算法分析与设计

课程信息发布平台

第二次上机

地址

第二次上机

题目列表

- 890 画个圈圈诅咒你
- 884 Bamboo的OS实验
- 872 AlvinZH的儿时梦想——坦克篇
- 892 Bamboo的饼干
- 862 AlvinZH的儿时梦想——运动员篇
- 891 ModricWang's Number Theory II
- 898 ModricWang's Real QuickSort

解题报告

890 画个圈圈诅咒你

思路

简单题。题目中的圆并没有什么实际作用，简化成**线段重合**问题会更好理解些。

暴力解法：使用双重for循环会T到想哭，记住最直接的方法一般是过不了题的。

解法一：二分查找。空间较小，时间更长。

把圆相离的问题转换为线段相交的问题，按先起点后终点的顺序升序排列这些圆（线段）。对于每条线段，向右找到第一条起点比这条线段终点大的线段，然后后面的线段都会满足要求，这里用二分去找。具体参考参考代码一。

解法二：线性查找。时间更短，空间更大。

同样是把圆相离的问题转换为线段的相交问题，把所有圆的左点和右点记录下来，并标记他们是左还是右，点的数量是圆的数量的两倍。排序：按所有点的位置排，如果点位置一样，则左边点优先（重要）。从头到尾遍历一次，用一变量（初始值为n）记录右边有多少个圆的左点，遇到左点时变量减1，遇到右点时用答案加上当前变量值，即是此圆右边与之相离的数量（左边的不须计算否则会产生重复）。具体参考参考代码二。

分析

两种方法都需要排序，排序时间 $O(NlogN)$ 。

查找时间：解法一是 $O(NlogN)$，解法二是 $O(N)$ 。

参考代码一

```
//
// Created by AlvinZH on 2017/10/24.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
using namespace std;

int n, x, r;

struct Circle {
    int left, right;

    bool operator < (const Circle a) const {
        if(left == a.left) return right < a.right;//其次右端
        return left < a.left;//左端优先
    }
}C[50005];

//二分查找右边最近的圆
int find(int l, int r, int x)
{
    int m;
    while(l <= r)
    {
        m = (l + r) >> 1;
        if(C[m].left < x) l = m + 1;
        else if(C[m].left >= x) r = m - 1;
    }
    return l;
}
```

```
int main()
{
    while(~scanf("%d",&n))
    {
        for(int i = 0; i < n; ++i) {
            scanf("%d %d", &x, &r);
            C[i].left = x - r;
            C[i].right = x + r;
        }
        sort(C, C+n);

        int ans=0;
        for(int i = 0; i < n-1; i++) {
            ans += n - find(i+1, n-1, C[i].right+1);
        }
        printf("%d\n", ans);
    }
}
```

参考代码二

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
using namespace std;

int n, x, r;

struct Node {
    int x;//位置
    int isR;//标记左右

    bool operator < (const Node n) const {
        if(x < n.x) return true;
        else if(x == n.x && isR == 0) return true;
        return false;
    }
}N[100010];

int main()
{
    while(~scanf("%d", &n))
    {
        int num = 0;
        for (int i = 0; i < n; ++i) {
            scanf("%d %d", &x, &r);
            N[num].x = x - r;
```

```
            N[num].isR = 0;
            num++;
            N[num].x = x + r;
            N[num].isR = 1;
            num++;
        }
        sort(N, N+num);

        int ans = 0;
        int sum = n;
        for (int i = 0; i < num; ++i) {
            if(N[i].isR == 0) sum--;
            else ans += sum;
        }
        printf("%d\n", ans);
    }
}
```

884 Bamboo的OS实验

分析

首先理解题意，要完成不同数量的不同命令，但是完成相同的命令之间必须有n个间隔，为使得时间最短，自然优先用其他命令来填充这n分钟的时间，由于数量少的命令可以用来填充空隙，所以次数最多的命令是起作用最大的。而且注意到，每次具体执行的是哪个命令并不影响时间，只与命令的数量有关（这有点贪心的思想，当预习吧）

基于以上分析，可以有以下几种方法：

1、按照命令数量从大到小排列，每次都是**从数量最多的命令开始新一轮周期**，这样是用时最少的。

举个栗子，命令1 2 3 4 5各有6 1 1 1 1个，n=2，如果采用 1->2->3->4->5->1_->..这样就会导致数量最多的命令1每个都要有2个空的时间段来填充；最佳的思想是1->2->3->1->4->5,这样尚未进入周期的命令1才需要额外的时间填充。

那么，将命令按数量从大到小排列后，总是选择当前数量最多的开始一轮周期，对选择的命令数量-1，时间++；每次这个周期一旦开始就要开始记录是否到n,周期结束后，要重新排序以保证从大到小的顺序，直到最多的命令也执行完毕，后面无需时间填充

核心代码如下：

