A. Three Decks

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

Monocarp placed three decks of cards in a row on the table. The first deck consists of a cards, the second deck consists of b cards, and the third deck consists of c cards, with the condition $a < b < c$.

Monocarp wants to take some number of cards (at least one, but no more than c) from the **third** deck and distribute them between the first two decks so that each of the taken cards ends up in either the first or the second deck. It is possible that all the cards taken from the third deck will go into the same deck.

Your task is to determine whether Monocarp can make the number of cards in all three decks equal using the described operation.

Input

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

The only line of each test case contains three integers a, b, and c ($1 \le a, b, c \le 10^8$) — the number of cards in the first, second, and third decks, respectively.

Additional constraint on the input: $a < b < c$.

Output

For each test case, output "YES" (without quotes) if Monocarp can make the number of cards in all three decks equal using the described operation. Otherwise, output "NO" (without quotes).

Standard Input	Standard Output
4 3 5 10 12 20 30 3 5 7 1 5 6	YES NO YES NO

Note

In the first test case, Monocarp has to take 4 cards from the third deck, put 3 cards in the first deck, and 1 card in the second deck. Thus, there will be 6 cards in all three decks.

In the second test case, it is impossible to make the number of cards in all three decks equal.

In the third test case, Monocarp has to take 2 cards from the third deck and put both in the first deck. Thus, there will be 5 cards in all three decks.

In the fourth test case, it is also impossible to make the number of cards in all three decks equal.

,

B. Move to the End

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

You are given an array a consisting of n integers.

For every integer k from 1 to n, you have to do the following:

1. choose an arbitrary element of a and move it to the right end of the array (you can choose the last element, then the array won't change);
2. print the sum of k last elements of a;
3. move the element you've chosen on the first step to its original position (restore the original array a).

For every k, you choose the element which you move so that the value you print is **the maximum possible**.

Calculate the value you print for every k.

Input

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

Each test case consists of two lines:

- the first line contains one integer n ($1 \le n \le 2 \cdot 10^5$);
- the second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$).

Additional constraint on the input: the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print n integers. The i-th of these integers should be equal to the maximum value you can print if $k = i$.

Standard Input	Standard Output
4 7 13 5 10 14 8 15 13 6 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1 42 2 7 5	15 28 42 50 63 73 78 1000000000 2000000000 3000000000 4000000000 5000000000 6000000000 42 7 12

Note

Let's consider the first test case from the statement:

- when $k = 1$, you can move the 6-th element to the end, the array becomes $[13, 5, 10, 14, 8, 13, 15]$, and the value you print is 15;

- when $k = 2$, you can move the 6-th element to the end, the array becomes $[13, 5, 10, 14, 8, 13, 15]$, and the value you print is $13 + 15 = 28$;
- when $k = 3$, you can move the 4-th element to the end, the array becomes $[13, 5, 10, 8, 15, 13, 14]$, and the value you print is $15 + 13 + 14 = 42$;
- when $k = 4$, you can move the 5-th element to the end, the array becomes $[13, 5, 10, 14, 15, 13, 8]$, and the value you print is $14 + 15 + 13 + 8 = 50$;
- when $k = 5$, you can move the 1-st element to the end, the array becomes $[5, 10, 14, 8, 15, 13, 13]$, and the value you print is $14 + 8 + 15 + 13 + 13 = 63$;
- when $k = 6$, you can move the 1-st element to the end, the array becomes $[5, 10, 14, 8, 15, 13, 13]$, and the value you print is $10 + 14 + 8 + 15 + 13 + 13 = 73$;
- when $k = 7$, you can move the 6-th element to the end, the array becomes $[13, 5, 10, 14, 8, 13, 15]$, and the value you print is $13 + 5 + 10 + 14 + 8 + 13 + 15 = 78$.

