$O(1)$。

首先从链表头开始，找到第一个值不等于 num 的节点，作为新的头节点，这个节点是肯定不用删除的，记为 newHead。继续往后遍历，假设当前节点为 cur，如果 cur 节点值等于 num，就将 cur 节点删除，删除的方式是将之前最近一个值不等于 num 的节点 pre 连接到 cur 的下一个节点，即 pre.next=cur.next；如果 cur 节点值不等于 num，就令 pre=cur，即更新最近一个值不等于 num 的节点。

具体实现过程请参看如下代码中的 removeValue2 方法。

```
public Node removeValue2(Node head, int num) {
        while (head != null) {
                if (head.value != num) {
                        break;
                }
                head = head.next;
        }
        Node pre = head;
        Node cur = head;
        while (cur != null) {
                if (cur.value == num) {
                        pre.next = cur.next;
                } else {
                        pre = cur;
                }
                cur = cur.next;
        }
        return head;
}
```

将搜索二叉树转换成双向链表

【题目】

对二叉树的节点来说，有本身的值域，有指向左孩子和右孩子的两个指针；对双向链表的节点来说，有本身的值域，有指向上一个节点和下一个节点的指针。在结构上，两种结构有相似性，现在有一棵搜索二叉树，请将其转换为一个有序的双向链表。

例如，节点定义为：

```
public class Node {
        public int value;
        public Node left;
        public Node right;
        public Node(int data) {
                this.value = data;
```

```
        }
    }
```

一棵搜索二叉树如图 2-7 所示。

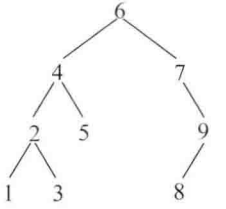

图 2-7

这棵搜索二叉树转换后的双向链表从头到尾依次是 1～9。对每一个节点来说，原来的 right 指针等价于转换后的 next 指针，原来的 left 指针等价于转换后的 last 指针，最后返回转换后的双向链表头节点。

【难度】

尉 ★★☆☆

【解答】

方法一：用队列等容器收集二叉树中序遍历结果的方法。时间复杂度为 $O(N)$，额外空间复杂度为 $O(N)$，具体过程如下：

1．生成一个队列，记为 queue，按照二叉树中序遍历的顺序，将每个节点放入 queue 中。

2．从 queue 中依次弹出节点，并按照弹出的顺序重连所有的节点即可。

方法一的具体实现请参看如下代码中的 convert1 方法。

```
public Node convert1(Node head) {
        Queue<Node> queue = new LinkedList<Node>();
        inOrderToQueue(head, queue);
        if (queue.isEmpty()) {
                return head;
        }
        head = queue.poll();
        Node pre = head;
        pre.left = null;
        Node cur = null;
        while (!queue.isEmpty()) {
                cur = queue.poll();
                pre.right = cur;
```

```
                cur.left = pre;
                pre = cur;
            }
            pre.right = null;
            return head;
    }

    public void inOrderToQueue(Node head, Queue<Node> queue) {
            if (head == null) {
                return;
            }
            inOrderToQueue(head.left, queue);
            queue.offer(head);
            inOrderToQueue(head.right, queue);
    }
```

方法二：利用递归函数，除此之外不使用任何容器的方法。时间复杂度为 $O(N)$，额外空间复杂度为 $O(h)$，h 为二叉树的高度，具体过程如下：

1．实现递归函数 process。process 的功能是将一棵搜索二叉树转换为一个结构有点特殊的有序双向链表。结构特殊是指这个双向链表尾节点的 right 指针指向该双向链表的头节点。函数 process 最终返回这个链表的尾节点。

例如：搜索二叉树只有一个节点时，在经过 process 处理后，形成如图 2-8 所示的形式，最后返回节点 1。

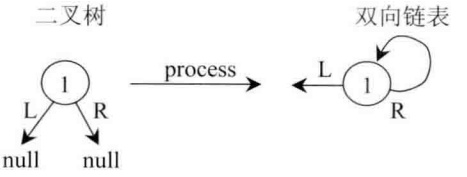

图 2-8

搜索二叉树较为一般的情况，在经过 process 处理后，变为如图 2-9 所示的形式，最后返回节点 3。

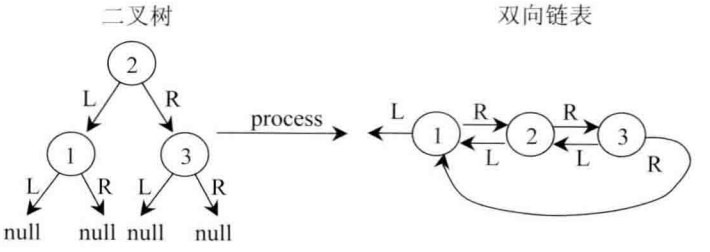

图 2-9

总之，process 函数的功能是将一棵搜索二叉树变成有序的双向链表，然后让最大值节点的 right 指针指向最小值节点，最后返回最大值节点。

那么递归函数 process 应该如何实现呢？

假设一棵搜索二叉树如图 2-10 所示。

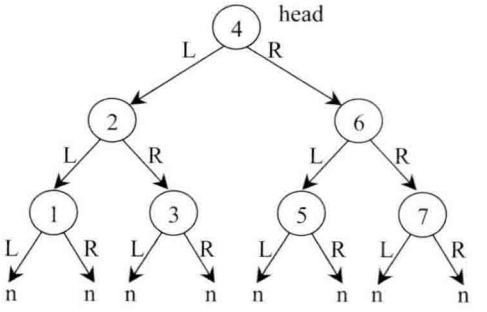

图 2-10

节点 4 为头节点，先用 process 函数处理左子树，就将左子树转换成了有序双向链表，同时返回尾节点，记为 leftE；再用 process 函数处理右子树，就将右子树转换成了有序双向链表，同时返回尾节点，记为 rightE，如图 2-11 所示。

