Codeforces Round 980 (Div. 2)

A. Profitable Interest Rate

1 second, 256 megabytes

Alice has a coins. She can open a bank deposit called "Profitable", but the minimum amount required to open this deposit is b coins.

There is also a deposit called "Unprofitable", which can be opened with **any** amount of coins. Alice noticed that if she opens the "Unprofitable" deposit with x coins, the minimum amount required to open the "Profitable" deposit decreases by $2x$ coins. However, these coins cannot later be deposited into the "Profitable" deposit.

Help Alice determine the maximum number of coins she can deposit into the "Profitable" deposit if she first deposits some amount of coins (possibly 0) into the "Unprofitable" deposit. If Alice can never open the "Profitable" deposit, output 0.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

A single line of each test case contains two integers a and b ($1 \le a, b \le 10^9$) — the number of coins Alice has and the initial minimum amount required to open the "Profitable" deposit.

Output

For each test case, output a single integer — the maximum number of coins that Alice can deposit into the "Profitable" deposit. If Alice can never open the "Profitable" deposit, output 0.

input
5 10 5 7 9 5 100 1 1 1 2

output
10 5 0 1 0

In the first test case, $a \ge b$, so Alice can immediately open the "Profitable" deposit with all 10 coins.

In the second test case, Alice can open the "Unprofitable" deposit with 2 coins. Then she will have 5 coins left, but the minimum amount required to open the "Profitable" deposit will decrease by 4 coins, making it equal to 5 coins. Thus, Alice will be able to open the "Profitable" deposit with 5 coins.

In the third test case, Alice will not be able to open the "Profitable" deposit.

B. Buying Lemonade

1 second, 256 megabytes

There is a vending machine that sells lemonade. The machine has a total of n slots. You know that initially, the i-th slot contains a_i cans of lemonade. There are also n buttons on the machine, each button corresponds to a slot, with exactly one button corresponding to each slot. Unfortunately, the labels on the buttons have worn off, so you **do not know** which button corresponds to which slot.

When you press the button corresponding to the i-th slot, one of two events occurs:

- If there is a can of lemonade in the i-th slot, it will drop out and you will take it. At this point, the number of cans in the i-th slot decreases by 1.
- If there are no cans of lemonade left in the i-th slot, nothing will drop out.

After pressing, the can drops out so quickly that it is impossible to track from which slot it fell. The contents of the slots are hidden from your view, so you cannot see how many cans are left in each slot. The only thing you know is the initial number of cans in the slots: $a_1, a_2, \ldots, a_n$.

Determine the minimum number of button presses needed to guarantee that you receive at least k cans of lemonade.

Note that you can adapt your strategy during the button presses based on whether you received a can or not. It is guaranteed that there are at least k cans of lemonade in total in the machine. In other words, $k \le a_1 + a_2 + \ldots + a_n$.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le n \le 2 \cdot 10^5$, $1 \le k \le 10^9$) — the number of slots in the machine and the required number of cans of lemonade.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the number of cans in the slots.

It is guaranteed that $k \le a_1 + a_2 + \ldots + a_n$, meaning there are at least k cans of lemonade in the machine.

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

Output

For each test case, output a single integer — the minimum number of button presses needed to guarantee that you receive at least k cans of lemonade.

input
5 2 1 1 1 2 2 1 2 3 4 2 1 3 10 50 1 1 3 8 8 9 12 13 27 27 2 1000000000 1000000000 500000000

output
1 2 5 53 1000000000

In the first test case, we can simply press the first button and receive one can of lemonade.

In the second test case, we can press each button once and guarantee that we receive 2 cans of lemonade. Note that if we simply press one button twice, we might not be lucky, and that button could correspond to the first slot, in which case we would only receive 1 can of lemonade for two presses.

