Task A. Make the word
In this problem, we first had to check the length of the input word and the length of the word “impact”. If the lengths are different, then the answer is "NO". Otherwise, let's look at all possible permutations of the letters of the word "impact" if one of these permutations is entered, then only in this case the answer is "YES".

Problem B. Abdu and an array
Let's look at each element of the array. If the number x was met, then we increase the counter by 1. If after viewing all the elements the counter value is equal to k, then the answer is “YES”, otherwise the answer is “NO”.

Task C. <blank>
Let's loop from x + 1 until we find a suitable number. We will break each number into digits and check that they consist of the same digits. If the number consists of the same digits, then print this number and close the program.

Problem D. Missing Number
Let's remember all the numbers that we met. Since the numbers n and k are up to 100000, the maximum answer cannot exceed 200000. Let's start the loop from 1 to 200000 and count the number of those numbers that were not in the input array. This can be done using the “Two Pointers” method, or using the map data structure, or simply by remembering in the array which elements were encountered and which were not.

Problem E. Landscape
Let's fix the position i and for each position i we will run the cycle j to find a hill or a ravine. Let's call the inflection point the coordinate that is the top of the hill or the bottom of the ravine. There must be no more than one inflection point on the segment from i to j. By sorting through all possible i and j, you can find all suitable hills and ravines.
C++ code example:

```
#include <bits/stdc++.h>
#define ll long long
```

```
#define ull unsigned long long
#define ld long double
#define uint unsigned int
#define f first
#define s second
#define pb push_back
#define mp make_pair
using namespace std;
const int INF = (int)(1e9 + 7);
const ll INFLL = (ll)(1e18 + 7);
const ld Pi = acos(-1.0);
const ld Eps = (ld)(1e-12);
mt19937 mrand(random_device{}());
int rnd(int x) { return mrand() % x;}
int a[6000];
int main(){
    srand(time(0));
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    //freopen("a.in", "r", stdin);
    //freopen("a.out", "w", stdout);
    //cout.precision(12);
    int n, ans = 1;
    cin >> n;
    for(int i = 1; i <= n; i ++){
        cin >> a[i];
    }
    for(int l = 1; l < n; l ++){
        bool typee = (a[l + 1] > a[l]), swapped = false;
        for(int r = l + 1; r <= n; r ++){
            if(a[r] == a[r - 1]){
                break;
            }
            if((a[r] > a[r - 1]) != typee){
                if(swapped){
                    break;
                }
                swapped = true;
```

```
            typee = !typee;
        }
        ans = max(ans, r - l + 1);
        //cout << l << ' ' << r << '\n';
      }
    }
    cout << ans << '\n';
    return 0;
}
```

Problem F. The Beauty of an Array

Let's count for each number how many times it occurred, and its last occurrence is r[a[i]]. Let's go through all the numbers and for each number that has occurred the maximum number of times we will find the value r[a[i]] - i + 1 Compare it with the best answer, and in order not to look at this number again, you can simply reduce the number by 1 occurrences of this number in the array.

C++ code example:

```
#include <bits/stdc++.h>
using namespace std;

int findShortestSubArray(vector<int> a) {
        map <int, int> cnt, r;
        int mx = 0;
        for(int i = 0; i < a.size(); ++i){
            cnt[a[i]]++, r[a[i]] = i;
            mx = max(mx, cnt[a[i]]);
        }
        int ans = a.size();
        for(int i = 0; i < a.size(); ++i) {
            if(mx == cnt[a[i]]){
                ans = min(ans, r[a[i]] - i + 1);
                cnt[a[i]]--;
            }
        }
        return ans;

int main(){
```

```
	int n;
	cin >> n;
	vector <int> a;
	for(int i = 1; i <= n; ++i){
		int x;
		cin >> x;
		a.push_back(x);
	}
	cout << findShortestSubArray(a);
	return 0;
}
```

Problem G. Table

This task is for implementation. Here it was necessary to simulate what is said in the condition. We will run through all the elements from the first column and the last row. From each element we will move to the right and up to pass along this diagonal, that is, if the row number is x and the column number is y, then we decrease x by 1 and increase y by 1 until x and y are within the matrix. Output all elements with x and y coordinates. See code for better understanding.