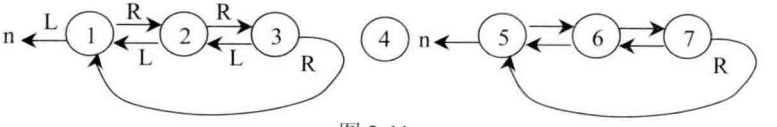

图 2-11

接下来，把节点 3（左子树 process 处理后的返回节点）的 right 指针连向节点 4，节点 4 的 left 指针连向节点 3，节点 4 的 right 指针连向节点 5（右子树 process 处理后的返回节点为节点 7，通过节点 7 的 right 指针可以找到节点 5），节点 5 的 left 指针连向节点 4，就完成了整个棵树向有序双向链表的转换。最后根据 process 函数的要求，把节点 7（右子树 process 处理后的返回节点）的 right 指针连向节点 1（左子树 process 处理后的返回节点为节点 3，通过节点 3 的 right 指针可以找到节点 1），然后返回节点 7 即可，如图 2-12 所示。

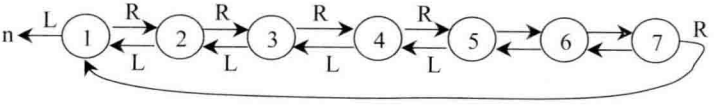

图 2-12

一开始时把整棵树的头节点作为参数传进 process 函数，然后每棵子树都会经历递归函数 process 的过程，具体过程请参看如下代码中的 process 方法。

为什么要将有序双向链表的尾节点连接头节点之后再返回尾节点呢？因为用这种方式可以快速找到双向链表的头尾两端，从而省去了通过遍历过程才能找到两端的麻烦。

2．通过 process 过程得到的双向链表是尾节点的 right 指针连向头节点的结构。所以，最终需要将尾节点的 right 指针设置为 null 来让双向链表变成正常的样子。

方法二的具体实现请参看如下代码中的 convert2 方法。

```
public Node convert2(Node head) {
        if (head == null) {
                return null;
        }
        Node last = process(head);
        head = last.right;
        last.right = null;
        return head;
}

public Node process(Node head) {
        if (head == null) {
                return null;
        }
        Node leftE = process(head.left); // 左边结束
        Node rightE = process(head.right); // 右边结束
        Node leftS = leftE != null ? leftE.right : null; // 左边开始
        Node rightS = rightE != null ? rightE.right : null; // 右边开始
        if (leftE != null && rightE != null) {
                leftE.right = head;
                head.left = leftE;
                head.right = rightS;
                rightS.left = head;
                rightE.right = leftS;
                return rightE;
        } else if (leftE != null) {
                leftE.right = head;
                head.left = leftE;
                head.right = leftS;
                return head;
        } else if (rightE != null) {
                head.right = rightS;
                rightS.left = head;
                rightE.right = head;
                return rightE;
        } else {
                head.right = head;
                return head;
        }
```

```
}
```

关于方法二中时间复杂度与空间复杂度的解释，可以用 process 递归函数发生的次数来估算时间复杂度，process 会处理所有的子树，子树的数量就是二叉树节点的个数。所以时间复杂度为 $O(N)$，process 递归函数最多占用二叉树高度为 h 的栈空间，所以额外空间复杂度为 $O(h)$。

【扩展】

相信读者已经注意到，本题在复杂度方面能够达到的程度完全取决于二叉树遍历的实现，如果一个二叉树遍历的实现在时间和空间复杂度上足够好，那么本题就可以做到在时间复杂度和空间复杂度上同样好。如果二叉树的节点数为 N，有没有时间复杂度为 $O(N)$、额外空间复杂度为 $O(1)$的遍历实现呢？如果有这样的实现，那本题也一定有时间复杂度为 $O(N)$、额外空间复杂度为 $O(1)$的方法。既不用栈，也不用递归函数，只用有限的几个变量就可以实现，这样的遍历实现是有的。欢迎有兴趣的读者阅读本书“遍历二叉树的神级方法”问题，然后结合神级的遍历方法再重新实现这道题。

单链表的选择排序

【题目】

给定一个无序单链表的头节点 head，实现单链表的选择排序。

要求：额外空间复杂度为 $O(1)$。

【难度】

士 ★☆☆☆

【解答】

既然要求额外空间复杂度为 $O(1)$，就不能把链表装进数组等容器中排序，排好序之后再重新连接，而是要求面试者在原链表上利用有限几个变量完成选择排序的过程。选择排序是从未排序的部分中找到最小值，然后放在排好序部分的尾部，逐渐将未排序的部分缩小，最后全部变成排好序的部分。本书实现的方法模拟了这个过程。

1．开始时默认整个链表都是未排序的部分，对于找到的第一个最小值节点，肯定是整