二分

二分法

定义

二分查找，也称折半搜索、对数搜索，是用来在一个**有序数组**中查找某一元素的算法。

过程

以在一个升序数组中查找一个数为例。

它每次考察数组当前部分的中间元素，如果中间元素刚好是要找的，就结束搜索过程；如果中间元素小于所查找的值，那么左侧的只会更小，不会有所查找的元素，只需到右侧查找；如果中间元素大于所查找的值同理，只需到左侧查找。

性质

时间复杂度

二分查找的最优时间复杂度为$O(1)$。

二分查找的平均时间复杂度和最坏时间复杂度均为log_2n。因为在二分搜索过程中，算法每次都把查询的区间减半，所以对于一个长度为n的数组，至多会进行log_2n次查找。

空间复杂度

迭代版本的二分查找的空间复杂度为$O(1)$。

递归（无尾调用消除）版本的二分查找的空间复杂度为$O(log_2n)$。

STL的二分查找

C++ 标准库中实现了查找首个**不小于**给定值的元素的函数 `std::lower_bound` 和查找首个**大于**给定值的元素的函数 `std::upper_bound`，二者均定义于头文件 `<algorithm>` 中。

二者均采用二分实现，所以调用前必须保证元素有序。

```
lower_bound(begin, end, val);
upper_bound(begin, end, val);
```

最大值最小化

注意，这里的有序是广义的有序，如果一个数组中的左侧或者右侧都满足某一种条件，而另一侧都不满足这种条件，也可以看作是一种**有序**（如果把满足条件看做1，不满足看做0，至少对于这个条件的这一维度是有序的）。换言之，二分搜索法可以用来查找满足某种条件的最大（最小）的值。

要求满足某种条件的最大值的最小可能情况（最大值最小化），首先的想法是从小到大枚举这个作为答案的「最大值」，然后去判断是否合法。若答案单调，就可以使用二分搜索法来更快地找到答案。因此，要想使用二分搜索法来解这种**最大值最小化**的题目，需要满足以下三个条件：

1. 答案在一个固定区间内；

2. 可能查找一个符合条件的值不是很容易，但是要求能**比较容易地判断某个值是否是符合条件**的；
3. 可行解对于区间满足一定的单调性。换言之，如果x是符合条件的，那么有$x+1$或者$x-1$也符合条件。（这样下来就满足了上面提到的单调性）

当然，最小值最大化是同理的。

二分答案

解题的时候往往会考虑枚举答案然后检验枚举的值是否正确。若满足单调性，则满足使用二分法的条件。把这里的枚举换成二分，就变成了「二分答案」。

[COCI2011-2012#5] 砍树

题目描述

伐木工人 Mirko 需要砍 M 米长的木材。对 Mirko 来说这是很简单的工作，因为他有一个漂亮的新伐木机，可以如野火一般砍伐森林。不过，Mirko 只被允许砍伐一排树。

Mirko 的伐木机工作流程如下：Mirko 设置一个高度参数 H（米），伐木机升起一个巨大的锯片到高度 H，并锯掉所有树比 H 高的部分（当然，树木不高于 H 米的部分保持不变）。Mirko 就得到树木被锯下的部分。例如，如果一排树的高度分别为 $20, 15, 10$ 和 17，Mirko 把锯片升到 15 米的高度，切割后树木剩下的高度将是 $15, 15, 10$ 和 15，而 Mirko 将从第 1 棵树得到 5 米，从第 4 棵树得到 2 米，共得到 7 米木材。

Mirko 非常关注生态保护，所以他不会砍掉过多的木材。这也是他尽可能高地设定伐木机锯片的原因。请帮助 Mirko 找到伐木机锯片的最大的整数高度 H，使得他能得到的木材至少为 M 米。换句话说，如果再升高 1 米，他将得不到 M 米木材。

输入格式

第 1 行 2 个整数 $N(1 \le N \le 10^6)$ 和 $M(1 \le M \le 2 \times 10^9)$，$N$ 表示树木的数量，M 表示需要的木材总长度。

