

4、

**算法思路：**

采用动态规划的思想，设置一个二维数组dp[n][n](n=A.size())，其代表的意义是：假设存在A[i]，A[j]。那么dp[i][j]代表的就是以A[i]，A[j]为结尾的斐波那契数列的长度，例如：A={1，2，3，5，8}，那么dp[3][4]=5。同时，如果我们存在A[k]=A[j]-A[i]且k<i，显然存在dp[i][j]=dp[k][i]+1。所以我们用两层循环遍历一下所有的i和j的组合情况，保存最大值即可。

**复杂度分析：**

时间复杂度：O(n2)，空间复杂度：O(n2)，n为A的长度。

**代码：**

class Solution {

public:

    int lenLongestFibSubseq(vector<int>& A) {

        unordered\_map<int,int> hash;

        int n=A.size();

        if(n==0)

        {

            return 0;

        }

        for(int i=0;i<n;i++)

        {

            hash[A[i]]=i;

        }

        vector<vector<int>> dp(n,vector<int>(n,0));

        for(int i=0;i<n;i++)

        {

            for(int j=i+1;j<n;j++)

            {

                dp[i][j]=2;

            }

        }

        int temp=0;

        int answer=0;

        for(int i=0;i<n;i++)

        {

            for(int j=i+1;j<n;j++)

            {

                temp=A[j]-A[i];

                if(hash.count(temp))

                {

                    int temp1=hash[temp];

                    if(temp1<i)

                    {

                        dp[i][j]=dp[temp1][i]+1;

                        answer=max(answer,dp[i][j]);

                    }

                }

            }

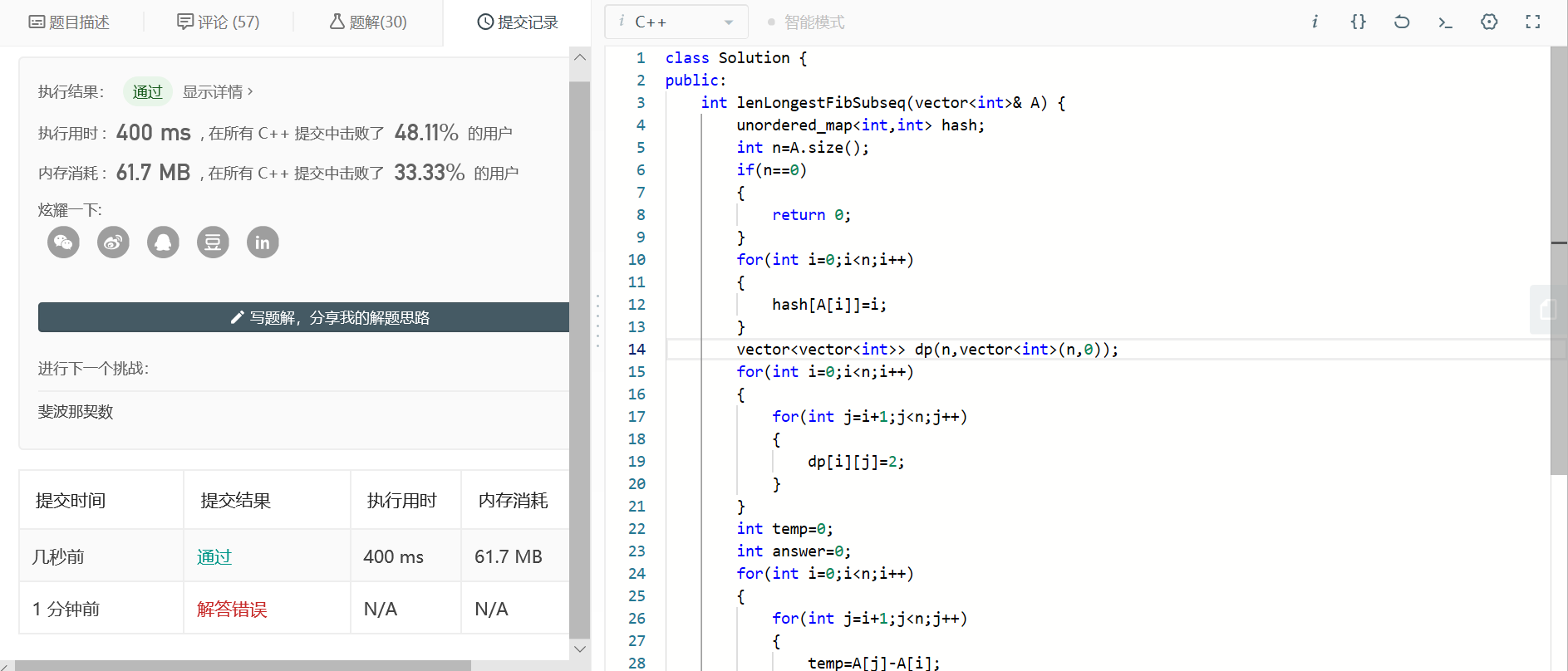
        }

        return answer;

