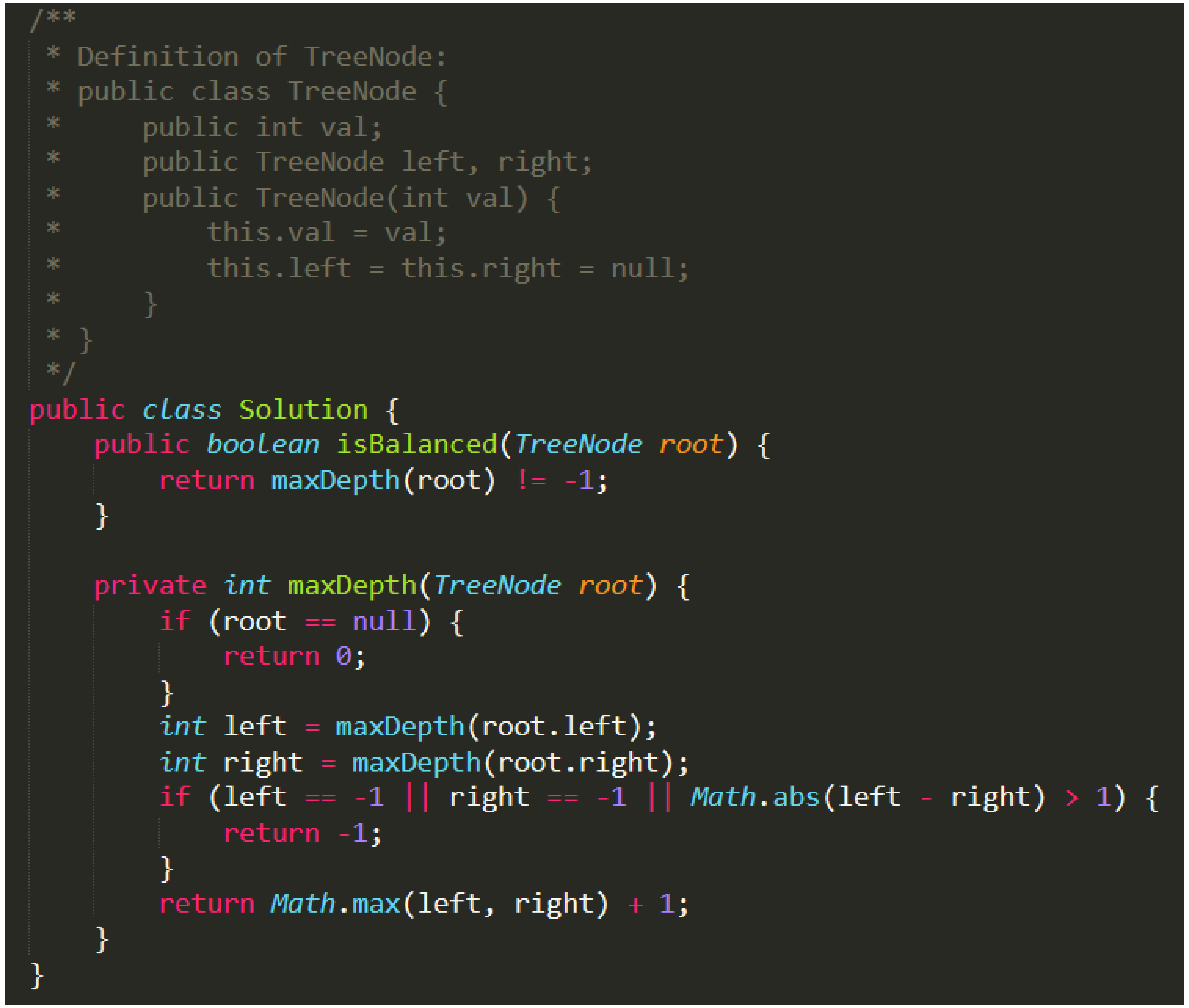
LintCode 93. 平衡二叉树



```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    public boolean isBalanced(TreeNode root) {
        return maxDepth(root) != -1;
    }

    private int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        if (left == -1 || right == -1 || Math.abs(left - right) > 1) {
            return -1;
        }
        return Math.max(left, right) + 1;
    }
}
```

复杂度是多少?

