A. CME

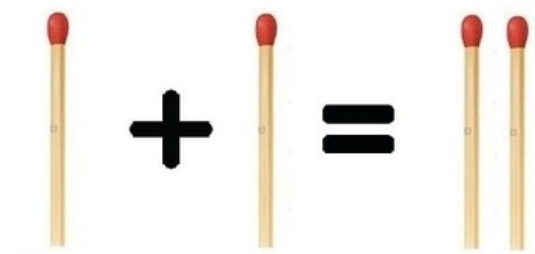
time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

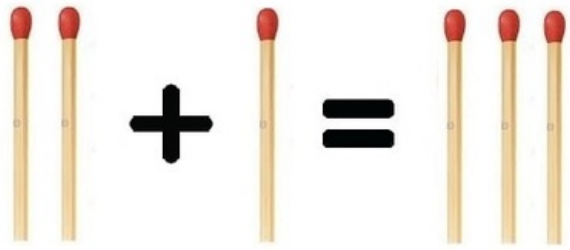
Let's denote *correct match equation* (we will denote it as CME) an equation $a + b = c$ there all integers a, b and c are greater than zero.

For example, equations $2 + 2 = 4$ (||+||=||||) and $1 + 2 = 3$ (|+||=|||) are CME but equations $1 + 2 = 4$ (|+||=||||), $2 + 2 = 3$ (||+||=|||), and $0 + 1 = 1$ (+|=|) are not.

Now, you have n matches. You want to assemble a CME using **all** your matches. Unfortunately, it is possible that you can't assemble the CME using all matches. But you can buy some extra matches and then assemble CME!

For example, if $n = 2$, you can buy two matches and assemble |+|=||, and if $n = 5$ you can buy one match and assemble ||+|=|||.





Calculate the minimum number of matches which you have to buy for assembling CME.

Note, that you have to answer q independent queries.

Input

The first line contains one integer q ($1 \le q \le 100$) — the number of queries.

The only line of each query contains one integer n ($2 \le n \le 10^9$) — the number of matches.

Output

For each test case print one integer in single line — the minimum number of matches which you have to buy for assembling CME.

Example

input
4 2 5 8 11
output
2 1 0 1

Note

The first and second queries are explained in the statement.

In the third query, you can assemble $1 + 3 = 4$ (|+|||=||||) without buying matches.

In the fourth query, buy one match and assemble $2 + 4 = 6$ (||+||||=||||||).

B. Strings Equalization

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

You are given two strings of equal length s and t consisting of lowercase Latin letters. You may perform any number (possibly, zero) operations on these strings.

During each operation you choose two **adjacent** characters in **any** string and assign the value of the first character to the value of the second or vice versa.

For example, if s is "acbc" you can get the following strings in **one** operation:

- "aabc" (if you perform $s_2 = s_1$);
- "ccbc" (if you perform $s_1 = s_2$);
- "accc" (if you perform $s_3 = s_2$ or $s_3 = s_4$);
- "abbc" (if you perform $s_2 = s_3$);
- "acbb" (if you perform $s_4 = s_3$);

Note that you can also apply this operation to the string t.

Please determine whether it is possible to transform s into t, applying the operation above any number of times.

Note that you have to answer q independent queries.

Input

The first line contains one integer q ($1 \le q \le 100$) — the number of queries. Each query is represented by two consecutive lines.

The first line of each query contains the string s ($1 \le |s| \le 100$) consisting of lowercase Latin letters.

The second line of each query contains the string t ($1 \le |t| \le 100$, $|t| = |s|$) consisting of lowercase Latin letters.

Output

For each query, print "YES" if it is possible to make s equal to t, and "NO" otherwise.

You may print every letter in any case you want (so, for example, the strings "yEs", "yes", "Yes", and "YES" will all be recognized as positive answer).

Example

input
3 xabb aabx technocup technocup a z
output
YES YES NO

Note

In the first query, you can perform two operations $s_1 = s_2$ (after it s turns into "aabb") and $t_4 = t_3$ (after it t turns into "aabb").

In the second query, the strings are equal initially, so the answer is "YES".

In the third query, you can not make strings s and t equal. Therefore, the answer is "NO".

C. Save the Nature

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

You are an environmental activist at heart but the reality is harsh and you are just a cashier in a cinema. But you can still do something!

You have n tickets to sell. The price of the i-th ticket is p_i. As a teller, you have a possibility to select the order in which the tickets will be sold (i.e. a permutation of the tickets). You know that the cinema participates in two ecological restoration programs applying

them **to the order you chose**:

- The $x\%$ of the price of each the a-th sold ticket (a-th, $2a$-th, $3a$-th and so on) *in the order you chose* is aimed for research and spreading of renewable energy sources.
- The $y\%$ of the price of each the b-th sold ticket (b-th, $2b$-th, $3b$-th and so on) *in the order you chose* is aimed for pollution abatement.