    }

};

**截图：**

****

**5、**

**算法思路：**

解决本题有两种方法：第一种是比较投机取巧的，将链表中的元素放到数组中，然后对数组进行插入排序（或者其他排序），最后放回链表。不过本题旨在对链表进行插入排序的操作，故在此采用这种方法。

用3个指针：s，q，p。其中s是指向头指针的指针，q是已经排好序的部分，p是需要插入到合适位置的。我们对p进行操作，用一个temp指针从头遍历一下，找到p应该插入的位置，然后将p插入即可。最后返回s->next即可。

**复杂度分析：**

时间复杂度：O(nlogn)，空间复杂度：O(1)。n为链表长度。

**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) {

      if(head==NULL||head->next==NULL)

      {

          return head;

      }

      ListNode \*s,\*q,\*p;

      s=new ListNode(-100);

      s->next=head;

      q=head;

      p=q->next;

      while(p!=NULL)

      {

          if(p->val<q->val)

          {

              ListNode \*temp=s;

              while(p->val>temp->next->val)

              {

                  temp=temp->next;

              }

              q->next=p->next;

              p->next=temp->next;

              temp->next=p;

              p=q->next;

          }

          else

          {

              q=q->next;

              p=p->next;

          }

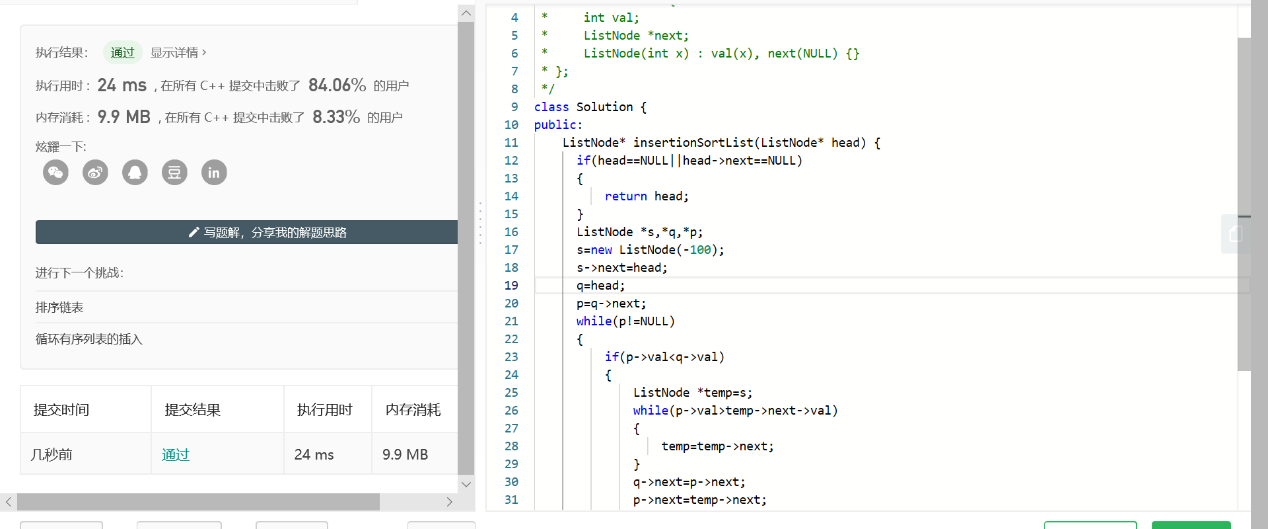
      }

      return s->next;

    }

};

**截图：**



6、

**算法思路：**

写出两个链表合并的算法，然后将lists里面的链表逐个合并即可。关键在于两个链表如何合并。

从两个链表头开始，每次比较两个值的大小，然后选择较小者加入新链表，同时较小者所在的原链表指向下一个next，直到有一个链表为空为止。最后将不为空的剩下的链表拼接到新链表的末尾即可。

**复杂度分析：**

时间复杂度：O(k2n)，空间复杂度：O(1)。k为lists.size()-1，n为lists里最长的链表的长度。

**代码：**

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode \*TwoLists(ListNode \*a,ListNode \*b)