LintCode 93. 平衡二叉树



```java
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    public boolean isBalanced(TreeNode root) {
        return maxDepth(root) != -1;
    }

    private int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        if (left == -1 || right == -1 || Math.abs(left - right) > 1) {
            return -1;
        }
        return Math.max(left, right) + 1;
    }
}
```

假设有n个结点；

时间复杂度：O(n)；

空间复杂度：O(1)。

LintCode 374. 螺旋矩阵

https://www.lintcode.com/problem/reconstruct-itinerary/description

LintCode 374. 螺旋矩阵

给定一个包含 m x n 个元素的矩阵，按照螺旋顺序，返回该矩阵中的所有要素。

螺旋顺序指的是：从矩阵左上角出发向右走，每次遇见边界或走过的数字都右转，直到走完整个矩阵。

样例 1:

输入: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

输出: [1,2,3,6,9,8,7,4,5]

样例 2:

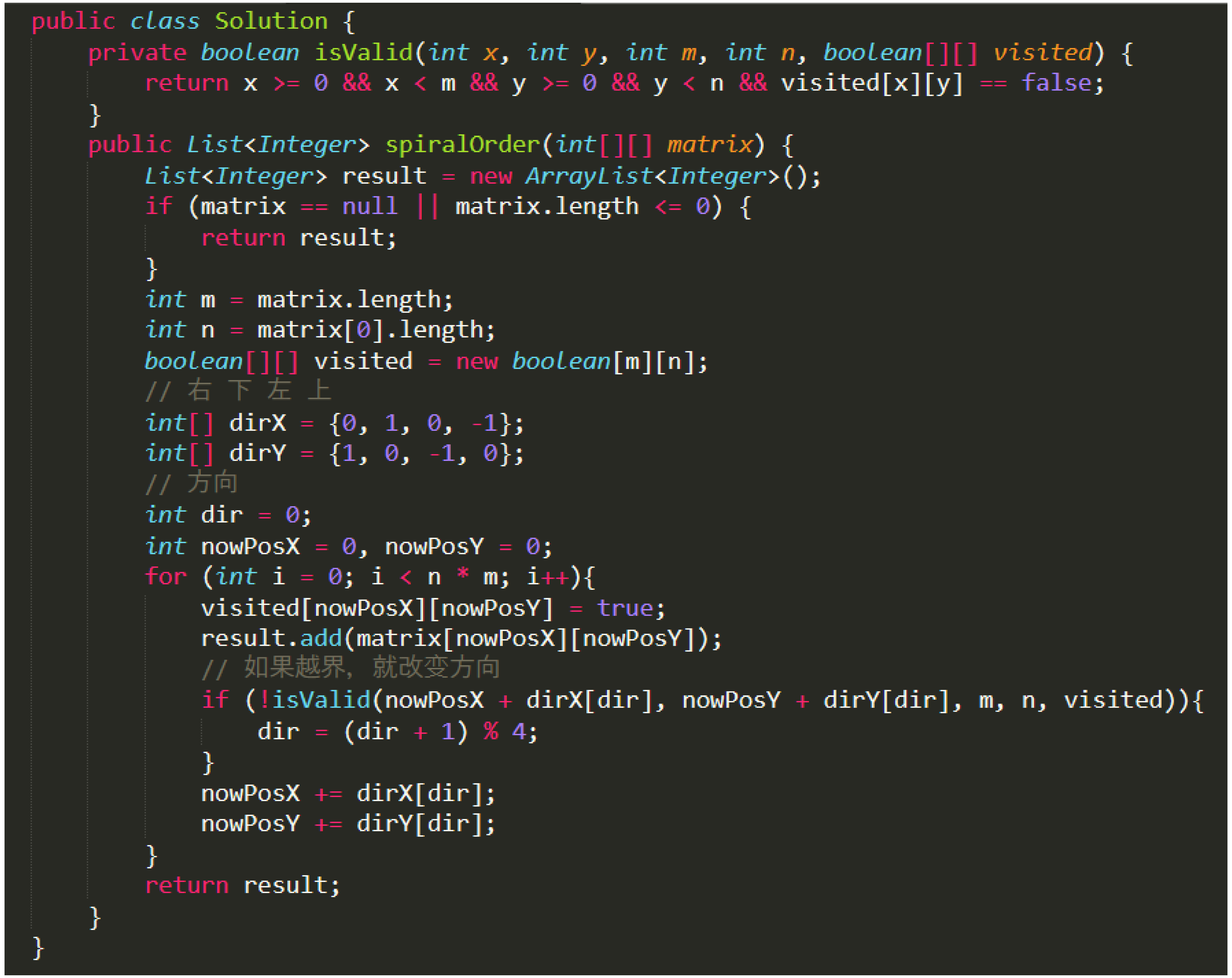
输入: [[6,4,1], [7,8,9]]