```
static bool cmp(int i, int j)
{
    return i>j;
```

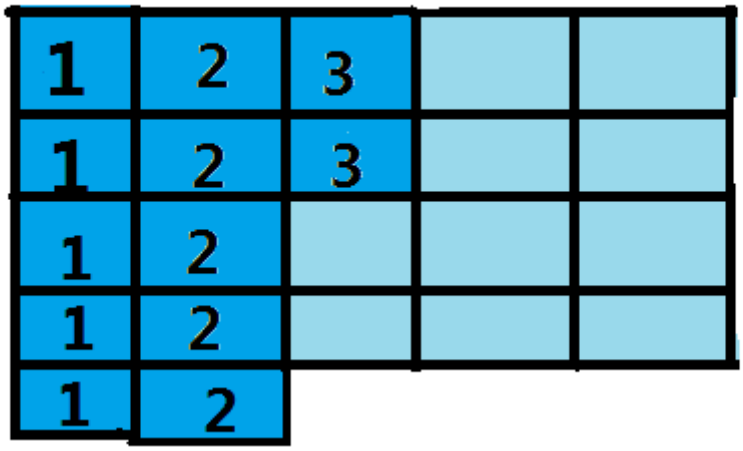
```
}
 int Map[35];
int main()
{
    int c,n,x;
    while(~scanf("%d",&x))
    {
         for(int i=0; i<35; i++)Map[i]=0;
        for(int i = 0; i<x; i++)
        {
            cin>>c;
            Map[c]++;
        }
        scanf("%d",&n);
        sort(Map,Map+35,cmp);
        int time = 0;
        while(Map[0]!=0)
        {
            int k =0;
            while(k<=n)
            {
                if(k<31&&Map[k]>0)
                {
                    Map[k]--;
                }
                else if(Map[0]==0)break;
                k++;
                time++;
            }
            sort(Map,Map+35,cmp);
        }

        printf("%d\n",time);
    }
}
```

2、核心思路还是上面的思路，但是上面每次都保持数组从大到小的性质可以用优先队列来实现。只是可能要借助临时的数组temp[]来存储从优先队列中pop出的数据。

3、也是大部分AC代码采用的思路。其实不管命令有多少，n等于几，这些命令总是要做完的，所花的时间一定是>=x的。所以只需要看需要填充多少思考人生时间。

上图:



//用画图画的，专治强迫症

深蓝色的就是用命令填充的，浅蓝色即为“思考人生”时间。

显然总的时间=任务数+思考时间

当只有第一排的1而后面全是浅蓝色时，是需要空闲时间最多的情况，max_num=n*$最多的命令数-1$，因为最后一轮后面是不填充的，所以-1；

我们要做的就是从这个最大值里逐列减去已经有命令的格子。

核心代码如下，可以看图体会：

```
int Map[35];
    for(int i=0; i<35; i++)Map[i]=0;
    for(int i = 0; i<tasks.size(); i++)
    {
        Map[tasks[i]]++;
    }
    sort(Map,Map+35,cmp);
    int max = Map[0]-1;
    int slot = max*n;
    for(int i = 1;i<31;i++)
    {
        slot -= min(Map[i],max);
    }
        int ans;
        if(slot>0)ans = slot+tasks.size();
        else ans = tasks.size();
```

872 AlvinZH的儿时梦想—-坦克篇

思路

简单题。仔细看题，题目意在找到直线穿过的矩形数最小，不能从两边穿过。那么我们只要知道每一行矩形之间的空隙位置就可以了。

如果这里用二维数组记住每一个空隙的位置，一是没有必要，二是记录了还要大量的处理才能得到答案。反正我是没想过要怎么处理。

可以发现，要得到本题的答案，只要找到空隙最多的哪个位置，我们取左边参考点，每一行的空隙位置我们可以记录到同一个数组里，即用A[pos]代表pos位置的直线有多少个空隙。但是发现总长度有点大，用数组是不可能了，有没有什么东西可以存下我这样的数对呢?

答案是map或轻便的pair。map的使用方法之前公邮里给大家发过了，不知道大家有没有好好学习。至于对组（pair），是一个稍微封装了一下的结构体模板，可以花一分钟看一下什么是对组。有着这个这题就简单了，两个值一个记录位置，一个记录出现次数，最后找到最大出现次数，n减去此数便可得到答案。具体可参考参考代码一。

非要这样做吗？？？我不会STL就做不了?

不是的，为什么非要把那么大的数当做索引呢？我就想把它当做数组的值，那下标是什么呢？答案是从0自增的一个计数变量。即A[cnt]记录空隙出现的位置，我们将它排序一下，相同位置会被放在一起，统计相同值的区间跨度一样可以找到空隙出现次数的最大值。具体见参考代码二。

分析

解法一使用map，时间复杂度将达到 $O(NMlogNM)$ 。

解法二由于需要手动排序，时间复杂度一样是 $O(NMlogNM)$ 。

参考代码一