第 2 行 N 个整数表示每棵树的高度$T_i(1 \le T_i \le 10^9, \sum T_i > M)$。

输出格式

1 个整数，表示锯片的最高高度。

样例输入

```
4 7
20 15 10 17
```

样例输出

```
15
```

样例输入

```
5 20
4 42 40 26 46
```

样例输出

```
36
```

Solution

很明显，随着锯片高度的升高，我们能够砍伐的木材是**越来越少**的，结合题意可得，存在一个中间值mid，使得高度低于mid时能超过M，反之亦然

考虑二分H_i后进行`check`，单次`check`操作实际上就是给定一个锯片高度，询问砍伐了多少木头，很容易实现

因此此题可以用二分答案的方法完成

Code

```
void solve(){
    ll n, m; cin >> n >> m;
    vector<int> tree(n);
    for (auto& v: tree) cin >> v;

    auto check = [&](int mid) {
        ll tmp = 0;
        for (auto v: tree) tmp += max(0, v - mid);
        return tmp >= m;
    };

    int l = 0, r = 4e5 + 10, ans = 0;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (check(mid)) l = mid + 1, ans = mid;
        else r = mid - 1;
    }

    cout << ans << endl;
}
```

[NOIP1999 提高组] 导弹拦截

题目描述

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度，计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

输入格式

一行，若干个整数($n \le 10^5, a_i \le 5 \times 10^4$)，中间由空格隔开。

输出格式

两行，每行一个整数，第一个数字表示这套系统最多能拦截多少导弹，第二个数字表示如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

样例输入

```
389 207 155 300 299 170 158 65
```

样例输出

```
6
2
```

Solution

第二问可以很简单地用贪心解决，我们接下来着重探讨第一问

第一问我们需要维护一个序列的**最长单调子序列**的长度，这个问题在$a_i < ser.back()$时很简单，但是$a_i > ser.back()$又该如何讨论呢

此时我们需要找到某个序列中最大的ser_j，使得$a_i > ser_j$，并且使用a_i将其替换

这种替换代表着：

1. 若某一时刻，后面的全部被替换掉：则代表我们之前的$ser_j, \cdots$及之后的全部放弃，实际的序列采用新的$a_i, \cdots$作为替换
2. 若某一时刻，后面的没有全部被替换掉：则代表我们实际的序列采用原先的$ser_j, \cdots$

在**单调序列中**查找序列中最大的ser_j，使得$a_i > ser_j$时，我们可以使用二分查找

Code

```
void solve(){
    while(cin >> a[++n]);
    n--;
    int len = 1;
    ans1[1] = a[1];
    for(int i = 2;i <= n; i++){
        if(a[i] <= ans1[len]) ans1[++len] = a[i];
        else{
            int l = 1,r = len;
            while(l < r){
                int mid = l + r >> 1;
                if(ans1[mid] < a[i]) r = mid;
                else l = mid + 1;
            }
            ans1[l] = a[i];
        }
    }
    cout << len << endl;
```

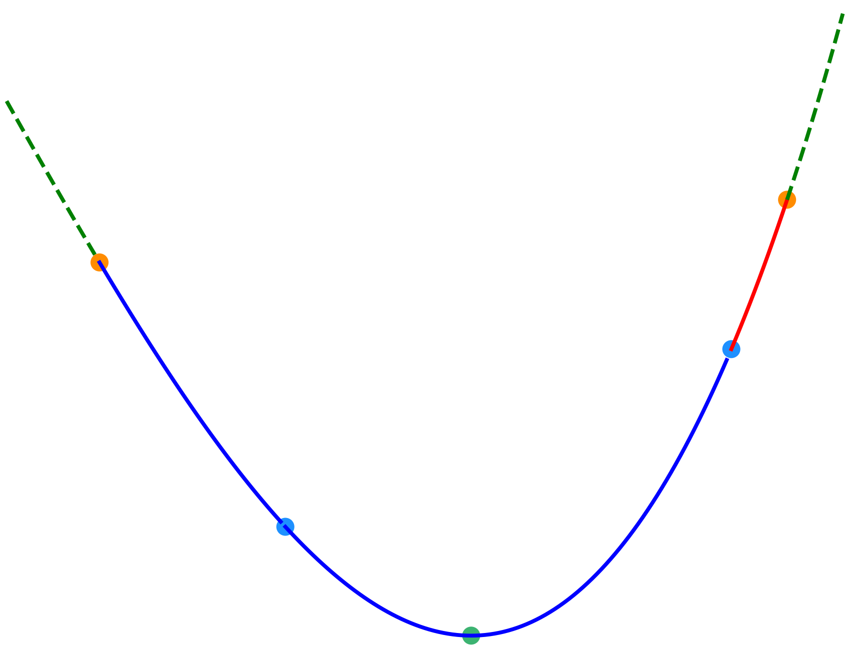
```
    len = 1;
    ans2[1] = a[1];
    for(int i = 2; i <= n; i++){
        if(a[i] > ans2[len]) ans2[++len] = a[i];
        else ans2[lower_bound(ans2+1, ans2+1+len ,a[i]) - ans2] = a[i];
    }
    cout << len << endl;

}
```