    {

        ListNode \*head=new ListNode(0);

        ListNode \*p=head;

        ListNode \*pa=a,\*pb=b;

        while(pa!=NULL&&pb!=NULL)

        {

            if(pa->val<pb->val)

            {

                head->next=pa,pa=pa->next;

            }

            else

            {

                head->next=pb,pb=pb->next;

            }

            head=head->next;

        }

        head->next=pa?pa:pb;

        return p->next;

    }

    ListNode\* mergeKLists(vector<ListNode\*>& lists) {

        if(lists.size()==0)

        {

            return NULL;

        }

        if(lists.size()==1)

        {

            return lists[0];

        }

        int len=lists.size();

        ListNode \*p=TwoLists(lists[0],lists[1]);

        for(int i=2;i<len;i++)

        {

            p=TwoLists(p,lists[i]);

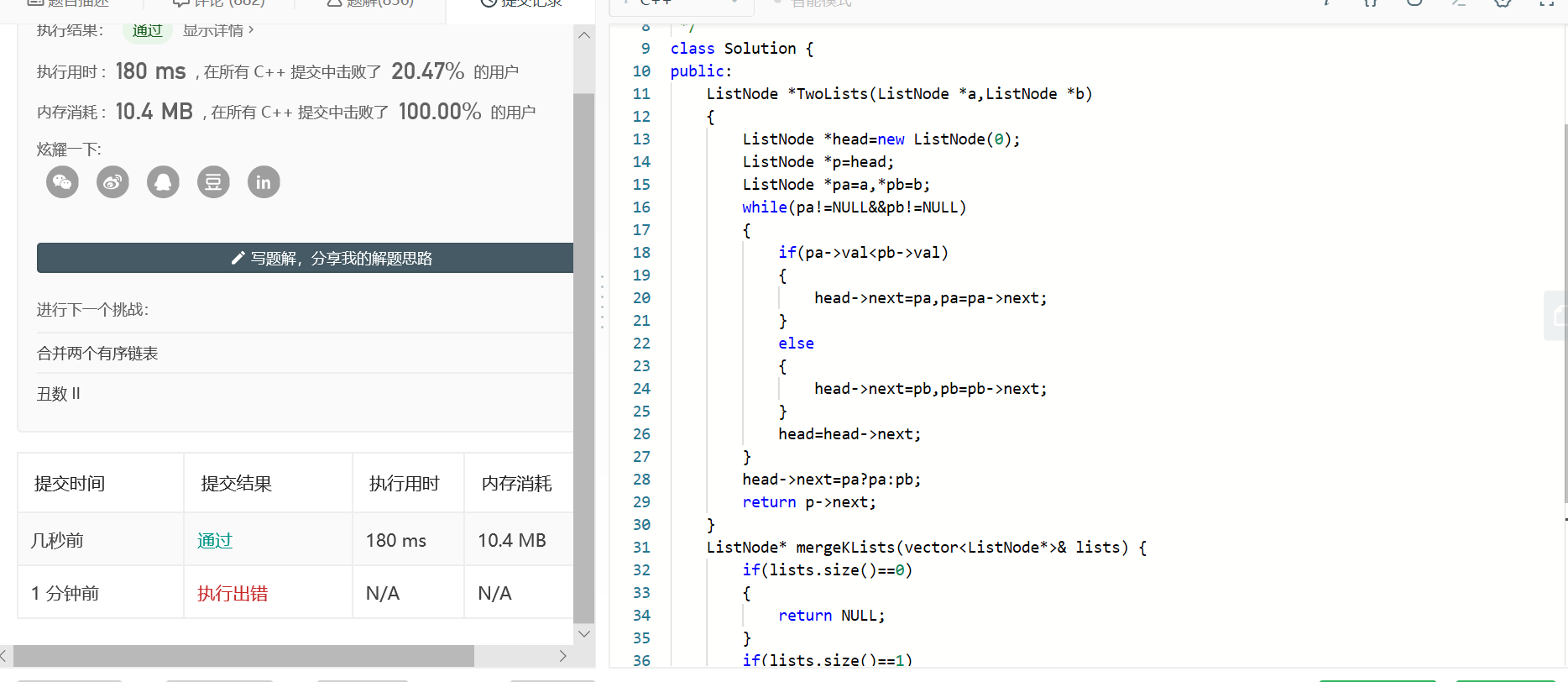
        }

        return p;

    }

};

**截图：**

****