Educational Codeforces Round 75 (Rated for Div. 2)

A. Broken Keyboard

1 second, 256 megabytes

Recently Polycarp noticed that some of the buttons of his keyboard are malfunctioning. For simplicity, we assume that Polycarp's keyboard contains 26 buttons (one for each letter of the Latin alphabet). Each button is either working fine or malfunctioning.

To check which buttons need replacement, Polycarp pressed some buttons in sequence, and a string s appeared on the screen. When Polycarp presses a button with character c, one of the following events happened:

- if the button was working correctly, a character c appeared at the end of the string Polycarp was typing;
- if the button was malfunctioning, **two** characters c appeared at the end of the string.

For example, suppose the buttons corresponding to characters `a` and `c` are working correctly, and the button corresponding to `b` is malfunctioning. If Polycarp presses the buttons in the order `a`, `b`, `a`, `c`, `a`, `b`, `a`, then the string he is typing changes as follows: `a` → `abb` → `abba` → `abbac` → `abbaca` → `abbacabb` → `abbacabba`.

You are given a string s which appeared on the screen after Polycarp pressed some buttons. Help Polycarp to determine which buttons are working correctly for sure (that is, this string could not appear on the screen if any of these buttons was malfunctioning).

You may assume that the buttons don't start malfunctioning when Polycarp types the string: each button either works correctly throughout the whole process, or malfunctions throughout the whole process.

Input

The first line contains one integer t ($1 \le t \le 100$) — the number of test cases in the input.

Then the test cases follow. Each test case is represented by one line containing a string s consisting of no less than 1 and no more than 500 lowercase Latin letters.

Output

For each test case, print one line containing a string res. The string res should contain all characters which correspond to buttons that work correctly **in alphabetical order, without any separators or repetitions**. If all buttons may malfunction, res should be empty.

input
4 a zzaaz ccff cbddbb

output
a z bc

B. Binary Palindromes

2 seconds, 256 megabytes

A palindrome is a string t which reads the same backward as forward (formally, $t[i] = t[|t| + 1 - i]$ for all $i \in [1, |t|]$). Here $|t|$ denotes the length of a string t. For example, the strings `010`, `1001` and `0` are palindromes.

You have n binary strings $s_1, s_2, \dots, s_n$ (each s_i consists of zeroes and/or ones). You can swap any pair of characters any number of times (possibly, zero). Characters can be either from the same string or from different strings — there are no restrictions.

Formally, in one move you:

- choose four integer numbers x, a, y, b such that $1 \le x, y \le n$ and $1 \le a \le |s_x|$ and $1 \le b \le |s_y|$ (where x and y are string indices and a and b are positions in strings s_x and s_y respectively),
- swap (exchange) the characters $s_x[a]$ and $s_y[b]$.

What is the maximum number of strings you can make palindromic simultaneously?

Input

The first line contains single integer Q ($1 \le Q \le 50$) — the number of test cases.

The first line on each test case contains single integer n ($1 \le n \le 50$) — the number of binary strings you have.

Next n lines contains binary strings $s_1, s_2, \dots, s_n$ — one per line. It's guaranteed that $1 \le |s_i| \le 50$ and all strings constist of zeroes and/or ones.

Output

Print Q integers — one per test case. The i-th integer should be the maximum number of palindromic strings you can achieve simultaneously performing zero or more swaps on strings from the i-th test case.

input
4 1 0 3 1110 100110 010101 2 11111 000001 2 001 11100111

output
1 2 2 2

In the first test case, s_1 is palindrome, so the answer is 1.

In the second test case you can't make all three strings palindromic at the same time, but you can make any pair of strings palindromic. For example, let's make $s_1 = 0110$, $s_2 = 111111$ and $s_3 = 010000$.

In the third test case we can make both strings palindromic. For example, $s_1 = 11011$ and $s_2 = 100001$.

In the last test case s_2 is palindrome and you can make s_1 palindrome, for example, by swapping $s_1[2]$ and $s_1[3]$.

C. Minimize The Integer

2 seconds, 256 megabytes

You are given a huge integer a consisting of n digits (n is between 1 and $3 \cdot 10^5$, inclusive). It may contain leading zeros.

