A. Great Graphs

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

Farmer John has a farm that consists of n pastures connected by one-directional roads. Each road has a weight, representing the time it takes to go from the start to the end of the road. The roads could have negative weight, where the cows go so fast that they go back in time! However, Farmer John guarantees that it is impossible for the cows to get stuck in a time loop, where they can infinitely go back in time by traveling across a sequence of roads. Also, each pair of pastures is connected by at most one road in each direction.

Unfortunately, Farmer John lost the map of the farm. All he remembers is an array d, where d_i is the smallest amount of time it took the cows to reach the i-th pasture from pasture 1 using a sequence of roads. The cost of his farm is the sum of the weights of each of the roads, and Farmer John needs to know the **minimal** cost of a farm that is consistent with his memory.

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 10^5$) — the number of pastures.

The second line of each test case contains n space separated integers $d_1, d_2, \ldots, d_n$ ($0 \le d_i \le 10^9$) — the array d. It is guaranteed that $d_1 = 0$.

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

Output

For each test case, output the minimum possible cost of a farm that is consistent with Farmer John's memory.

Example

input

```
3
3
0 2 3
2
0 1000000000
1
0
```

output

```
-3
0
0
```

Note

In the first test case, you can add roads

- from pasture 1 to pasture 2 with a time of 2,
- from pasture 2 to pasture 3 with a time of 1,
- from pasture 3 to pasture 1 with a time of -3,
- from pasture 3 to pasture 2 with a time of -1,
- from pasture 2 to pasture 1 with a time of -2.

The total cost is $2 + 1 + -3 + -1 + -2 = -3$.
In the second test case, you can add a road from pasture 1 to pasture 2 with cost 1000000000 and a road from pasture 2 to pasture 1 with cost -1000000000. The total cost is $1000000000 + -1000000000 = 0$.

In the third test case, you can't add any roads. The total cost is 0.

B. Tree Array

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

You are given a tree consisting of n nodes. You generate an array from the tree by marking nodes one by one.

Initially, when no nodes are marked, a node is equiprobably chosen and marked from the entire tree.

After that, until all nodes are marked, a node is equiprobably chosen and marked from the set of unmarked nodes with at least one edge to a marked node.

It can be shown that the process marks all nodes in the tree.

The final array a is the list of the nodes' labels in order of the time each node was marked.

Find the expected number of inversions in the array that is generated by the tree and the aforementioned process.

The number of inversions in an array a is the number of pairs of indices (i, j) such that $i < j$ and $a_i > a_j$. For example, the array $[4, 1, 3, 2]$ contains 4 inversions: $(1, 2)$, $(1, 3)$, $(1, 4)$, $(3, 4)$.

Input

The first line contains a single integer n ($2 \le n \le 200$) — the number of nodes in the tree.

The next $n - 1$ lines each contains two integers x and y ($1 \le x, y \le n$; $x \ne y$), denoting an edge between node x and y.

It's guaranteed that the given edges form a tree.

Output

Output the expected number of inversions in the generated array modulo $10^9 + 7$.

Formally, let $M = 10^9 + 7$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Examples

input

```
3
1 2
1 3
```

output

```
166666669
```

input

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

output

```
500000009
```

input

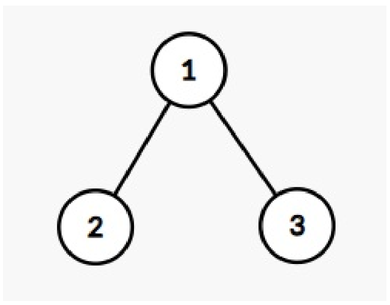
```
5
1 2
1 3
1 4
2 5
```

output

```
500000007
```

Note

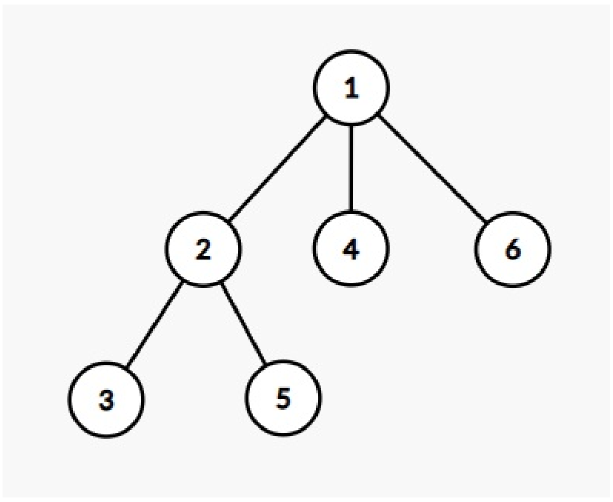
This is the tree from the first sample:



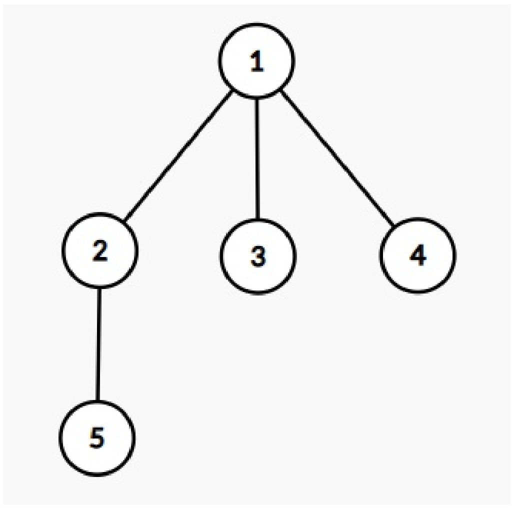
For the first sample, the arrays are almost fixed. If node 2 is chosen initially, then the only possible array is $[2, 1, 3]$ (1 inversion). If node 3 is chosen initially, then the only possible array is $[3, 1, 2]$ (2 inversions). If node 1 is chosen initially, the arrays $[1, 2, 3]$ (0 inversions) and $[1, 3, 2]$ (1 inversion) are the only possibilities and equiprobable. In total, the expected number of inversions is $\frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 2 + \frac{1}{3} \cdot (\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1) = \frac{7}{6}$.

$166666669 \cdot 6 = 7 \pmod{10^9 + 7}$, so the answer is 166666669.

This is the tree from the second sample:



This is the tree from the third sample:



C1. Converging Array (Easy Version)

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

This is the easy version of the problem. The only difference is that in this version $q = 1$. You can make hacks only if both versions of the problem are solved.

There is a process that takes place on arrays a and b of length n and length $n-1$ respectively.

The process is an infinite sequence of operations. Each operation is as follows:

- First, choose a random integer i ($1 \le i \le n-1$).
- Then, simultaneously set $a_i = \min\left(a_i, \frac{a_i+a_{i+1}-b_i}{2}\right)$ and $a_{i+1} = \max\left(a_{i+1}, \frac{a_i+a_{i+1}+b_i}{2}\right)$ without any rounding (so values may become non-integer).

See notes for an example of an operation.
It can be proven that array a converges, i. e. for each i there exists a limit a_i converges to. Let function $F(a, b)$ return the value a_1 converges to after a process on a and b.

You are given array b, but not array a. However, you are given a third array c. Array a is good if it contains only **integers** and satisfies $0 \le a_i \le c_i$ for $1 \le i \le n$.

Your task is to count the number of good arrays a where $F(a, b) \ge x$ for q values of x. Since the number of arrays can be very large, print it modulo $10^9 + 7$.

Input

The first line contains a single integer n ($2 \le n \le 100$).

The second line contains n integers $c_1, c_2 \ldots, c_n$ ($0 \le c_i \le 100$).

The third line contains $n-1$ integers $b_1, b_2, \ldots, b_{n-1}$ ($0 \le b_i \le 100$).

The fourth line contains a single integer q ($q = 1$).

The fifth line contains q space separated integers $x_1, x_2, \dots, x_q$ ($-10^5 \le x_i \le 10^5$).

Output

Output q integers, where the i-th integer is the answer to the i-th query, i. e. the number of good arrays a where $F(a, b) \ge x_i$ modulo $10^9 + 7$.

Example

input

```
3
2 3 4
2 1
1
-1
```

output

```
56
```

Note

The following explanation assumes $b = [2, 1]$ and $c = [2, 3, 4]$ (as in the sample).

Examples of arrays a that are **not** good:

- $a = [3, 2, 3]$ is not good because $a_1 > c_1$;
- $a = [0, -1, 3]$ is not good because $a_2 < 0$.

One possible good array a is $[0, 2, 4]$. We can show that no operation has any effect on this array, so $F(a, b) = a_1 = 0$.

Another possible good array a is $[0, 1, 4]$. In a single operation with $i = 1$, we set $a_1 = \min(\frac{0+1-2}{2}, 0)$ and $a_2 = \max(\frac{0+1+2}{2}, 1)$. So, after a single operation with $i = 1$, a becomes equal to $[-\frac{1}{2}, \frac{3}{2}, 4]$. We can show that no operation has any effect on this array, so $F(a, b) = -\frac{1}{2}$.

C2. Converging Array (Hard Version)

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

This is the hard version of the problem. The only difference is that in this version $1 \le q \le 10^5$. You can make hacks only if both versions of the problem are solved.