C++ code example:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;
```

```
int n;
int a[510][510];

void solve() {
	cin >> n;

	for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++)
		cin >> a[i][j];

	vector <int> ans;
	for (int it = 1; it <= 2 * n - 1; it++) {
		int x = min(it, n);
		int y = 1;
		if (it > n) y = it - n + 1;
		while (x >= 1 && y <= n) {
			ans.pb(a[x][y]);
			x--, y++;
		}
	}

	for (int x: ans)
		cout << x << ' ';
}

int main() {
	ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
}
```

Problem H. Maximum Ones

In this problem, it was necessary to use either a binary search or two pointers. For each position i, we find by binary search the maximum position j such that there are at most k zeros between i and j.
The number of zeros can be maintained using an array of prefix sums. Among all i and j, find the longest segment.
C++ code example:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;

void solve() {
        int n; cin >> n;
        vector <int> a(n);
        for (int i = 0; i < n; i++)
                cin >> a[i];
        int k; cin >> k;

    int l = 0, r = n + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        bool ok = 0;
        int cnt = 0;
        for (int i = 0; i < m - 1; i++) cnt += a[i];
        for (int i = m - 1; i < n; i++) {
```

```
            cnt += a[i];
            if (m - cnt <= k) ok = 1;
            cnt -= a[i - m + 1];
        }
        if (ok) l = m;
        else r = m;
    }

    cout << l;
}

int main() {
        ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
}
```

Task I. Lunch break

Let's assume an array prev[i] that will store the position of the previous "matched" element of the array for element number i. We call "suitable" any element whose difference with the current element does not exceed x. That is, prev[i] = j if 1 <= j < i, |a[j] - a[i]| <= x. Among all suitable j (And it’s enough to simply enumerate them), we take the maximum. If there is no suitable j, leave prev[i] = 0.

The prev array makes it easy to respond to queries. Let's iterate over all l <= i <= r and look at prev[i]. If there is at least one i such that l <= prev[i] <= r (that is, the nearest matching element to the left of i is in the range), then the answer is "YES", otherwise - "NO".

Problem J. Find a horse

This was an interactive task in which it was necessary to maintain the rules for working with the interactor and, after each data output, update the stream buffer using the flush command. I remind you that in C ++ in

the endl command there is also an update of the data stream. For other languages, it is written in more detail in the problem statement.

Solution:
Initially, let's imagine that a horse could be in each cell, that is, initially all cells are under suspicion.
Let's sort through the first question and pick the best one. The best question is the question after which, in the worst case, there will be a minimum number of suspected cells.
By asking a question, we will mark all the cells that could give exactly the same answer as the jury program gave you. In exactly the same way, we will continue to ask questions until only one cell remains in the suspected cells.

C++ code example:

```
#pragma comment(linker, "/stack:2000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")

#include <bits/stdc++.h>

#define mp make_pair
#define pb push_back
#define IOS ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define ll long long
#define f first
#define s second

using namespace std;
int n, m, k, a[50][50];
const int INF = (int)(1e9 + 7);
const int MAXN = (int)(3e5 + 7);
int ans, x, y, b[10];
int dx[9] = {0, -2, -2, -1, -1, 1, 1, 2, 2};
int dy[9] = {0, -1, 1, -2, 2, -2, 2, -1, 1};

bool check(int l1, int r1, int l2, int r2, int x, int y){
```

```
    return (x >= l1 && x <= l2 && y >= r1 && y <= r2);
}

void rec(int z){

    int xx1, xx2, yy1, yy2, poos;
    int minn = INF;
    for (int i1 = 1; i1 <= 8; i1++)
        for (int j1 = 1; j1 <= 8; j1++)
            for (int i2 = i1; i2 <= 8; i2++)
                for (int j2 = j1; j2 <= 8; j2++){
                    for (int i = 0; i <= 9; i++)
                        b[i] = 0;
                    for (int x = 1; x <= 8; x++)
                        for (int y = 1; y <= 8; y++)
                            if (a[x][y] == 1){
                                k = 0;
                                for (int i = 0; i < 9; i++)
                                    if (check(i1, j1, i2, j2, x + dx[i], y + dy[i]))
                                        k++;
                                b[k]++;
                            }
                    int maxx = 0;
                    int pos = -1;
                    for (int i = 0; i <= 9; i++){
                        if (b[i] > maxx){
                            maxx = b[i];
                            pos = i;
                        }
                    }
                    if (maxx < minn)
                    {
                        minn = maxx;
                        poos = pos;
                        xx1 = i1;
                        xx2 = i2;
                        yy1 = j1;
```

```
                    yy2 = j2;
                }
            }
    int cntk;
     cout << "? " << char(xx1 + 'A' - 1) << yy1 << ' ' << char(xx2 + 'A' - 1)
<< yy2 << endl;
     cin >> cntk;

     for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1){
                k = 0;
                for (int i = 0; i < 9; i++)
                    if (check(xx1, yy1, xx2, yy2, x + dx[i], y + dy[i]))
                        k++;
                if (k != cntk)
                    a[x][y] = 0;
            }

    if (minn == 1){
        for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1)
                cout << "! " << char(x + 'A' - 1) << y << endl;
        return;
    }
    rec(z + 1);
}

int main() {
    IOS;

    for (int i = 1; i <= 8; i++)
        for (int j = 1; j <= 8; j++)
            a[i][j] = 1;

    rec(1);
```

```
    return 0;
}
```

