**北京邮电大学软件学院**

**2022-2023学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称：** 实验项目五

**项目完成人：**

**姓名：** 王宇涵 **学号：** 2020211730

**指导教师：** 李朝晖

**日 期： 2023年 4 月 27 日**

1. **实验目的**

1、 深刻理解并掌握分治法、贪心法的设计思想；

2、 提高应用分治法、贪心法的技能；

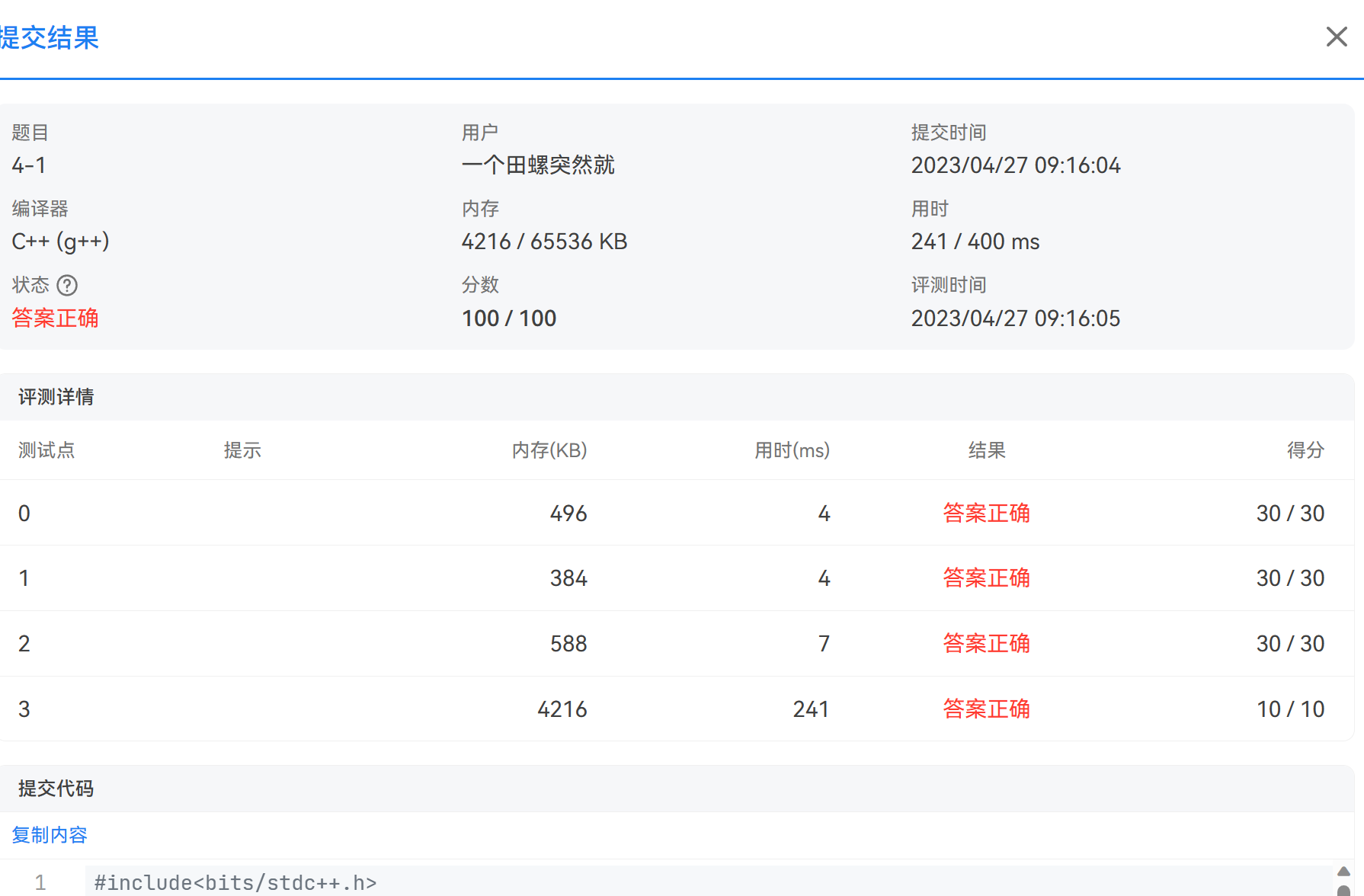
1. **实验内容**
2. (必做) PTA1,PTA2,PTA3
3. **实验环境**