There is a process that takes place on arrays a and b of length n and length $n - 1$ respectively.

The process is an infinite sequence of operations. Each operation is as follows:

- First, choose a random integer i ($1 \le i \le n - 1$).
- Then, simultaneously set $a_i = \min\left(a_i, \frac{a_i + a_{i+1} - b_i}{2}\right)$ and $a_{i+1} = \max\left(a_{i+1}, \frac{a_i + a_{i+1} + b_i}{2}\right)$ without any rounding (so values may become non-integer).

See notes for an example of an operation.
It can be proven that array a converges, i. e. for each i there exists a limit a_i converges to. Let function $F(a, b)$ return the value a_1 converges to after a process on a and b.

You are given array b, but not array a. However, you are given a third array c. Array a is good if it contains only **integers** and satisfies $0 \le a_i \le c_i$ for $1 \le i \le n$.

Your task is to count the number of good arrays a where $F(a, b) \ge x$ for q values of x. Since the number of arrays can be very large, print it modulo $10^9 + 7$.

Input

The first line contains a single integer n ($2 \le n \le 100$).

The second line contains n integers $c_1, c_2 \dots, c_n$ ($0 \le c_i \le 100$).

The third line contains $n - 1$ integers $b_1, b_2, \dots, b_{n-1}$ ($0 \le b_i \le 100$).

The fourth line contains a single integer q ($1 \le q \le 10^5$).

The fifth line contains q space separated integers $x_1, x_2, \dots, x_q$ ($-10^5 \le x_i \le 10^5$).

Output

Output q integers, where the i-th integer is the answer to the i-th query, i. e. the number of good arrays a where $F(a, b) \ge x_i$ modulo $10^9 + 7$.

Example

input

```
3
2 3 4
2 1
5
-1 0 1 -100000 100000
```

output

```
56
28
4
60
0
```

Note

The following explanation assumes $b = [2, 1]$ and $c = [2, 3, 4]$ (as in the sample).

Examples of arrays a that are **not** good:

- $a = [3, 2, 3]$ is not good because $a_1 > c_1$;
- $a = [0, -1, 3]$ is not good because $a_2 < 0$.

One possible good array a is $[0, 2, 4]$. We can show that no operation has any effect on this array, so $F(a, b) = a_1 = 0$.

Another possible good array a is $[0, 1, 4]$. In a single operation with $i = 1$, we set $a_1 = \min(\frac{0+1-2}{2}, 0)$ and $a_2 = \max(\frac{0+1+2}{2}, 1)$. So, after a single operation with $i = 1$, a becomes equal to $[-\frac{1}{2}, \frac{3}{2}, 4]$. We can show that no operation has any effect on this array, so $F(a, b) = -\frac{1}{2}$.

D. Inverse Inversions

time limit per test: 5 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You were playing with permutation p of length n, but you lost it in Blair, Alabama!

Luckily, you remember some information about the permutation. More specifically, you remember an array b of length n, where b_i is the number of indices j such that $j < i$ and $p_j > p_i$.

You have the array b, and you want to find the permutation p. However, your memory isn't perfect, and you constantly change the values of b as you learn more. For the next q seconds, one of the following things happen:

1. $1\ i\ x$ — you realize that b_i is equal to x;
2. $2\ i$ — you need to find the value of p_i. If there's more than one answer, print any. It can be proven that there's always at least one possible answer under the constraints of the problem.

Answer the queries, so you can remember the array!

Input

The first line contains a single integer n ($1 \le n \le 10^5$) — the size of permutation.

The second line contains n integers $b_1, b_2 \ldots, b_n$ ($0 \le b_i < i$) — your initial memory of the array b.

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

The next q lines contain the queries, each with one of the following formats:

- $1\ i\ x$ ($0 \le x < i \le n$), representing a query of type 1.
- $2\ i$ ($1 \le i \le n$), representing a query of type 2.

It is guaranteed that there's at least one query of type 2.

Output

For each query of type 2, print one integer — the answer to the query.

Examples

input

```
3
0 0 0
7
2 1
2 2
2 3
1 2 1
2 1
2 2
2 3
```

output

```
1
2
3
2
1
3
```

input

```
5
0 1 2 3 4
15
2 1
2 2
1 2 1
2 2
2 3
2 5
1 3 0
1 4 0
2 3
2 4
2 5
1 4 1
2 3
2 4
2 5
```

output

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

Note

For the first sample, there's initially only one possible permutation that satisfies the constraints: $[1, 2, 3]$, as it must have 0 inversions.

After the query of type 1, the array b is $[0, 1, 0]$. The only permutation p that produces this array is $[2, 1, 3]$. With this permutation, b_2 is equal to 1 as $p_1 > p_2$.