Problem K. Color table

O(n^2 * k):
It is enough to know the optimal paths from cells (1, 1) and (n, n) to all the others. Path data must be calculated for all colors. Let's denote the number of different colors as k. Let's go through the color, calculate the dynamics for each and get answers for the cells of this color.

O(n^4):
Denote by d[i] the number of cells of color i. Let's learn to solve for color i in O(d[i]^2). Sort cells by row number, if equal - by column number. We will calculate the same dynamics as before, but instead of two transitions we will consider d[i].

O(n^3):
We optimize the decisive color algorithm in O(d[i]^2) to O(d[i] * n). Note that in each column we are only interested in the bottom cell of those already considered. Let b[i] be the current best result in column i. Then we create f[i][j] = 1 + max(b[k]), (1 <= k <= j). Then we assign b[i] = f[i][j].

O(n^2 * log(n)) (Full solution):
Note that to determine the value of f[i][j] it is necessary to request the maximum on some prefix b[i]. We implement the maximum query and the change operation in O(log(n)). For this, for example, you can use a segment tree.

Solution 1: https://pastebin.com/8yyB84W3
Solution 2: https://pastebin.com/zND1yapW

Задача A. Сөзді алыңыз
Бұл тапсырмада алдымен енгізілген сөздің ұзындығын және "impact" сөзінің ұзындығын тексеру қажет болды. Егер ұзындықтар әртүрлі болса, онда жауап "NO". Әйтпесе, "impact" сөзінің әріптерін ауыстырудың барлық түрлерін қарастырайық, егер осы ауыстырулардың бірі енгізілсе, онда бұл жағдайда ғана "YES" жауабы болады.

Задача B. Абду және массив
Массивтің әр элементін қарастырайық. Егер x саны кездессе, онда есептегішті 1-ге көбейтеміз. Егер барлық элементтерді көргеннен кейін есептегіштің мәні k болса, онда жауап "YES", әйтпесе жауап "NO" болады.

Задача C. <blank>
x + 1-ден бастап циклді қолайлы санды тапқанға дейін іске қосыңыз. Біз әр санды цифрларға бөліп, олардың бірдей сандардан тұратындығын тексереміз. Егер сан бірдей цифрлардан тұрса, онда біз бұл санды алып, бағдарламаны жабамыз.

Задача D. Жетіспейтін сан
Біз кездестірген барлық сандарды есте сақтаймыз. N және k сандары 100000-ға дейін болғандықтан, максималды жауап 200000-нан аспауы керек. Біз 1-ден 200000-ға дейінгі циклды іске қосамыз және енгізілген массивте болмаған сандардың санын есептейміз. Мұны "екі көрсеткіш" әдісі арқылы немесе map деректер құрылымын қолдану арқылы немесе массивте қандай элементтер кездескенін және қайсысы жоқ екенін есте сақтау арқылы жасауға болады.

Задача E. Ландшафт
Біз i позицияны бекітеміз және әр i позиция үшін төбені немесе шатқалды табу үшін j циклін бастаймыз. Төбенің шыңы немесе шатқалдың түбі болып табылатын координатаны иілу нүктесі деп атайық. i-ден j-ге дейінгі аралықта бір иілу нүктесінен артық болмауы керек. Барлық мүмкін i және j-ді аралап, сіз барлық қолайлы төбелер мен шатқалдарды таба аласыз.
C++ кодының мысалы:

```
#include <bits/stdc++.h>
#define ll long long
#define ull unsigned long long
#define ld long double
#define uint unsigned int
#define f first
#define s second
#define pb push_back
#define mp make_pair
using namespace std;
const int INF = (int)(1e9 + 7);
const ll INFLL = (ll)(1e18 + 7);
const ld Pi = acos(-1.0);
const ld Eps = (ld)(1e-12);
mt19937 mrand(random_device{}());
int rnd(int x) { return mrand() % x;}
int a[6000];
int main(){
    srand(time(0));
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    //freopen("a.in", "r", stdin);
    //freopen("a.out", "w", stdout);
    //cout.precision(12);
    int n, ans = 1;
    cin >> n;
    for(int i = 1; i <= n; i ++){
        cin >> a[i];
    }
    for(int l = 1; l < n; l ++){
        bool typee = (a[l + 1] > a[l]), swapped = false;
        for(int r = l + 1; r <= n; r ++){
            if(a[r] == a[r - 1]){
                break;
            }
            if((a[r] > a[r - 1]) != typee){
                if(swapped){
```

```
                break;
            }
            swapped = true;
            typee = !typee;
        }
        ans = max(ans, r - l + 1);
        //cout << l << ' ' << r << '\n';
    }
  }
  cout << ans << '\n';
  return 0;
}
```