三分

如果需要求出**单峰函数**的极值点，通常使用二分法**衍生**出的三分法求单峰函数的极值点

三分法与二分法的基本思想类似，但每次操作需在当前区间$[l, r]$（下图中除去虚线范围内的部分）内任取两点$lmid, rmid(lmid < rmid)$（下图中的两蓝点）。如下图，如果$f(lmid < f(rmid))$，则在$[rmid, r]$（下图中的红色部分）中函数必然单调递增，最小值所在点（下图中的绿点）必然不在这一区间内，可舍去这一区间。反之亦然。



三分法每次操作会舍去两侧区间中的其中一个。为减少三分法的操作次数，应使两侧区间尽可能大。因此，每一次操作时的$lmid$和$rmid$分别取$mid + eps$和$mid - eps$是一个不错的选择。

[RdOI R2] 称重(weigh)

题目描述

rui_er 为了准备体测，买了 n 个实心球准备练习，但是却发现在发货时混入了两个质量明显较轻但外观相似的球（这两个球质量相等），且已知这两个球的质量之和大于一个正常的球。为了防止影响训练效果，现在需要找出这两个球。因为手动找太慢了，现在拿来了一个天平，可以在两侧各放上若干个球，得到两侧的质量大小关系。请你帮帮 rui_er，在使用不超过 k 次天平的情况下，找出这两个较轻的球。

这里 k 是每个测试点的属性，你不必也不应该读入。

交互方式

本题采用 I/O 交互。

你可以选择进行称量操作，此时向标准输出打印一行 `1 p a1 a2 ... ap q b1 b2 ... bq`，表示在天平左盘放入编号为 $a_1, a_2, \cdots, a_p$ 的 p 个球，在天平右盘放入编号为 $b_1, b_2, \cdots, b_q$ 的 q 个球。随后清空缓冲区，并从标准输入读入一个 `<>=` 之一的字符，表示左盘与右盘的质量关系。

对于每次此类询问，你需要保证 $1 \le p, q \le n$，$p + q \le n$，所有 a_i 和 b_i 互不相同，且你最多进行此类询问 k 次。