,

C. Card Game

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   512 megabytes
```

Alice and Bob are playing a game. They have n cards numbered from 1 to n. At the beginning of the game, some of these cards are given to Alice, and the rest are given to Bob.

Card with number i beats card with number j if and only if $i > j$, **with one exception**: card 1 beats card n.

The game continues as long as each player has at least one card. During each turn, the following occurs:

1. Alice chooses one of her cards and places it face up on the table;
2. Bob, seeing Alice's card, chooses one of his cards and places it face up on the table;
3. if Alice's card beats Bob's card, both cards are taken by Alice. Otherwise, both cards are taken by Bob.

A player can use a card that they have taken during one of the previous turns.

The player who has no cards at the beginning of a turn loses. Determine who will win if both players play optimally.

Input

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

Each test case consists of two lines:

- the first line contains a single integer n ($2 \le n \le 50$) — the number of cards;
- the second line contains n characters, each either A or B. If the i-th character is A, then card number i is initially given to Alice; otherwise, it is given to Bob.

Additional constraint on the input: in each test case, at least one card is initially given to Alice, and at least one card is initially given to Bob.

Output

For each test case, output `Alice` if Alice wins with optimal play, or `Bob` if Bob wins. It can be shown that if both players play optimally, the game will definitely end in a finite number of turns with one of the players winning.

Standard Input	Standard Output
8 2 AB 2 BA 4 ABAB 4 BABA 3 BAA	Alice Bob Bob Bob Alice Alice Bob Alice

5 AAAAB 5 BAAAB 6 BBBAAA	

Note

In the first test case, Alice has only one card, and Bob has only one card. Since Alice's card beats Bob's card, she wins after the first turn.

In the second test case, Alice has only one card, and Bob has only one card. Since Bob's card beats Alice's card, he wins after the first turn.

In the third test case, there are two possible game scenarios:

- if Alice plays the card 1 on the first turn, Bob can respond with the card 2 and take both cards. Then, Alice has to play the card 3 on the second turn, and Bob will respond by playing the card 4. Then, he wins;
- if Alice plays the card 3 on the first turn, Bob can respond with the card 4 and take both cards. Then, Alice has to play the card 1, and Bob can respond either with the card 2 or the card 3. Then, he wins.

In the fourth test case, there are two possible game scenarios:

- if Alice plays the card 2 on the first turn, Bob can respond with the card 3 and take both cards. Then, Alice has to play the card 4 on the second turn, and Bob will respond by playing the card 1. Then, he wins;
- if Alice plays the card 4 on the first turn, Bob can respond with the card 1 and take both cards. Then, Alice has to play the card 2, and Bob can respond either with the card 3 or the card 4. Then, he wins.

,

D. Array and GCD

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

You are given an integer array a of size n.

You can perform the following operations any number of times (possibly, zero):

- pay one coin and increase any element of the array by 1 (you must have at least 1 coin to perform this operation);
- gain one coin and decrease any element of the array by 1.

Let's say that an array is *ideal* if both of the following conditions hold:

- each element of the array is at least 2;
- for each pair of indices i and j ($1 \le i, j \le n$; $i \neq j$) the greatest common divisor (GCD) of a_i and a_j is equal to 1. If the array has less than 2 elements, this condition is automatically satisfied.

Let's say that an array is *beautiful* if it can be transformed into an ideal array using the aforementioned operations, provided that you initially have no coins. If the array is already ideal, then it is also beautiful.

The given array is not necessarily beautiful or ideal. You can remove any elements from it (including removing the entire array or not removing anything at all). Your task is to calculate the minimum number of elements you have to remove (possibly, zero) from the array a to make it **beautiful**.

Input

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

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

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($2 \le a_i \le 10^9$).

Additional constraint on the input: the sum of n over all test cases doesn't exceed $4 \cdot 10^5$.

Output

For each test case, print a single integer — the minimum number of elements you have to remove (possibly, zero) from the array a to make it **beautiful**.

Standard Input	Standard Output
5 3 5 5 5 4 2 3 2 4 1 3 3 2 100 2 5	0 2 0 0 1

2 4 2 11 2	

Note

In the first example, you don't need to delete any elements, because the array is already beautiful. It can be transformed into an ideal array as follows: $[5, 5, 5] \rightarrow [4, 5, 5] \rightarrow [4, 4, 5] \rightarrow [4, 3, 5]$ (you end up with 3 coins).

In the second example, you need to remove 2 elements so that the array becomes beautiful. If you leave the elements $[2, 3]$ and delete the other elements, then the given array is already ideal (and therefore, beautiful).

