[Problem - A - Codeforces]

题意是说有n只怪物，每个怪物有h的血量和p的伤害，Genos初始能对所有怪物造成k的伤害，之后存活的怪物中，p值最小的怪物会削弱Genos的攻击能力，使Genos的攻击力减少p，重复若干轮直到Genos的伤害减为0或者怪物死光

由于每次对Genos产生削弱的是杀伤力最小的怪物，那么我们可以用一个小根堆来维护所有的怪物，始终将伤害最小的队伍置于堆顶，对于每一轮战斗，可以从堆顶取出一个怪物，先判断这个怪物死没死，如果已经死了则丢弃continue掉就行，否则的话讲Genos的攻击力削弱，直到k减为0或者堆空

时间复杂度取决于while循环执行的次数和小根堆维护的代价

由于k最多被削k次就会变为0，所以while执行最多不会超过k次，时间复杂度就成了$n \times log(n) + k$，即$O(nlogn)$

当然排序也能写，时间复杂度是一样的，而且可能更好写，大家可以尝试一下

```cpp
#include <bits/stdc++.h>

using std::cin;
using std::cout;
using std::string;
using ll = long long;
using ull = unsigned long long;
using uint = unsigned int;
const int N = 1e5 + 10;

int n, k;
struct Monster {
    int h, p;
    bool operator<(const Monster &m) const {
        return p > m.p;
    }
};
int h[N], p[N];

void solve() {
    cin >> n >> k;
    std::priority_queue<Monster> q;
    for (int i = 1; i <= n; i ++ ) cin >> h[i];
    for (int i = 1; i <= n; i ++ ) cin >> p[i];
    for (int i = 1; i <= n; i ++ ) q.push({h[i], p[i]});

    long long nowAtk = k;
    while (q.size()) {
        auto [h, p] = q.top();
        if (nowAtk >= h) {
            q.pop();
            continue;
        } else {
            if (p >= k) {
                k = 0;
                break;
            } else {
                k -= p;
                nowAtk += k;
```

```
            }
        }
    }

    if (q.size()) cout << "NO\n";
    else cout << "YES\n";
}
int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int T = 1;
    cin >> T;
    while (T --)
    {
        solve();
    }
}
```

Problem - B - Codeforces

题意是说，有一棵链状的树（因为正好有两个叶子节点），问你从起点开始，只能往没激活过的节点（起点视作被激活）的方向走，问最后A赢还是B赢。

由于只能往没激活的点走，并且这是一条链，那么只要出发的方向确定，就不能再走回头路了，考虑检查起点的两个分支有没有一个分支的叶子节点的深度的奇偶性，奇数深度A必胜，偶数深度B必胜

时间复杂度遍历整棵树$O(n)$没啥好说的

```
#include <bits/stdc++.h>

using std::cin;
using std::cout;
using std::string;
using std::vector;
using ll = long long;
using ull = unsigned long long;
using uint = unsigned int;
const int N = 2e5 + 10;
int n, _;
vector<int> g[N];
int dep[N];
bool flag;

void dfs(int u, int fa) {
    dep[u] = fa == -1 ? 0 : dep[fa] + 1;
    if (g[u].size() == 1 && dep[u] % 2) {
        flag = true;
        return;
    }
    for (auto v: g[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}
void solve() {
    cin >> n >> _;
```

```
    for (int i = 1; i < n; i ++ ) {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }

    int s;
    cin >> s;
    flag = false;
    dfs(s, -1);

    cout << (flag?"Ron\n" : "Hermione\n");
}
int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int T = 1;
    //cin >> T;
    while (T --)
    {
        solve();
    }
}
```

Problem - C - Codeforces

题意是说，问你数组中有多少个子段可以满足子段内所有数的异或和不为完全平方数。

为什么是完全平方数，因为根据因数分解的方法，c = a * b，若a <= b，则只有当c是完全平方数的时候，a=b才有可能成立，此时会产生重复，使得因数的个数为奇数个。

如何快速知道一段区间的异或和呢，由于$a \oplus b = c$时，$c \oplus b = a$，则可以考虑使用异或前缀和，$a_i \oplus a_{i+1} \oplus \ldots \oplus a_j$就等于$a_j \oplus (a_1 \oplus a_2 \oplus \ldots \oplus a_{i-1})$

我们记$s_i = a_1 \oplus a_2 \oplus \ldots \oplus a$，$s_{i,j} = a_i \oplus a_{i+1} \oplus \ldots \oplus a_j$