在得到答案后，向标准输出打印一行 `2 x y` 来提交答案，表示编号为 x 的球和编号为 y 的球质量偏轻。

你需要保证 $1 \le x < y \le n$（注意需要严格按照从小到大顺序输出），且在进行完这一操作后立即终止程序。

交互库在一开始就已经确定小球的情况，不会随着你的询问而改变。

输入格式

第一行一个整数 $n (2 \le n \le 500)$，表示球的数量。这里 $k (12 \le k)$ 是每个测试点的属性，你不必也不应该读入。

接下来若干行，见【交互方式】。

输出格式

若干行，见【交互方式】。

样例输入

```
6

=

<

>
```

样例输出

```
1 1 1 1 2

1 1 3 1 4

1 1 5 1 6

2 3 6
```

Solution

我们将整道题分为下面四种情况来完成：

1. 在$2i$个球中找1个球，这样只需要将球分为**均匀的两堆**和剩下**不那么均匀的一堆**，称重**均匀的两堆**即可
2. 在$2i$个球中找2个球，这样只需要将球分为**均匀的两堆**，就可以把问题归结为两个**单球单堆**或一个规模减半的**双球单堆**
3. 在$2i+1$个球中找1个球，这样只需要将球分为**均匀的两堆**和剩下**不那么均匀的一堆**，称重**均匀的两堆**即可
4. 在$2i+1$个球中找2个球，这样只需要将球分为**均匀的两堆**和剩下**一个**，称重**均匀的两堆**即可，若两堆重量相等，则归结为两个**单球单堆**，否则归结为规模减半的**双球单堆**

可以看出，我们的操作至多有1次未将问题规模减半，因此能够通过本题

Code

```
int interact(int *a, int p, int *b, int q) {
    printf("1 %d ", p);
    rep(i, 1, p) printf("%d ", a[i]);
    printf("%d ", q);
    rep(i, 1, q) printf("%d%c", b[i], " \n"[i==q]);
    fflush(stdout);
    char tmp[2];
    scanf("%s", tmp);
    if(tmp[0] == '<') return -1;
    if(tmp[0] == '=') return 0;
    return 1;
}
void searchAns(int L, int R, int k) {
//  printf("searchAns %d %d %d\n", L, R, k);
    int len = R - L + 1, M = (L + R) >> 1, tA = 0, tB = 0;
    int ML = L + (R - L) / 3, MR = R - (R - L) / 3;
//  printf("%d %d\n", ML, MR);
    if(len == k) {
        if(k == 1) ans.push_back(L);
        else ans.push_back(L), ans.push_back(R);
        return;
    }
    if(k == 1) {
        rep(i, L, ML) a[++tA] = i;
        rep(i, MR, R) b[++tB] = i;
        int res = interact(a, tA, b, tB);
```

```
        if(!res) return searchAns(ML+1, MR-1, 1);
        if(res == 1) return searchAns(MR, R, 1);
        return searchAns(L, ML, 1);
    }
    if(len & 1) {
        rep(i, L, M-1) a[++tA] = i;
        rep(i, M, R-1) b[++tB] = i;
        int res = interact(a, tA, b, tB);
        if(!res) {
            searchAns(L, M-1, 1);
            searchAns(M, R-1, 1);
        }
        else if(res == 1) {
            a[1] = L; b[1] = R;
            int qwq = interact(a, 1, b, 1);
            if(qwq == 1) {
                ans.push_back(R);
                searchAns(M, R-1, 1);
            }
            else searchAns(M, R-1, 2);
        }
        else {
            a[1] = R - 1; b[1] = R;
            int qwq = interact(a, 1, b, 1);
            if(qwq == 1) {
                ans.push_back(R);
                searchAns(L, M-1, 1);
            }
            else searchAns(L, M-1, 2);
        }
    }
    else {
        rep(i, L, M) a[++tA] = i;
        rep(i, M+1, R) b[++tB] = i;
        int res = interact(a, tA, b, tB);
        if(!res) {
            searchAns(L, M, 1);
            searchAns(M+1, R, 1);
        }
        else if(res == 1) searchAns(M+1, R, 2);
        else searchAns(L, M, 2);
    }
}
```

