1. $\because n! = n(n-1)\cdots 2\cdot 1 \le \underbrace{n\cdot n\cdots n\cdot n}_{n\uparrow n} = n^n$

$\therefore n! \le n^n$

$\log(n!) \le n\log n$

$\log(n!) = O(n\log n)$

$\because n! = n(n-1)\cdots 2\cdot 1 \ge n(n-1)\cdots(n/2)$

$\ge \underbrace{(n/2)\cdot(n/2)\cdots(n/2)}_{n/2\uparrow n/2}$

$= (n/2)^{n/2}$

$\therefore \log n! \ge (n/2)^{n/2}$

$\log(n!) \ge \frac{n}{2}\log\left(\frac{n}{2}\right)$

~~$\log((2n)!) \ge n\log n$~~

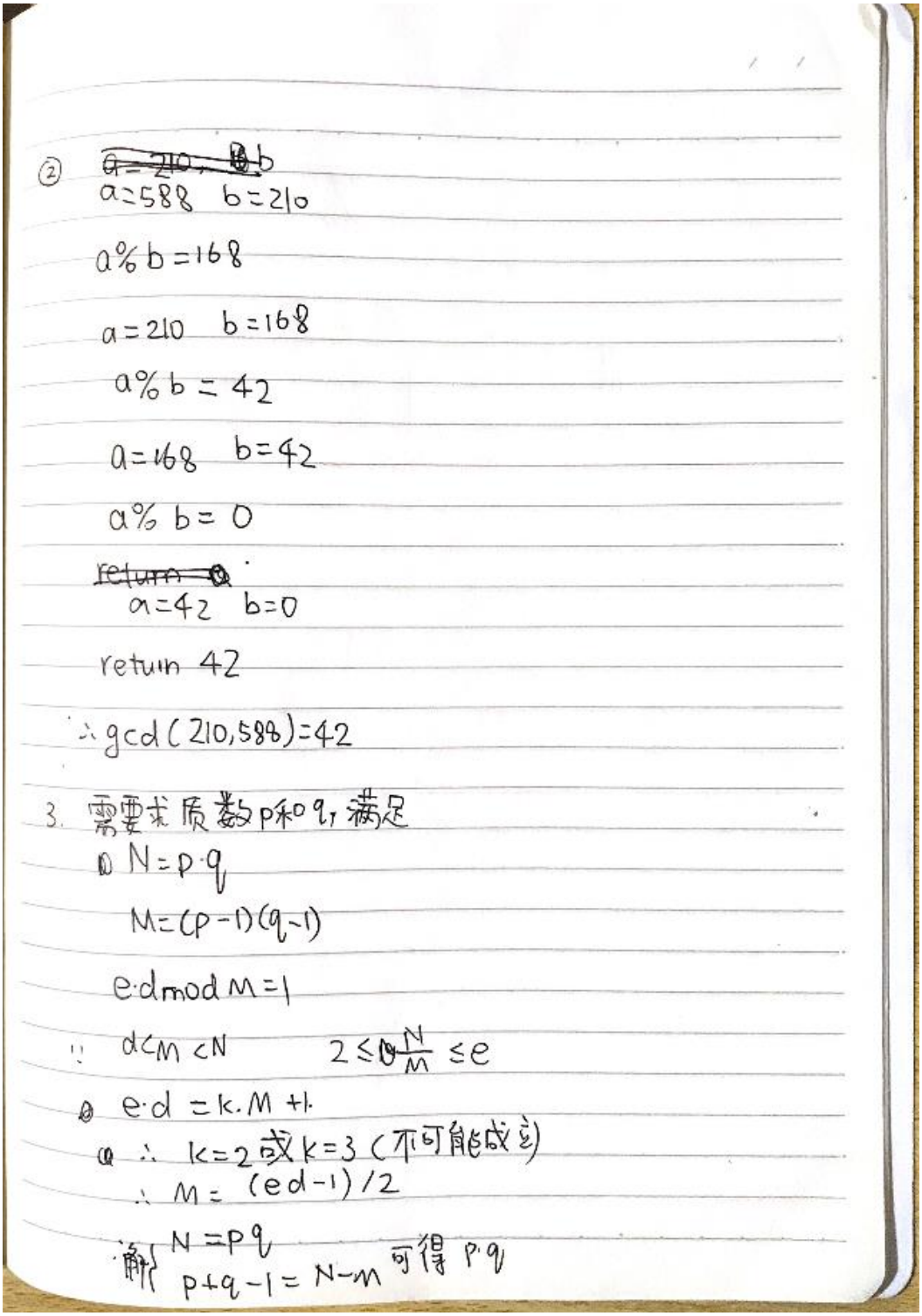
$\therefore \log(n!) = \Omega(n\log n)$

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

2. ① $210 = 7\times 5\times 3\times 2$

$588 = 7^2\times 3\times 2^2$

$\therefore \gcd(210, 588) = 7\times 3\times 2 = 42$



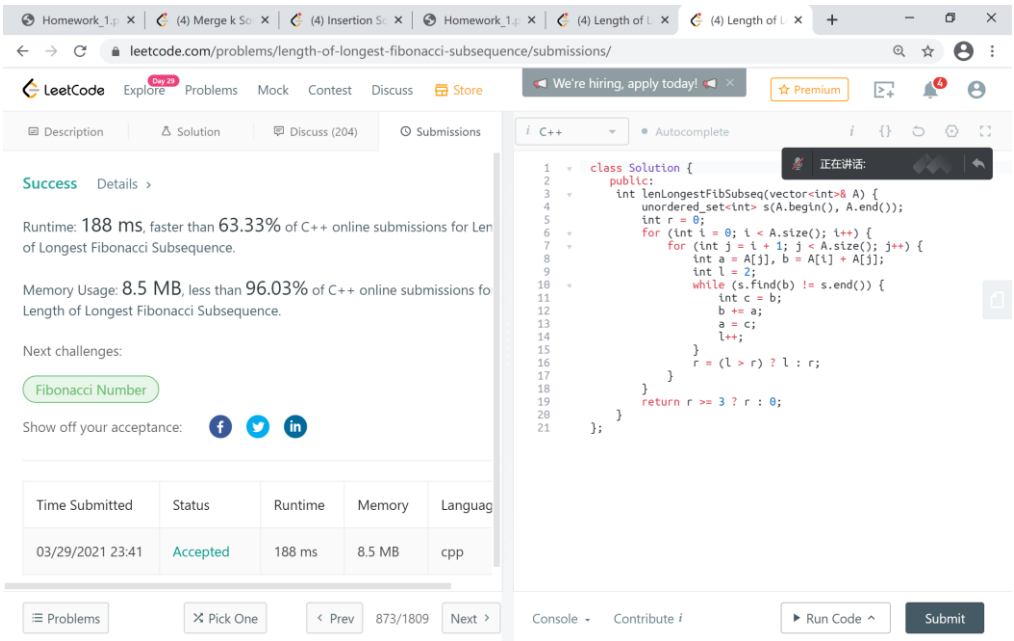
1. 思路: 使用 c++ unordered_set 的暴力解法
   直接搜索两数之和是否在数组中

   复杂度: $O(N^2 \log M)$

```
#include <unordered_set>
#include <vector>
```

```
using namespace std;

class Solution {
   public:
    int lenLongestFibSubseq(vector<int>& A) {
        unordered_set<int> s(A.begin(), A.end());
        int r = 0;
        for (int i = 0; i < A.size(); i++) {
            for (int j = i + 1; j < A.size(); j++) {
                int a = A[j], b = A[i] + A[j];
                int l = 2;
                while (s.find(b) != s.end()) {
                    int c = b;
                    b += a;
                    a = c;
                    l++;
                }
                r = (l > r) ? l : r;
            }
        }
        return r >= 3 ? r : 0;
    }
};
```

