题目：

题目描述 | 通过 | 笔记 | 题解 | 提交记录

300. 最长递增子序列

已解答

中等 相关标签 相关企业

给你一个整数数组 `nums` ，找到其中最长严格递增子序列的长度。

子序列 是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，`[3,6,2,7]` 是数组 `[0,3,1,6,2,2,7]` 的子序列。

示例 1：

> **输入：** nums = [10,9,2,5,3,7,101,18]
> **输出：** 4
> **解释：** 最长递增子序列是 [2,3,7,101]，因此长度为 4 。

示例 2：

> **输入：** nums = [0,1,0,3,2,3]
> **输出：** 4

示例 3：

> **输入：** nums = [7,7,7,7,7,7,7]
> **输出：** 1

提示：

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

3.7K 1.6K

解题思路一：

数组+二分：使用一个等长数组来存储各个长度的子序列的右边的最小结尾值， 如lenthValue[i] 表示长度为i+1的递增子序列的最小结尾值，相同长度下有更小值就更新，使用 二分查找找到插入位置或替换位置，如果当前位置是最长子序列的右边界，那么当前最长子序列长度加一，直到结束过后，最后的maxlenth就是最长自增子序列的长度

```
#include <iostream>
#include <vector>
using namespace std;

class Solution {
```

```
public:
    int lengthOfLIS(vector<int>& nums) {
        if (nums.empty()) return 0; // 如果数组为空，直接返回0，因为最长递增子序列长度为0
        int len = nums.size();
        // lenthValue[i] 表示长度为i+1的递增子序列的最小结尾值
        vector<int> lenthValue(len, 0);
        int maxlenth = 0;

        for (int i = 0; i < len; ++i) {
            int left = 0, right = maxlenth;
            // 二分查找找到插入位置或替换位置
            while (left < right) {
                int mid = left + (right - left) / 2;
                if (lenthValue[mid] < nums[i])
                    left = mid + 1;
                else
                    right = mid;
            }
            lenthValue[left] = nums[i];
            if (left == maxlenth)
                ++maxlenth; // 如果当前位置是最长长度的右边界，则递增子序列长度加一
        }

        return maxlenth; // 返回最长递增子序列的长度
    }
};


int main()
{
    vector<int>test= { 4,10,4,3,8,9 };
    Solution s;
    int answer = s.lengthOfLIS(test);
    cout << answer;
    return 0;
}
```

解题思路二：

动态规划：我们把数组以长度拆分，如果当前下标元素大于之前下标的元素，当前长度下的数组一定是在之间长度的数组的最大值+1，否则当前下标的元素是dp[i]=1，也就是说大于时，状态转移方程是dp[i]=max(dp[i],dp[j]+1)，在循环中设置dp[i]初值为1，因为第一个数的前面没有子序列

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = (int)nums.size();
        if (n == 0) {
            return 0;
        }
        //初始化
        vector<int> dp(n, 0);
        for (int i = 0; i < n; ++i) {
            //预防前面没有子序列，或者直接i=1,然后dp[i]=max(1,dp[j]+1)
            dp[i] = 1;
            for (int j = 0; j < i; ++j) {
```

```
                if (nums[j] < nums[i]) {
                    dp[i] = max(dp[i], dp[j] + 1);
                }
            }
        }
        //返回结果取dp的最大值或者直接返回dp[n-1]
        return *max_element(dp.begin(), dp.end());
    }
};
```

。

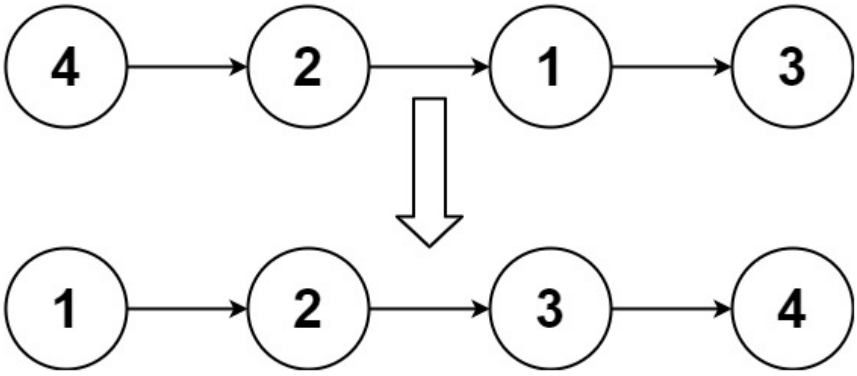
题目二：

148. 排序链表

中等 相关标签 相关企业 A文

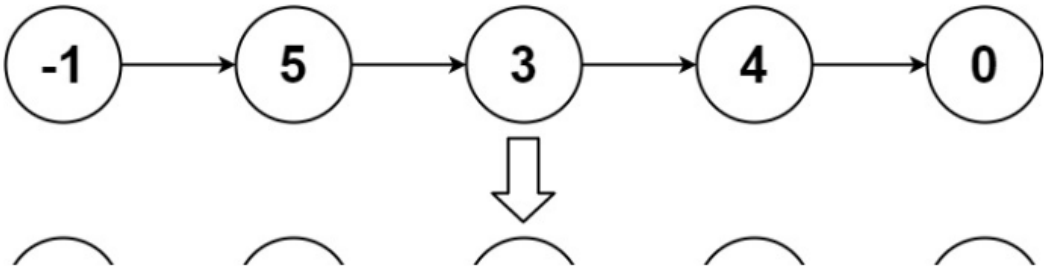
给你链表的头结点 `head` ，请将其按 **升序** 排列并返回 **排序后的链表** 。

示例 1：




