2025 SYNU 4月周赛 Round III

题目由易到难排序为：**D(1300)->C(1400)->A(1800)->B(2100)**

题解报告

D. 嗷呜嗷呜事务所IV

tag

枚举，模拟

解题思路

题目要求求出一个序列 w_i 出现次数最多的数，即众数。且定义 w_i 的值为 a_i 在序列 a 中出现的次数。

只需要一次循环求出 w_i ，然后再用一次循环统计 w_i 值中各个值出现的次数，取最大即可。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
// #define int long long
#define endl '\n'
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using mat = vector<vector<int>>;
using u64 = unsigned long long;
constexpr int N = 1e7 + 5;
// dont use umap!!!

int cnt_c[N], cnt_w[N];
signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> c(n + 1);
    for(int i = 1; i <= n; i ++)
    {
        cin >> c[i];
        cnt_c[c[i]] ++;
    }
    int ans = -1, ans_cnt = 0;
    for(int i = 1; i <= n; i ++)
    {
        cnt_w[cnt_c[c[i]]]++;
        if(cnt_w[cnt_c[c[i]]] > ans_cnt or (cnt_w[cnt_c[c[i]]] == ans_cnt and cnt_c[c[i]] > ans))
            ans = cnt_c[c[i]], ans_cnt = cnt_w[cnt_c[c[i]]];
    }
    cout << ans << '\n';
    return 0;
}
```

C. 嗷呜嗷呜事务所III

tag

前缀和，排序

解题思路

首先考虑没有奇美拉不参加排队的情况，题目是一个很简单的贪心问题，只需要所有奇美拉按照从小到大的顺序排列总时间一定最小（证明省略）。

现在思考若存在有一只奇美拉不参加排队的情况：我们钦定这只奇美拉的疗愈时间为 v，那么实际上就是在排好序的队列中移除这只奇美拉后重新计算当前队伍的总等待时间即可。

如上删除和维护总时间可以使用平衡树完成，但这过于复杂了。我们来考虑更加简单的方法，考虑每只奇美拉的等待时间等于他前面的所有奇美拉的等待时间之和。那么这只退出奇美拉对总等待时间的贡献实际上为它后方奇美拉数量的和乘以他的等待时间。只需要计算出排序后 v 的位置 p，那么我们只需要用总时间 sum 减去它后方所有奇美拉的数量 $n-p$ 乘以 v 再减去他自己的等待时间 $\sum_{i=1}^{p} a_{ord[i]}$ 即为他的答案 ($a_{ord[i]}$ 表示排序后的 a_i)，$\sum_{i=1}^{p} a_{ord[i]}$ 的计算可以用**前缀和优化**。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
// #define int long long
#define endl '\n'
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using u64 = unsigned long long;
// dont use umap!!!

signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<pii> pa(n + 1);
    vector<i64> pre(n + 1);
    i64 sum = 0;
    for(int i = 1; i <= n; i ++)
    {
        cin >> pa[i].first;
        pa[i].second = i;
    }
    sort(pa.begin() + 1, pa.end());
    for(int i = 1; i <= n; i ++)
    {
        pre[i] = pre[i - 1] + pa[i].first;
        sum += pre[i - 1];
    }
    vector<i64> ans(n + 1);
    for(int i = 1; i <= n; i ++)
    {
        auto [v, idx] = pa[i];
        ans[idx] = sum - pre[i - 1] - (i64)v * (n - i);
    }
    for(int i = 1; i <= n; i ++) cout << ans[i] << '\n';
    return 0;
}
```

A. 嗷呜嗷呜事务所

tag

贪心，暴力枚举，数据结构

一种错误的思路

注意到每次取行和列都互相不影响（应该说，假设取走某一行后，相当于再每一列都取走了一个固定的 p，不影响每一列和的相对大小）。所以想到一个贪心策略：每次取和最大的行或者列。这样很容易实现，利用2个优先队列维护行和列的和，每次去除两个堆顶比较，取较大的取出，取出后（假设取出的是行），将他减去 $m \times p$ 后再加入优先队列，然后再给列的tag加上 p(表示给所有列都减去了一个 p)，这样就可以保证每次取的都是当前最大的行或者列。

