

# 2019 Team Formation Test

## Task Overview

ID	Name	Time Limit	Memory Limit	Subtasks
T191	Distributing Cards	3.000 s	512 MB	11 + 17 + 19 + 24 + 29
T192	Colorful Strip	2.000 s	512 MB	10 + 25 + 30 + 28 + 7
T193	Liquid Layers	1.000 s	256 MB	100 (Partial score available)
T194	Roulette	1.000 s	256 MB	8 + 9 + 12 + 34 + 37

**Notice:**

Unless otherwise specified, inputs and outputs shall follow the format below:

- One space between a number and another number or character in the same line.
- No space between characters in the same line.
- Each string shall be placed in its own separate line.
- Outputs will be automatically fixed as follows: Trailing spaces in each line will be removed and an end-of-line character will be added to the end of the output if not present. All other format errors will not be fixed.

C++ programmers should be aware that using C++ streams (`cin` / `cout`) may lead to I/O bottlenecks and substantially lower performance.

For some problems 64-bit integers may be required. In Pascal it is `int64`. In C/C++ it is `long long` and its token for `scanf` / `printf` is `%lld`.

All tasks are divided into subtasks. You need to pass all test cases in a subtask to get points.

## T191 - DISTRIBUTING CARDS

Time Limit: 3.000 s / Memory Limit: 512 MB

Recently, a Trading Card Game (TCG) called Shadowverse blew up the whole Byteland! Both Alice and Bob are addicted to this card game. Today, they are going to buy the newly released Shadowverse deck. Due to the budget constraint, they decide to chip in for a single deck instead of buying one deck for themselves on their own.

After getting the new deck, their next task is to discuss how to distribute the cards in the deck. Alice and Bob then come up with a method to distribute the cards fairly. Initially, they put all  $N$  cards in the deck and arrange them in a ring. More precisely, they put down the 1<sup>st</sup> card at first, then put the 2<sup>nd</sup> card to the right of the 1<sup>st</sup> card, then the 3<sup>rd</sup> card, 4<sup>th</sup> card, and so on. Finally, they put down the  $N^{\text{th}}$  card so that the 1<sup>st</sup> card is to its right.

Alice then lets Bob take any card (and any number of cards) he wants one by one. However, she also sets a cost  $c_i$  on each card such that, after Bob takes the  $x^{\text{th}}$  card, Alice will take the next  $c_x$  cards to the right of the  $x^{\text{th}}$  card. If there will be less than  $c_x$  cards after Bob takes the  $x^{\text{th}}$  card, Bob is not permitted to take it.

Bob is now thinking about what strategy can be used for obtaining all the cards he likes. Each card belongs to one of the two types: Follower or Spell. Bob likes Follower cards only and he would like to obtain all Follower cards. In contrast, he hates Spell card so much and he refuses to take any Spell card.

Bob finds that different orders of taking cards may affect whether he can obtain all the Follower cards. Therefore, Bob is now curious about how many different card-taking orders there are, such that he can take all the Follower cards according to the order. Note that Bob will never take any Spell card.

## INPUT

The first line contains an integer  $N$ , the number of cards in the deck.

The second line contains a string with  $N$  characters. Each character is either **F** or **S**. If the  $i^{\text{th}}$  character is **F**, it means that the  $i^{\text{th}}$  card is a Follower card; otherwise, the  $i^{\text{th}}$  card is a Spell card.

The third line contains  $N$  integers  $c_i$ , denoting the cost of the  $i^{\text{th}}$  card.

## OUTPUT

Output a single integer in one line, which is the number of different card-taking orders such that Bob can obtain all the Follower cards. As the number may be large, print it modulo  $10^9 + 7$ .

## SAMPLE TESTS

	Input	Output
1	<div>6</div> <div>FSSSFS</div> <div>2 0 0 0 2 0</div>	<div>1</div>

There is only one valid card-taking order: take the 1<sup>st</sup> card and give the 2<sup>nd</sup> and 3<sup>rd</sup> cards to Alice; then, take the 5<sup>th</sup> card and give the 6<sup>th</sup> and 4<sup>th</sup> cards to Alice.

2	<div>4</div> <div>FFFS</div> <div>1 0 0 0</div>	<div>2</div>
---	---	--------------

The two valid ways are 2<sup>nd</sup> → 3<sup>rd</sup> → 1<sup>st</sup> and 3<sup>rd</sup> → 2<sup>nd</sup> → 1<sup>st</sup>.