```
输入：head = [4,2,1,3]
输出：[1,2,3,4]
```

示例 2：



解题思路一：

冒泡排序，每次比较完整个链表，如果有不等的那就换（题目要求升序，所以这里是大于交换，小于不处理），并标记变量表示当前指针的元素已经交换过，然后走到下一个指向开始循环比交，直到所有的变量都被标记，结束交换，链表排序完成

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        bool swapped = true;

        // 冒泡排序实现
        while (swapped) {
            swapped = false;
            ListNode* list1 = head;
//结束条件是所以元素比较完毕
            while (list1->next != nullptr) {
                if (list1->val > list1->next->val) {
                    swap(list1->val, list1->next->val);
                    swapped = true;
                }
                list1 = list1->next;
            }
        }

        return head;
    }
};
```

解题思路二：

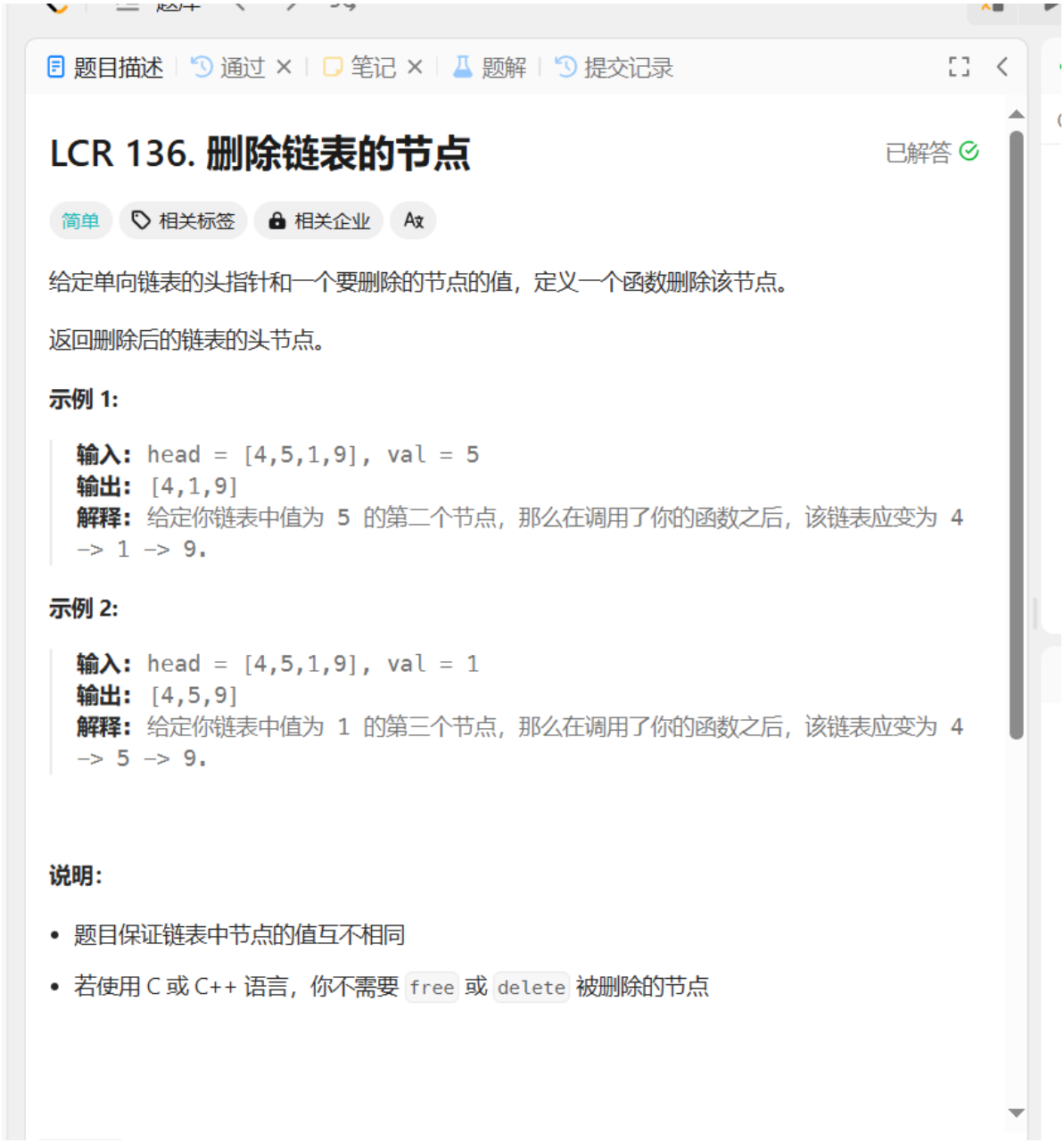
归并排序：

使用双指针，当快指针走完时，慢指针刚好走一半，然后将链表分成两半（参考数组的归并排序），这个时候使用一个指针mid来接取slow的next，再把next置为空指针，则链表就分为了两半，再分为left和right对链表进行递归处理，到最后剩下一两个元素的时候开始自底向上归并链表

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        // 如果链表为空或只有一个节点，直接返回
        if (!head || !head->next) return head;

        // 使用快慢指针找到链表的中点
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // 将链表从中点断开，分成两个链表
        ListNode* mid = slow->next;
        slow->next = nullptr;

        // 递归排序两个子链表
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);

        // 合并两个已排序的子链表
```

```
        return merge(left, right);
    }

private:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        // 创建一个虚拟头节点
        ListNode dummy(0);
        ListNode* tail = &dummy;

        // 合并两个有序链表
        while (l1 && l2) {
            if (l1->val < l2->val) {
                tail->next = l1;
                l1 = l1->next;
