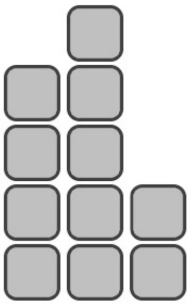
A. Print a Pedestal (Codeforces logo?)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Given the integer n — the number of available blocks. You must use **all** blocks to build a *pedestal*.

The *pedestal* consists of 3 platforms for 2-nd, 1-st and 3-rd places respectively. The platform for the 1-st place must be **strictly** higher than for the 2-nd place, and the platform for the 2-nd place must be **strictly** higher than for the 3-rd place. Also, the height of each platform must be greater than zero (that is, each platform must contain at least one block).



Example pedestal of $n = 11$ blocks: second place height equals 4 blocks, first place height equals 5 blocks, third place height equals 2 blocks.

Among all possible pedestals of n blocks, deduce one such that the platform height for the 1-st place **minimum** as possible. If there are several of them, output any of them.

Input

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

Each test case contains a single integer n ($6 \le n \le 10^5$) — the total number of blocks for the pedestal. All n blocks must be used.

It is guaranteed that the sum of n values over all test cases does not exceed 10^6.

Output

For each test case, output 3 numbers h_2, h_1, h_3 — the platform heights for 2-nd, 1-st and 3-rd places on a pedestal consisting of n blocks ($h_1 + h_2 + h_3 = n$, $0 < h_3 < h_2 < h_1$).

Among all possible pedestals, output the one for which the value of h_1 **minimal**. If there are several of them, output any of them.

Example

input

```
6
11
6
10
100000
7
8
```

output

```
4 5 2
2 3 1
4 5 1
33334 33335 33331
2 4 1
3 4 1
```

Note

In the first test case we can not get the height of the platform for the first place less than 5, because if the height of the platform for the first place is not more than 4, then we can use at most $4 + 3 + 2 = 9$ blocks. And we should use $11 = 4 + 5 + 2$ blocks. Therefore, the answer 4 5 2 fits.

In the second set, the only suitable answer is: 2 3 1.

B. Array Decrements

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Kristina has two arrays a and b, each containing n non-negative integers. She can perform the following operation on array a any number of times:

- apply a decrement to each non-zero element of the array, that is, replace the value of each element a_i such that $a_i > 0$ with the value $a_i - 1$ ($1 \le i \le n$). If a_i was 0, its value does not change.

Determine whether Kristina can get an array b from an array a in some number of operations (probably zero). In other words, can she make $a_i = b_i$ after some number of operations for each $1 \le i \le n$?

For example, let $n = 4$, $a = [3, 5, 4, 1]$ and $b = [1, 3, 2, 0]$. In this case, she can apply the operation twice:

- after the first application of the operation she gets $a = [2, 4, 3, 0]$;
- after the second use of the operation she gets $a = [1, 3, 2, 0]$.

Thus, in two operations, she can get an array b from an array a.

Input

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

The descriptions of the test cases follow.

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

The second line of each test case contains exactly n non-negative integers $a_1, a_2, \dots, a_n$ ($0 \le a_i \le 10^9$).

The third line of each test case contains exactly n non-negative integers $b_1, b_2, \dots, b_n$ ($0 \le b_i \le 10^9$).

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

Output

For each test case, output on a separate line:

- YES, if by doing some number of operations it is possible to get an array b from an array a;
- NO otherwise.

You can output YES and NO in any case (for example, strings yEs, yes, Yes and YES will be recognized as a positive response).

Example

input

```
6
4
3 5 4 1
1 3 2 0
3
1 2 1
0 1 0
4
5 3 7 2
1 1 1 1
5
1 2 3 4 5
1 2 3 4 6
1
8
0
1
4
6
```

output

```
YES
YES
NO
NO
YES
NO
```

Note

The first test case is analyzed in the statement.

In the second test case, it is enough to apply the operation to array a once.

In the third test case, it is impossible to get array b from array a.