```
//
// Created by AlvinZH on 2017/10/8.
// Copyright (c) AlvinZH. All rights reserved.
//

#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <map>
using namespace std;

int main()
{
    //freopen("in.txt", "r", stdin);
```

```
    //freopen("out.txt", "w", stdout);
    int n, m, x;
    while(~scanf("%d %d", &n, &m))
    {
        map<int, int> sameSum;//统计相同和的个数

        for (int i = 0; i < n; ++i) {
            int sum = 0;
            for (int j = 0; j < m; ++j) {
                scanf("%d", &x);
                if(j == m - 1) continue;
                sum += x;
                sameSum[sum]++;
            }
        }
        int maxSame = 0;
        for(map<int, int>::iterator it = sameSum.begin(); it != sameSum.end(); it++)
            if(maxSame < it->second) maxSame = it->second;

        printf("%d\n", n - maxSame);
    }
}
```

参考代码二

```
/*
 Author: 蒋泳波(41)
 Result: AC    Submission_id: 343393
 Created at: Thu Oct 26 2017 14:48:19 GMT+0800 (CST)
 Problem: 872    Time: 106    Memory: 5292
*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 1e6 + 10;

int a[maxn],cnt,x,n,m;

int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        cnt = 0;
        for(int i = 1; i <= n; i++)
        {
            int now = 0;
            for(int j = 1; j < m; j++)
```

```
            {
                scanf("%d",&x);
                now += x;
                a[cnt++] = now;
            }
            scanf("%d",&x);
        }
        sort(a,a+cnt);
        int last = 0,ans = 0;
        a[cnt] = -1;//注意初始化值
        for(int i = 1; i <= cnt; i++)
        {
            if(a[i] != a[i-1])//找到分界位置
            {
                if(ans < i - last)
                    ans = i - last;
                last = i;
            }
        }
        printf("%d\n",n - ans);
    }
    return 0;
}
```

892 Bamboo的饼干

分析

从两个数组中各取一个数，使两者相加等于给定值。要注意去重和排序

难度不大，方法很多，基本只要不大于On^2 的都可以过。本意想考察二分搜索

还可以借助stl中的map，set以及lower_bound等，当然只用数组也可以做。由于数据范围不大，也可以直接用数组下标来计数。

提起去重，有同学似乎一直纠结 （2,3）和（3,2）算不算重复数对..不算!只有（2,1）（2,1）这样的是真·重复对

map

这是很多AC代码用到的方法。因为map的key值是不重复且有序的，因此很适合本题。

参考代码

```
int n, t, x;
map<int, int> m;
int A[MaxSize];

int main()
{
    while(~scanf("%d", &n))
    {
        m.clear();
        int ans = 0;
        for (int i = 0; i < n; ++i) {
            scanf("%d", &x);
            m[x] = 1;
        }
        for (int i = 0; i < n; ++i) {
            scanf("%d", &A[i]);
        }
        scanf("%d", &t);
        sort(A, A+n);

        for (int i = n-1; i >= 0; --i) {
            long long tem = (long long)t - (long long)A[i];
            if(m[tem] > 0)//查询方便
            {
                ans++;
                printf("%lld %d\n", tem, A[i]);
                m[tem] = 0;
            }
        }

        if(ans == 0) printf("OTZ\n");
        printf("\n");
    }
}
```

数组

此处**@周宏建**，数组下标计数的方式，与上面map功能相似，注意map键值可为负但是数组下标不可以。当数据范围过大时该方法可能受限。
下面是这位同学上机时的AC代码：

```
#define bias 10000001
using namespace std;
bool a[20000010];
int b[100005];
int main()
{
        int n,x,target;
        while(~scanf("%d",&n))
```

