算法设计与应用基础：作业 1

1. Show that

$$log(n!) = \Theta(nlogn)$$

(*Hint*: To show an upper bound, compare $n!$ with n^n. To show a lower bound, compare it with $(n/2)^{n/2}$.)

证明：

因为 log(n!) = log(1) + log(2) + ... + log(n-1) + log(n)

<= log(n) + log(n) + ... + log(n)= n*log(n)

所以 log(n!) = O(n log n)

因为 log(n!) = log(1) + ... + log(n/2) + ... + log(n)

>= log(n/2) + ... + log(n)

>= log(n/2) + ... + log(n/2)

= n/2 * log(n/2)

= n/2 * log(n) - n/2

>= n/4 * log(n)

所以 log(n!) = Ω(*n* log *n*)

所以 log(n!) = Θ(*n* log *n*)

2. Compute gcd(210, 588) two different ways: by finding the factorization of each number, and by using Euclid’ s algorithm.

方法一：因式分解

210 = 2 * 3 * 5 * 7 = 42 * 5

588 = 2 * 2 * 3 * 7 * 7 = 42 * 14

所以 gcd(210,588) = 42

方法二：欧几里得算法

gcd(210,588) = gcd(210,588 mod 210) = gcd(210,168) = gcd(168,42) = gcd(42,168 mod 42) = gcd(42,0) = 42

3. In the RSA cryptosystem, Alice’ s public key (N, e) is available to everyone. Suppose that her private key d is compromised and becomes known to Eve. Show that if $e = 3$ (a common choice) then Eve can efficiently factor N.

假设 N = xy，M = (x-1)(y-1)，那么 ed = 1 mod M，由 e = 3，3d = kM + 1，即 k = (3d - 1) / M。因为 0 < d < M，于是 k = 1,2。用 k 的值试探可以求得 M，由 N，M 可以解得 x，y。

4. Length of Longest Fibonacci Subsequence

A sequence $X_1, X_2, ..., X_n$ is fibonacci-like if:

- $n \geq 3$
- $X_i + X_{i+1} = X_{i+2}$, for all $i + 2 \leq n$

Given a **strictly increasing** array A of positive integers forming a sequence, find the **length** of the longest fibonacci-like subsequence of A. If one does not exist, return 0.

(Recall that a subsequence is derived from another sequence A by deleting any number of elements (including none) from A, without changing the order of the remaining elements. For example, $[3, 5, 8]$ is a subsequence of $[3, 4, 5, 6, 7, 8]$.)

Example:

Input : $[1, 2, 3, 4, 5, 6, 7, 8]$

Output : 5

Explanation: The longest subsequence that is fibonacci-like: $[1, 2, 3, 5, 8]$.

【算法思路】

dp[i][j]：表示以 arr[i],arr[j]结尾的斐波那契数列的最大长度

考虑在 arr[i]之前有某个数字 arr[k]，应该满足 arr[k] + arr[i] == arr[j]，可以写出来状态转移方程

dp[i][j] = max(dp[k][i] + 1) 其中 arr[k] + arr[i] == arr[j]