In the third test case, one of the optimal strategies is as follows:

Press the first button twice. After the first press, a can of lemonade will definitely drop out. Then there are two options:

- If no can of lemonade drops after the second press, we know that this button must correspond to the second slot, since $a_2 = 1$ and $a_1, a_3 > 1$. Then we can press the second button twice and the third button once. Since $a_1, a_3 \ge 2$, we will definitely receive three cans of lemonade for these three presses. Thus, after 5 presses, we will have 4 cans of lemonade.
- If a can of lemonade drops after the second press, we can make one press on the second button and one press on the third button. After each of these presses, we will definitely receive a can of lemonade. Thus, after 4 presses, we will have 4 cans of lemonade.

It can be shown that it is impossible to guarantee receiving 4 cans of lemonade with only 4 presses, so the answer is 5.

C. Concatenation of Arrays

2 seconds, 256 megabytes

You are given n arrays $a_1, \ldots, a_n$. The length of each array is two. Thus, $a_i = [a_{i,1}, a_{i,2}]$. You need to concatenate the arrays into a single array of length $2n$ such that the number of inversions[†] in the resulting array is minimized. Note that you **do not need** to count the actual number of inversions.

More formally, you need to choose a permutation[‡] p of length n, so that the array $b = [a_{p_1,1}, a_{p_1,2}, a_{p_2,1}, a_{p_2,2}, \ldots, a_{p_n,1}, a_{p_n,2}]$ contains as few inversions as possible.

[†] The number of inversions in an array c is the number of pairs of indices i and j such that $i < j$ and $c_i > c_j$.

[‡] A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 10^5$) — the number of arrays.

Each of the following n lines contains two integers $a_{i,1}$ and $a_{i,2}$ ($1 \le a_{i,j} \le 10^9$) — the elements of the i-th array.

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

Output

For each test case, output $2n$ integers — the elements of the array you obtained. If there are multiple solutions, output any of them.

input

```
4
2
1 4
2 3
3
3 2
4 3
2 1
5
5 10
2 3
9 6
4 1
8 7
1
10 20
```

output

```
2 3 1 4
2 1 3 2 4 3
4 1 2 3 5 10 8 7 9 6
10 20
```

In the first test case, we concatenated the arrays in the order $2, 1$. Let's consider the inversions in the resulting array $b = [2, 3, 1, 4]$:

- $i = 1$, $j = 3$, since $b_1 = 2 > 1 = b_3$;
- $i = 2$, $j = 3$, since $b_2 = 3 > 1 = b_3$.

Thus, the number of inversions is 2. It can be proven that this is the minimum possible number of inversions.

In the second test case, we concatenated the arrays in the order $3, 1, 2$. Let's consider the inversions in the resulting array $b = [2, 1, 3, 2, 4, 3]$:

- $i = 1$, $j = 2$, since $b_1 = 2 > 1 = b_2$;
- $i = 3$, $j = 4$, since $b_3 = 3 > 2 = b_4$;
- $i = 5$, $j = 6$, since $b_5 = 4 > 3 = b_6$.

Thus, the number of inversions is 3. It can be proven that this is the minimum possible number of inversions.

In the third test case, we concatenated the arrays in the order $4, 2, 1, 5, 3$.

D. Skipping

2 seconds, 256 megabytes

It is already the year 3024, ideas for problems have long run out, and the olympiad now takes place in a modified individual format. The olympiad consists of n problems, numbered from 1 to n. The i-th problem has its own score a_i and a certain parameter b_i ($1 \le b_i \le n$).

Initially, the testing system gives the participant the **first** problem. When the participant is given the i-th problem, they have two options:

- They can submit the problem and receive a_i points;
- They can skip the problem, in which case they will never be able to submit it.

Then, the testing system selects the next problem for the participant from problems with indices j, such that:

- If he submitted the i-th problem, it looks at problems with indices $j < i$;
- If he skipped the i-th problem, it looks at problems with indices $j \le b_i$.

Among these problems, it selects the problem with the **maximum** index that it has **not previously given** to the participant (he has neither submitted nor skipped it before). If there is no such problem, then the competition for the participant **ends**, and their result is equal to the sum of points for all submitted problems. In particular, if the participant submits the first problem, then the competition for them ends. Note that the participant receives each problem **at most once**.