3	<div>4</div> <div>FFFF</div> <div>1 0 0 0</div>	<div>0</div>
---	---	--------------

4	<div>5</div> <div>FSSSF</div> <div>4 0 0 0 1</div>	<div>0</div>
---	--	--------------

## SUBTASKS

For all cases:

$$1 \leq N \leq 10^6$$

$$0 \leq c_i \leq N$$

$c_i = 0$  if the  $i^{\text{th}}$  card is a Spell card.

It is guaranteed that there will be at least one Follower card.

	Points	Constraints
1	11	$1 \leq N \leq 10$
2	17	Number of Follower cards $\leq 10$
3	19	$0 \leq c_i \leq 1$ For any card $i$ , $c_i = 0$ if the card in its left is a Follower card.
4	24	$1 \leq N \leq 1000$
5	29	No additional constraints

## T192 - COLORFUL STRIP

Time Limit: 2.000 s / Memory Limit: 512 MB

Mr Tung Cow received an electronic strip as his graduation gift! The cool thing about this strip is that it is divided into  $N$  cells. Initially, each cell is either red (R), green (G) or blue (B).

"Wow! What an amazing strip!", Mr Tung Cow thought. But the coolest thing is yet to come: the colour in each cell could be changed! There are two ways to change the cells' colours. The first way is to change a continuous segment of cells to a specific colour. The second way is to shift the colours of a continuous segment of cells.

Formally, for the first way, Mr Tung Cow can choose two integers  $x, y$  and a color  $C$ , where  $1 \leq x \leq y \leq N$  and  $C$  is either R, G, or B. Then, the colours of all cells from  $x$  to  $y$  will be changed to  $C$ .

For the second way, Mr Tung Cow can choose two integers  $x, y$  where  $1 \leq x \leq y \leq N$ . For  $x \leq i \leq y$ , if cell  $i$  is red (R), it will become green (G); if cell  $i$  is green (G), it will become blue (B); if cell  $i$  is blue (B), it will become red (R). In other words, shifting is the simultaneous applications of the three color-change rules to all cells from  $x$  to  $y$ :  $R \rightarrow G, G \rightarrow B, B \rightarrow R$ .

Also, Mr Tung Cow loves colourful things. And as a mathematician (mathematicow?), he loves defining things. He defines a colourful strip as a continuous sequence of cells containing all three colours (R, G and B). For example, the strip "RGGRB" has three colourful strips:

RGGRB (1, 5)

GGRB (2, 5)

GRB (3, 5)

Now there are  $Q$  events happening in chronological order.

For the  $i^{th}$  event, it can only be one of the three types below:

- Type 1: Mr Tung Cow changes the colours of the  $x_i^{th}$  to the  $y_i^{th}$  cells to  $C_i$  (first way of changing colour mentioned above).
- Type 2: Mr Tung Cow shifts the colours of the  $x_i^{th}$  to the  $y_i^{th}$  cells (second way of changing colour mentioned above).
- Type 3: Mr Tung Cow gives you two integers  $x_i, y_i$ , asking you how many colourful strips  $(p, q)$  exists such that  $x_i \leq p \leq q \leq y_i$ .

As he is Mr Tung Cow, you have to answer him. Please output the answers to all the type 3 events in chronological order.

## INPUT

The first line contains an integer  $N$ , which is the length of the strip.

The second line contains a length- $N$  string. Each character is either R, G, or B. The  $i^{th}$  character represents the initial color of the  $i^{th}$  cell of the strip.

The third line contains an integer  $Q$ , which is the number of events.

Then  $Q$  lines follow. On the  $i^{th}$  line, the first integer  $t_i$  denotes the type of event  $i$ .

- If  $t_i = 1$ , then two integers  $x_i, y_i$  and a character  $C_i$  follow, where  $1 \leq x_i \leq y_i \leq N$  and  $C_i$  is either R, G, or B.
- If  $t_i = 2$  or  $t_i = 3$ , then two integers  $x_i, y_i$  follow, where  $1 \leq x_i \leq y_i \leq N$ .

## OUTPUT

Answer all type 3 events in order, one per line.