从而可以写出代码

【复杂度分析】

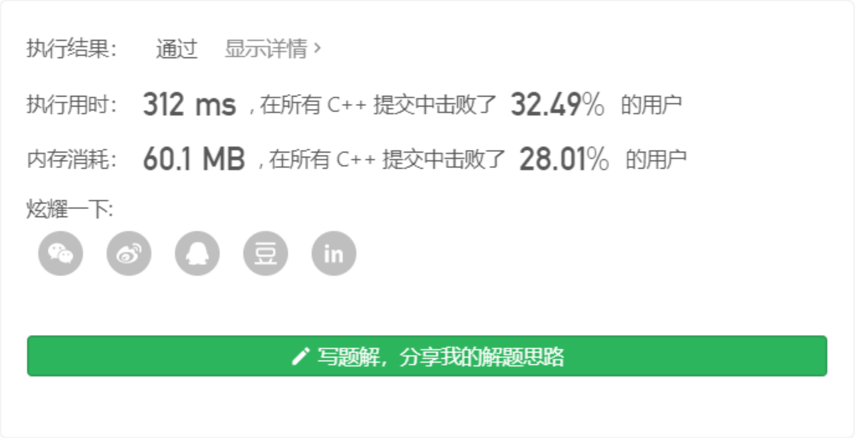
时间复杂度：O(nˆ2)，n 是 arr 的长度。

空间复杂度：O(n log m)，其中 m 是 arr 中最大的元素。

【代码】

```
class Solution {
public:
    int lenLongestFibSubseq(vector<int>& arr) {
        int n = arr.size();
        if (n < 3) return 0;
        int ans = 0;
        unordered_map<int, int> mp;
        vector<vector<int>> dp(n, vector<int>(n, 0));
        for (int i = 0; i < n; ++i)
            mp[arr[i]] = i;
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                dp[i][j] = 2;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                int d = arr[j] - arr[i];
                if (mp.count(d)) {
                    int index = mp[d];
                    if (index < i)
                        dp[i][j] = max(dp[i][j], dp[index][i] + 1
);
                }
            ans = max(ans, dp[i][j]);
            }
        }
        return ans > 2 ? ans : 0;
    }
};
```

【Accepted 截图】

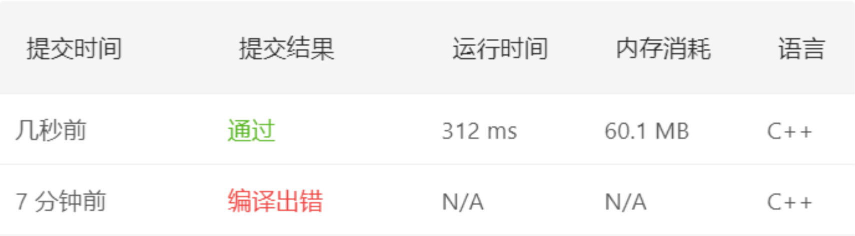


执行结果： 通过 显示详情 ›

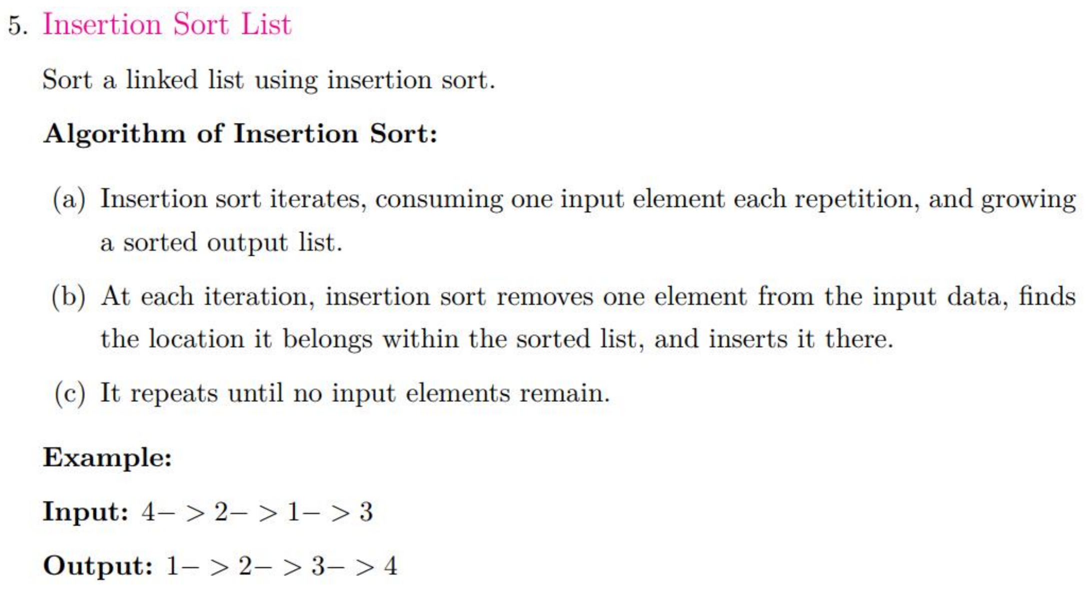
执行用时： 312 ms ，在所有 C++ 提交中击败了 32.49% 的用户

内存消耗： 60.1 MB ，在所有 C++ 提交中击败了 28.01% 的用户

炫耀一下:



提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	312 ms	60.1 MB	C++
7 分钟前	编译出错	N/A	N/A	C++



5. Insertion Sort List

Sort a linked list using insertion sort.

Algorithm of Insertion Sort:

(a) Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.

(b) At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.

(c) It repeats until no input elements remain.

Example:

Input: 4− > 2− > 1− > 3

Output: 1− > 2− > 3− > 4

【算法思路】

遍历原始列表，一次获取一个元素并将其插入排序列表中的适当位置。指针 last_sorted 代表已排序链表的最后一个元素。指针 first_unsorted 代表尚未插入排序链表的第一个元素。指针 current 搜索已排序链表来确定在哪插入 first _unsorted 指针。如果指针 first_unsorted 应在头指针 head 前插入,那么就插入在链表头结点之前。

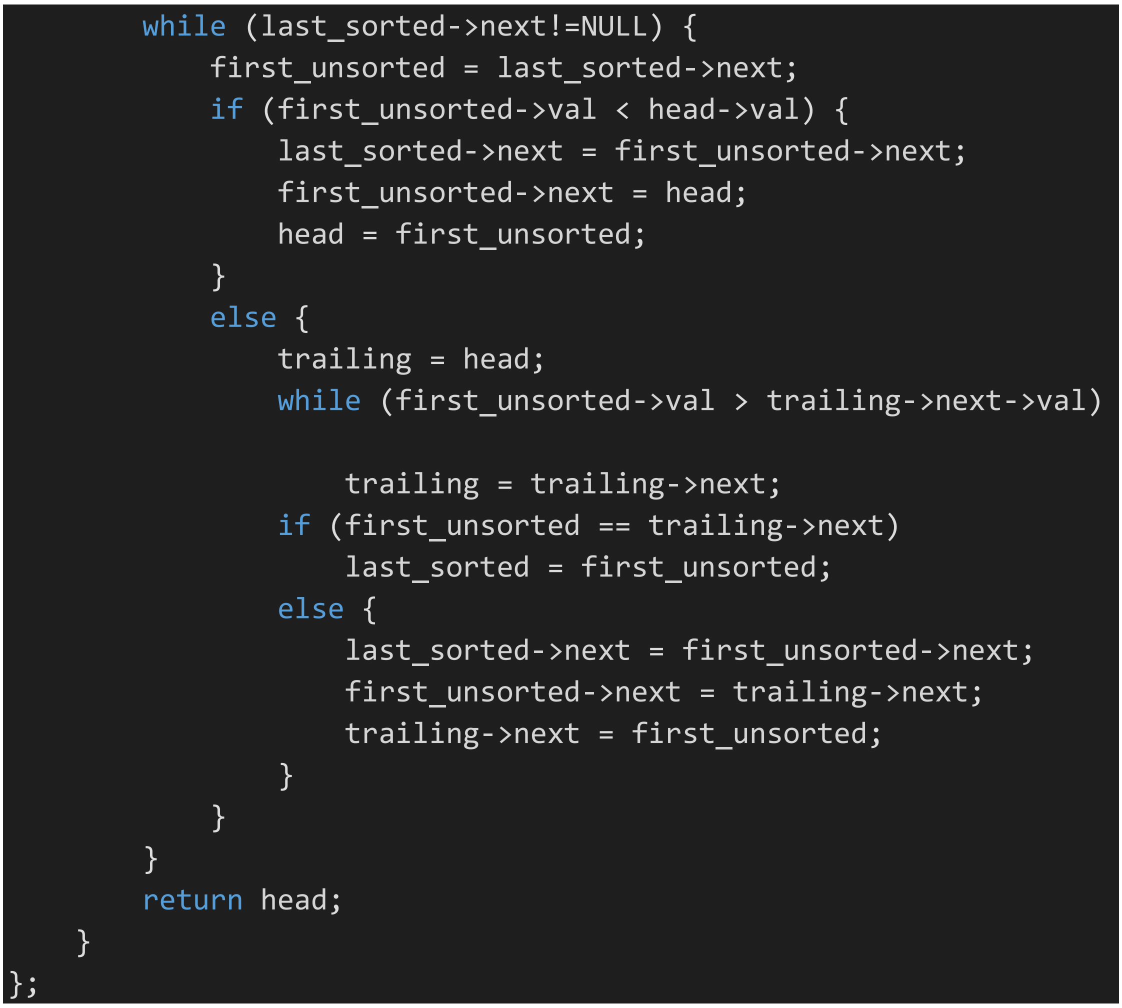
【复杂度分析】

时间复杂度：O(nˆ2)，其中 n 是链表的长度。

空间复杂度：O(1)。

【代码】

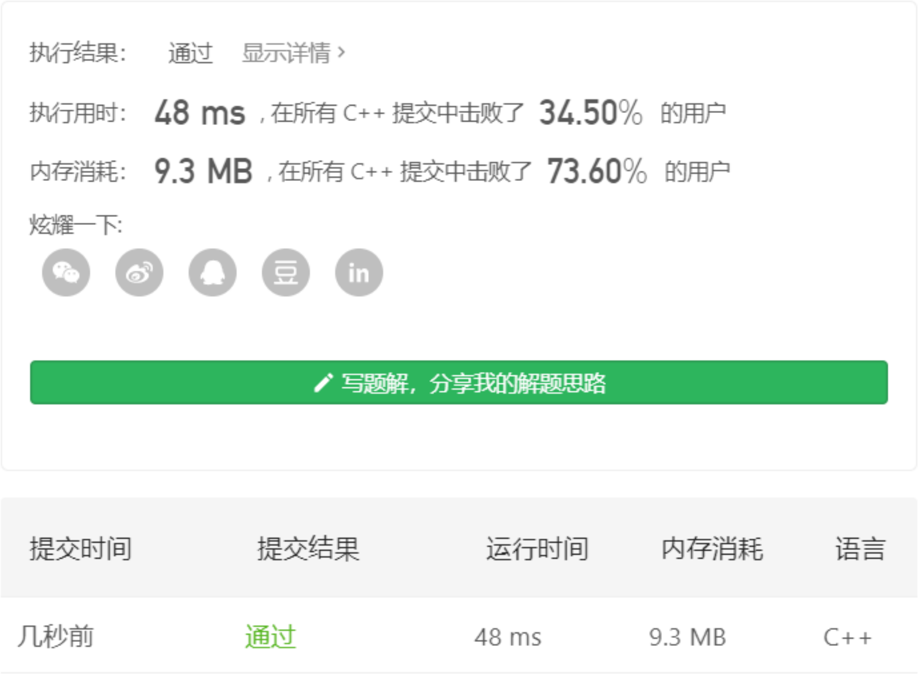
```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        ListNode *first_unsorted,*last_sorted,*trailing;
        if (head == NULL) return NULL;
        last_sorted = head;
```



```
        while (last_sorted->next!=NULL) {
            first_unsorted = last_sorted->next;
            if (first_unsorted->val < head->val) {
                last_sorted->next = first_unsorted->next;
                first_unsorted->next = head;
                head = first_unsorted;
            }
            else {
                trailing = head;
                while (first_unsorted->val > trailing->next->val)

                    trailing = trailing->next;
                if (first_unsorted == trailing->next)
                    last_sorted = first_unsorted;
                else {
                    last_sorted->next = first_unsorted->next;
                    first_unsorted->next = trailing->next;
                    trailing->next = first_unsorted;
                }
            }
        }
        return head;
    }
};
```