You can swap two digits on adjacent (neighboring) positions if the swapping digits are of different parity (that is, they have different remainders when divided by 2).

For example, if a = 032867235 you can get the following integers in a single operation:

- 302867235 if you swap the first and the second digits;
- 023867235 if you swap the second and the third digits;
- 032876235 if you swap the fifth and the sixth digits;
- 032862735 if you swap the sixth and the seventh digits;
- 032867325 if you swap the seventh and the eighth digits.

Note, that you can't swap digits on positions 2 and 4 because the positions are not adjacent. Also, you can't swap digits on positions 3 and 4 because the digits have the same parity.

You can perform any number (possibly, zero) of such operations.

Find the minimum integer you can obtain.

Note that the resulting integer also may contain leading zeros.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases in the input.

The only line of each test case contains the integer a, its length n is between 1 and $3 \cdot 10^5$, inclusive.

It is guaranteed that the sum of all values n does not exceed $3 \cdot 10^5$.

Output

For each test case print line — the minimum integer you can obtain.

input
3 0709 1337 246432

output
0079 1337 234642

In the first test case, you can perform the following sequence of operations (the pair of swapped digits is highlighted): 0**70**9 → 0079.

In the second test case, the initial integer is optimal.

In the third test case you can perform the following sequence of operations: 246**43**2 → 24**63**42 → 2**43**642 → 234642.

D. Salary Changing

3 seconds, 256 megabytes

You are the head of a large enterprise. n people work at you, and n is odd (i.e. n is not divisible by 2).

You have to distribute salaries to your employees. Initially, you have s dollars for it, and the i-th employee should get a salary from l_i to r_i dollars. You have to distribute salaries in such a way that the median salary is *maximum possible*.

To find the median of a sequence of odd length, you have to sort it and take the element in the middle position after sorting. For example:

- the median of the sequence [5, 1, 10, 17, 6] is 6,
- the median of the sequence [1, 2, 1] is 1.

It is guaranteed that you have enough money to pay the minimum salary, i.e $l_1 + l_2 + \dots + l_n \le s$.

Note that you don't have to spend all your s dollars on salaries.

You have to answer t test cases.

Input

The first line contains one integer t ($1 \le t \le 2 \cdot 10^5$) — the number of test cases.

The first line of each query contains two integers n and s ($1 \le n < 2 \cdot 10^5$, $1 \le s \le 2 \cdot 10^{14}$) — the number of employees and the amount of money you have. The value n is not divisible by 2.

The following n lines of each query contain the information about employees. The i-th line contains two integers l_i and r_i ($1 \le l_i \le r_i \le 10^9$).

It is guaranteed that the sum of all n over all queries does not exceed $2 \cdot 10^5$.

It is also guaranteed that you have enough money to pay the minimum salary to each employee, i.e. $\sum_{i=1}^{n} l_i \le s$.

Output

For each test case print one integer — the maximum median salary that you can obtain.

input
3 3 26 10 12 1 4 10 11 1 1337 1 1000000000 5 26 4 4 2 4 6 8 5 6 2 7

output
11 1337 6

In the first test case, you can distribute salaries as follows: $sal_1 = 12, sal_2 = 2, sal_3 = 11$ (sal_i is the salary of the i-th employee). Then the median salary is 11.

In the second test case, you have to pay 1337 dollars to the only employee.

In the third test case, you can distribute salaries as follows: $sal_1 = 4, sal_2 = 3, sal_3 = 6, sal_4 = 6, sal_5 = 7$. Then the median salary is 6.

E1. Voting (Easy Version)

2 seconds, 512 megabytes

The only difference between easy and hard versions is constraints.

Now elections are held in Berland and you want to win them. More precisely, you want everyone to vote for you.

There are n voters, and two ways to convince each of them to vote for you. The first way to convince the i-th voter is to pay him p_i coins. The second way is to make m_i other voters vote for you, and the i-th voter will vote for free.