C. Restoring the Duration of Tasks

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Recently, Polycarp completed n successive tasks.

For each completed task, the time s_i is known when it was given, no two tasks were given at the same time. Also given is the time f_i when the task was completed. For each task, there is an unknown value d_i ($d_i > 0$) — **duration of task execution**.

It is known that the tasks were completed in the order in which they came. Polycarp performed the tasks as follows:

- As soon as the very first task came, Polycarp immediately began to carry it out.
- If a new task arrived before Polycarp finished the previous one, he put the new task at the end of the queue.
- When Polycarp finished executing the next task and the queue was not empty, he **immediately** took a new task from the head of the queue (if the queue is empty — he just waited for the next task).

Find d_i (duration) of each task.

Input

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

The descriptions of the input data sets follow.

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

The second line of each test case contains exactly n integers $s_1 < s_2 < \dots < s_n$ ($0 \le s_i \le 10^9$).

The third line of each test case contains exactly n integers $f_1 < f_2 < \dots < f_n$ ($s_i < f_i \le 10^9$).

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

Output

For each of t test cases print n positive integers $d_1, d_2, \dots, d_n$ — the duration of each task.

Example

input

```
4
3
0 3 7
2 10 11
2
10 15
11 16
9
12 16 90 195 1456 1569 3001 5237 19275
13 199 200 260 9100 10000 10914 91066 5735533
1
0
1000000000
```

output

```
2 7 1
1 1
1 183 1 60 7644 900 914 80152 5644467
1000000000
```

Note

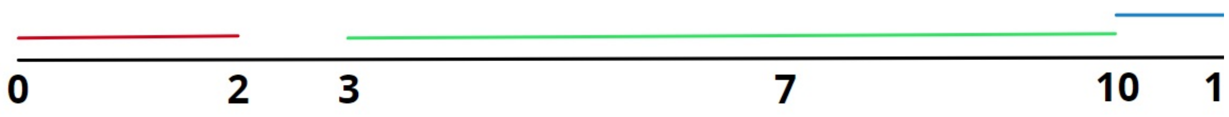
First test case:

The queue is empty at the beginning: $[\ ]$. And that's where the first task comes in. At time 2, Polycarp finishes doing the first task, so the duration of the first task is 2. The queue is empty so Polycarp is just waiting.

At time 3, the second task arrives. And at time 7, the third task arrives, and now the queue looks like this: $[7]$.

At the time 10, Polycarp finishes doing the second task, as a result, the duration of the second task is 7.

And at time 10, Polycarp immediately starts doing the third task and finishes at time 11. As a result, the duration of the third task is 1.



An example of the first test case.

D. Black and White Stripe

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a stripe of checkered paper of length n. Each cell is either white or black.

What is the minimum number of cells that must be recolored from white to black in order to have a segment of k consecutive black cells on the stripe?

If the input data is such that a segment of k consecutive black cells already exists, then print 0.

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

Next, descriptions of t test cases follow.

The first line of the input contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$). The second line consists of the letters 'W' (white) and 'B' (black). The line length is n.

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

Output
For each of t test cases print an integer — the minimum number of cells that need to be repainted from white to black in order to have a segment of k consecutive black cells.

Example

input
```
4
5 3
BBWBW
5 5
BBWBW
5 1
BBWBW
1 1
W
```
output
```
1
2
0
1
```

Note
In the first test case, s="BBWBW" and $k = 3$. It is enough to recolor s_3 and get s="BBBBW". This string contains a segment of length $k = 3$ consisting of the letters 'B'.

In the second test case of the example s="BBWBW" and $k = 5$. It is enough to recolor s_3 and s_5 and get s="BBBBB". This string contains a segment of length $k = 5$ consisting of the letters 'B'.

In the third test case of the example s="BBWBW" and $k = 1$. The string s already contains a segment of length $k = 1$ consisting of the letters 'B'.