## SAMPLE TESTS

	Input	Output
<b>1</b>	5 RGGRB 8 3 1 5 3 1 3 2 3 3 3 1 5 3 1 3 1 3 4 G 3 1 5 3 2 5	3 0 5 1 1 0

After event 1: RGGRB

After event 2: RGGRB

After event 3: RGGRB

After event 4: RGGRB

After event 5: RGGRB

After event 6: RGGGB

After event 7: RGGGB

After event 8: RGGGB

<b>2</b>	6 RGBRGB 5 3 1 6 2 1 3 3 2 5 1 1 1 R 3 1 6	10 1 6
----------	---	--------------

## SUBTASKS

For all cases:  $1 \leq N, Q \leq 4 \times 10^5$ 

	Points	Constraints
<b>1</b>	10	$1 \leq N \leq 100$ $1 \leq Q \leq 1000$
<b>2</b>	25	$1 \leq N, Q \leq 5000$
<b>3</b>	30	$1 \leq N, Q \leq 2 \times 10^5$ There is no type 2 event and $x_i = y_i$ for all type 1 events.
<b>4</b>	28	$1 \leq N, Q \leq 2 \times 10^5$
<b>5</b>	7	No additional constraints.

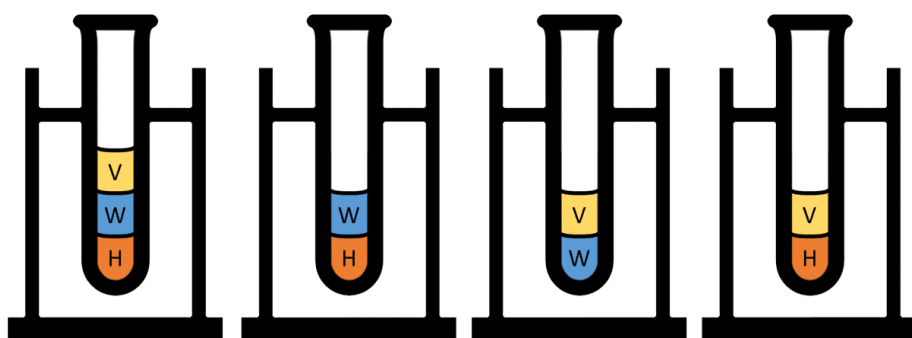
## T193 - LIQUID LAYERS

Time Limit: 1.000 s / Memory Limit: 256 MB

Your laboratory has just received a secret mission from the Byteland Government. They have sent you a box of  $N$  bottles, labeled from 1 to  $N$ , where  $2 \leq N \leq 100$ . It is known that each of these bottles contains a unique type of liquid. The  $N$  types of liquids are all colorless and tasteless, with densities different from each other.

It is also known that these liquids do not mix with each other, meaning that if you pour any combination of these liquids into a container, they will separate into layers. The layers are always ordered according to their densities: liquid with higher density goes down, whereas liquid with lower density goes up. In the end, the layers should satisfy that: for any pair of liquid types in the same container, the one with higher density will have its layer at a position closer to the bottom.

