Educational Codeforces Round 111 (Rated for Div. 2)

A. Find The Array

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

Let's call an array a consisting of n positive (greater than 0) integers *beautiful* if the following condition is held for every i from 1 to n: either $a_i = 1$, or at least one of the numbers $a_i - 1$ and $a_i - 2$ exists in the array as well.

For example:

- the array $[5, 3, 1]$ is beautiful: for a_1, the number $a_1 - 2 = 3$ exists in the array; for a_2, the number $a_2 - 2 = 1$ exists in the array; for a_3, the condition $a_3 = 1$ holds;
- the array $[1, 2, 2, 2, 2]$ is beautiful: for a_1, the condition $a_1 = 1$ holds; for every other number a_i, the number $a_i - 1 = 1$ exists in the array;
- the array $[1, 4]$ is not beautiful: for a_2, neither $a_2 - 2 = 2$ nor $a_2 - 1 = 3$ exists in the array, and $a_2 \neq 1$;
- the array $[2]$ is not beautiful: for a_1, neither $a_1 - 1 = 1$ nor $a_1 - 2 = 0$ exists in the array, and $a_1 \neq 1$;
- the array $[2, 1, 3]$ is beautiful: for a_1, the number $a_1 - 1 = 1$ exists in the array; for a_2, the condition $a_2 = 1$ holds; for a_3, the number $a_3 - 2 = 1$ exists in the array.

You are given a positive integer s. Find the minimum possible size of a beautiful array with the sum of elements equal to s.

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

Then t lines follow, the i-th line contains one integer s ($1 \le s \le 5000$) for the i-th test case.

Output
Print t integers, the i-th integer should be the answer for the i-th testcase: the minimum possible size of a beautiful array with the sum of elements equal to s.

Example

input
4 1 8 7 42
output
1 3 3 7

Note
Consider the example test:

1. in the first test case, the array $[1]$ meets all conditions;
2. in the second test case, the array $[3, 4, 1]$ meets all conditions;
3. in the third test case, the array $[1, 2, 4]$ meets all conditions;
4. in the fourth test case, the array $[1, 4, 6, 8, 10, 2, 11]$ meets all conditions.

B. Maximum Cost Deletion

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

You are given a string s of length n consisting only of the characters `0` and `1`.

You perform the following operation until the string becomes empty: choose some **consecutive** substring of **equal** characters, erase it from the string and glue the remaining two parts together (any of them can be empty) in the same order. For example, if you erase the substring `111` from the string `111010`, you will get the string `010`. When you delete a substring of length l, you get $a \cdot l + b$ points.

Your task is to calculate the maximum number of points that you can score in total, if you have to make the given string empty.

Input
The first line contains a single integer t ($1 \le t \le 2000$) — the number of testcases.

The first line of each testcase contains three integers n, a and b ($1 \le n \le 100$; $-100 \le a, b \le 100$) — the length of the string s and the parameters a and b.

The second line contains the string s. The string s consists only of the characters `0` and `1`.

Output
For each testcase, print a single integer — the maximum number of points that you can score.

Example

input
3 3 2 0 000 5 -2 5 11001 6 1 -4 100111
output
6 15 -2

Note
In the first example, it is enough to delete the entire string, then we will get $2 \cdot 3 + 0 = 6$ points.

In the second example, if we delete characters one by one, then for each deleted character we will get $(-2) \cdot 1 + 5 = 3$ points, i. e. 15 points in total.

In the third example, we can delete the substring `00` from the string `100111`, we get $1 \cdot 2 + (-4) = -2$ points, and the string will be equal to `1111`, removing it entirely we get $1 \cdot 4 + (-4) = 0$ points. In total, we got -2 points for 2 operations.

C. Manhattan Subarrays

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

Suppose you have two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$. Let's denote the Manhattan distance between them as $d(p, q) = |x_p - x_q| + |y_p - y_q|$.

Let's say that three points p, q, r form a *bad* triple if $d(p, r) = d(p, q) + d(q, r)$.

Let's say that an array $b_1, b_2, \ldots, b_m$ is *good* if it is impossible to choose three **distinct** indices i, j, k such that the points (b_i, i), (b_j, j) and (b_k, k) form a bad triple.

