A. Wizard of Orz

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

There are n digital panels placed in a straight line. Each panel can show any digit from 0 to 9. Initially, all panels show 0.

Every second, the digit shown by each panel increases by 1. In other words, at the end of every second, a panel that showed 9 would now show 0, a panel that showed 0 would now show 1, a panel that showed 1 would now show 2, and so on.

When a panel is paused, the digit displayed on the panel does not change in the subsequent seconds.

You must pause exactly one of these panels, at any second you wish. Then, the panels adjacent to it get paused one second later, the panels adjacent to those get paused 2 seconds later, and so on. In other words, if you pause panel x, panel y (for all valid y) would be paused exactly $|x-y|$ seconds later.

For example, suppose there are 4 panels, and the 3-rd panel is paused when the digit 9 is on it.

- The panel 1 pauses 2 seconds later, so it has the digit 1;
- the panel 2 pauses 1 second later, so it has the digit 0;
- the panel 4 pauses 1 second later, so it has the digit 0.

The resulting 4-digit number is 1090. **Note that this example is not optimal for $n = 4$.**

Once all panels have been paused, you write the digits displayed on them from left to right, to form an n digit number (it can consist of leading zeros). What is the largest possible number you can get? Initially, all panels show 0.

Input

The first line of the input contains a single integer t ($1 \le t \le 100$) — the number of test cases. Each test case consists of a single line containing a single integer n ($1 \le n \le 2 \cdot 10^5$).

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

Output

For each test case, print the largest number you can achieve, if you pause one panel optimally.

Example

input

```
2
1
2
```

output

```
9
98
```

Note

In the first test case, it is optimal to pause the first panel when the number 9 is displayed on it.

In the second test case, it is optimal to pause the second panel when the number 8 is displayed on it.

B. Hills And Valleys

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

You are given a sequence of n integers a_1, a_2, ..., a_n. Let us call an index j ($2 \le j \le n-1$) a *hill* if $a_j > a_{j+1}$ and $a_j > a_{j-1}$; and let us call it a *valley* if $a_j < a_{j+1}$ and $a_j < a_{j-1}$.

Let us define the *intimidation value* of a sequence as the sum of the number of hills and the number of valleys in the sequence. You can change **exactly one** integer in the sequence to any number that you want, or let the sequence remain unchanged. What is the minimum *intimidation value* that you can achieve?

Input

The first line of the input contains a single integer t ($1 \le t \le 10000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 3 \cdot 10^5$).

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

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

Output

For each test case, print a single integer — the minimum intimidation value that you can achieve.

Example

input

```
4
3
1 5 3
5
2 2 2 2 2
6
1 6 2 5 2 10
5
1 6 2 5 1
```

output

```
0
0
1
0
```

Note

In the first test case, changing a_2 to 2 results in no hills and no valleys.

In the second test case, the best answer is just to leave the array as it is.

In the third test case, changing a_3 to 6 results in only one valley (at the index 5).

In the fourth test case, changing a_3 to 6 results in no hills and no valleys.

C. Three Bags

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

You are given **three** bags. Each bag contains a non-empty multiset of numbers. You can perform a number of operations on these bags. In one operation, you can choose any two non-empty bags, and choose one number from each of the bags. Let's say that you choose number a from the first bag and number b from the second bag. Then, you remove b from the second bag and replace a with $a - b$ in the first bag. Note that if there are multiple occurrences of these numbers, then you shall only remove/replace exactly one occurrence.

You have to perform these operations in such a way that you have exactly one number remaining in exactly one of the bags (the other two bags being empty). It can be shown that you can always apply these operations to receive such a configuration in the end. Among all these configurations, find the one which has the maximum number left in the end.

Input

The first line of the input contains three space-separated integers n_1, n_2 and n_3 ($1 \le n_1, n_2, n_3 \le 3 \cdot 10^5$, $1 \le n_1 + n_2 + n_3 \le 3 \cdot 10^5$) — the number of numbers in the three bags.

The i-th of the next three lines contain n_i space-separated integers $a_{i,1}$, $a_{i,2}$, ..., a_{i,n_i} ($1 \le a_{i,j} \le 10^9$) — the numbers in the i-th bag.

Output

Print a single integer — the maximum number which you can achieve in the end.