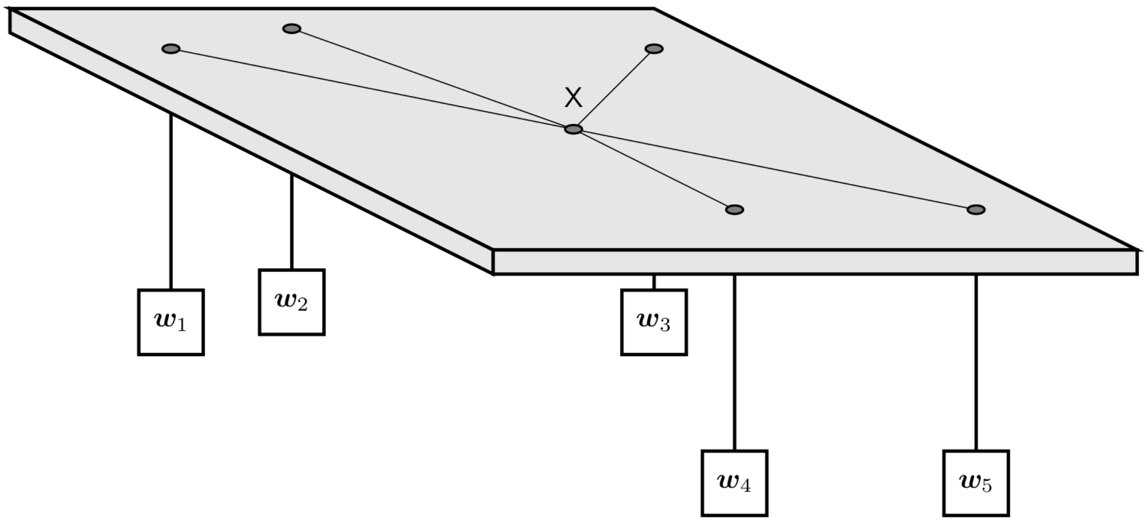
[JSOI2004] 平衡点

题目描述

如图，有 n 个重物，每个重物系在一条足够长的绳子上。

每条绳子自上而下穿过桌面上的洞，然后系在一起。图中 x 处就是公共的绳结。假设绳子是完全弹性的（即不会造成能量损失），桌子足够高（重物不会垂到地上），且忽略所有的摩擦，求绳结 x 最终平衡于何处。

注意：桌面上的洞都比绳结 x 小得多，所以即使某个重物特别重，绳结 x 也不可能穿过桌面上的洞掉下来，最多是卡在某个洞口处。



输入格式

文件的第一行为一个正整数 n（$1 \leq n \leq 1000$），表示重物和洞的数目。

接下来的 n 行，每行是 3 个整数 x_i, y_i, w_i，分别表示第 i 个洞的坐标以及第 i 个重物的重量。（$-10000 \leq x_i, y_i \leq 10000, 0 < w_i \leq 1000$)

输出格式

你的程序必须输出两个浮点数（保留小数点后三位），分别表示处于最终平衡状态时绳结 x 的横坐标和纵坐标。两个数以一个空格隔开。

样例输入

```
3
0 0 1
0 2 1
1 1 1
```

样例输出

```
0.577 1.000
```

Solution

我们对x和y轴分别进行分析，可以发现，对于单个轴而言，其**坐标 - 势能**图都满足**单峰**的样式

且x, y轴互不干扰

因此只需要**三分套三分**即可

Code

```
db ansx,ansy;

db dis(db x1,db y1,db x2,db y2){
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

db cal(db x,db y){
    db rt = 0;
    for (int i = 1; i <= n; i++)rt += p[i].w * dis(x, y, p[i].x, p[i].y);
    return rt;
}

db tre(double x){
    db l = -10000, r = 10000;
    for (int i = 1; i <= 100; i++){
        db dt = (r - l) / 3.0;
        db p1 = l + dt;
        db p2 = r - dt;
        db wp1 = cal(x, p1);
        db wp2 = cal(x, p2);
        if(wp1 < wp2) r = p2;
        else l = p1;
    }
    ansy = l;
    return cal(x, l);
}

void solve(){
    cin >> n;
    db ox = 0, oy = 0;
    for (int i = 1; i <= n; i++){
        cin >> p[i].x >> p[i].y >> p[i].w;
    }
    db l = -10000, r = 10000;
    for (int i = 1; i <= 100; i++){
        db dt = (r - l) / 3.0;
        db p1 = l + dt;
        db p2 = r - dt;
        db wp1 = tre(p1);
        db wp2 = tre(p2);
        if(wp1 < wp2) r = p2;
        else l = p1;
    }
    ansx=l;
    printf("%.3Lf %.3Lf\n",ansx,ansy);
}
```