You are given an array $a_1, a_2, \ldots, a_n$. Calculate the number of *good* subarrays of a. A subarray of the array a is the array $a_l, a_{l+1}, \ldots, a_r$ for some $1 \le l \le r \le n$.

Note that, according to the definition, subarrays of length 1 and 2 are *good*.

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 ($1 \le n \le 2 \cdot 10^5$) — the length of array a.

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

It's guaranteed that the sum of n doesn't exceed $2 \cdot 10^5$.

Output
For each test case, print the number of *good* subarrays of array a.

Example

input
3 4 2 4 1 3 5 6 9 1 9 6 2 13 37
output
10

```
12
3
```

Note

In the first test case, it can be proven that any subarray of a is *good*. For example, subarray $[a_2, a_3, a_4]$ is good since it contains only three elements and:

- $d((a_2, 2), (a_4, 4)) = |4 - 3| + |2 - 4| = 3 < d((a_2, 2), (a_3, 3)) + d((a_3, 3), (a_4, 4)) = 3 + 1 + 2 + 1 = 7$;
- $d((a_2, 2), (a_3, 3)) < d((a_2, 2), (a_4, 4)) + d((a_4, 4), (a_3, 3))$;
- $d((a_3, 3), (a_4, 4)) < d((a_3, 3), (a_2, 2)) + d((a_2, 2), (a_4, 4))$;

In the second test case, for example, subarray $[a_1, a_2, a_3, a_4]$ is **not** good, since it contains a *bad* triple $(a_1, 1)$, $(a_2, 2)$, $(a_4, 4)$:

- $d((a_1, 1), (a_4, 4)) = |6 - 9| + |1 - 4| = 6$;
- $d((a_1, 1), (a_2, 2)) = |6 - 9| + |1 - 2| = 4$;
- $d((a_2, 2), (a_4, 4)) = |9 - 9| + |2 - 4| = 2$;

So, $d((a_1, 1), (a_4, 4)) = d((a_1, 1), (a_2, 2)) + d((a_2, 2), (a_4, 4))$.

D. Excellent Arrays

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

Let's call an integer array $a_1, a_2, \ldots, a_n$ *good* if $a_i \neq i$ for each i.

Let $F(a)$ be the number of pairs (i, j) $(1 \leq i < j \leq n)$ such that $a_i + a_j = i + j$.

Let's say that an array $a_1, a_2, \ldots, a_n$ is *excellent* if:

- a is *good*;
- $l \leq a_i \leq r$ for each i;
- $F(a)$ is the maximum possible among all *good* arrays of size n.

Given n, l and r, calculate the number of *excellent* arrays modulo $10^9 + 7$.

Input

The first line contains a single integer t $(1 \leq t \leq 1000)$ — the number of test cases.

The first and only line of each test case contains three integers n, l, and r $(2 \leq n \leq 2 \cdot 10^5$; $-10^9 \leq l \leq 1$; $n \leq r \leq 10^9)$.

It's guaranteed that the sum of n doesn't exceed $2 \cdot 10^5$.

Output

For each test case, print the number of excellent arrays modulo $10^9 + 7$.

Example

input
4 3 0 3 4 -3 5 42 -33 55 69 -42 146
output
4 10 143922563 698570404

Note

In the first test case, it can be proven that the maximum $F(a)$ among all *good* arrays a is equal to 2. The excellent arrays are:

1. $[2, 1, 2]$;
2. $[0, 3, 2]$;
3. $[2, 3, 2]$;
4. $[3, 0, 1]$.

E. Stringforces

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

You are given a string s of length n. Each character is either one of the first k lowercase Latin letters or a question mark.

You are asked to replace every question mark with one of the first k lowercase Latin letters in such a way that the following value is maximized.

Let f_i be the maximum length substring of string s, which consists entirely of the i-th Latin letter. A substring of a string is a contiguous subsequence of that string. If the i-th letter doesn't appear in a string, then f_i is equal to 0.

The value of a string s is the minimum value among f_i for all i from 1 to k.

What is the maximum value the string can have?