Задача F. Массивтің сұлулығы

Біз әр сан үшін оның қанша рет кездескенін және оның соңғы пайда болуын есептейміз r [a [i]]. Біз барлық сандар бойынша жүреміз және ең көп рет кездескен әр сан үшін r[a[i]] - i + 1 мәнін табамыз, оны ең жақсы жауаппен салыстырамыз және бұл санға тағы бір рет қарамау C++ кодындағы мысал:

```
#include <bits/stdc++.h>
using namespace std;

int findShortestSubArray(vector<int> a) {
    map <int, int> cnt, r;
    int mx = 0;
    for(int i = 0; i < a.size(); ++i){
        cnt[a[i]]++, r[a[i]] = i;
        mx = max(mx, cnt[a[i]]);
    }
    int ans = a.size();
    for(int i = 0; i < a.size(); ++i) {
        if(mx == cnt[a[i]]){
            ans = min(ans, r[a[i]] - i + 1);
            cnt[a[i]]--;
        }
    }
    return ans;
```

```
int main(){
	int n;
	cin >> n;
	vector <int> a;
	for(int i = 1; i <= n; ++i){
		int x;
		cin >> x;
		a.push_back(x);
	}
	cout << findShortestSubArray(a);
	return 0;
}
```

Задача G. Кесте

Мұнда шартта айтылғандарды модельдеу қажет болды. Біз бірінші бағаннан және соңғы жолдан барлық элементтер бойынша жүгіреміз. Әр элементтен біз осы диагональ бойынша өту үшін оңға және жоғарыға жылжимыз, яғни егер x жолының нөмірі және y бағанының нөмірі болса, онда x-ті 1-ге азайтып, x және y матрица шегінде болғанша y-ді 1-ге көбейтеміз. Жақсы түсіну үшін кодты қараңыз.

C++ кодының мысалы:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
```

```
const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;

int n;
int a[510][510];

void solve() {
	cin >> n;

	for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++)
		cin >> a[i][j];

	vector <int> ans;
	for (int it = 1; it <= 2 * n - 1; it++) {
		int x = min(it, n);
		int y = 1;
		if (it > n) y = it - n + 1;
		while (x >= 1 && y <= n) {
			ans.pb(a[x][y]);
			x--, y++;
		}
	}

	for (int x: ans)
		cout << x << ' ';
}

int main() {
	ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
}
```

Задача H. Бірлердің максималды саны

Бұл тапсырмада екілік іздеуді немесе екі көрсеткішті қолдану қажет болды. i мен j арасындағы нөлдер саны k-дан аспайтындай әрбір i позициясы үшін біз екілік іздеу арқылы максималды j позициясын табамыз.
Нөлдер санын префикс қосындыларының массиві арқылы сақтауға болады. Барлық i және j арасында біз ең ұзын кесіндіні табамыз.

C++ кодындағы мысал:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;

void solve() {
        int n; cin >> n;
        vector <int> a(n);
        for (int i = 0; i < n; i++)
                cin >> a[i];
        int k; cin >> k;

    int l = 0, r = n + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        bool ok = 0;
```

```
        int cnt = 0;
        for (int i = 0; i < m - 1; i++) cnt += a[i];
        for (int i = m - 1; i < n; i++) {
            cnt += a[i];
            if (m - cnt <= k) ok = 1;
            cnt -= a[i - m + 1];
        }
        if (ok) l = m;
        else r = m;
    }

    cout << l;
}

int main() {
         ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
}
```

Задача I. Түскі үзіліс

i нөмір бар элемент үшін алдыңғы "сәйкес" массив элементінің орнын сақтайтын prev[i] массивіне артықшылық беріңіз. "Сәйкес" деп ағымдағы элементпен айырмашылығы x-ден аспайтын кез-келген элементті атаймыз. Яғни, prev[i] = j, егер 1 <= j < i, |a[j] - a[i]| < = x. барлық қолайлы j арасында (және оларды сұрыптау жеткілікті), максимумды алайық. Егер бірде-бір сәйкес j болмаса, prev[i] = 0 қалдырыңыз. prev массивінің көмегімен сұрауларға жауап беру оңай.