[CF1749C] Number Game

题目描述

有一个序列 a ，Alice和Bob它进行操作，一共进行 k 个回合。在第 i 个回合中，Alice必须在序列 a 不为空的情况下删除序列 a 任意一个**小于等于** $k-i+1$ 的数。接着，Bob必须在序列 a 不为空的情况下将 $k-i+1$ 加到序列 a 中任意一个数上。当 k 个回合结束时，如果Alice每一回合都操作过了，他就赢了，否则就输了。求保证让Alice赢的情况下 k 最大是多少。

输入格式

第一行一个整数 t ($1 \le t \le 100$)，代表数据组数

每组数据第一行一个整数 n ($1 \le n \le 100$)，表示序列长度

每组数据第二行包含 n 个整数 $a_1, a_2, ..., a_n$ ($1 \le a_i \le n$).

输出格式

对于每组数据，输出一行一个整数k，在Alice和Bob都采取最优策略下，k的最大值

样例输入

```
4
3
1 1 2
4
4 4 4 4
1
1
5
1 3 2 1 1
```

样例输出

```
2
0
1
3
```

Solution

可以看出，对于某个k，如果`Alice`此时**失败**的话，对于$k+1$，`Alice`也必然失败

同样的，对于某个k，如果`Alice`此时**成功**的话，对于$k-1$，`Alice`也必然成功

因此，可以对k二分的同时，进行游戏结果模拟，从而确定二分区间如何缩限

Code

```
void solve(){
    int n; cin >> n;
    vector<int> a(n);
    for(auto& v : a) cin >> v;
    sort(a.begin(), a.end());

    auto play = [&](int k){
        int l = 0, r = upper_bound(a.begin(), a.end(), k) - a.begin() - 1;
        for(int i = k; i >= 1; i--){
            while(r >= l && a[r] > i) r--;
            if(r < l) return false;
            r--; l++;
        }
        return true;
    };

    int ans = 0, l = 0, r = n;
    while(l <= r){
        int mid = (l + r) >> 1;
        if(play(mid)) l = mid + 1, ans = mid;
        else r = mid - 1;
    }
    cout << ans << '\n';
}
```

[CF1760F] Quests

题目描述

有 n 个任务，你每一天都可以选择其中的一个任务完成或不选。当你完成了第 i 个任务，你将获得 a_i 元。但是如果你今天完成了一个任务，那么你之后 k 天内都不能再完成这个任务。

给出两个数 c，d，要求求出满足在 d 天内可以收集至少 c 元的最大的 k。

如果不存在这样的 k，输出 Impossible。

如果 k 可以无限大，输出 Infinity。

否则输出最大的 k。

输入格式

第一行一个整数$t(1 \le t \le 10^4)$，代表数据组数

每组数据第一行是 n, c, d ($2 \le n \le 2 \cdot 10^5$; $1 \le c \le 10^{16}$; $1 \le d \le 2 \cdot 10^5$) 代表任务总数、金钱需求数和总时间

第二行包含 n 个整数 $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$)，代表每个任务的回报

保证 $\sum n \le 2 \cdot 10^5$

输出格式

如果k不存在，输出`Impossible`，如果任意的k都能满足要求，输出`Infinity`，否则输出**最大**的k

样例输入

```
6
2 5 4
1 2
2 20 10
100 10
3 100 3
7 2 6
4 20 3
4 5 6 7
4 100000000000 2022
8217734 927368 26389746 627896974
2 20 4
5 1
```

样例输出

```
2
Infinity
Impossible
1
12
0
```

Solution

首先，如果我们每天都完成**回报最高**的任务都无法达到需求，则直接输出`Impossible`

如果我们在**不重复**任务的情况下都能达到要求，则直接输出`Infinity`

否则我们考虑二分k

以$\lfloor \frac{d}{k} \rfloor$为周期数，$d \mod k$为最后一周期长度，我们采用受益最高的的$k$个任务进行试算，如果能满足要求，则将$k$的二分区间向小侧缩限，否则将$k$的二分区间向大侧缩限