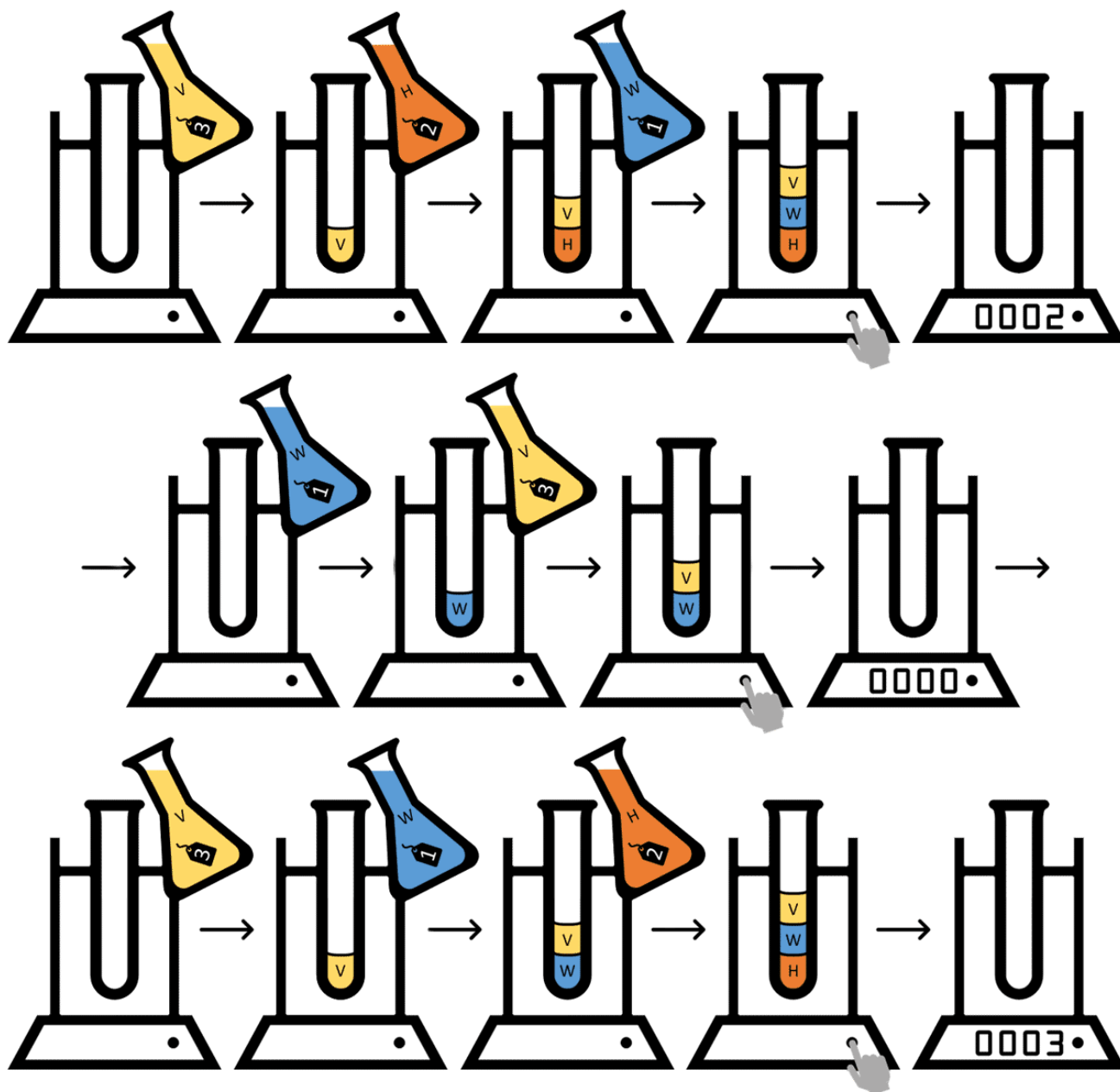
Here are a few examples of pouring honey ( $\text{H}$ ), water ( $\text{W}$ ) and vegetable oil ( $\text{V}$ ) into the same container: (order of density from highest to lowest: honey, water, vegetable oil)



Unfortunately, your laboratory is not equipped with any machine that can directly measure the density of liquids. The lucky thing is that, there is a special machine that may help. This special machine is composed of a long test tube, a button, a counter, and a display. Initially, the test tube is empty and the counter is set to zero. To use the machine, you can pour one type of liquid into the test tube, as long as the test tube does not contain that type of liquid. The suggested volume is 3mL, but actually how much you pour does not matter. Because of density difference between the liquids, the newly-added liquid may swap positions with other liquid layers. Once the swapping is done, the counter will be increased by the number of swaps, and you can then pour another type of liquid into the test tube, or press the button (the effect of pressing the button will be described in the next paragraph).

Due to the constraints of this machine, the value of the counter will not be shown unless you press the button. Once you press the button, the value of the counter will be shown on the display for a short period of time so that you could read it, but the side effect is that the machine will automatically empty the test tube and reset the counter to zero.

The following figure illustrates how this special machine works. You may refer to the section "Sample Run" for detailed explanation. Please note that the colors and letters on the liquids are just for better visualization.



## YOUR TASK

You are asked to order the  $N$  ( $2 \leq N \leq 100$ ) given liquids by their densities, from highest to lowest, by utilizing the special machine. To obtain the correct order as quickly as possible, you are required to minimize the number of times of pouring liquid into the special machine.

It is guaranteed that, before the start of the experiment, the test tube is empty and the counter is set to zero.

## IMPLEMENTATION

You should implement one subprogram `experiment(int N)` to simulate the process of conducting the experiment with  $N$  bottles of liquid given.

To simulate the experiment, your subprogram `experiment(int N)` is allowed to make calls to the following two grader functions (you do not have to implement these grader functions):

- procedure `pourLiquid(int index)`
  - It simulates the process of pouring some amount of the liquid in the bottle labeled as *index* into the test tube of the special machine.
  - You are not allowed to pour a liquid if, at that moment, the test tube already contains that type of liquid.
- function `getReading()`
  - It simulates the process of pressing the button of the special machine.
  - The return value of this function is the value shown on the display.
  - After this function is called, the test tube will be cleared and the counter will be reset to zero.
  - You are not allowed to call this function when the test tube is empty.