Біз барлық l <= i <= r сұрыптап, prev[i] - ге қараймыз. Егер l <= prev[i] <= r орындалатын кем дегенде біреуі болса (яғни, i-нің сол жағындағы ең жақын сәйкес элемент диапазонға кіреді), онда жауап "YES", әйтпесе "NO".

Задача J. Атты табу
Бұл интерактивті тапсырма болды, онда интерактормен жұмыс істеу ережелерін сақтау керек және әр деректер шыққаннан кейін flush пәрмені арқылы ағындық буферді жаңарту қажет. Естеріңізге сала кетейін, C++ пәрменінде деректер ағынының жаңартуы да бар. Басқа тілдер үшін тапсырма шартында толығырақ жазылған.

Решение:
Бастапқыда әр торда ат болуы мүмкін деп елестетіп көрейік, яғни бастапқыда барлық торлар күдіктенеді.
Бірінші сұрақты іздейік, ең жақсысын таңдайық. Ең жақсы сұрақ - сұрақтан кейін күдікті торлардың ең аз саны қалатындай сұрақ. Сұрақ қойғаннан кейін біз қазылар алқасының бағдарламасы бізге дәл осындай жауап берген барлық торларды белгілейміз. Сол сияқты, күдікті торлардың саны біреу болмайынша сұрақтар қоюды жалғастырамыз.

C++ кодындағы мысал:
```
#pragma comment(linker, "/stack:2000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")

#include <bits/stdc++.h>

#define mp make_pair
#define pb push_back
#define IOS ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define ll long long
#define f first
#define s second

using namespace std;
int n, m, k, a[50][50];
const int INF = (int)(1e9 + 7);
const int MAXN = (int)(3e5 + 7);
int ans, x, y, b[10];
int dx[9] = {0, -2, -2, -1, -1, 1, 1, 2, 2};
```

```
int dy[9] = {0, -1, 1, -2, 2, -2, 2, -1, 1};

bool check(int l1, int r1, int l2, int r2, int x, int y){
    return (x >= l1 && x <= l2 && y >= r1 && y <= r2);
}

void rec(int z){

    int xx1, xx2, yy1, yy2, poos;
    int minn = INF;
    for (int i1 = 1; i1 <= 8; i1++)
        for (int j1 = 1; j1 <= 8; j1++)
            for (int i2 = i1; i2 <= 8; i2++)
                for (int j2 = j1; j2 <= 8; j2++){
                    for (int i = 0; i <= 9; i++)
                        b[i] = 0;
                    for (int x = 1; x <= 8; x++)
                        for (int y = 1; y <= 8; y++)
                            if (a[x][y] == 1){
                                k = 0;
                                for (int i = 0; i < 9; i++)
                                    if (check(i1, j1, i2, j2, x + dx[i], y + dy[i]))
                                        k++;
                                b[k]++;
                            }
                    int maxx = 0;
                    int pos = -1;
                    for (int i = 0; i <= 9; i++){
                        if (b[i] > maxx){
                            maxx = b[i];
                            pos = i;
                        }
                    }
                    if (maxx < minn)
                    {
                        minn = maxx;
                        poos = pos;
```

```
                    xx1 = i1;
                    xx2 = i2;
                    yy1 = j1;
                    yy2 = j2;
                }
            }
    int cntk;
     cout << "? " << char(xx1 + 'A' - 1) << yy1 << ' ' << char(xx2 + 'A' - 1)
<< yy2 << endl;
     cin >> cntk;

     for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1){
                k = 0;
                for (int i = 0; i < 9; i++)
                    if (check(xx1, yy1, xx2, yy2, x + dx[i], y + dy[i]))
                        k++;
                if (k != cntk)
                    a[x][y] = 0;
            }

    if (minn == 1){
        for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1)
                cout << "! " << char(x + 'A' - 1) << y << endl;
        return;
    }
    rec(z + 1);
}

int main() {
    IOS;

    for (int i = 1; i <= 8; i++)
        for (int j = 1; j <= 8; j++)
```

```
            a[i][j] = 1;

    rec(1);

    return 0;
}
```

Задача K. Түрлі-түсті кетсе

O(n^2 * k) :
(1, 1) және (n, n) ұяшықтарынан басқа ұяшықтарға оңтайлы жолдарды білу жеткілікті. Бұл жолдар барлық түстер үшін есептелуі керек. Біз әр түрлі түстердің санын k деп белгілейміз. Біз түсті сұрыптаймыз, әрқайсысы үшін динамиканы есептейміз және берілген түсті жасушалар үшін жауаптар аламыз.