E. Tasty Dishes

time limit per test: 10 seconds
memory limit per test: 64 megabytes
input: standard input
output: standard output

Note that the memory limit is unusual.

There are n chefs numbered $1, 2, \ldots, n$ that must prepare dishes for a king. Chef i has skill i and initially has a dish of tastiness a_i where $|a_i| \leq i$. Each chef has a list of other chefs that he is allowed to copy from. To stop chefs from learning bad habits, the king makes sure that chef i can only copy from chefs of larger skill.

There are a sequence of days that pass during which the chefs can work on their dish. During each day, there are two stages during which a chef can change the tastiness of their dish.

1. At the beginning of each day, each chef can *choose* to work (or not work) on their own dish, thereby multiplying the tastiness of their dish of their skill ($a_i := i \cdot a_i$) (or doing nothing).
2. After all chefs (who wanted) worked on their own dishes, each start observing the other chefs. In particular, for each chef j on chef i's list, chef i can *choose* to copy (or not copy) j's dish, thereby adding the tastiness of the j's dish to i's dish ($a_i := a_i + a_j$) (or doing nothing). It can be assumed that all copying occurs simultaneously. Namely, if chef i chooses to copy from chef j he will copy the tastiness of chef j's dish at the end of stage 1.

All chefs work to maximize the tastiness of their *own* dish in order to please the king.

Finally, you are given q queries. Each query is one of two types.

1. $1\ k\ l\ r$ — find the sum of tastiness $a_l, a_{l+1}, \ldots, a_r$ after the k-th day. Because this value can be large, find it modulo $10^9 + 7$.
2. $2\ i\ x$ — the king adds x tastiness to the i-th chef's dish before the 1-st day begins ($a_i := a_i + x$). Note that, because the king wants to see tastier dishes, he only adds positive tastiness ($x > 0$).

Note that queries of type 1 are independent of each all other queries. Specifically, each query of type 1 is a *scenario* and does not change the initial tastiness a_i of any dish for future queries. Note that queries of type 2 are cumulative and only change the initial tastiness a_i of a dish. See notes for an example of queries.

Input

The first line contains a single integer n ($1 \le n \le 300$) — the number of chefs.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-i \le a_i \le i$).

The next n lines each begin with a integer c_i ($0 \le c_i < n$), denoting the number of chefs the i-th chef can copy from. This number is followed by c_i distinct integers d ($i < d \le n$), signifying that chef i is allowed to copy from chef d during stage 2 of each day.

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

Each of the next q lines contains a query of one of two types:

- $1\ k\ l\ r$ ($1 \le l \le r \le n$; $1 \le k \le 1000$);
- $2\ i\ x$ ($1 \le i \le n$; $1 \le x \le 1000$).

It is guaranteed that there is at least one query of the first type.

Output

For each query of the first type, print a single integer — the answer to the query.

Example

input

```
5
1 0 -2 -2 4
4 2 3 4 5
1 3
1 4
1 5
0
7
1 1 1 5
2 4 3
1 1 1 5
2 3 2
1 2 2 4
2 5 1
1 981 4 5
```

output

```
57
71
316
278497818
```

Note

Below is the set of chefs that each chef is allowed to copy from:

- 1: $\{2, 3, 4, 5\}$
- 2: $\{3\}$
- 3: $\{4\}$
- 4: $\{5\}$
- 5: $\emptyset$ (no other chefs)

Following is a description of the sample.

For the first query of type 1, the initial tastiness values are $[1, 0, -2, -2, 4]$.

The final result of the first day is shown below:

1. $[1, 0, -2, -2, 20]$ (chef 5 works on his dish).
2. $[21, 0, -2, 18, 20]$ (chef 1 and chef 4 copy from chef 5).

So, the answer for the 1-st query is $21 + 0 - 2 + 18 + 20 = 57$.

For the 5-th query (3-rd of type 1). The initial tastiness values are now $[1, 0, 0, 1, 4]$.

Day 1

1. $[1, 0, 0, 4, 20]$ (chefs 4 and 5 work on their dishes).
2. $[25, 0, 4, 24, 20]$ (chef 1 copies from chefs 4 and 5, chef 3 copies from chef 4, chef 4 copies from chef 5).

Day 2

1. $[25, 0, 12, 96, 100]$ (all chefs but chef 2 work on their dish).
2. $[233, 12, 108, 196, 100]$ (chef 1 copies from chefs 3, 4 and 5, chef 2 from 3, chef 3 from 4, chef 4 from chef 5).
   So, the answer for the 5-th query is $12 + 108 + 196 = 316$.

It can be shown that, in each step we described, all chefs moved optimally.