【Accepted 截图】



执行结果：　通过　显示详情 ›

执行用时：　48 ms ，在所有 C++ 提交中击败了 34.50% 的用户

内存消耗：　9.3 MB ，在所有 C++ 提交中击败了 73.60% 的用户

炫耀一下:

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	48 ms	9.3 MB	C++

6. Merge k Sorted Lists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

Input:

[

1− > 4− > 5,

1− > 3− > 4,

2− > 6

]

Output: 1− > 1− > 2− > 3− > 4− > 4− > 5− > 6

【算法思路】

将 n 个链表配对并将同一对中的链表合并；

第一轮合并以后，n 个链表被合并成了 k/2 个链表，平均长度为 2n/k，然后是 k/4 个链表，k/8 个链表等等。

重复这一过程，直到我们得到了最终的有序链表。

【复杂度分析】

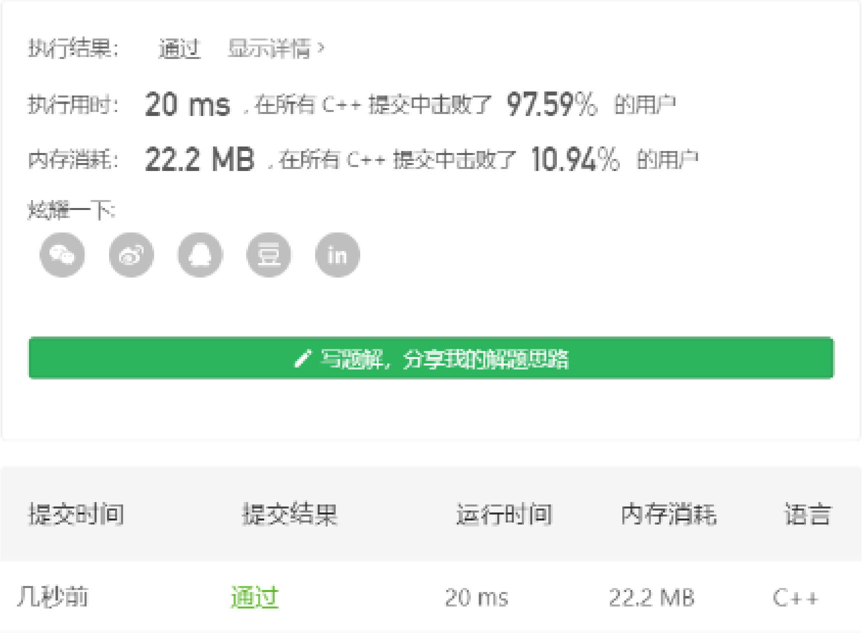
时间复杂度：O(kn log n)，n 是 lists 的元素个数，k 是 lists 中一个元素长度

空间复杂度：O(log n)

【代码】

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwo(ListNode* a, ListNode* b) {
        if (a == NULL) return b;
        if (b == NULL) return a;
        ListNode* head = new ListNode(0), *c = head;
        while (a && b)
            if(a->val < b->val) {
                c->next = a;
                c = a;
                a = a->next;
            }
            else {
                c->next = b;
                c = b;
                b = b->next;
            }
        c->next = (a == NULL) ? b : a;
        return head->next;
    }
    ListNode* merge(vector<ListNode*>& lists, int l, int r) {
        if (l == r) return lists[l];
        if (l > r) return NULL;
        int mid = (l + r) / 2;
        return mergeTwo(merge(lists, l, mid), merge(lists, mid +
1, r));
    }
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        int n = lists.size();
        if (n == 0) return NULL;
        return merge(lists, 0, n - 1);
    }
};
```

【Accepted 截图】



执行结果： 通过 显示详情 ›

执行用时： 20 ms ，在所有 C++ 提交中击败了 97.59% 的用户

内存消耗： 22.2 MB ，在所有 C++ 提交中击败了 10.94% 的用户

炫耀一下：

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	20 ms	22.2 MB	C++