If the ticket is in both programs then the $(x + y)\%$ are used for environmental activities. Also, it's known that all prices are multiples of 100, so there is no need in any rounding.

For example, if you'd like to sell tickets with prices $[400, 100, 300, 200]$ and the cinema pays 10% of each 2-nd sold ticket and 20% of each 3-rd sold ticket, then arranging them in order $[100, 200, 300, 400]$ will lead to contribution equal to $100 \cdot 0 + 200 \cdot 0.1 + 300 \cdot 0.2 + 400 \cdot 0.1 = 120$. But arranging them in order $[100, 300, 400, 200]$ will lead to $100 \cdot 0 + 300 \cdot 0.1 + 400 \cdot 0.2 + 200 \cdot 0.1 = 130$.

Nature can't wait, so you decided to change the order of tickets in such a way, so that the **total** contribution to programs will reach at least k in **minimum** number of sold tickets. Or say that it's impossible to do so. In other words, find the minimum number of tickets which are needed to be sold in order to earn at least k.

Input

The first line contains a single integer q ($1 \le q \le 100$) — the number of independent queries. Each query consists of 5 lines.

The first line of each query contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of tickets.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($100 \le p_i \le 10^9$, $p_i \bmod 100 = 0$) — the corresponding prices of tickets.

The third line contains two integers x and a ($1 \le x \le 100$, $x + y \le 100$, $1 \le a \le n$) — the parameters of the first program.

The fourth line contains two integers y and b ($1 \le y \le 100$, $x + y \le 100$, $1 \le b \le n$) — the parameters of the second program.

The fifth line contains single integer k ($1 \le k \le 10^{14}$) — the required total contribution.

It's guaranteed that the total number of tickets per test doesn't exceed $2 \cdot 10^5$.

Output

Print q integers — one per query.

For each query, print the minimum number of tickets you need to sell to make the total ecological contribution of at least k if you can sell tickets in any order.

If the total contribution can not be achieved selling all the tickets, print -1.

Example

input
4 1 100 50 1 49 1 100 8 100 200 100 200 100 200 100 100 10 2 15 3 107 3 1000000000 1000000000 1000000000 50 1 50 1 3000000000 5 200 100 100 100 100 69 5 31 2 90
output
-1 6 3 4

Note

In the first query the total contribution is equal to $50 + 49 = 99 < 100$, so it's impossible to gather enough money.

In the second query you can rearrange tickets in a following way: $[100, 100, 200, 200, 100, 200, 100, 100]$ and the total contribution from the first 6 tickets is equal to $100 \cdot 0 + 100 \cdot 0.1 + 200 \cdot 0.15 + 200 \cdot 0.1 + 100 \cdot 0 + 200 \cdot 0.25 = 10 + 30 + 20 + 50 = 110$.

In the third query the full price of each ticket goes to the environmental activities.

In the fourth query you can rearrange tickets as $[100, 200, 100, 100, 100]$ and the total contribution from the first 4 tickets is $100 \cdot 0 + 200 \cdot 0.31 + 100 \cdot 0 + 100 \cdot 0.31 = 62 + 31 = 93$.

D. Sequence Sorting

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

You are given a sequence $a_1, a_2, \ldots, a_n$, consisting of integers.

You can apply the following operation to this sequence: choose some integer x and move **all** elements equal to x either to the beginning, or to the end of a. Note that you have to move all these elements in **one** direction in **one** operation.

For example, if $a = [2, 1, 3, 1, 1, 3, 2]$, you can get the following sequences in one operation (for convenience, denote elements equal to x as x-elements):

- $[1, 1, 1, 2, 3, 3, 2]$ if you move all 1-elements to the beginning;
- $[2, 3, 3, 2, 1, 1, 1]$ if you move all 1-elements to the end;
- $[2, 2, 1, 3, 1, 1, 3]$ if you move all 2-elements to the beginning;
- $[1, 3, 1, 1, 3, 2, 2]$ if you move all 2-elements to the end;
- $[3, 3, 2, 1, 1, 1, 2]$ if you move all 3-elements to the beginning;
- $[2, 1, 1, 1, 2, 3, 3]$ if you move all 3-elements to the end;

You have to determine the minimum number of such operations so that the sequence a becomes sorted in non-descending order. Non-descending order means that for all i from 2 to n, the condition $a_{i-1} \le a_i$ is satisfied.

Note that you have to answer q independent queries.