```
        {
                memset(a,0,sizeof(a));
                bool flag=false;
                for(int i=0;i<n;++i)
                {
                        scanf("%d",&x);
                        a[x+bias]=1;
                }
                for(int i=0;i<n;++i)
                scanf("%d",b+i);
                scanf("%d",&target);
                sort(b,b+n);
                int tmp;
                for(int i=n-1;i>=0;--i)
                {
                        tmp=target-b[i]+bias;
                        if(tmp>=0&&tmp<20000010&&a[tmp])
                        {
                                a[tmp]=0;
                                flag=true;
                                printf("%d %d\n",target-b[i],b[i]);
                        }
                }
                if(!flag)
                printf("OTZ\n");
                printf("\n");
        }
}
```

数组+二分

这也是非常常见的思路。因为只有两个数，确定一个查找另一个，本质就是个查找。当普通查找TLE时应当会想到用二分查找来做。手写和STL均可。

参考代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<vector>
using namespace std;
const int maxx = 100004;
int a[maxx],b[maxx];
int BinarySearch(int a[], int l ,int r,int val)
{
    int mid  ;
    while(l<=r)
    {
        mid = (l+r)/2;
```

```
        if(a[mid]==val)return mid;
        else if(a[mid]>val)r = mid-1;
        else l  = mid+1;
    }
    return -1;
}
int main()
{
   int n,t,tt,pos;
   while(~scanf("%d",&n))
   {
       for(int i = 0;i<n;i++)
        scanf("%d",&a[i]);
       for(int j = 0;j<n;j++)
        scanf("%d",&b[j]);
       scanf("%d",&t);
       sort(a,a+n);sort(b,b+n);
       bool flag = false;
       for(int i = 0;i<n;i++)
       {
           if(i>0&&a[i]==a[i-1])continue;
           tt = t-a[i];
           pos = BinarySearch(b,0,n-1,tt);
           if(pos>-1&&pos<n)
           {
               printf("%d %d\n",a[i],b[pos]);
               flag = true;
           }
       }
       if(flag==false)printf("OTZ\n");
       printf("\n");
   }
}
```

当然，因为查找一个数，哈希表还能更快

拓展

请大家思考一下，如果给一个数组找三个数之和为某一值呢？四个数之和呢？

862-AlvinZH的儿时梦想——运动员篇

思路

难题。

应该想到，不管给出的数据如何，每一个淘汰的人不会对最终答案产生任何影响，所以每次淘汰就把人除掉就可以了，最后剩下的两个人计算它们从开始到相遇需要的时间就可以了。

首先对每个人根据初始位置进行排序，因为相遇总是先发生在相邻的两个人身上的，所以一开始先对**相邻的人**两两计算相遇时间，然后把相遇时间放进优先队列里（保证时间短的优先出队），然后依次出队，判定见面的两个人中哪个会被淘汰，然后把淘汰的人除去，维护新建立起来的相邻关系，以及新的相遇时间放进优先队列，一直处理直到队列只剩最后一对，然后取出来计算时间就可以了。

需要使用循环链表记录每一个人的相邻位置是谁，简单使用两个数组即可模拟循环链表。

本题还要注意的是环形跑道，也就是说在计算时间的时候记得相应处理，比如对跑道长度取模。注意看下列的求时间函数：

```
double getTime(int rear, int front)
{
    int dx = (P[front].pos - P[rear].pos + L) % L;//相对距离
    int dv = P[rear].v - P[front].v;//相对速度
    if (dv < 0)//front追rear
    {
        dv = -dv;
        dx = L - dx;
    }
    return (dx * 1.0 / dv);
}
```

分析

考察的是**优先队列**和**循环链表**。

最初状态环上有n个人，每次淘汰的必然是环上相邻的选手。注意到第一个被淘汰的人不会对后续过程有任何影响，所以找到这个人并把他从状态环上删去，就能把问题变成一个只有n-1人的子问题，此子问题与原问题有相同的答案。

利用优先队列维护状态环上所有相邻的人相遇的时间，每次取出最小值，可以淘汰一人，注意淘汰一人后，原本不相邻的人就相邻了，需要求得新的相遇时间入队，重复这一过程，直到队列剩余元素为1时结束。

考察了大家的模拟能力和手速，代码挺长，想起来还是挺简单的，对吧？

参考代码

```
//
// Created by AlvinZH on 2017/9/25.
// Copyright (c) AlvinZH. All rights reserved.
//
```