O(n^4) :
O(d[i]^2) үшін і түсін шешуді үйреніңіз. Біз ұяшықтарды жол нөмірі бойынша, тең болған кезде баған нөмірі бойынша сұрыптаймыз. Біз бұрынғыдай динамиканы есептейміз, бірақ екі ауысудың орнына d[i] қарастырамыз.

O(n^3) :
Біз алгоритмді оңтайландырамыз шешуші түс O(d[i]^2) үшін O (d[i] * n) дейін.
Әр бағанда бізді тек қарастырылған төменгі ұяшық қызықтыратынын ескеріңіз. b[i] Қазіргі уақытта і бағандағы ең жақсы нәтиже болсын. Содан кейін біз жасаймыз f[i][j] = 1 + max(b[k]), (1 <= k <= j). b[i] = f[i][j] тағайындағаннан кейін.

O(n^2 * log(n)) (Толық шешім):
f[i][j] мәнін анықтау үшін кейбір b[i] префиксінен максимумды сұрау керек екенін ескеріңіз. Біз Максимум сұрауын және O(log(n)) үшін өзгерту операциясын жүзеге асырамыз. Ол үшін, мысалы, сегменттер ағашын қолдануға болады.

Решение 1: https://pastebin.com/8yyB84W3
Решение 2: https://pastebin.com/zND1yapW

Задача A. Получи слово
В этой задаче сначала надо было сверить длину вводимого слова и длину слова “impact”. Если длины различные, то ответ “NO”. В противном случае давайте посмотрим всевозможные перестановки букв слова “impact” если введена одна из этих перестановок, то только в этом случае ответ “YES”.

Задача B. Абду и массив
Посмотрим на каждый элемент массива. Если было встречено число x, то увеличим счётчик на 1. Если после просмотра всех элементов значение счётчика равно k, то ответ “YES” в противном случае ответ “NO”.

Задача C. <blank>
Запустим цикл от x + 1 до тех пор пока не найдём подходящие число. Каждое число будем разбивать на цифры и проверять, то что они состоят из одних и тех же цифр. Если число состоит из одинаковых цифр, то выведем это число и закроем программу.

Задача D. Отсутствующее число
Запомним все числа которые мы встречали. Так как числа n и k до 100000, то максимальный ответ не может превышать 200000. Запустим цикл от 1 до 200000 и будем считать количество тех чисел, которых не было в введенном массиве. Это можно делать с помощью метода “Двух указателей” или используя структуру данных map, или просто запомнив в массиве какие элементы встречались а какие нет.

Задача E. Ландшафт
Зафиксируем позицию i и для каждой позиции i будем запускать цикл j для нахождения холма или оврага. Назовём точкой перегиба ту координату, которая является вершиной холма или дном оврага. На отрезке от i до j должно быть не более одной точки перегиба. Перебрав все возможные i и j можно можно найти все подходящие холмы и овраги.
Пример кода на C++:

```
#include <bits/stdc++.h>
#define ll long long
#define ull unsigned long long
#define ld long double
#define uint unsigned int
#define f first
#define s second
#define pb push_back
#define mp make_pair
using namespace std;
const int INF = (int)(1e9 + 7);
const ll INFLL = (ll)(1e18 + 7);
const ld Pi = acos(-1.0);
const ld Eps = (ld)(1e-12);
mt19937 mrand(random_device{}());
int rnd(int x) { return mrand() % x;}
int a[6000];
int main(){
    srand(time(0));
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    //freopen("a.in", "r", stdin);
    //freopen("a.out", "w", stdout);
    //cout.precision(12);
    int n, ans = 1;
    cin >> n;
    for(int i = 1; i <= n; i ++){
        cin >> a[i];
    }
    for(int l = 1; l < n; l ++){
        bool typee = (a[l + 1] > a[l]), swapped = false;
        for(int r = l + 1; r <= n; r ++){
            if(a[r] == a[r - 1]){
                break;
            }
            if((a[r] > a[r - 1]) != typee){
                if(swapped){
                    break;
```

```
            }
            swapped = true;
            typee = !typee;
        }
        ans = max(ans, r - l + 1);
        //cout << l << ' ' << r << '\n';
    }
  }
  cout << ans << '\n';
  return 0;
}
```

Задача F. Красота массива

Посчитаем для каждого числа сколько раз оно встречалось, и его последнее вхождение r[a[i]]. Пройдём по всем числам и для каждого чисела, которое встречалось максимальное количество раз найдём значение r[a[i]] - i + 1 Сравним его с лучшим ответом, и для того чтобы не смотреть на это число ещё раз можно просто уменьшить на 1 количество вхождений этого числа в массив.