事实上，这个思路是错误的，或者说不好处理这种情况：**存在行最大值和列最大值相同的情况**。例如：

```
input1:
3 3 2 4
5 4 3
5 4 4
1 2 6
ans1:
25

input2:
3 3 2 4
5 5 1
4 4 2
3 4 6
ans2:
25
```

在第一个样例中应该先取行，而第二个中应该先取列。

题解思路

回到正确的思路，我们还是有结论：**取行和列互不影响**。我们错误的贪心在于无法处理行和列相等时，应该先取行还是先取列。我们跳过这个问题，既然行和列互不影响，我们考虑答案的结构：**答案一定是取 x 次行最大和 y 次列最大**。那么很显然，x 和 y 是我们可以枚举的，而取 x 次行最大和 y 次列最大，这是我们可以先行预处理出来的，再做前缀和即可。然后枚举 x，那么 $y=k-x$，每次的答案就是 $f(x)=pre_{row_x}+pre_{col_{k-x}}$。但是，我们还需要考虑行和列会有交叉操作：假设我们取了 x 次行，那么每一列的值都应该相应的减少 $x\times p$，所以我们取 $k-x$ 次列时，这一部分是多计算的（也就是把 $x\times p$ 多算了 $k-x$ 次）。所以最后的答案应该减掉 $x\times p\times (k-x)$，那么最后的答案应该是:

$$ans=\max_{i=0}^{k}(pre_{row_i}+pre_{col_{k-i}}-i\times p\times (i-p))$$

> tips：$-\inf$ 要开到 $-1e18$。

参考代码

```
#define matrix(_x, _y, _z) vector<vector<int>>(_x, vector<int>(_y, _z))
#define debug(_x) cout << #_x << '=' << _x << endl
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using mat = vector<vector<int>>;
using u64 = unsigned long long;
constexpr int N = 2e5 + 10;
// dont use umap!!!

signed main()
{
    IOS;
    int n,m,k,p;
    cin >> n >> m >> k >> p;
    auto g = matrix(n + 1, m + 1, 0);
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= m; j ++)
            cin >> g[i][j];
    priority_queue<int> heaprow, heapcol;
    for(int i = 1; i <= n; i ++) // row
    {
        int sum = 0;
        for(int j = 1; j <= m; j ++)
        {
            sum += g[i][j];
        }
        heaprow.push(sum);
    }
    for(int i = 1; i <= m; i ++) // col
    {
        int sum = 0;
        for(int j = 1; j <= n; j ++)
        {
            sum += g[j][i];
        }
        heapcol.push(sum);
    }
    i64 ans = -1e18;
    vector<int> precol(k + 1), prerow(k + 1);
    for(int i = 1; i <= k; i ++)
    {
        auto colsum = heapcol.top();
        auto rowsum = heaprow.top();
        heapcol.pop();
        heaprow.pop();
        precol[i] = precol[i - 1] + colsum;
        prerow[i] = prerow[i - 1] + rowsum;
        heapcol.push(colsum - n * p);
        heaprow.push(rowsum - m * p);
    }
    for(int i = 0; i <= k; i ++)
    {
        ans = max(ans, prerow[i] + precol[k - i] - i * (k - i) * p);
    }
    cout << ans << '\n';
    return 0;
}
```

B. 嗷呜嗷呜事务所II

tag

逆序对，归并排序/树状数组，模拟，枚举

解题思路

本题本质上的题意为：每次让所有数自增1，且对 p 取模，求逆序对能到达最大值以及最小需要多少次操作让逆序对最大。

最开始我们可以用归并排序或树状数组求出原序列逆序对的个数。让我们回忆一下，逆序对个数为 $\sum_{i=1}^{n}\sum_{j=i+1}^{n}[a_j < a_i]$