Windows 10；Dev-C++

1. **实验过程及结果**

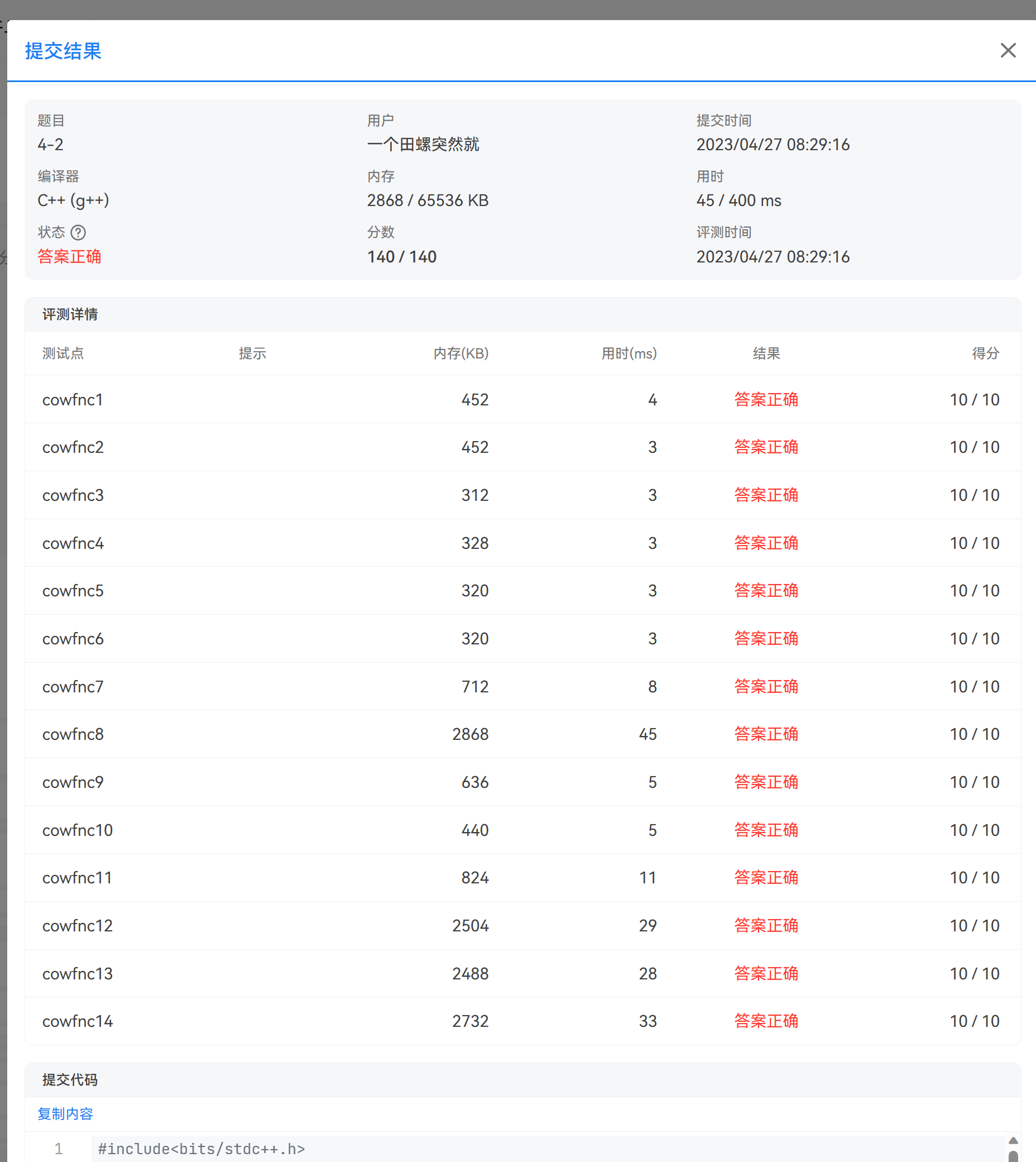


**PTA1**

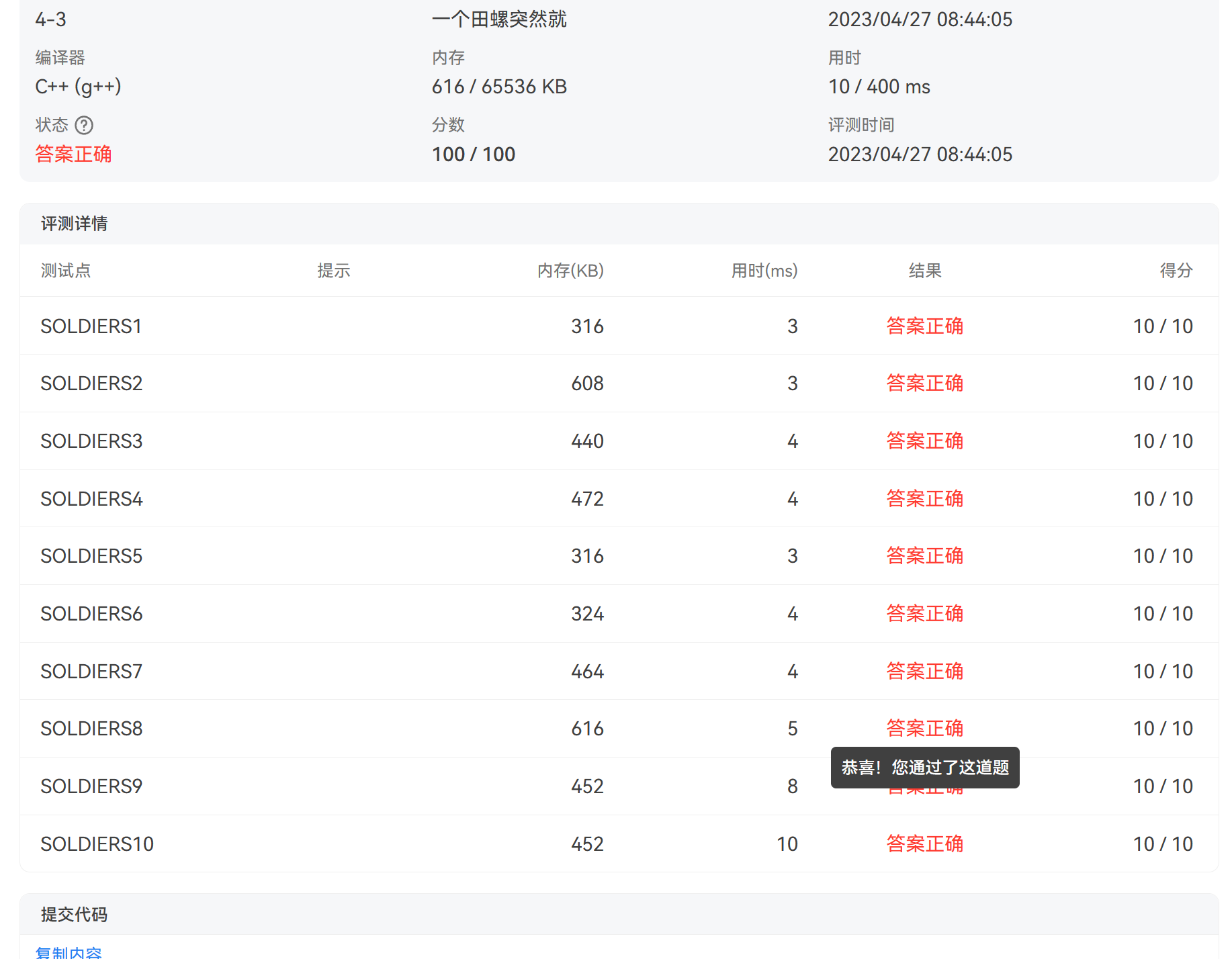


方法2：  


PTA2



**PTA-3**



1. **附录**

**项目1：第k小元素**

给定一个大小为*n*(1≤*n*≤1000000)且无序的整型数组，数组中可能存在相同元素，请找出该数组第*k*(1≤*k*≤*n*)小的元素，注意这里的第*k*小元素指的是按从小到大排序后的第*k*个位置上的元素。