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <functional>
#include <algorithm>
#define MaxSize 100005
using namespace std;

struct Person {
    int pos,v,power;
    bool operator < (const Person& x) const {
        return pos < x.pos;//按位置从小到大排序，pos小的优先级大（sort函数）
    }
};

struct Race {
    int front,rear;//两个人对应下标
    double time;//相遇时间
    Race(int r = 0,int f = 0,double t = 0.0) {
        rear = r;front = f;time = t;
    }

    bool operator < (const Race& r) const {
        return time > r.time;//相遇时间小的位于队首（优先队列）
    }
};

int n,L;
Person P[MaxSize];
int nextP[MaxSize],lastP[MaxSize];//记录下一个和上一个人的标号
bool isOUT[MaxSize];//记录是否被淘汰
priority_queue<Race> Q;//优先队列

double getTime(int rear, int front)
{
    int dx = (P[front].pos - P[rear].pos + L) % L;//相对距离
    int dv = P[rear].v - P[front].v;//相对速度
    if (dv < 0)//front追rear
    {
        dv = -dv;
        dx = L - dx;
    }
    return (dx * 1.0 / dv);
}

int main()
{
    //freopen("in1.txt", "r", stdin);
    //freopen("out2.txt", "w", stdout);
    int T;
```

```
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d %d", &n, &L);

        for (int i = 0; i < n; i++) {
            scanf("%d %d %d", &P[i].pos, &P[i].v, &P[i].power);
        }
        sort(P, P + n);

        while (!Q.empty()) Q.pop();

        for (int i = 0; i < n; i++) {
            double t = getTime(i, (i + 1) % n);
            Q.push(Race(i,(i+1)%n,t));

            lastP[i] = (i - 1 + n) % n;
            nextP[i] = (i + 1) % n;
        }

        int All = n;//剩余比赛人数
        memset(isOUT, false, sizeof(isOUT));

        while (Q.size() > 1)
        {
            Race tmp = Q.top();
            Q.pop();
            int rear = tmp.rear,front = tmp.front;
            if(isOUT[rear] || isOUT[front])
                continue;
            if(lastP[rear] == front && nextP[front] == rear)//剩余最后两人
                break;

            if(P[rear].power > P[front].power)//rear追上front，淘汰front
            {
                isOUT[front] = true;
                nextP[rear] = nextP[front];
                lastP[nextP[front]] = rear;
                double tt = getTime(rear, nextP[front]);
                Q.push(Race(rear, nextP[rear], tt));
            }
            else
            {
                isOUT[rear] = true;
                lastP[front] = lastP[rear];
                nextP[lastP[rear]] = front;
                double tt = getTime(lastP[front], front);
                Q.push(Race(lastP[front], front, tt));
            }

            if(--All <= 0)
                break;
```

```
        }

        Race tmp = Q.top();
        Q.pop();
        printf("%.3lf\n", tmp.time);
    }
}

/*
 * 考点：优先队列
 * 坑：数据量较大
 */
```

891 ModricWang's Number Theory II

思路

使得序列的最大公约数不为1，就是大于等于2，就是找到一个大于等于2的数，它能够整除序列中的所有数。 考虑使得一个数d整除数组中所有数的代价：

如果一个数不能被b整除，那么可以花费x的代价删掉它，或者通过多次加1使得它可以被d整除，代价应该为 $(d-a[i]\%d)*y$, $(a[i]\%d==0s$时特判，应该为$0)$

令 $l=x/y$

如果$d-a[i]\%d<=l$ $(a[i]\%d!=0)$, 这个数产生的代价是 $(d-a[i]\%d)*y$, 否则是x。

所有代价求和就是总代价，最小的总代价就是答案。

但是这样枚举了d和a[i], 复杂度是$O(n^2)$ 的。 考虑将a[i]换一种方式存储：b[i]表示值为i的数出现的次数。 这样d可以将b分成如下若干段：

$[0,d-1],[d,d*2-1],[d*2,d*3-1],\ldots,[d*i,d*(i+1)-1]$

对于每一段而言， $[d*(i+1)-l,d*(i+1)-1]$ 内的数应该通过多次加1变成$d*(i+1)$,

代价应为 $(该区间内数的个数*d*(i+1)-该区间内的数之和)*y$

$[d*i+1,d*(i+1)-l-1]$ 内的数应该直接删除,

代价应为 $该区间内的个数*x$

通过构造相应的前缀和数组，上述操作均可以在$O(1)$ 的时间复杂度内完成

具体操作时应该注意边界

因为合数会被质数整除，因此d可以只枚举质数。

计算时间复杂度需要一些数论知识。首先素数密度也就是$\frac{小于n的素数}{n}$可以参见oeis A006880,一个近似解析式为 $\frac{1}{ln(n)}$，那么小于n的素数的总个数可以近似为 $\frac{n}{ln(n)}$

设小于等于n的素数为$prime[i]$，素数总数为P,取近似$P = \frac{n}{ln(n)}$

求结果部分的复杂度可以写为 $\sum_1^P \frac{n}{prime[i]}$

参见wikipedia,素数的倒数和又可以近似为 $\sum_1^P \frac{1}{prime[i]} = ln(ln(n))$

因此 $\sum_1^P \frac{n}{prime[i]} = O(n * ln(ln(n)))$

这里得到了计算结果部分的复杂度，还需要加上求素数这个过程的时间复杂度。如果使用朴素筛法，求复杂度的过程正好的上文所述的完全一致，其复杂度为$O(n * ln(ln(n)))$。如果使用欧拉筛求素数，复杂度为$O(n)$。

因此$O(运行时间) = O(求素数) + O(计算结果) = O(n * ln(ln(n)))$

代码