首先，假设没有取模操作，很显然，所有数自增并不改变相对大小，因此逆序对不会发生改变。我们考虑逆序对第1次发生变化的时刻，此时一定是所有 a_i 中的最大值发生了取模操作，变成了0。相应的，逆序对第 k 次发生变化的时刻，此时一定是所有 a_i 中的第 k 大发生了取模操作。

然后，我们来考虑每次逆序对变化的影响：假设当前是逆序对发生的第 k 次变换，钦定序列第 k 大值为 X，其所有的出现位置的集合为 Pos_X，序列中所有的 X 则会变成 0。我们不妨考虑其中的一个 X 和他的位置 p，此时它对逆序对的新贡献为位于 p 前方所有大于 0 的数的个数(即 $p-1-\sum_{i=1}^{p-1}[a_i=0]$)，同理应该减去它原来的贡献，即他后方小于 X 的数的个数 $n-p-\sum_{i=p+1}^{n}[a_i=X]$ (因为此时 X原来一定是操作前序列的最大值)。注意！这里的大于小于是只在逆序对发生了 k 次变换之后的新序列。同理，我们对所有 X 的位置都进行上述操作，就得到了逆序对的变化值，进而求出当前逆序对的值。

我们只需要枚举 P 次逆序对变化，即可求出所有的情况，对所有情况的逆序对数量取max，并记录最早到达最大值的操作次数即可。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS                         \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);                   \
    cout.tie(0);
// #define int long long
#define endl '\n'
#define all(_x) _x.begin(), _x.end()
#define range(_x, _st, _ed) (_x.begin() + _st), (_x.begin() + _ed)
#define rep(_x, _y, _z) for (int _x = _y; _x <= _z; _x++)
#define matrix(_x, _y, _z) vector<vector<int>>(_x, vector<int>(_y, _z))
#define debug(_x) cout << #_x << '=' << _x << endl
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using mat = vector<vector<int>>;
using u64 = unsigned long long;
constexpr int N = 2e5 + 10;
// dont use umap!!!

int mergesort(vector<int> &a) // 归并求逆序对数量
{
    if (a.size() == 1) return 0;
    int mid = a.size() / 2;
    vector<int> L, R;
    for (int i = 0; i < mid; i++) L.push_back(a[i]);
    for (int i = mid; i < a.size(); i++) R.push_back(a[i]);
    int res = mergesort(L) + mergesort(R);
    int i = 0, j = 0, p = 0;
    while (i < L.size() && j < R.size())
    {
        if (L[i] <= R[j]) a[p++] = L[i++];
        else a[p++] = R[j++], res += mid - i; // 此时左半剩余元素个数为mid - i
    }
    while (i < L.size()) a[p++] = L[i++];
    while (j < R.size()) a[p++] = R[j++];
    return res;
}

void solve()
{
    int n, m;
    cin >> n >> m;
    vector<int> a(n);
    map<int, vector<int>> pos; //pos[x]记录x出现过的所有位置
    for (int i = 0; i < n; i++)
        cin >> a[i], pos[-a[i]].push_back(i + 1); // 因为要从大到小枚举，所以map中存负数
    i64 ans = mergesort(a); // 求最开始逆序对的数量
    i64 t = ans, cnt = 0;
    i64 tcnt = 0; // 记录当前操作了多少次
    for (auto [v, poss] : pos) // 枚举所有逆序对发生变化的时刻
    {
        for (int i = 0; i < poss.size(); i++) // 枚举当前最大值的所有位置
        {
            // 计算其对逆序对的影响
            t += poss[i] - 1 - i; // 正贡献为其前方所有非0数的个数
            t -= (n - poss[i]) - (poss.size() - i - 1); // 负贡献为其原序列中后方所有小于它的个数
        }
        // 记录操作次数
        tcnt += m - (-v) - tcnt;
        if (t > ans) // 更新答案
        {
            cnt = tcnt;
            ans = t;
        }
    }
```

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

signed main()
{
    IOS;
    int _ = 1;
    cin >> _;

    while (_--)
        solve();

    return 0;
}
```