Input

The first line contains one integer q ($1 \le q \le 3 \cdot 10^5$) — the number of the queries. Each query is represented by two consecutive lines.

The first line of each query contains one integer n ($1 \le n \le 3 \cdot 10^5$) — the number of elements.

The second line of each query contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements.

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

Output

For each query print one integer — the minimum number of operation for sorting sequence a in non-descending order.

Example

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

Note

In the first query, you can move all 1-elements to the beginning (after that sequence turn into $[1, 1, 1, 3, 6, 6, 3]$) and then move all 6-elements to the end.

In the second query, the sequence is sorted initially, so the answer is zero.

In the third query, you have to move all 2-elements to the beginning.

E. Paint the Tree

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

You are given a weighted tree consisting of n vertices. Recall that a tree is a connected graph without cycles. Vertices u_i and v_i are connected by an edge with weight w_i.

Let's define the k-coloring of the tree as an assignment of exactly k colors to **each** vertex, so that each color is used no more than two times. You can assume that you have infinitely many colors available. We say that an edge is *saturated* in the given k-coloring if its endpoints share at least one color (i.e. there exists a color that is assigned to both endpoints).

Let's also define the *value* of a k-coloring as the sum of weights of *saturated* edges.

Please calculate the maximum possible *value* of a k-coloring of the given tree.

You have to answer q independent queries.

Input

The first line contains one integer q ($1 \le q \le 5 \cdot 10^5$) – the number of queries.

The first line of each query contains two integers n and k ($1 \le n, k \le 5 \cdot 10^5$) — the number of vertices in the tree and the number of colors to assign to each vertex, respectively.

Each of the next $n - 1$ lines describes an edge of the tree. Edge i is denoted by three integers u_i, v_i and w_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$, $1 \le w_i \le 10^5$) — the labels of vertices it connects and the weight of the edge. It is guaranteed that the given edges form a tree.

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

Output

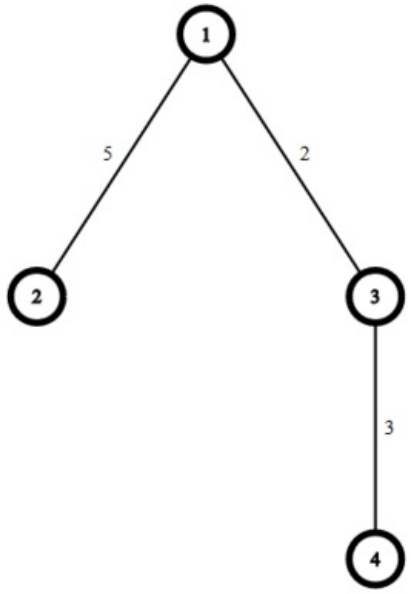
For each query print one integer — the maximum *value* of a k-coloring of the given tree.

Example

input
2 4 1 1 2 5 3 1 2 3 4 3 7 2 1 2 5 1 3 4 1 4 2 2 5 1 2 6 2 4 7 3
output
8 14

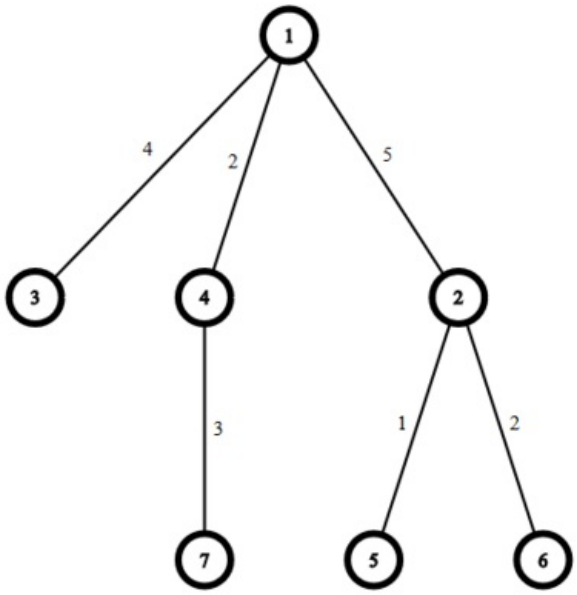
Note

The tree corresponding to the first query in the example:



One of the possible k-colorings in the first example: $(1), (1), (2), (2)$, then the 1-st and the 3-rd edges are saturated and the sum of their weights is 8.

The tree corresponding to the second query in the example:



One of the possible k-colorings in the second example: $(1, 2), (1, 3), (2, 4), (5, 6), (7, 8), (3, 4), (5, 6)$, then the 1-st, 2-nd, 5-th and 6-th edges are saturated and the sum of their weights is 14.