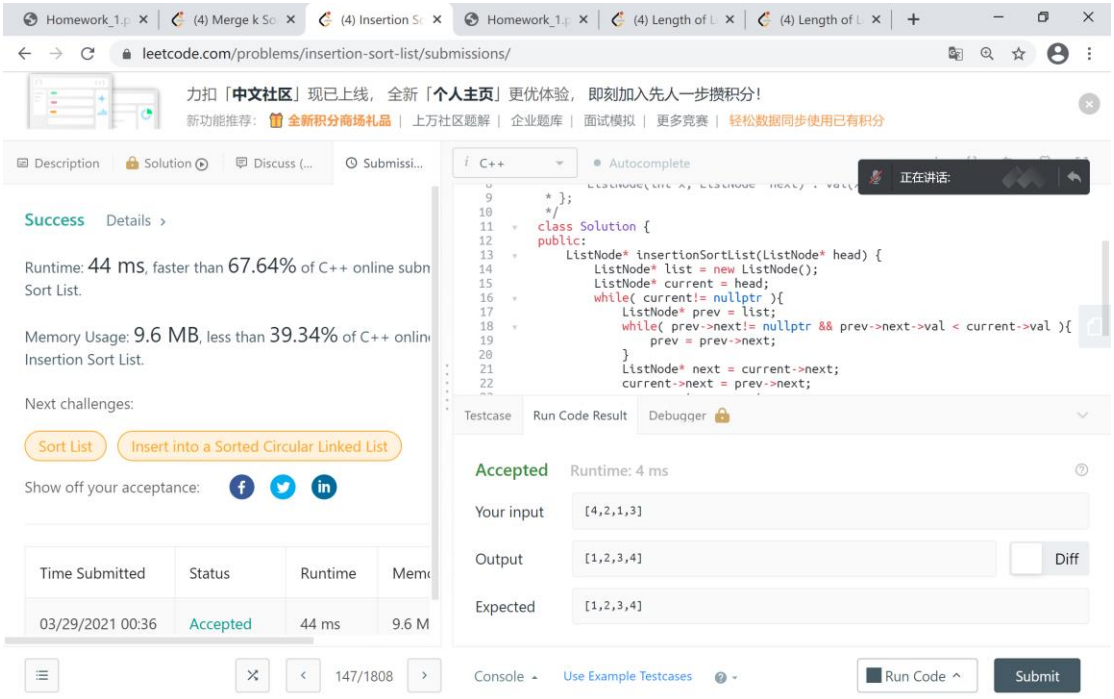


2. 思路：新建链表，对原链表每个元素插入新链表

   复杂度：$O(N^2)$

```
class Solution {
public:
```

```
    ListNode* insertionSortList(ListNode* head) {
        ListNode* list = new ListNode();
        ListNode* current = head;
        while( current!= nullptr ){
            ListNode* prev = list;
            while( prev->next!= nullptr && prev->next->val < current
->val ){
                prev = prev->next;
            }
            ListNode* next = current->next;
            current->next = prev->next;
            prev->next = current;
            current = next;
        }
    ListNode* l = list->next;
    delete list;
    return l;
    }
};
```



3. 思路：遍历各个链表，找到最小的元素，加在返回的链表后面

   复杂度：$O(kN)$

```
class Solution {
   public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.size() == 0) {
            return nullptr;
```

```
        }
        vector<ListNode*> _lists = lists;
        ListNode* current = new ListNode;
        ListNode* head = current;
        while (true) {
            int m, j = 0;
            while (j < _lists.size() && _lists[j] == nullptr) {
                j++;
            }
            if (j == _lists.size()) {
                break;
            }
            m = j;
            for (int i = j; i < _lists.size(); i++) {
                if (_lists[i] != nullptr && _lists[i]->val < _lists[
m]->val) {
                    m = i;
                }
            }
            ListNode* next = new ListNode(_lists[m]->val);
            current->next = next;
            current = current->next;
            _lists[m] = _lists[m]->next;
        }
        current = head->next;
        delete head;
        return current;
    }
};
```