Code

```
void solve(){
    ll n, c, d; cin >> n >> c >> d;
    vector<ll> a(n), ps(n);
    for(auto& v: a) cin >> v;

    sort(all(a), [](auto x, auto y){
        return x > y;
    });

    for(int i = 0; i < n; i++) ps[i] = (i ? ps[i - 1] : 0ll) + a[i];

    if(a[0] * d < c) {
```

```
        cout << "Impossible\n"; return;
    }

    ll tot = 0;
    for(int i = 0; i < min(n, d); i++) tot += a[i];
    if(tot >= c) {
        cout << "Infinity\n"; return;
    }

    int l = 2, r = d, ans = 0;

    while(l <= r){
        auto mid = (l + r) >> 1;
        ll res = 0;
        auto t1 = d / mid;
        auto m1 = d - t1 * mid;
        res += t1 * ps[min(mid - 1ll, n - 1)];
        res += m1 ? ps[min(m1 - 1, n - 1)] : 0ll;

        if(res < c) r = mid - 1;
        else l = mid + 1, ans = mid - 1;
    }

    cout << ans << '\n';
}
```

[CF1732C1] Sheikh (Easy version)

题目描述

给定一个数组$a_1, a_2, \cdots, a_n$，一次询问，求 $[L, R]$ 内**任意子区间的和**减去**该区间的异或值**得到的差的最大值，若有多个最大值，取区间长度最短的，若仍有多个区间，任取一个输出即可。在简单版本中保证 $q=1$，L = 1，$R=n$。

输入格式

第一行一个整数$t(1 \le t \le 10^4)$，代表数据组数

每组数据第一行是 n 和 $ q $ ($1 \le n \le 10^5$, $q=1$) 代表数组的长度和询问的次数

第二行包含 n 个整数 $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$)，代表数组元素

接下来的q行，每行两个整数 L_i 和R_i ($1 \le L_i \le R_i \le n$) 代表区间起点 / 终点

保证 $\sum n \le 2 \cdot 10^5$.

保证 $L_1=1$ 和 $R_1=n$.

输出格式

每次询问输出一行两个整数，答案区间左边界和答案区间右边界

样例输入

```
6
1 1
0
1 1
2 1
5 10
1 2
3 1
0 2 4
1 3
4 1
0 12 8 3
1 4
5 1
21 32 32 32 10
1 5
7 1
0 1 0 1 0 1 0
1 7
```

样例输出

```
1 1
1 1
1 1
2 3
2 3
2 4
```

Solution

对于区间$[L, R]$的数值之和，我们可以使用**前缀和**做到$O(n)$预处理，$O(1)$查询

而对于区间$[L, R]$的异或和，我们利用$a_1 \oplus a_2 \cdots a_l \oplus \cdots\ a_r \oplus a_1 \cdots a_{l-1} = a_l \oplus \cdots a_r$这样的性质，同样使用**前缀和**做到$O(n)$预处理，$O(1)$查询

我们使用$f(l, r) = sum(l, r) - xorsum(l, r)$代表我们答案所求的值

此时可以看到，若左端点l固定，$f(l, r) - f(l, r-1) \geq a_r$

因此我们可以对枚举l，并对r进行二分，对于每个l得到最优的r，汇总答案即可

Code

```
void solve(){
    int n, q; cin >> n >> q;
    vector<ll> a(n + 1), sum(n + 1), xr(n + 1);
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        sum[i] = sum[i - 1] + a[i];
        xr[i] = (xr[i - 1] ^ a[i]);
    }
```

```cpp
    auto f = [&](int l, int r){
        return sum[r] - sum[l - 1] - (xr[r] ^ xr[l - 1]);
    };

    while(q--){
        int l, r; cin >> l >> r;
        ll ans = f(l, r), ansl = l, ansr = r;

        for(int i = l; i <= r; i++){
            int L = i, R = r;
            while(L <= R){
                int mid = (L + R) >> 1;
                if(f(i, mid) == ans){
                    if(mid - i + 1 < ansr - ansl + 1) ansl = i, ansr = mid;
                    R = mid - 1;
                }
                else L = mid + 1;
            }
        }

        cout << ansl << ' ' << ansr << endl;
    }
}
```