Prokhor has prepared thoroughly for the olympiad, and now he can submit any problem. Help him determine the maximum number of points he can achieve.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 4 \cdot 10^5$) — the number of problems in the olympiad.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the scores of the problems.

The third line of each test case contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$) — the parameters of the problems.

It is guaranteed that the sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output

For each test case, output a single integer — the maximum number of points that Prokhor can achieve.

input

```
4
2
15 16
2 1
5
10 10 100 100 1000
3 4 1 1 1
3
100 49 50
3 2 2
4
100 200 300 1000
2 3 4 1
```

output

```
16
200
100
1000
```

In the first test case, Prokhor can skip the first problem; then he will receive the problem with index $b_1 = 2$. Prokhor can submit it and receive $a_2 = 16$ points. After that, the competition will end because Prokhor has already received all problems. Note that if Prokhor submits the first problem, he will receive $a_1 = 15$ points, but the competition will end immediately.

In the second test case, Prokhor can skip the first problem; then he will receive the problem with index $b_1 = 3$. Prokhor can submit it and receive $a_3 = 100$ points. After that, Prokhor will receive the second problem, which he can skip to receive the problem with index $b_2 = 4$. Prokhor can submit the fourth problem and receive another $a_4 = 100$ points. After that, the competition ends because Prokhor has already received all problems with indices not exceeding 4. Thus, Prokhor will receive a total of 200 points.

In the third test case, Prokhor can submit the first problem and receive 100 points, after which the competition will end immediately.

E. C+K+S

3 seconds, 256 megabytes

You are given two strongly connected[†] directed graphs, each with exactly n vertices, but possibly different numbers of edges. Upon closer inspection, you noticed an important feature — the length of any cycle in these graphs is divisible by k.

Each of the $2n$ vertices belongs to exactly one of two types: *incoming* or *outgoing*. For each vertex, its type is known to you.

You need to determine whether it is possible to draw exactly n directed edges between the source graphs such that the following four conditions are met:

- The ends of any added edge lie in different graphs.
- From each outgoing vertex, exactly one added edge originates.
- Into each incoming vertex, exactly one added edge enters.
- In the resulting graph, the length of any cycle is divisible by k.

[†] A strongly connected graph is a graph in which there is a path from every vertex to every other vertex.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and k ($2 \le k \le n \le 2 \cdot 10^5$) — the number of vertices in each graph and the value by which the length of each cycle is divisible.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($a_i \in \{0, 1\}$). If $a_i = 0$, then vertex i of the first graph is incoming. If $a_i = 1$, then vertex i of the first graph is outgoing.

The third line of each test case contains a single integer m_1 ($1 \le m_1 \le 5 \cdot 10^5$) — the number of edges in the first graph.

The next m_1 lines contain descriptions of the edges of the first graph. The i-th of them contains two integers v_i and u_i ($1 \le v_i, u_i \le n$) — an edge in the first graph leading from vertex v_i to vertex u_i.

Next, in the same format, follows the description of the second graph.

The next line contains n integers $b_1, b_2, \ldots, b_n$ ($b_i \in \{0, 1\}$). If $b_i = 0$, then vertex i of the second graph is incoming. If $b_i = 1$, then vertex i of the second graph is outgoing.

The next line contains a single integer m_2 ($1 \le m_2 \le 5 \cdot 10^5$) — the number of edges in the second graph.

The next m_2 lines contain descriptions of the edges of the second graph. The i-th of them contains two integers v_i and u_i ($1 \le v_i, u_i \le n$) — an edge in the second graph leading from vertex v_i to vertex u_i.

It is guaranteed that both graphs are strongly connected, and the lengths of all cycles are divisible by k.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$. It is guaranteed that the sum of m_1 and the sum of m_2 over all test cases does not exceed $5 \cdot 10^5$.

Output

For each test case, output "YES" (without quotes) if it is possible to draw n new edges such that all conditions are met, and "NO" (without quotes) otherwise.