### 输入格式:

每个输入文件为一个测试用例，每个文件的第一行给出两个正整数*n*和*k*，第二行给出*n*个整数，其间以空格分隔。

### 输出格式:

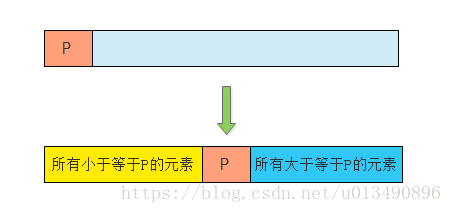
输出第*k*小元素的值。

【整体思路】

**方法一：**

**利用划分求第k小元素**

假设列表是以数组实现的，其元素索引从0开始，那么第k小的元素就是把此列表从小到大排序后，索引在k-1位置上的元素。

我们可以将给定的列表根据某个值p（例如列表的第一个元素）进行划分。一般来说，这是对列表元素的重新整理，使左边部分包含所有小于等于p的元素，紧接着是中轴（pivot）p本身，再接着是所有大于等于p的元素。如下图所示  


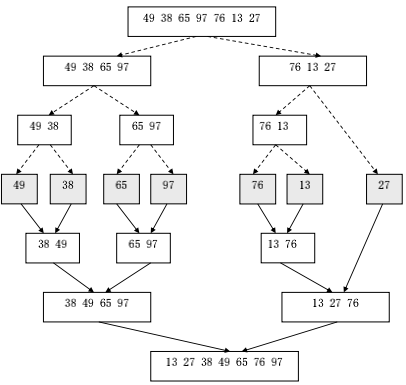
**方法2：**

比较直观的想法是使用堆数据结构来辅助得到最小的 k 个数。堆的性质是每次可以找出最大或最小的元素。我们可以使用一个大小为 k 的最大堆（大顶堆），将数组中的元素依次入堆，当堆的大小超过 k 时，便将多出的元素从堆顶弹出。

【算法分析设计】

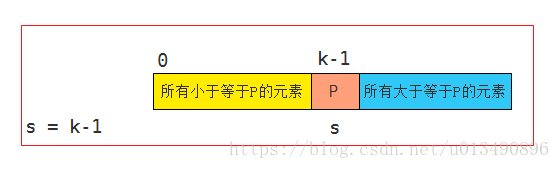
方法一：

* 根据下图可以很容易看出设计策略。

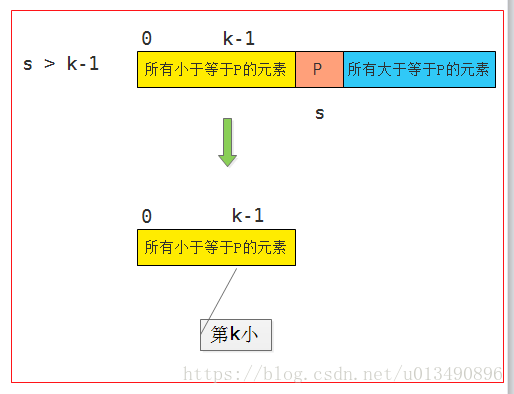


* 分：将待排序序列划分为两个长度相等的子序列。
* 治：分别对这两个子序列进行升序排列，得到两个有序子序列。
* 合：将两个有序子序列合并成一个有序序列。

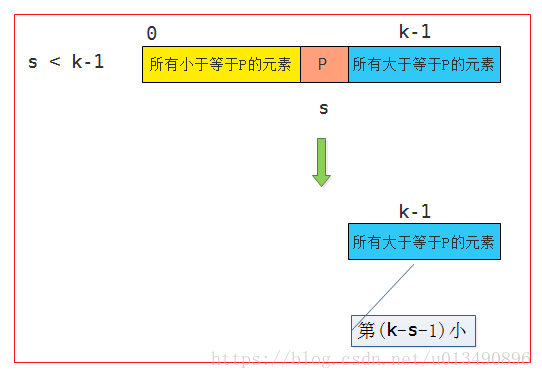
假设首次划分此列表，s是分割位置，也就是划分后中轴元素的索引。我们分3种情况进行讨论：

[1]. 当s=k-1 ,那么中轴p本身显然就是第k小的元素；

[2]. 如果s>k-1，那么整个列表的第k小元素就是左边部分的第k小元素；



[3]. 如果s<k-1，那么问题就转换为求右边部分的第(k-s-1)小元素；推导过程是这样的：本来是求第k小，通过划分，筛除了最前面的(s+1)个元素，所以只用求右边部分（蓝色）的第 k-(s+1)小。



可以看出，第2种情况和第3种情况虽然没有彻底解决问题，但是使问题的实例变小了。对于这个较小的实例可以用同样的方法来解决，即递归求解。

函数kthMin:

array: 传入的数组

Low: 数组左端

high: 数组右端

k: 相对于left的后移位置

方法2：

我们用一个大根堆实时维护数组的前 k 小值。首先将前 k 个数插入大根堆中，随后从第 k+1个数开始遍历，如果当前遍历到的数比大根堆的堆顶的数要小，就把堆顶的数弹出，再插入当前遍历到的数。最后将大根堆里的数存入数组返回即可。在下面的代码中，由于 C++ 语言中的堆（即优先队列）为大根堆，我们可以这么做。而 Python 语言中的堆为小根堆，因此我们要对数组中所有的数取其相反数，才能使用小根堆维护前 kk 小值。