Moreover, the process of such voting takes place in several steps. For example, if there are five voters with $m_1 = 1$, $m_2 = 2$, $m_3 = 2$, $m_4 = 4$, $m_5 = 5$, then you can buy the vote of the fifth voter, and eventually everyone will vote for you. Set of people voting for you will change as follows: $5 \to 1, 5 \to 1, 2, 3, 5 \to 1, 2, 3, 4, 5$.

Calculate the minimum number of coins you have to spend so that everyone votes for you.

Input

The first line contains one integer t ($1 \le t \le 5000$) — the number of test cases.

The first line of each test case contains one integer n ($1 \le n \le 5000$) — the number of voters.

The next n lines contains the description of voters. i-th line contains two integers m_i and p_i ($1 \le p_i \le 10^9, 0 \le m_i < n$).

It is guaranteed that the sum of all n over all test cases does not exceed 5000.

Output

For each test case print one integer — the minimum number of coins you have to spend so that everyone votes for you.

input
3 3 1 5 2 10 2 8 7 0 1 3 1 1 1 6 1 1 1 4 1 4 1 6 2 6 2 3 2 8 2 7 4 4 5 5

output
8 0 7

In the first test case you have to buy vote of the third voter. Then the set of people voting for you will change as follows: $3 \to 1, 3 \to 1, 2, 3$.

In the second example you don't need to buy votes. The set of people voting for you will change as follows:
$1 \to 1, 3, 5 \to 1, 2, 3, 5 \to 1, 2, 3, 5, 6, 7 \to 1, 2, 3, 4, 5, 6, 7$.

In the third test case you have to buy votes of the second and the fifth voters. Then the set of people voting for you will change as follows:
$2, 5 \to 1, 2, 3, 4, 5 \to 1, 2, 3, 4, 5, 6$.

E2. Voting (Hard Version)

2 seconds, 256 megabytes

The only difference between easy and hard versions is constraints.

Now elections are held in Berland and you want to win them. More precisely, you want everyone to vote for you.

There are n voters, and two ways to convince each of them to vote for you. The first way to convince the i-th voter is to pay him p_i coins. The second way is to make m_i other voters vote for you, and the i-th voter will vote for free.

Moreover, the process of such voting takes place in several steps. For example, if there are five voters with $m_1 = 1$, $m_2 = 2$, $m_3 = 2$, $m_4 = 4$, $m_5 = 5$, then you can buy the vote of the fifth voter, and eventually everyone will vote for you. Set of people voting for you will change as follows: $5 \to 1, 5 \to 1, 2, 3, 5 \to 1, 2, 3, 4, 5$.

Calculate the minimum number of coins you have to spend so that everyone votes for you.

Input

The first line contains one integer t ($1 \le t \le 2 \cdot 10^5$) — the number of test cases.

The first line of each test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of voters.

The next n lines contains the description of voters. i-th line contains two integers m_i and p_i ($1 \le p_i \le 10^9, 0 \le m_i < n$).

It is guaranteed that the sum of all n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case print one integer — the minimum number of coins you have to spend so that everyone votes for you.

input
3 3 1 5 2 10 2 8 7 0 1 3 1 1 1 6 1 1 1 4 1 4 1 6 2 6 2 3 2 8 2 7 4 4 5 5

output
8 0 7

In the first test case you have to buy vote of the third voter. Then the set of people voting for you will change as follows: $3 \to 1, 3 \to 1, 2, 3$.

In the second example you don't need to buy votes. The set of people voting for you will change as follows:
$1 \to 1, 3, 5 \to 1, 2, 3, 5 \to 1, 2, 3, 5, 6, 7 \to 1, 2, 3, 4, 5, 6, 7$.

In the third test case you have to buy votes of the second and the fifth voters. Then the set of people voting for you will change as follows:
$2, 5 \to 1, 2, 3, 4, 5 \to 1, 2, 3, 4, 5, 6$.