In the third example, you don't need to delete any elements because the array is already ideal (and thus, beautiful).

In the fourth example, the array is beautiful. It can be transformed into an ideal array as follows: $[2, 100, 2] \rightarrow [2, 99, 2] \rightarrow [2, 99, 3] \rightarrow [2, 98, 3] \rightarrow [2, 97, 3]$ (you end up with 2 coins).

,

E. Unpleasant Strings

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 512 megabytes

Let's call a letter *allowed* if it is a lowercase letter and is one of the first k letters of the Latin alphabet.

You are given a string s of length n, consisting only of allowed letters.

Let's call a string t *pleasant* if t is a subsequence of s.

You are given q strings $t_1, t_2, \ldots, t_q$. All of them consist only of allowed letters. For each string t_i, calculate the minimum number of allowed letters you need to append to it on the right so that it **stops** being pleasant.

A sequence t is a subsequence of a sequence s if t can be obtained from s by the deletion of several (possibly, zero or all) element from arbitrary positions.

Input

The first line contains two integers n and k ($1 \le n \le 10^6$; $1 \le k \le 26$) — the length of the string s and the number of allowed letters.

The second line contains the string s, consisting of n lowercase Latin letters. Each character of the string is one of the first k letters of the Latin alphabet.

The third line contains one integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries.

The next q lines contain queries: one query per line. The i-th line contains the string t_i, consisting only of allowed letters.

Additional constraint on input: the total length of all t_i does not exceed 10^6.

Output

For each query, output one integer — the minimum number of allowed letters that need to be appended to the string on the right so that it stops being pleasant.

Standard Input	Standard Output
7 3 abacaba 3 cc bcb b	0 1 2
5 1 aaaaa 6 a aa aaa aaaa	5 4 3 2 1 0

aaaaa aaaaaa	

Note

In the first example:

1. The string cc is already unpleasant, so nothing needs to be appended to it;
2. bcb is pleasant, so at least one letter needs to be appended to the right: $bcba$ will not work, but $bcbb$ and $bcbc$ are unpleasant.
3. To b, at least two letters need to be appended, since ba, bb, and bc are pleasant. For example, we can obtain an unpleasant string bbb.

,

F. Numbers and Strings

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 512 megabytes

For each integer x from 1 to n, we will form the string $S(x)$ according to the following rules:

- compute $(x + 1)$;
- write x and $x + 1$ next to each other in the decimal system without separators and leading zeros;
- in the resulting string, sort all digits in non-decreasing order.

For example, the string $S(139)$ is `011349` (before sorting the digits, it is `139140`). The string $S(99)$ is `00199`.

Your task is to count the number of distinct strings among $S(1), S(2), \ldots, S(n)$.

Input

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

Each test case consists of one line containing a single integer n ($1 \le n \le 10^9 - 2$).

Output

For each test case, output a single integer — the number of distinct strings among $S(1), S(2), \ldots, S(n)$.

Standard Input	Standard Output
2 42 1337	42 948

,

G. Modulo 3

Input file:	standard input
Output file:	standard output
Time limit:	4 seconds
Memory limit:	512 megabytes

Surely, you have seen problems which require you to output the answer modulo $10^9 + 7$, $10^9 + 9$, or 998244353. But have you ever seen a problem where you have to print the answer modulo 3?

You are given a functional graph consisting of n vertices, numbered from 1 to n. It is a directed graph, in which each vertex has exactly one outgoing arc. The graph is given as the array $g_1, g_2, \ldots, g_n$, where g_i means that there is an arc that goes from i to g_i. For some vertices, the outgoing arcs might be self-loops.

Initially, all vertices of the graph are colored in color 1. You can perform the following operation: select a vertex and a color from 1 to k, and then color this vertex and all vertices that are reachable from it. You can perform this operation any number of times (even zero).

You should process q queries. The query is described by three integers x, y and k. For each query, you should:

- assign $g_x := y$;
- then calculate the number of different graph colorings for the given value of k (two colorings are different if there exists at least one vertex that is colored in different colors in these two colorings); since the answer can be very large, print it **modulo** 3.

Note that in every query, the initial coloring of the graph is reset (all vertices initially have color 1 in each query).

Input

The first line contains two integers n and q ($1 \le n, q \le 2 \cdot 10^5$).

The second line contains n integers $g_1, g_2, \ldots, g_n$ ($1 \le g_i \le n$).

The q lines follow. The i-th line contains three integers x_i, y_i and k_i ($1 \le x_i, y_i \le n$; $1 \le k_i \le 10^9$).

Output

For each query, print a single integer — the number of different graph colorings for the given value of k, taken modulo 3.

Standard Input	Standard Output
4 5 2 3 1 4 4 3 1 2 1 2 3 4 3 4 1 5 2 4 4	1 2 0 2 1
8 10 7 4 6 8 7 7 1 4 1 7 5	1 0 1

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