算法的复杂度分析：

由于使用了一个大小为 k 的堆，空间复杂度为 O(k)；

入堆和出堆操作的时间复杂度均为 O(log k)，每个元素都需要进行一次入堆操作，故算法的时间复杂度为 O(nlogk)。

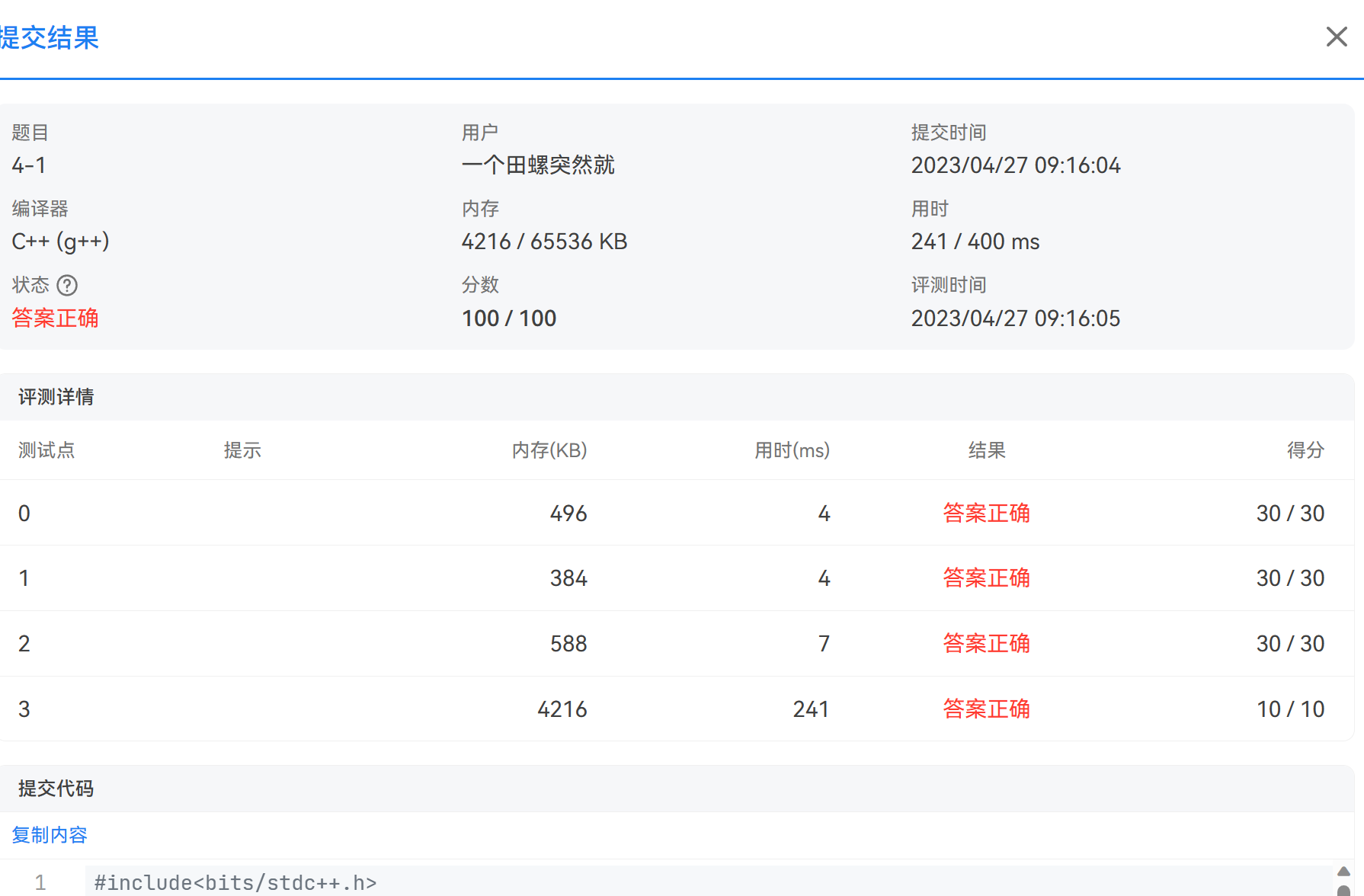
【核心代码及解释】

下面的函数时通过递归排序，采用类似于快速排序的方法进行排序，然后找出弟K个元素

int partition(int nums[],int low, int high)//划分
  
{
  
 int randin=low+rand()%(high-low+1);
  
 swap(nums[randin],nums[low]);
  
 int pivot=nums[low];
  
 // debug(pivot);
  
 int l=low,r=high;
  
 while(l<r)
  
 {
  
 while(l<r&&nums[r]>pivot)r--;
  
 // debug(r);
  
 // debug(l);
  
 if(l<r)
  
 {
  
 nums[l] = nums[r];
  
 l++;
  
 }
  
 while(l<r&&nums[l]<pivot)l++;
  
 if(l<r)
  
 {
  
 nums[r] = nums[l];
  
 r--;
  
 }
  
 // debug(l);
  
 // debug(nums[l]) ;
  
 }
  
 nums[r]=pivot;
  
 // debug(l);
  
 return l;
  
 }
  
  
void qfind(int a[],int left,int right,int k)
  
{
  
 int mid=partition(a,left,right);
  
 if(k-1==mid)//第k小数，数组下标为k-1
  
 cout<<a[k-1];
  
 else if(k-1<mid)//在左区接着找
  
 qfind(a,left,mid-1,k);
  
 else qfind(a,mid+1,right,k);//在右区接着找
  
}

**方法2：**  
int getLeastNumbers(int nums[], int k)
  
{
  
 //优先队列
  
  
 priority\_queue<int,vector<int>,less<int>> q;
  
 for(int i =0; i< k; i++)//前k个元素入队
  
 {
  
 q.push(nums[i]);
  
 }
  
 if(!k)return {};
  
 //将数组剩下的元素入队,则top就是第k个元素
  
 //如果使用大根堆，则需要n个
  
 for(int i = k;i < n ;i++)
  
 {
  
 int t = q.top();//取最顶上的元素
  
 if(nums[i] < t)
  
 {
  
 q.pop();
  
 q.push(nums[i]);
  
 }
  
 //q.top现在是前i个元素的第k个最大元素
  
 }
  
 return q.top();
  