F. Red-White Fence

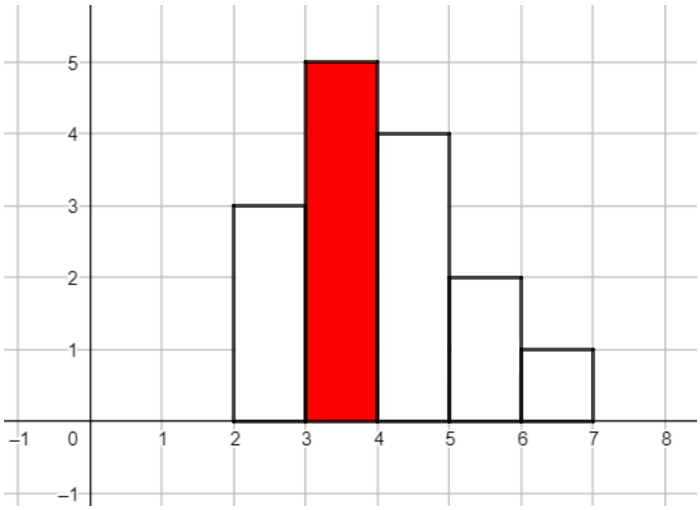
5 seconds, 512 megabytes

Polycarp wants to build a fence near his house. He has n white boards and k red boards he can use to build it. Each board is characterised by its length, which is an integer.

A good fence should consist of **exactly one** red board and several (possibly zero) white boards. The red board should be the longest one in the fence (every white board used in the fence should be strictly shorter), and the sequence of lengths of boards should be ascending before the red board and descending after it. Formally, if m boards are used, and their lengths are $l_1, l_2, \ldots, l_m$ in the order they are placed in the fence, from left to right (let's call this array $[l_1, l_2, \ldots, l_m]$ *the array of lengths*), the following conditions should hold:

- there should be exactly one red board in the fence (let its index be j);
- for every $i \in [1, j - 1]$ $l_i < l_{i+1}$;
- for every $i \in [j, m - 1]$ $l_i > l_{i+1}$.

When Polycarp will build his fence, he will place all boards from left to right on the same height of 0, without any gaps, so these boards compose a polygon:



Example: a fence with [3, 5, 4, 2, 1] as the array of lengths. The second board is red. The perimeter of the fence is 20.

Polycarp is interested in fences of some special perimeters. He has q **even** integers he really likes (these integers are Q_1, Q_2, ..., Q_q), and for every such integer Q_i, he wants to calculate the number of different fences with perimeter Q_i he can build (two fences are considered different if their **arrays of lengths** are different). Can you help him calculate these values?

Input

The first line contains two integers n and k ($1 \le n \le 3 \cdot 10^5$, $1 \le k \le 5$) — the number of white and red boards Polycarp has.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le 3 \cdot 10^5$) — the lengths of white boards Polycarp has.

The third line contains k integers b_1, b_2, ..., b_k ($1 \le b_i \le 3 \cdot 10^5$) — the lengths of red boards Polycarp has. **All b_i are distinct**.

The fourth line contains one integer q ($1 \le q \le 3 \cdot 10^5$) — the number of special integers.

The fifth line contains q integers Q_1, Q_2, ..., Q_q ($4 \le Q_i \le 12 \cdot 10^5$, every Q_i is even) — the special integers Polycarp likes.

Output

For each Q_i, print one integer — the number of good fences with perimeter Q_i Polycarp can build, taken modulo 998244353.

input
5 2 3 3 1 1 1 2 4 7 6 8 10 12 14 16 18

output
1 2 2 4 6 4 1

input
5 5 1 2 3 4 5 1 2 3 4 5 4 4 8 10 14

output
1 3 5 20

Possible fences in the first example denoted by their arrays of lengths (the length of the red board is highlighted):

- with perimeter 6: [**2**];
- with perimeter 8: [1, **2**], [**2**, 1];
- with perimeter 10: [1, **2**, 1], [**4**];
- with perimeter 12: [1, **4**], [3, **4**], [**4**, 1], [**4**, 3];
- with perimeter 14: [1, **4**, 1], [1, **4**, 3], [3, **4**, 1], [3, **4**, 3], [1, 3, **4**], [**4**, 3, 1];
- with perimeter 16: [1, **4**, 3, 1], [3, **4**, 3, 1], [1, 3, **4**, 1], [1, 3, **4**, 3];
- with perimeter 18: [1, 3, **4**, 3, 1].