```
#include<iostream>
#include<cstring>

using namespace std;

const long long Max_Ai = 1000000*2;
long long n, x, y, l;
long long nums[Max_Ai + 10];
long long s[Max_Ai + 10], sum[Max_Ai + 10];

bool valid[Max_Ai + 10];
long long prime[Max_Ai + 10];
long long tot;

//线性筛求素数
void init_prime() {
        memset(valid, true, sizeof(valid));

        for (int i = 2; i <= Max_Ai; i++) {
                if (valid[i]) prime[++tot] = i;

                for (int j = 1; j <= tot && i*prime[j] <= Max_Ai; j++) {
                        valid[i*prime[j]] = false;
                        if (i%prime[j]==0) break;
                }
        }
}
```

```cpp
int main() {
#ifdef ONLINE_JUDGE
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
#endif

        init_prime();

        cin >> n >> x >> y;
        l = x/y;
        for (long long i = 1; i <= n; i++) {
                long long p;
                cin >> p;
                nums[p]++;    //这是一种比较特别的数字记录方法，原理类似于基数排序radix sort
        }

        for (long long i = 1; i <= Max_Ai; i++) {
                s[i] = s[i - 1] + nums[i];    //数量和
                sum[i] = sum[i - 1] + nums[i]*i;    //前缀和
        }

        auto min_cost = (long long) 1e18;
        for (long long i = 1; i <= tot; i++) {
                long long k = prime[i];
                long long now_cost = 0;

                for (long long j = 0; j <= Max_Ai; j += k) {
                        long long mid = max(j + k - l - 1, j);
                        long long bound = min(j + k - 1, Max_Ai);

                        if (bound > mid) {
                                now_cost += ((s[bound] - s[mid])*(j + k) - (sum[bound] -
                                now_cost += (s[mid] - s[j])*x;
                        } else {
                                now_cost += (s[bound] - s[j])*x;
                        }
                }

                min_cost = min(min_cost, now_cost);
        }

        cout << min_cost << "\n";

        return 0;
}
```

873 ModricWang's Real QuickSort

思路

这是一道非常基础的题，目的是帮助大家回顾快排相关的知识。大家完成此题之后应该就对快排有比较深刻的印象了。

对于整个快排的流程，题目描述中已经给了清晰完整的伪代码。需要自己加工的部分就是，需要手动记录下每次划分后的分界线，也就是划分时的变量i。

由于数据较为简单，要求的层数也较浅，实现划分函数后手工调用即可。

时间复杂度$O(n)$，空间复杂度$O(n)$

代码

```
#include <iostream>

using namespace std;

const int MaxN = (int) 1e7 + 10;

int n;
int nums[MaxN];

int partition(int *arr, int n) {
        int mid = arr[n/2];
        int i = 0, j = n - 1;
        while (i <= j) {
                while (arr[i] < mid) i++;
                while (arr[j] > mid) j--;
                if (i <= j) {
                        swap(arr[i], arr[j]);
                        i++;
                        j--;
                }
        }
        return i;
}

int main() {
#ifdef ONLINE_JUDGE
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
#endif

        cin >> n;
        for (int i = 0; i < n; i++)
                cin >> nums[i];
```

```
        int r = partition(nums, n);
        int l = partition(nums, r);

        for (int i = l; i < r; i++)
                cout << nums[i] << " ";

        cout << "\n";

}
```