 }

【实验结果】



方法2：  


**项目二：PTA-2子序列的平均值**

给定一个长度为*n*的非负序列*A*，请你找出一个长度不小于*L*的子段（子段是序列A中一些连续的元素构成的集合），使得子段中数值的平均值最大。最终输出这个最大的平均值。

### 输入格式:

第一行两个整数*n*,*L*(1<=*L*<=*n*<=100,000)

以下*n*行，每行一个非负整数，表示序列*A*中每个元素的值。

### 输出格式:

一个整数，**欲求的最大平均值乘以1000后的结果（注意不要四舍五入，直接输出）**。

### 输入样例:

10 6  
6   
4  
2  
10  
3  
8  
5  
9  
4  
1

### 输出样例:

6500

【整体思路】

1. 首先，在二分答案的过程中，将最小值赋为序列中的最小值，将最大值赋为序列中的最大值。接着进行二分答案，每次选取左右端点的中点mid，然后将mid作为平均值，将序列中每个数减去mid。然后，用前缀和的方式求出每个数的前缀和。这个操作的目的是将每个数减去mid，然后在求前缀和的过程中，实现O(n)的时间复杂度。
2. 在check函数中，用最小前缀和mins记录序列中最小的前缀和，然后用sum[i]-mins>=0的方式判断平均值是否可以更大。如果sum[i]-mins>=0，说明以i为结尾的子序列的平均值大于等于mid，即平均值可以更大。如果不满足条件，则说明平均值不够大，需要将右端点r设为mid。
3. 在二分答案的过程中，每次循环都会调用check函数进行判断，而check函数的时间复杂度为O(n)，所以整个算法的时间复杂度为O(nlogn)。

【算法分析设计】

给定一个序列 a 和一个区间长度 L，要求找到一个平均数 mid，使得序列中的某个区间的平均数大于等于 mid。我们可以通过二分答案的方式来解决这个问题：

* 先将平均数的搜索区间设为序列中最小值和最大值之间，即 [minn, maxx]。
* 然后每次取平均数 mid 作为当前猜测答案，将序列 a 中每个元素减去 mid 得到一个新序列 b，然后对 b 做前缀和，得到一个新序列 sum，其中 sum[i] 表示序列 b 的前缀和，即 b[0] + b[1] + ... + b[i-1]。
* 接下来对于每个右端点 i，我们需要找到左端点 j，使得区间 [j, i] 的平均数大于等于 mid。为了实现这一点，我们可以通过维护一个区间最小值的前缀最小值数组 mins，来实现在 $O(1)$ 时间内找到满足条件的左端点 j。具体地，我们维护一个变量 mins 表示目前为止 sum[0..i-L] 中的最小值，然后对于每个右端点 i，我们检查 sum[i]-mins 是否大于等于 0，如果是，则说明区间 [i-L+1, i] 的平均数大于等于 mid，此时可以更新答案，将搜索区间的左端点设为 mid，否则说明区间 [i-L+1, i] 的平均数小于 mid，此时将搜索区间的右端点设为 mid。直到左右端点的差小于 1e-5，说明答案已经找到，输出结果即可。