F. Stack Exterminable Arrays

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

Let's look at the following process: initially you have an empty stack and an array s of the length l. You are trying to push array elements to the stack in the order $s_1, s_2, s_3, \ldots s_l$. Moreover, if the stack is empty or the element at the top of this stack is not equal to the current element, then you just push the current element to the top of the stack. Otherwise, you don't push the current element to the stack and, moreover, pop the top element of the stack.

If after this process the stack remains empty, the array s is considered *stack exterminable*.

There are samples of stack exterminable arrays:

- $[1, 1]$;
- $[2, 1, 1, 2]$;
- $[1, 1, 2, 2]$;
- $[1, 3, 3, 1, 2, 2]$;
- $[3, 1, 3, 3, 1, 3]$;
- $[3, 3, 3, 3, 3, 3]$;
- $[5, 1, 2, 2, 1, 4, 4, 5]$;

Let's consider the changing of stack more details if $s = [5, 1, 2, 2, 1, 4, 4, 5]$ (the top of stack is highlighted).

1. after pushing $s_1 = 5$ the stack turn into $[\mathbf{5}]$;
2. after pushing $s_2 = 1$ the stack turn into $[5, \mathbf{1}]$;
3. after pushing $s_3 = 2$ the stack turn into $[5, 1, \mathbf{2}]$;
4. after pushing $s_4 = 2$ the stack turn into $[5, \mathbf{1}]$;
5. after pushing $s_5 = 1$ the stack turn into $[\mathbf{5}]$;
6. after pushing $s_6 = 4$ the stack turn into $[5, \mathbf{4}]$;
7. after pushing $s_7 = 4$ the stack turn into $[\mathbf{5}]$;
8. after pushing $s_8 = 5$ the stack is empty.

You are given an array $a_1, a_2, \ldots, a_n$. You have to calculate the number of its subarrays which are stack exterminable.

Note, that you have to answer q independent queries.

Input

The first line contains one integer q ($1 \leq q \leq 3 \cdot 10^5$) — the number of queries.

The first line of each query contains one integer n ($1 \leq n \leq 3 \cdot 10^5$) — the length of array a.

The second line of each query contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq n$) — the elements.

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

Output
For each test case print one integer in single line — the number of stack exterminable subarrays of the array a.

Example

input
3 5 2 1 1 2 2 6 1 2 1 1 3 2 9 3 1 2 2 1 6 6 3 3
output
4 1 8

Note
In the first query there are four stack exterminable subarrays: $a_{1\ldots4} = [2, 1, 1, 2], a_{2\ldots3} = [1, 1], a_{2\ldots5} = [1, 1, 2, 2], a_{4\ldots5} = [2, 2]$.

In the second query, only one subarray is exterminable subarray — $a_{3\ldots4}$.

In the third query, there are eight stack exterminable subarrays: $a_{1\ldots8}, a_{2\ldots5}, a_{2\ldots7}, a_{2\ldots9}, a_{3\ldots4}, a_{6\ldots7}, a_{6\ldots9}, a_{8\ldots9}$.

G. Wooden Raft

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

Suppose you are stuck on a desert island. The only way to save yourself is to craft a wooden raft and go to the sea. Fortunately, you have a hand-made saw and a forest nearby. Moreover, you've already cut several trees and prepared it to the point that now you have n logs and the i-th log has length a_i.

The wooden raft you'd like to build has the following structure: 2 logs of length x and x logs of length y. Such raft would have the area equal to $x \cdot y$. Both x **and** y **must be integers** since it's the only way you can measure the lengths while being on a desert island. And both x **and** y **must be at least** 2 since the raft that is one log wide is unstable.

You can cut logs in pieces but you can't merge two logs in one. What is the maximum area of the raft you can craft?

Input
The first line contains the only integer n ($1 \le n \le 5 \cdot 10^5$) — the number of logs you have.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($2 \le a_i \le 5 \cdot 10^5$) — the corresponding lengths of the logs.

It's guaranteed that you can always craft at least 2×2 **raft.**

Output
Print the only integer — the maximum area of the raft you can craft.

Examples

input
1 9
output
4

input
9 9 10 9 18 9 9 9 28 9
output
90

Note
In the first example, you can cut the log of the length 9 in 5 parts: $2 + 2 + 2 + 2 + 1$. Now you can build 2×2 raft using 2 logs of length $x = 2$ and $x = 2$ logs of length $y = 2$.

In the second example, you can cut $a_4 = 18$ into two pieces $9 + 9$ and $a_8 = 28$ in three pieces $10 + 9 + 9$. Now you can make 10×9 raft using 2 logs of length 10 and 10 logs of length 9.