E. Price Maximization

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A batch of n goods (n — an even number) is brought to the store, i-th of which has weight a_i. Before selling the goods, they must be packed into packages. After packing, the following will be done:

- There will be $\frac{n}{2}$ packages, each package contains exactly two goods;
- The weight of the package that contains goods with indices i and j ($1 \le i, j \le n$) is $a_i + a_j$.

With this, the cost of a package of weight x is always $\left\lfloor \frac{x}{k} \right\rfloor$ burles (rounded down), where k — a fixed and given value.

Pack the goods to the packages so that the revenue from their sale is maximized. In other words, make such $\frac{n}{2}$ pairs of given goods that the sum of the values $\left\lfloor \frac{x_i}{k} \right\rfloor$, where x_i is the weight of the package number i ($1 \le i \le \frac{n}{2}$), is **maximal**.

For example, let $n = 6, k = 3$, weights of goods $a = [3, 2, 7, 1, 4, 8]$. Let's pack them into the following packages.

- In the first package we will put the third and sixth goods. Its weight will be $a_3 + a_6 = 7 + 8 = 15$. The cost of the package will be $\left\lfloor \frac{15}{3} \right\rfloor = 5$ burles.
- In the second package put the first and fifth goods, the weight is $a_1 + a_5 = 3 + 4 = 7$. The cost of the package is $\left\lfloor \frac{7}{3} \right\rfloor = 2$ burles.
- In the third package put the second and fourth goods, the weight is $a_2 + a_4 = 2 + 1 = 3$. The cost of the package is $\left\lfloor \frac{3}{3} \right\rfloor = 1$ burle.

With this packing, the total cost of all packs would be $5 + 2 + 1 = 8$ burles.

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

The descriptions of the test cases follow.

The first line of each test case contains two integers n ($2 \le n \le 2 \cdot 10^5$) and k ($1 \le k \le 1000$). The number n — is even.

The second line of each test case contains exactly n integers $a_1, a_2, \dots, a_n$ ($0 \le a_i \le 10^9$).

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

Output

For each test case, print on a separate line a single number — the maximum possible total cost of all the packages.

Example

input

```
6
6 3
3 2 7 1 4 8
4 3
2 1 5 6
4 12
0 0 0 0
2 1
1 1
6 10
2 0 0 5 9 4
6 5
5 3 8 6 3 2
```

output

```
8
4
0
2
1
5
```

Note

The first test case is analyzed in the statement.

In the second test case, you can get a total value equal to 4 if you put the first and second goods in the first package and the third and fourth goods in the second package.

In the third test case, the cost of each item is 0, so the total cost will also be 0.

F. Shifting String

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarp found the string s and the permutation p. Their lengths turned out to be the same and equal to n.

A permutation of n elements — is an array of length n, in which every integer from 1 to n occurs exactly once. For example, $[1, 2, 3]$ and $[4, 3, 5, 1, 2]$ are permutations, but $[1, 2, 4]$, $[4, 3, 2, 1, 2]$ and $[0, 1, 2]$ are not.

In one operation he can multiply s by p, so he replaces s with string new, in which for any i from 1 to n it is true that $new_i = s_{p_i}$. For example, with $s = wmbe$ and $p = [3, 1, 4, 2]$, after operation the string will turn to $s = s_3 s_1 s_4 s_2 = bwem$.

Polycarp wondered after how many operations the string would become equal to its initial value for the first time. Since it may take too long, he asks for your help in this matter.

It can be proved that the required number of operations always exists. It can be very large, so use a 64-bit integer type.

Input

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

The first line of each case contains single integer n $(1 \le n \le 200)$ — the length of string and permutation.

The second line of each case contains a string s of length n, containing lowercase Latin letters.

The third line of each case contains n integers — permutation p $(1 \le p_i \le n)$, all p_i are different.

Output

Output t lines, each of which contains the answer to the corresponding test case of input. As an answer output single integer — the minimum number of operations, after which the string s will become the same as it was before operations.