输出: [6,4,1,9,8,7]

LintCode 374. 螺旋矩阵

模拟这个顺时针螺旋的过程依次遍历矩阵元素。

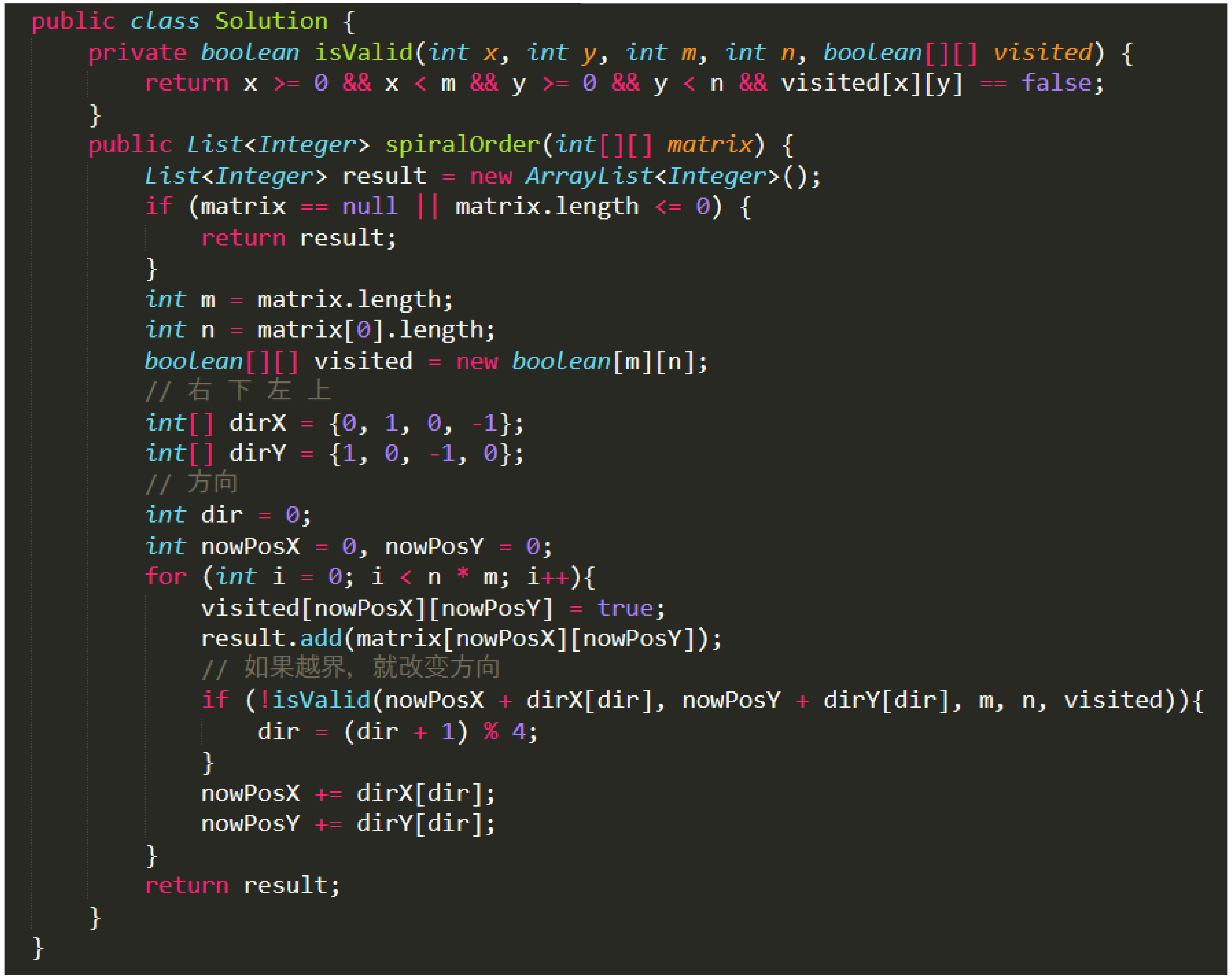
LintCode 374. 螺旋矩阵



```java
public class Solution {
    private boolean isValid(int x, int y, int m, int n, boolean[][] visited) {
        return x >= 0 && x < m && y >= 0 && y < n && visited[x][y] == false;
    }
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<Integer>();
        if (matrix == null || matrix.length <= 0) {
            return result;
        }
        int m = matrix.length;
        int n = matrix[0].length;
        boolean[][] visited = new boolean[m][n];
        // 右 下 左 上
        int[] dirX = {0, 1, 0, -1};
        int[] dirY = {1, 0, -1, 0};
        // 方向
        int dir = 0;
        int nowPosX = 0, nowPosY = 0;
        for (int i = 0; i < n * m; i++){
            visited[nowPosX][nowPosY] = true;
            result.add(matrix[nowPosX][nowPosY]);
            // 如果越界，就改变方向
            if (!isValid(nowPosX + dirX[dir], nowPosY + dirY[dir], m, n, visited)){
                dir = (dir + 1) % 4;
            }
            nowPosX += dirX[dir];
            nowPosY += dirY[dir];
        }
        return result;
    }
}
```

复杂度是多少?

LintCode 374. 螺旋矩阵



```java
public class Solution {
    private boolean isValid(int x, int y, int m, int n, boolean[][] visited) {
        return x >= 0 && x < m && y >= 0 && y < n && visited[x][y] == false;
    }
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<Integer>();
        if (matrix == null || matrix.length <= 0) {
            return result;
        }
        int m = matrix.length;
        int n = matrix[0].length;
        boolean[][] visited = new boolean[m][n];
        // 右 下 左 上
        int[] dirX = {0, 1, 0, -1};
        int[] dirY = {1, 0, -1, 0};
        // 方向
        int dir = 0;
        int nowPosX = 0, nowPosY = 0;
        for (int i = 0; i < n * m; i++){
            visited[nowPosX][nowPosY] = true;
            result.add(matrix[nowPosX][nowPosY]);
            // 如果越界, 就改变方向
            if (!isValid(nowPosX + dirX[dir], nowPosY + dirY[dir], m, n, visited)){
                dir = (dir + 1) % 4;
            }
            nowPosX += dirX[dir];
            nowPosY += dirY[dir];
        }
        return result;
    }
}
```

时间复杂度：O(m*n)；

空间复杂度：O(m*n)。