算法分析：

时间复杂度：由于二分的次数为

每次二分需要线性的时间 O(n) 来对序列求前缀和和维护前缀最小值，因此总时间复杂度为

空间复杂度：除了输入序列 a 和一些变量之外，算法需要 O(n) 的额外空间来存储前缀和数组 sum 和前缀最小值数组 mins。

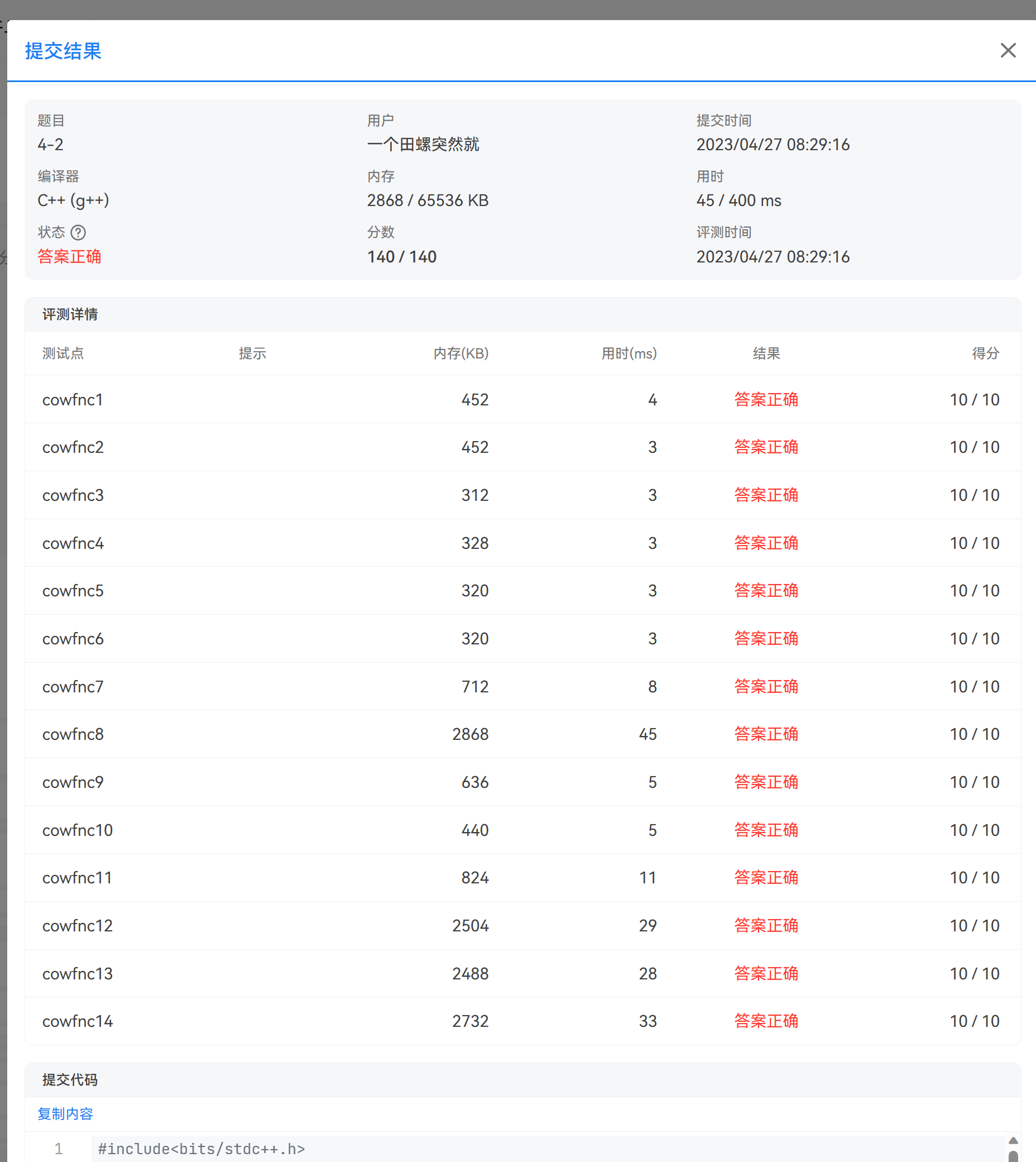
【核心代码及解释】

int check(double mid)
  
{
  
 double mins;
  
 for(int i = 1; i <= n; i ++ )
  
 {//求i到j的和应该是sum[j]-sum[i-1]
  
 b[i - 1] = a[i - 1] - mid;
  
 sum[i] = sum[i - 1] + b[i - 1];
  
 }
  
 //前缀和处理
  
 mins = 1e9;
  
 for(int i = L; i <= n; i ++ )
  
 {
  
  
 mins = min(mins, sum[i - L]);
  
 if(sum[i] - mins >= 0)
  
 {
  
 return 1;
  
 }
  
 }
  
 return 0;
  
}
  
int main()
  
{
  
 cin >> n >> L;
  
 double minn = 1e9, maxx = 0;
  
 for(int i = 0; i < n; i ++ )
  
 {
  
 cin>>a[i];
  
 minn = min(a[i], minn);
  
 maxx = max(a[i], maxx);
  
 }
  
  
 double l = minn;
  
 double r = maxx;
  
  
 while(r - l > 1e-5)
  
 {
  
 double mid = l+ (( r - l ) / 2);
  
 if(check(mid))
  
 { //满足条件说明平均数可以更大
  
 l = mid;
  
 }
  
 else
  
 {
  
 r = mid;
  
 }
  
 }
  
 l += 1e-5;
  
 printf("%d", int(l\*1000));
  
 return 0;
  
}

1. 首先，定义函数 check(mid) 用于判断给定的mid是否满足条件。函数首先将原始数组a中的每个元素减去mid，然后计算a数组的前缀和sum。接着，用变量mins记录sum[0]到sum[i-L]的最小值，并在计算sum[i] - mins，若结果大于等于0，说明a数组中以i结尾的长度为L的连续子段的平均值大于等于mid，即check(mid)返回1；否则返回0。
2. 在主函数中，首先输入原始数组a的长度n和子段长度L，并记录a数组中的最小值minn和最大值maxx。然后，设定查找区间[l,r]为[minn,maxx]，使用二分查找算法查找最长的连续子段的平均值大于等于mid，直到r - l < 1e-5为止。每次二分查找时，计算中点mid = l + ((r - l) / 2)，如果check(mid)返回1，说明平均值可以更大，则将区间左端点l更新为mid；否则，将区间右端点r更新为mid。最终，当l找到满足条件的值后，将其乘以1000取整，作为结果输出。

值得注意的是，为了防止精度误差，每次更新区间左端点时，将mid加上一个足够小的值（例如1e-5）作为新的左端点l。

**运行结果如下：**



**项目三：**

在一个划分成网格的操场上，n个士兵散乱地站在网格点上。网格点用整数坐标(x,y)表示。士兵们可以沿网格边往上、下、左、右移动一步，但在同一时刻任一网格点上只能有一名士兵。按照军官的命令，士兵们要整齐地列成一个水平队列，即排列成(x,y),(x+1,y),…,(x+n-1,y)。如何选择x和y的值才能使士兵们以最少的总移动步数排成一行。

编程计算使所有士兵排成一行需要的最少移动步数。

**题目引自POJ**

### 输入格式:

第1行是士兵数n，1≤n≤10000。接下来n行是士兵的初始位置，每行有2个整数x和y，-10000≤x，y≤10000。

### 输出格式:

一个数据，即士兵排成一行需要的最少移动步数。

### 输入样例:

5  
1 2  
2 2  
1 3  
3 -2  
3 3

### 输出样例:

8

【整体思路】

由于是求曼哈顿距离，所以可以将 x , y 分开考虑。\*\*

* y ：要将所有的 y\_i集中到某个 y\_0上。
* 可以通过微量法证明 y\_0一定在某个已知 y\_i上。再通过微量法证明 y\_0 一定是 y\_i的中位数。
* x ：要将 x\_1集中到某个 x\_0 。其他 x\_i依次相间 1 排下去。
* 先将 x排序，可以证明 x的顺序一定就是最终的序列的顺序（因为交叉位置的话解更差）。
* 因为定了序，所以有
* ，则可以将问题转化为
* x 就是与 y同样的问题了，求
* 的中位数 x\_0 就可以了。

【算法分析设计】

## **纵坐标处理**

我们先看纵坐标（其实是纵坐标比较简单）

* 我们假设士兵要移动到的目标纵坐标为m；

**士兵在垂直于x轴方向上的移动距离为：**

|y1-m|+|y2-m|+|y3-m|+……+|yn-1-m|+|yn-m|

那么m取什么值最小呢？显而易见，就是y1~yn序列的中位数，实现代码如下：

int rey;  
 sort(y+1,y+n+1);  
 if(!n%2) rey=(y[n/2]+y[n/2+1])/2;  
 //!n%2意为 n%2 =0(在if语句里写2个'=')，即 n%2 为假  
 else rey=y[n/2+1];