Пример кода на C++:

```
#include <bits/stdc++.h>
using namespace std;

int findShortestSubArray(vector<int> a) {
    map <int, int> cnt, r;
    int mx = 0;
    for(int i = 0; i < a.size(); ++i){
        cnt[a[i]]++, r[a[i]] = i;
        mx = max(mx, cnt[a[i]]);
    }
    int ans = a.size();
    for(int i = 0; i < a.size(); ++i) {
        if(mx == cnt[a[i]]){
            ans = min(ans, r[a[i]] - i + 1);
            cnt[a[i]]--;
        }
    }
```

```
        return ans;

int main(){
        int n;
        cin >> n;
        vector <int> a;
        for(int i = 1; i <= n; ++i){
                int x;
                cin >> x;
                a.push_back(x);
        }
        cout << findShortestSubArray(a);
        return 0;
}
```

Задача G. Таблица

Эта задача на реализацию. Тут надо было смоделировать то что сказано в условии. Будем бежать по всем элементам из первого столбца и последней строки. От каждого элемента будем двигаться вправо и вверх для прохода по этой диагонали, то есть если номер строки x, а номер столбца y, то уменьшаем x на 1 и увеличиваем y на 1 до тех пор пока x и y находятся в пределах матрицы. Выводим все элементы с координатами x и y. Для лучшего понимания смотрите код.

Пример кода на C++:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
```

```
typedef pair<int, int> pii;

const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;

int n;
int a[510][510];

void solve() {
	cin >> n;

	for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++)
		cin >> a[i][j];

	vector <int> ans;
	for (int it = 1; it <= 2 * n - 1; it++) {
		int x = min(it, n);
		int y = 1;
		if (it > n) y = it - n + 1;
		while (x >= 1 && y <= n) {
			ans.pb(a[x][y]);
			x--, y++;
		}
	}

	for (int x: ans)
		cout << x << ' ';
}

int main() {
	ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
```

}

Задача H. Максимум единиц

В этой задаче нужно было использовать либо бинарный поиск, либо два указателя. Для каждой позиции i найдём бинарным поиском максимальную позицию j такую что между i и j не более k нулей. Количество нулей можно поддерживать с помощью массива префикс сумм. Среди всех i и j найдём самый длинный отрезок.

Пример кода на C++:

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

#define pb push_back
#define fi first
#define se second
#define all(x) x.begin(), x.end()
#define sz(x) (int)x.size()
#define en '\n'

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

const int N = 1e6, M = N + 7;
const int MOD = 1e9 + 7;

void solve() {
        int n; cin >> n;
        vector <int> a(n);
        for (int i = 0; i < n; i++)
                cin >> a[i];
        int k; cin >> k;

    int l = 0, r = n + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        bool ok = 0;
        int cnt = 0;
```

```
        for (int i = 0; i < m - 1; i++) cnt += a[i];
        for (int i = m - 1; i < n; i++) {
            cnt += a[i];
            if (m - cnt <= k) ok = 1;
            cnt -= a[i - m + 1];
        }
        if (ok) l = m;
        else r = m;
    }

    cout << l;
}

int main() {
         ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int tt = 1;
    //cin >> tt;
    while (tt--) {
        solve();
        cout << en;
    }
}
```

Задача I. Обеденный перерыв

Предпосчитаем массив prev[i], который будет хранить позицию предыдущего «подходящего» элемента массива для элемента номер i. «Подходящим» мы называем любой элемент, разность которого с текущим элментом не превышает x. То есть, prev[i] = j, если 1 <= j < i, |a[j] - a[i]| <= x. Среди всех подходящих j (А их достаточно просто перебрать), возьмем максимальный. Если нет ни одного подходящего j, оставим prev[i] = 0.

С помощью массива prev легко отвечать на запросы. Переберём все l <= i <= r и посмотрим на prev[i]. Если есть хотя бы один такой i, что выполняется l <= prev[i] <= r (То есть, ближайший подходящий элемент слева от i входит в диапазон), то ответ «YES», иначе – «NO».

Задача J. Найти коня
Это была интерактивная задача в которой необходимо поддерживать правила работы с интерактором и после каждого вывода данных обновлять потоковый буфер с помощью команды flush. Напоминаю, что в C++ в команде endl так же присутствует обновление потока данных. Для других языков подробнее написано в условии задачи.

Решение:
Изначально представим, что в каждой клетки мог находится конь, то есть изначально все клетки под подозрением.
Давайте переберём первый вопрос и выберем лучший. Лучшим вопросом назовём тот вопрос после которого в худшем случае останется минимальное количество подозреваемых клеток.
Задав вопрос пометим все клетки которые могли дать точно такой же ответ, как вам выдала программа жюри. Точно такими же действиями продолжим задавать вопросы до тех пор пока в подозреваемых клетках не останется только одна клетка.