            }
            else {
                tail->next = l2;
                l2 = l2->next;
            }
            tail = tail->next;
        }

        // 将剩余的节点接到合并后的链表上
        tail->next = l1 ? l1 : l2;

        return dummy.next;
    }
};
```

题目三：



题目描述 | 通过 × | 笔记 × | 题解 | 提交记录

LCR 136. 删除链表的节点

已解答

简单 相关标签 相关企业

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

示例 1:

> **输入:** head = [4,5,1,9], val = 5
> **输出:** [4,1,9]
> **解释:** 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.

示例 2:

> **输入:** head = [4,5,1,9], val = 1
> **输出:** [4,5,9]
> **解释:** 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

说明:

- 题目保证链表中节点的值互不相同
- 若使用 C 或 C++ 语言，你不需要 `free` 或 `delete` 被删除的节点

解题思路一：

单指针：使用一个单指针遍历，如果到指针遍历结束之前有对应的值，当前指针指向next的next

```
class Solution {
public:
    ListNode* deleteNode(ListNode* head, int val) {
        //特殊情况
        if(head->val == val) {
            return head->next;
        }

        ListNode* pre = head;
        while ((pre->next != nullptr) && (pre->next->val != val)){
            pre = pre->next;
        }

        if(pre->next != nullptr) {
            pre->next = pre->next->next;
```

```
        }
        return head;
    }
};
```

解题思路二：

递归：

把链表视为一个个长度不等的小链表构成，即一个长度为5的链表可以由一个长度为4和长度为1的链表拼接而成，以此类推，不断递归，最后链表就变成了一个一个由长度为递减的小链表构成，如果当前链表的值等于删除元素，那么返回删除元素的下一个节点的节点，即相当于链表长度在这个过程中减一

```
class Solution {
public:
    ListNode* deleteNode(ListNode* head, int val) {
        if(nullptr == head) {
            return head;
        }

        head->next = deleteNode(head->next, val);
        return head->val == val ? head->next : head;
    }
};
```