## **横坐标处理**

**区别在于两位士兵的横坐标可能相同，而纵坐标的相同对于计算并没有影响**

* 我们假设第一位士兵站的位置是k，因为x从x1开始，那么我们假设成起始位置为k+1

那么：第二位士兵的位置是 k+2，接着是k+3,k+4,……,k+n;

所以，士兵横向(即平行于y轴方向)移动的距离为：

|x1-(k+1)|+|x2-(k+2)|+|x3-(k+3)|+……+|x(n-1)-(k+n-1)|+|xn-(k+n)|

变形一下：

|(x1-1)-k)|+|(x2-2)-k)|+|(x3-3)-k)|+……+|(x(n-1)-(n-1))-k)|+|(xn-n)-k)|
  
 //x(n-1)中 n-1 是x的下标

**结论：我们只需要取 k=xi-i的中位数就好了！**

【核心代码及解释】

int main()
  
{
  
 cin >> n;
  
 for (int i = 1; i <= n; i++)
  
 {
  
 cin >> x[i] >> y[i];
  
 }
  
  
 sort(y + 1, y + n + 1); // 按y坐标排序
  
 sort(x + 1, x + n + 1); // 按x坐标排序
  
 for (int i = 1; i <= n; i++) x[i] -= i; // 消除水平方向偏差
  
 sort(x + 1, x + n + 1); // 按y方向偏差排序
  
  
 for (int i = 1; i <= n; i++)
  
 ans += abs(x[i] - x[(1 + n) / 2]) + abs(y[i] - y[(1 + n) / 2]); // 计算步数
  
  
 cout << ans << endl;
  
 return 0;
  
}

输入数据部分：

* 从标准输入读取一个整数n，表示士兵的数量。
* 接下来n行，每行输入两个整数x[i]和y[i]，表示第i个士兵的二维坐标。

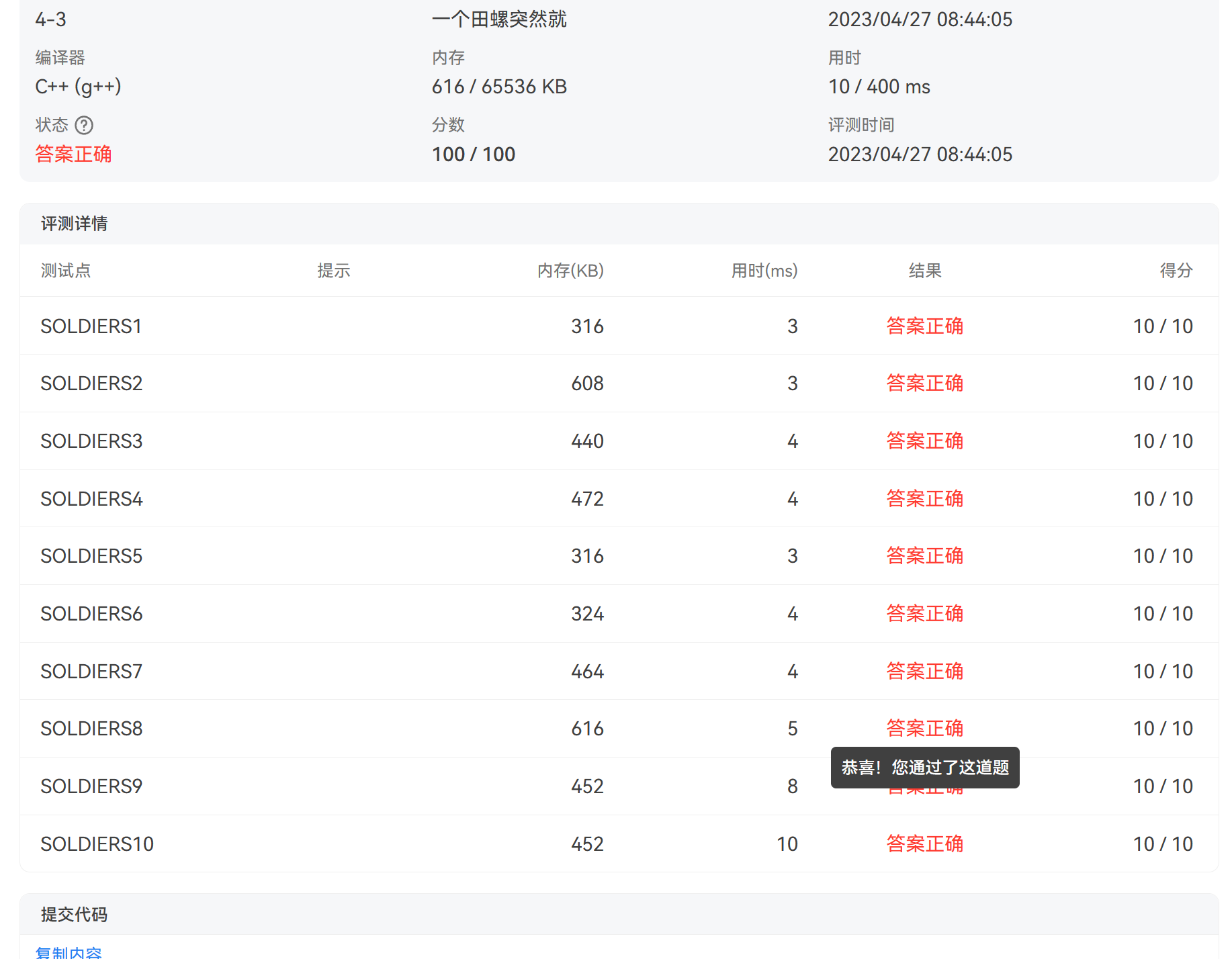
处理数据部分：

* 将所有士兵的y坐标按从小到大排序，保存在数组y中。
* 将所有士兵的x坐标按从小到大排序，保存在数组x中。
* 对数组x进行调整，让每个士兵的x坐标减去它的编号i，以消除“水平方向”的偏差。
* 再次对数组x进行排序，此时x中的元素已经表示每个士兵在“垂直方向”上的偏差。