Пример кода на C++:
```
#pragma comment(linker, "/stack:2000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")

#include <bits/stdc++.h>

#define mp make_pair
#define pb push_back
#define IOS ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define ll long long
#define f first
#define s second

using namespace std;
int n, m, k, a[50][50];
const int INF = (int)(1e9 + 7);
const int MAXN = (int)(3e5 + 7);
```

```
int ans, x, y, b[10];
int dx[9] = {0, -2, -2, -1, -1, 1, 1, 2, 2};
int dy[9] = {0, -1, 1, -2, 2, -2, 2, -1, 1};

bool check(int l1, int r1, int l2, int r2, int x, int y){
    return (x >= l1 && x <= l2 && y >= r1 && y <= r2);
}

void rec(int z){

    int xx1, xx2, yy1, yy2, poos;
    int minn = INF;
    for (int i1 = 1; i1 <= 8; i1++)
        for (int j1 = 1; j1 <= 8; j1++)
            for (int i2 = i1; i2 <= 8; i2++)
                for (int j2 = j1; j2 <= 8; j2++){
                    for (int i = 0; i <= 9; i++)
                        b[i] = 0;
                    for (int x = 1; x <= 8; x++)
                        for (int y = 1; y <= 8; y++)
                            if (a[x][y] == 1){
                                k = 0;
                                for (int i = 0; i < 9; i++)
                                    if (check(i1, j1, i2, j2, x + dx[i], y + dy[i]))
                                        k++;
                                b[k]++;
                            }
                    int maxx = 0;
                    int pos = -1;
                    for (int i = 0; i <= 9; i++){
                        if (b[i] > maxx){
                            maxx = b[i];
                            pos = i;
                        }
                    }
                    if (maxx < minn)
                    {
```

```
                    minn = maxx;
                    poos = pos;
                    xx1 = i1;
                    xx2 = i2;
                    yy1 = j1;
                    yy2 = j2;
                }
            }
    int cntk;
     cout << "? " << char(xx1 + 'A' - 1) << yy1 << ' ' << char(xx2 + 'A' - 1)
<< yy2 << endl;
     cin >> cntk;

     for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1){
                k = 0;
                for (int i = 0; i < 9; i++)
                    if (check(xx1, yy1, xx2, yy2, x + dx[i], y + dy[i]))
                        k++;
                if (k != cntk)
                    a[x][y] = 0;
            }

    if (minn == 1){
        for (int x = 1; x <= 8; x++)
        for (int y = 1; y <= 8; y++)
            if (a[x][y] == 1)
                cout << "! " << char(x + 'A' - 1) << y << endl;
        return;
    }
    rec(z + 1);
}

int main() {
    IOS;
```

```
    for (int i = 1; i <= 8; i++)
        for (int j = 1; j <= 8; j++)
            a[i][j] = 1;

    rec(1);

    return 0;
}
```

Задача K. Цветная таблица

$O(n^2 * k)$:
Достаточно знать оптимальные пути от клеток $(1, 1)$ и (n, n) до всех остальных. Данные пути необходимо вычислить для всех цветов. Обозначим количество различных цветов как k.
Переберем цвет, для каждого вычислим динамику и получим ответы для клеток данного цвета.

$O(n^4)$:
Обозначим как $d[i]$ количество клеток цвета i. Научимся решать для цвета i за $O(d[i]^2)$. Отсортируем клетки по номеру строку, при равенстве - по номеру столбца. Будем вычислять такую же динамику как и до этого, но вместо двух переходов будем рассматривать $d[i]$.

$O(n^3)$:
Оптимизируем алгоритм решающий цвет за $O(d[i]^2)$ до $O(d[i] * n)$. Заметим, что в каждом столбце нас интересует только нижняя клетка из уже рассмотренных. Пусть $b[i]$ - лучший на текущий момент результат в столбце i. Тогда создадим $f[i][j] = 1 + max(b[k]), (1 <= k <= j)$. После присвоим $b[i] = f[i][j]$.

$O(n^2 * log(n))$ (Полное решение):

Заметим, что для определения значения $f[i][j]$ необхидом запросить максимум на некотором префиксе $b[i]$. Реализуем запрос максимума и операцию изменения за $O(log(n))$. Для этого к примеру, можно использовать дерево отрезков.

Решение 1: https://pastebin.com/8yyB84W3
Решение 2: https://pastebin.com/zND1yapW