Input
The first line contains two integers n and k ($1 \le n \le 2 \cdot 10^5$; $1 \le k \le 17$) — the length of the string and the number of first Latin letters used.

The second line contains a string s, consisting of n characters. Each character is either one of the first k lowercase Latin letters or a question mark.

Output
Print a single integer — the maximum value of the string after every question mark is replaced with one of the first k lowercase Latin letters.

Examples

input
10 2 a??ab????b
output
4

input
9 4 ?????????
output
2

input
2 3 ??
output
0

input
15 3 ??b?babbc??b?aa
output
3

input
4 4 cabd
output
1

Note
In the first example the question marks can be replaced in the following way: "aaaababbbb". $f_1 = 4$, $f_2 = 4$, thus the answer is 4. Replacing it like this is also possible: "aaaabbbbbb". That way $f_1 = 4$, $f_2 = 6$, however, the minimum of them is still 4.

In the second example one of the possible strings is "aabbccdda".

In the third example at least one letter won't appear in the string, thus, the minimum of values f_i is always 0.

F. Jumping Around

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is an infinite pond that can be represented with a number line. There are n rocks in the pond, numbered from 1 to n. The i-th rock is located at an integer coordinate a_i. The coordinates of the rocks are pairwise distinct. The rocks are numbered in the

increasing order of the coordinate, so $a_1 < a_2 < \cdots < a_n$.

A robot frog sits on the rock number s. The frog is programmable. It has a base jumping distance parameter d. There also is a setting for the jumping distance range. If the jumping distance range is set to some integer k, then the frog can jump from some rock to any rock at a distance from $d - k$ to $d + k$ inclusive in any direction. The distance between two rocks is an absolute difference between their coordinates.

You are assigned a task to implement a feature for the frog. Given two integers i and k determine if the frog can reach a rock number i from a rock number s performing a sequence of jumps with the jumping distance range set to k. The sequence can be arbitrarily long or empty.

You will be given q testcases for that feature, the j-th testcase consists of two integers i and k. Print "Yes" if the i-th rock is reachable and "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 answer).

Input

The first line contains four integers n, q, s and d ($1 \le n, q \le 2 \cdot 10^5$; $1 \le s \le n$; $1 \le d \le 10^6$) — the number of rocks, the number of testcases, the starting rock and the base jumping distance parameter.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$) — the coordinates of the rocks. The coordinates of the rocks are pairwise distinct. The rocks are numbered in the increasing order of distance from the land, so $a_1 < a_2 < \cdots < a_n$.

Each of the next q lines contains two integers i and k ($1 \le i \le n$; $1 \le k \le 10^6$) — the parameters to the testcase.

Output

For each of the testcases print an answer. If there is a sequence of jumps from a rock number s to a rock number i with the jumping distance range set to k, then print "Yes". Otherwise, print "No".

Examples

input
7 4 4 5 1 5 10 13 20 22 28 4 1 7 2 7 3 3 2
output
Yes No Yes Yes

input
10 8 6 11 1 2 4 7 8 9 11 13 19 20 2 13 5 8 8 1 6 15 1 15 2 7 7 6 8 9
output
Yes Yes No Yes Yes Yes Yes Yes

input
6 9 6 6 1 2 4 9 18 19 2 17 1 18 5 4 2 11 5 17 6 8 4 3 3 3 6 6
output
Yes

Yes
Yes
Yes
Yes
Yes
No
No
Yes

input
4 1 1 10 1 8 10 19 2 1
output
Yes

Note

Explanation of the first example:

In the first testcase the destination rock is the same as the starting rock, thus no jumps are required to reach it.

In the second testcase the frog can jump any distance in the range $[5-2; 5+2]$. Thus, it can reach rock number 5 (by jumping 7 to the right) and rock number 3 (by jumping 3 to the left). From rock number 3 it can reach rock number 2 (by jumping 5 to the left). From rock number 2 it can reach rock number 1 (by jumping 4 to the left). However, there is no way to reach rock number 7.

In the third testcase the frog can jump any distance in the range $[5-3; 5+3]$. Thus, it can reach rock number 7 by jumping to rock 5 first and to 7 afterwards.

The fourth testcase is shown in the explanation for the second testcase.