Examples

input

```
2 4 1
1 2
6 3 4 5
5
```

output

```
20
```

input

```
3 2 2
7 5 4
```

```
2 9
7 1
```

output

```
29
```

Note

In the first example input, let us perform the following operations:

$[1,2],[6,3,4,5],[5]$

$[-5,2],[3,4,5],[5]$ (Applying an operation to $(1,6)$)

$[-10,2],[3,4],[5]$ (Applying an operation to $(-5,5)$)

$[2],[3,4],[15]$ (Applying an operation to $(5,-10)$)

$[-1],[4],[15]$ (Applying an operation to $(2,3)$)

$[-5],[],[15]$ (Applying an operation to $(-1,4)$)

$[],[],[20]$ (Applying an operation to $(15,-5)$)

You can verify that you cannot achieve a bigger number. Hence, the answer is 20.

D. Sum of Paths

time limit per test: 3 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

There are n cells, numbered $1,2,\ldots,n$ from left to right. You have to place a robot at any cell initially. The robot must make **exactly** k moves.

In one move, the robot must move one cell to the left or right, provided that it doesn't move out of bounds. In other words, if the robot was in the cell i, it must move to either the cell $i-1$ or the cell $i+1$, as long as it lies between 1 and n (endpoints inclusive). The cells, in the order they are visited (including the cell the robot is placed), together make a *good path*.

Each cell i has a value a_i associated with it. Let $c_0, c_1, \ldots, c_k$ be the sequence of cells in a *good path* in the order they are visited (c_0 is the cell robot is initially placed, c_1 is the cell where the robot is after its first move, and so on; more formally, c_i is the cell that the robot is at after i moves). Then the value of the path is calculated as $a_{c_0}+a_{c_1}+\cdots+a_{c_k}$.

Your task is to calculate the sum of values over all possible *good paths*. Since this number can be very large, output it modulo 10^9+7. Two *good paths* are considered different if the starting cell differs or there exists an integer $i\in[1,k]$ such that the current cell of the robot after exactly i moves is different in those paths.

You must process q updates to a and print the updated sum each time. Each update changes the value of exactly one cell. See the input format and the sample input-output for more details.

Input

The first line of the input contains three space-separated integers n, k and q ($2\le n\le 5000$; $1\le k\le 5000$; $1\le q\le 2\cdot 10^5$).

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

q lines follow. Each line contains two space-separated integers i and x ($1\le i\le n$; $1\le x\le 10^9$) indicating that you must change the value of a_i to x.

Output

Print q integers. The i-th integer should be the sum of values over all *good paths* after the first i updates are performed. Since the answers may be large, print them modulo 10^9+7.

Examples

input

```
5 1 5
3 5 1 4 2
1 9
2 4
3 6
4 6
5 2
```

output

```
62
58
78
86
86
```

input

```
5 2 5
3 5 1 4 2
1 9
2 4
3 6
4 6
5 2
```

output

```
157
147
207
227
227
```

input

```
4 40 6
92 21 82 46
3 56
1 72
4 28
1 97
2 49
2 88
```

output

```
239185261
666314041
50729936
516818968
766409450
756910476
```

Note

In the first example, the *good paths* are $(1,2),(2,1),(2,3),(3,2),(3,4),(4,3),(4,5),(5,4)$.

Initially the values of a are $[3,5,1,4,2]$. After the first update, they become $[9,5,1,4,2]$. After the second update, they become $[9,4,1,4,2]$, and so on.

E. Distinctive Roots in a Tree

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given a tree with n vertices. Each vertex i has a value a_i associated with it.

Let us root the tree at some vertex v. The vertex v is called a *distinctive root* if the following holds: in all paths that start at v and end at some other node, all the values encountered are distinct. Two different paths may have values in common but a single path must have all distinct values.

Find the number of *distinctive roots* in the tree.

Input

The first line of the input contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

The next line contains n space-separated integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$).

The following $n-1$ lines each contain two space-separated integers u and v ($1 \le u$, $v \le n$), denoting an edge from u to v.

It is guaranteed that the edges form a tree.

Output

Print a single integer — the number of *distinctive roots* in the tree.

Examples

input

```
5
2 5 1 1 4
1 2
1 3
2 4
2 5
```

output

```
3
```

input

```
5
2 1 1 1 4
1 2
1 3
2 4
2 5
```

output

```
0
```

Note

In the first example, 1, 2 and 5 are *distinctive roots*.