输出结果部分：

* 循环遍历所有士兵，计算每个士兵到达最优位置需要的移动步数，并将步数累加到ans变量中。
* 最终输出ans变量的值，即为所有士兵移动的最少步数。

**实验结果：**



1. **附录**

**实验心得：**

在本次实验中，我们学习了三个算法问题：子序列的平均值、士兵排队、第k小元素。通过实验，我掌握了以下知识和技能：

对于子序列的平均值问题，可以通过二分答案的方法来解决。具体实现时，我们可以先求出每个位置减去mid后的前缀和，然后枚举左端点i，右端点j=i+L-1，找到一个区间使得该区间的平均值大于等于mid。如果找到这样的区间，则说明mid可以更大，否则mid应该减小。通过不断缩小区间，最终得到答案。

对于士兵排队问题，我们可以使用贪心对坐标分别分析并排序，求出中位数。

对于第k小元素问题，我们可以利用快速选择算法来解决。快速选择算法和快速排序算法类似，但是在快速选择算法中我们只需要排序数组的一部分，而不是整个数组。具体实现时，我们可以利用快排的思想，每次选取一个pivot，将数组分为两个部分，如果pivot的位置正好为k-1，则pivot即为第k小的元素；否则，我们根据pivot的位置继续在左半部分或右半部分中查找第k小的元素。

通过本次实验，我掌握了以上三个算法问题的解决方法和实现技巧，同时也提高了自己的编程能力和算法设计能力。

1. **源代码：**

## PTA-1:

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
using namespace std;
  
  
int a[1000010];
  
int partition(int nums[],int low, int high)//划分
  
{
  
 int randin=low+rand()%(high-low+1);
  
 swap(nums[randin],nums[low]);
  
 int pivot=nums[low];
  
 // debug(pivot);
  
 int l=low,r=high;
  
 while(l<r)
  
 {
  
 while(l<r&&nums[r]>pivot)r--;
  
 // debug(r);
  
 // debug(l);
  
 if(l<r)
  
 {
  
 nums[l] = nums[r];
  
 l++;
  
 }
  
 while(l<r&&nums[l]<pivot)l++;
  
 if(l<r)
  
 {
  
 nums[r] = nums[l];
  
 r--;
  
 }
  
 // debug(l);
  
 // debug(nums[l]) ;
  
 }
  
 nums[r]=pivot;
  
 // debug(l);
  
 return l;
  
 }
  
  
void qfind(int a[],int left,int right,int k)
  
{
  
 int mid=partition(a,left,right);
  
 if(k-1==mid)//第k小数，数组下标为k-1
  
 cout<<a[k-1];
  
 else if(k-1<mid)//在左区接着找
  
 qfind(a,left,mid-1,k);
  
 else qfind(a,mid+1,right,k);//在右区接着找
  
}
  
int main()
  
{
  
 int n,k;
  
 cin>>n>>k;
  
 for(int i=0;i<n;i++)
  
 cin>>a[i];
  
 qfind(a,0,n-1,k);
  
 return 0;
  
 }

方法2：

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
using namespace std;
  
int n,k;
  
int a[1000010];
  
int getLeastNumbers(int nums[], int k)
  
{
  
 //优先队列
  
  
 priority\_queue<int,vector<int>,less<int>> q;
  
 for(int i =0; i< k; i++)//前k个元素入队
  
 {
  
 q.push(nums[i]);
  
 }
  
 if(!k)return {};
  
 //将数组剩下的元素入队,则top就是第k个元素
  
 //如果使用大根堆，则需要n个
  
 for(int i = k;i < n ;i++)
  
 {
  
 int t = q.top();//取最顶上的元素
  
 if(nums[i] < t)
  
 {
  
 q.pop();
  
 q.push(nums[i]);
  
 }
  
 //q.top现在是前i个元素的第k个最大元素
  
 }
  
 return q.top();
  
 }
  
int main()
  
{
  
  
 cin>>n>>k;
  
 for(int i=0;i<n;i++)
  
 cin>>a[i];
  
 cout<<getLeastNumbers(a, k) <<endl;
  
 return 0;
  
 }

## PTA-2:

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
using namespace std;
  
double a[100010];
  
double b[100010];
  
double sum[100010];
  
  
int n, L;
  
  
int check(double mid)
  
{
  
 double mins;
  
 for(int i = 1; i <= n; i ++ )
  
 {//求i到j的和应该是sum[j]-sum[i-1]
  
 b[i - 1] = a[i - 1] - mid;
  
 sum[i] = sum[i - 1] + b[i - 1];
  
 }
  
 //前缀和处理
  
 mins = 1e9;
  
 for(int i = L; i <= n; i ++ )
  
 {
  
  
 mins = min(mins, sum[i - L]);
  
 if(sum[i] - mins >= 0)
  
 {
  
 return 1;
  
 }
  
 }
  
 return 0;
  
}
  
int main()
  
{
  
 cin >> n >> L;
  
 double minn = 1e9, maxx = 0;
  
 for(int i = 0; i < n; i ++ )
  
 {
  
 cin>>a[i];
  
 minn = min(a[i], minn);
  
 maxx = max(a[i], maxx);
  
 }
  
  
 double l = minn;
  
 double r = maxx;
  
  
 while(r - l > 1e-5)
  
 {
  
 double mid = l+ (( r - l ) / 2);
  
 if(check(mid))
  
 { //满足条件说明平均数可以更大
  
 l = mid;
  
 }
  
 else
  
 {
  
 r = mid;
  
 }
  
 }
  
 l += 1e-5;
  
 printf("%d", int(l\*1000));
  
 return 0;
  
}

## PTA-3

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
using namespace std;
  
const int N = 10010;
  
int n,x[N],y[N],ans;
  
int main()
  
{
  
 cin>>n;
  
 for (int i=1; i<=n; i++)
  
 {
  
 cin>>x[i]>>y[i];
  
 }
  
 sort(y+1,y+n+1);
  
 sort(x+1,x+n+1);
  
 for (int i=1; i<=n; i++) x[i]-=i;
  
 sort(x+1,x+n+1);
  
 for (int i=1; i<=n; i++)
  
 ans += abs(x[i]-x[(1+n)/2])+abs(y[i]-y[(1+n)/2]);
  
 cout<<ans<<endl;
  
 return 0;
  
}