那么如何统计答案呢？我们知道，一个长度为n的数组，字段个数为$n \times (n+1)/2$个，4e5范围内的完全平方数也不是很多，大约是600多个，我们可以考虑再次利用一下异或的性质，计算对于每一个a_j，由于$s_{i,j} = s_j \oplus s_{i-1}$，而$s_{i,j}$又是一个完全平方数，我们只需要记录在$a_j$之前，有多少个$s_{i-1}$满足$s_{i-1} = s_{i,j} \oplus s_j$即可

这题还有一个很重要的性质，对于一个数x，将它和小于x的数进行异或，得到的结果一定小于$2x$，并且题目中说$a_i < n$，我们可以进行一些优化

n的完全平方数的个数为$\sqrt{n}$个，维护前缀和以及枚举a_j都是$O(n)$的，总时间复杂度是$O(n\sqrt{n})$，实际上是2e5 * 600 = 1e8，一开始没注意到$a_i <= n$用map统计的个数还会多一个log就TLE了，后来才发现用数组就能存下。

```
#include <bits/stdc++.h>

using std::cin;
using std::cout;
using std::string;
using ll = long long;
using ull = unsigned long long;
```

```cpp
using uint = unsigned int;
const int N = 2e5 + 10;

int n;
int a[N];
void solve() {
    cin >> n;
    std::vector<int> cnt(n * 2);
    for (int i = 1; i <= n; i ++ ) {
        cin >> a[i];
        a[i] ^= a[i - 1];
    }

    cnt[0] = 1;
    long long ans = 0;
    for (int i = 1; i <= n; i ++ ) {
        for (int j = 0; j * j < 2 * n; j ++ ) {
            int t = j * j ^ a[i];
            if (t >= 2 * n) continue;
            ans += cnt[t];
        }
        cnt[a[i]] ++;
    }
    cout << (long long) n * (n + 1) / 2 - ans << "\n";
}

int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int T = 1;
    cin >> T;
    while (T --)
    {
        solve();
    }
}
```

Problem - D - Codeforces

题意是说，要找到满足边长为l的正方形区域，且在这个$l \times l$的正方形区域中，最矮的建筑的高度要大于等于l，现在要你最大化l的值

很显然，当l取的越小，越容易满足条件，当l越大，越不容易满足条件，可以考虑二分枚举l的值，然后扫描整个数组检查是否能找到至少一个正方形能满足条件

由于需要多次扫描矩阵，就意味着每次扫描矩阵的耗费不能太长，如果暴力的去维护区间最小值，肯定会超时，我们考虑将一维的区间最小值问题扩展到二维，我们知道，一维区间最小值可以用st表、树状数组、线段树来维护，由于这题不涉及到修改，我们考虑使用二维st表

我们令$st[i][j][l]$代表点(i,j)为正方形的左上角，正方形的边长为2^l的区间最小值，那么将一个边长为2^l的矩阵分成左上、右上、坐下、右下四块区域，形成四个小的2^{l-1}大小的矩阵，就可以维护出st表

由于最小值有可重复贡献的性质，对于一个任意大小的正方形，我们一定可以用四个2^l的正方形将这个正方形覆盖，从而实现$O(1)$查询

时间复杂度：构建st表的复杂度为$n \times m \times log_2(min(n,m))$，二分答案的复杂度为$log_2(min(n,m)) \times n \times m$

```cpp
#include <bits/stdc++.h>

using std::cin;
using std::cout;
using std::string;
using std::vector;
using ll = long long;
using ull = unsigned long long;
using uint = unsigned int;

const int N = 1e6 + 10;

int n, m;
int lg2[N];
vector<vector<int>> a;
vector<vector<vector<int>>> st;

void init_lg2() {
    lg2[1] = 0;
    for (int i = 2; i < N; i ++ ) {
        lg2[i] = lg2[i / 2] + 1;
    }
}
bool check(int l, vector<vector<vector<int>>> &st) {
    for (int i = 1; i + l - 1 <= n; i ++ ) {
        for (int j = 1; j + l - 1 <= m; j ++ ) {
            int lo = lg2[l];
            int x = i, y = j;
            int t = st[x][y][lo];
            x = i + l - (1 << lo), y = j;
            t = std::min(t, st[x][y][lo]);
            x = i, y = j + l - (1 << lo);
            t = std::min(t, st[x][y][lo]);
            x = i + l - (1 << lo), y = j + l - (1 << lo);
            t = std::min(t, st[x][y][lo]);
            if (t >= l) return true;
        }
    }
    return false;
}

void solve() {
    cin >> n >> m;
    a.resize(n + 1);
    for (int i = 1; i <= n; i ++ ) a[i].resize(m + 1);
    st.resize(n + 1);
    for (int i = 1; i <= n; i ++ ) {
        st[i].resize(m + 1);
        for (int j = 1; j <= m; j ++ ) st[i][j].resize(16);
    }
    for (int i = 1; i <= n; i ++ ) for (int j = 1; j <= m; j ++ ) cin >>
a[i][j];
    for (int i = 1; i <= n; i ++ ) {
        for (int j = 1; j <= m; j ++ ) {
            st[i][j][0] = a[i][j];
```

```
        }
    }
    for (int l = 1; l < 16; l ++ ) {
        for (int i = 1; i <= n; i ++ ) {
            for (int j = 1; j <= m; j ++ ) {
                int ml = 1 << (l - 1);
                st[i][j][l] = st[i][j][l - 1];
                st[i][j][l] = std::min({
                    st[i][j][l - 1],
                    st[std::min(n, i + ml)][j][l - 1],
                    st[i][std::min(m, j + ml)][l - 1],
                    st[std::min(n, i + ml)][std::min(m, j + ml)][l - 1]
                });
            }
        }
    }

    int l = 1, r = std::min(n, m);
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (check(mid, st)) l = mid;
        else r = mid - 1;
    }
    cout << l << "\n";

}
int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(0);
    int T = 1;
    init_lg2();
    cin >> T;
    while (T --)
    {
        solve();
    }
}
```