You may output the answer in any case (for example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as a positive answer).

input

```
3
4 2
1 0 0 1
4
1 2
2 3
3 4
4 1
1 0 0 1
4
1 3
3 2
2 4
4 1
3 3
0 0 0
3
1 2
2 3
3 1
1 1 0
3
1 2
2 3
3 1
4 2
1 1 1 1
4
1 2
2 3
3 4
4 1
0 0 0 0
6
1 2
2 1
1 3
3 1
1 4
4 1
```

output

```
YES
NO
YES
```

In the first test case, it is possible to draw edges from the first graph to the second graph as $(1, 3)$ and $(4, 2)$ (the first number in the pair is the vertex number in the first graph, and the second number in the pair is the vertex number in the second graph), and from the second graph to the first graph as $(1, 2)$, $(4, 3)$ (the first number in the pair is the vertex number in the second graph, and the second number in the pair is the vertex number in the first graph).

In the second test case, there are a total of 4 incoming vertices and 2 outgoing vertices, so it is not possible to draw 3 edges.

F. Many Games

2 seconds, 256 megabytes

Recently, you received a rare ticket to the only casino in the world where you can actually earn something, and you want to take full advantage of this opportunity.

The conditions in this casino are as follows:

- There are a total of n games in the casino.
- You can play each game **at most once**.
- Each game is characterized by two parameters: p_i ($1 \le p_i \le 100$) and w_i — the probability of winning the game in percentage and the winnings for a win.
- If you lose in any game you decide to play, you will receive nothing at all (even for the games you won).

You need to choose a set of games in advance that you will play in such a way as to maximize the expected value of your winnings.

In this case, if you choose to play the games with indices $i_1 < i_2 < \ldots < i_k$, you will win in all of them with a probability of $\prod_{j=1}^{k} \frac{p_{i_j}}{100}$, and in that case, your winnings will be equal to $\sum_{j=1}^{k} w_{i_j}$.

That is, the expected value of your winnings will be $\left(\prod_{j=1}^{k} \frac{p_{i_j}}{100}\right) \cdot \left(\sum_{j=1}^{k} w_{i_j}\right)$.

To avoid going bankrupt, the casino owners have limited the expected value of winnings for each individual game. Thus, for all i ($1 \le i \le n$), it holds that $w_i \cdot p_i \le 2 \cdot 10^5$.

Your task is to find the maximum expected value of winnings that can be obtained by choosing some set of games in the casino.

Input

The first line contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of games offered to play.

The i-th of the following n lines contains two integers p_i and w_i ($1 \le p_i \le 100$, $1 \le w_i, p_i \cdot w_i \le 2 \cdot 10^5$) — the probability of winning and the size of the winnings in the i-th game.

Output

Output a single number — the maximum expected value of winnings in the casino that can be obtained by choosing some subset of games.

Your answer will be accepted if the relative or absolute error does not exceed 10^{-6}. Formally, if a is your answer and b is the jury's answer, it will be accepted if $\frac{|a-b|}{\max(b,1)} \le 10^{-6}$.

input

```
3
80 80
70 100
50 200
```

output

```
112.00000000
```

input

```
2
100 1
100 1
```

output

```
2.00000000
```

input

```
4
1 100
2 1000
2 100
3 1
```

output

```
20.00000000
```

input

```
5
34 804
78 209
99 191
61 439
90 79
```

output

```
395.20423800
```

In the first example, you can choose the first and third games. In this case, the expected value of winnings will be $\left(\frac{p_1}{100} \cdot \frac{p_3}{100}\right) \cdot (w_1 + w_3) = \left(\frac{80}{100} \cdot \frac{50}{100}\right) \cdot (80 + 200) = 112$.

In the second example, you can choose the first and second games. In this case, the expected value of winnings will be $\left(\frac{p_1}{100} \cdot \frac{p_2}{100}\right) \cdot (w_1 + w_2) = \left(\frac{100}{100} \cdot \frac{100}{100}\right) \cdot (1 + 1) = 2$.

In the third example, you can choose only the second game. In this case, the expected value of winnings will be $\frac{p_2}{100} \cdot w_2 = \frac{2}{100} \cdot 1000 = 20$.