Example

input

```
3
5
ababa
3 4 5 2 1
5
ababa
2 1 4 5 3
10
codeforces
8 6 1 7 5 2 9 3 10 4
```

output

```
1
6
12
```

Note

In the first sample operation doesn't change the string, so it will become the same as it was after 1 operations.

In the second sample the string will change as follows:

- s = babaa
- s = abaab
- s = baaba
- s = abbaa
- s = baaab
- s = ababa

G. Count the Trains

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are n of independent carriages on the rails. The carriages are numbered from left to right from 1 to n. The carriages are not connected to each other. The carriages move to the left, so that the carriage with number 1 moves ahead of all of them.

The i-th carriage has its own engine, which can accelerate the carriage to a_i km/h, but the carriage cannot go faster than the carriage in front of it. See example for explanation.

All carriages start moving to the left at the same time, and they naturally form **trains**. We will call **trains** — consecutive moving carriages having the same speed.

For example, we have $n = 5$ carriages and array $a = [10, 13, 5, 2, 6]$. Then the final speeds of the carriages will be $[10, 10, 5, 2, 2]$. Respectively, 3 of the train will be formed.

There are also messages saying that some engine has been corrupted:

- message "k d" means that the speed of the k-th carriage has decreased by d (that is, there has been a change in the maximum speed of the carriage $a_k = a_k - d$).

Messages arrive sequentially, the processing of the next message takes into account the changes from all previous messages.

After each message determine the number of formed trains.

Input

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

This is followed by descriptions of the test cases.

The first line of each test case is empty.

The second line of the test case contains two integers n and m ($1 \le n, m \le 10^5$) —the number of carriages and the number of messages to slow down the carriage, respectively.

The third line contains n integers: $a_1, a_2, \dots, a_n$ ($0 \le a_i \le 10^9$) — the number a_i means that the carriage with number i can reach a speed of a_i km/h.

The next m lines contain two integers k_j and d_j ($1 \le k_j \le n$, $0 \le d_j \le a_{k_j}$) —this is the message that the speed of the carriage with number k_j has decreased by d_j. In other words, there has been a change in its maximum speed $a_{k_j} = a_{k_j} - d_j$. Note that at any time the speed of each carriage is non-negative. In other words, $a_i \ge s_i$, where s_i —is the sum of such d_j that $k_j = i$.

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

Output

Print t lines. On each line print the answer for the corresponding test case.

For each test case print m numbers: the number of trains formed after each message.

Example

input

```
3

4 2
6 2 3 7
3 2
4 7

5 4
10 13 5 2 6
2 4
5 2
1 5
3 2

13 4
769 514 336 173 181 373 519 338 985 709 729 702 168
12 581
6 222
7 233
5 117
```

output

```
3 4
4 4 2 3
5 6 6 5
```

Note

For the first test case:

- Initially array $a = [6, 2, 3, 7]$.
- After the first message, the array $a = [6, 2, 1, 7]$. Accordingly, the speeds of the carriages are $[6, 2, 1, 1]$ and will form 3 of the train.
- After the second message the array $a = [6, 2, 1, 0]$. Accordingly, the speeds of the carriages are $[6, 2, 1, 0]$, and 4 of the train will be formed.

For the second test case:

- Initially, the array $a = [10, 13, 5, 2, 6]$.
- After the first message, the array $a = [10, 9, 5, 2, 6]$. Accordingly, the speeds of the carriages are equal: $[10, 9, 5, 2, 2]$, and 4 of the train will be formed.
- After the second message the array $a = [10, 9, 5, 2, 4]$. Accordingly, the speeds of the carriages are $[10, 9, 5, 2, 2]$, and 4 of the train will be formed.
- After the third message the array $a = [5, 9, 5, 2, 4]$. Accordingly, the speeds of the carriages are $[5, 5, 5, 2, 2]$, and 2 of the train will be formed.
- After the fourth message the array $a = [5, 9, 3, 2, 4]$. Accordingly, the speeds of the carriages are $[5, 5, 3, 2, 2]$, and 3 of the train will be formed.