hyh思路：前缀和，将正方形内大于等于l的位置都标记为1，只需要统计标记成1的个数是否为$l \times l$即可

```
#include <bits/stdc++.h>
using std::cin;
using std::cout;
using i64 = long long;
using db = double;
using ldb = long double;

#ifdef ONLINE_JUDGE
    constexpr int N = 1e5+7;
#else
    constexpr int N = 1e3+7;
#endif

int n, m;

```

```cpp
bool check(int x, std::vector<std::vector<int>> &a) {
    std::vector<std::vector<int>> sum(n+1, std::vector<int>(m+1));
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++) {
            int t = a[i][j] >= x;
            sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + t;
        }
    }

    for(int i=x;i<=n;i++) {
        for(int j=x;j<=m;j++) {
            if(sum[i][j] - sum[i-x][j] - sum[i][j-x] + sum[i-x][j-x] == x * x) {
                return true;
            }
        }
    }
    return false;
}

void solve(){
    cin>>n>>m;
    std::vector<std::vector<int>> a(n+1, std::vector<int>(m+1));
    for(int i=1;i<=n;i++) {
        for(int j=1;j<=m;j++) {
            cin>>a[i][j];
        }
    }

    int l = 1, r = std::min(n, m), mid;
    while(l <= r) {
        mid = l + r >> 1;
        if(check(mid, a)) {
            l = mid + 1;
        } else {
            r = mid - 1;
        }
    }

    cout<<r<<'\n';
}

signed main(){
    std::ios::sync_with_stdio(0);cin.tie(NULL);cout.tie(NULL);
    int T=1;
    cin>>T;
    while(T--){  solve();  }
    //system("pause");
    return 0;
}

```

ryx思路：单调队列

```cpp
#include <bits/stdc++.h>
```

```cpp
#define endl '\n'
#define buff ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);
using namespace std;
const int N = 1e3 + 10;
const int M = 1e6 + 10;
vector<int> a[N],hmin[N],minn[N];
int tou,wei,stt[M],n,m;
int check(int len){
    for(int i=1;i<=n;i++){
        hmin[i].clear();
        minn[i].clear();
        tou = 1,wei = 0;
        for(int j=0;j<m;j++){
            while(tou<=wei&&a[i][j]<a[i][stt[wei]]) wei--;
            stt[++wei] = j;
            while(tou<=wei&&stt[wei]-stt[tou]+1>len) tou++;
            if(j>=len-1) hmin[i].push_back(a[i][stt[tou]]);
        }
    }
    for(int i=0;i<(int)hmin[1].size();i++){
        tou = 1,wei = 0;
        for(int j=1;j<=n;j++){
            while(tou<=wei&&hmin[j][i]<hmin[stt[wei]][i]) wei--;
            stt[++wei] = j;
            while(tou<=wei&&stt[wei]-stt[tou]+1>len) tou++;
            if(j>=len) minn[j].push_back(hmin[stt[tou]][i]);
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=0;j<(int)minn[i].size();j++){
            if(minn[i][j]>=len) return 1;
        }
    }
    return 0;
}
int main(){
    buff;
    int t;
    cin>>t;
    while(t--){
        //int n,m;
        cin>>n>>m;
        for(int i=1;i<=n;i++){
            a[i].clear();
            for(int j=1;j<=m;j++){
                int temp;
                cin>>temp;
                a[i].push_back(temp);
            }
        }
        int l=1,r=min(n,m);
        while(l<=r){
            int mid = (l+r)/2;
            if(check(mid)) l=mid+1;
            else r=mid-1;
        }
```

```
58          cout<<r<<endl;
59      }
60      return 0;
61  }
```