Once you are confident in the density order of the  $N$  liquids, you should call the grader procedure `answer(int order[]) EXACTLY ONCE`. `order` should be an array of length  $N$ , where `order[i]` is the label of the bottle that contains the liquid with the  $(i + 1)$ -th highest density. In other words:

- the liquid with label `order[0]` has the highest density.
- the liquid with label `order[1]` has the second highest density.
- ...
- the liquid with label `order[N-1]` has the lowest density.

After calling `answer(int order[])`, your subprogram `experiment(int N)` should return immediately. You can assume that, for each test case, the grader will call your subprogram `experiment(int N)` exactly once.

## SAMPLE RUN

Suppose  $N = 3$ , liquid 1 is water (second highest density), liquid 2 is honey (highest density), liquid 3 is vegetable oil (lowest density).

Function Call	Returns	Explanation
<code>experiment(3)</code>		<i>is called.</i>
<code>pourLiquid(3)</code>		Pouring vegetable oil into the test tube. The test tube contains (from bottom to top): vegetable oil The value of counter: 0
<code>pourLiquid(2)</code>		Pouring honey into the test tube. The test tube contains (from bottom to top): honey, vegetable oil The value of counter: 1
<code>pourLiquid(1)</code>		Pouring water into the test tube. The test tube contains (from bottom to top): honey, water, vegetable oil The value of counter: 2
<code>getReading()</code>	2	Returned the value of counter. Cleared the test tube. Reset the counter to zero.
<code>pourLiquid(1)</code>		Pouring water into the test tube. The test tube contains (from bottom to top): water The value of counter: 0
<code>pourLiquid(3)</code>		Pouring vegetable oil into the test tube. The test tube contains (from bottom to top): water, vegetable oil The value of counter: 0
<code>getReading()</code>	0	Returned the value of counter. Cleared the test tube. Reset the counter to zero.
<code>pourLiquid(3)</code>		Pouring vegetable oil into the test tube. The test tube contains (from bottom to top): vegetable oil The value of counter: 0
<code>pourLiquid(1)</code>		Pouring water into the test tube. The test tube contains (from bottom to top): water, vegetable oil The value of counter: 1
<code>pourLiquid(2)</code>		Pouring honey into the test tube. The test tube contains (from bottom to top): honey, water, vegetable oil The value of counter: 3
<code>getReading()</code>	3	Returned the value of counter. Cleared the test tube. Reset the counter to zero.
<code>answer([2, 1, 3])</code>		It can be deduced that liquid 2 has the highest density, followed by liquid 1, and then liquid 3.
<code>experiment(3)</code>		<i>returns.</i>

## SCORING

**IMPORTANT:** In some test cases the behavior of the grader is adaptive. This means that in these test cases the grader does not have a fixed order of density of the  $N$  liquids. Instead, the answers given by the grader may depend on the questions asked by your solution. It is guaranteed that the grader answers in such a way that after each answer there is at least one order of density consistent with all the answers given so far.

On each test case, you will receive zero score if your program:

- does not exit correctly, or
- violates any rules mentioned in the section "Implementation", or
- does not order the liquid correctly.

Otherwise, your score will depend on  $C$ , the number of times `pourLiquid(int index)` is called.

$$score = \begin{cases} 100, & \text{if } C \leq 1016 \\ 60 + 40 \times \frac{1083 - C}{1083 - 1016}, & \text{if } 1016 < C \leq 1083 \\ 48 + 12 \times \frac{1146 - C}{1146 - 1083}, & \text{if } 1083 < C \leq 1146 \\ 5 + 40 \div (1 + e^{\frac{C - 4200}{500}}), & \text{if } 1146 < C \leq 9900 \\ 0, & \text{if } C > 9900 \end{cases}$$

Your score on this task is the lowest score you get among all test cases.

## TEMPLATE

The template below allows you to implement the necessary procedure.

(Please refer to the online version)

## SAMPLE GRADER

In order to test your program, you may download the sample grader files. To use the sample grader, you should implement your program under the folder corresponding to the programming language, and follow the instructions below:

Language	Source Code Filename	Compilation Command	Execution Command
Pascal	experiment.pas	./compile_pas.sh	./experiment
C	experiment.c	./compile_c.sh	./experiment
C++11	experiment.cpp	./compile_cpp.sh	./experiment

Please be advised that although the sample grader is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. For instance, the sample grader is not adaptive.

When testing your programs with the sample grader, your input should match the format and constraints from the task statement. Otherwise, unspecified behaviors may occur. The sample grader reads the input in the following format:

- line 1:  $N$
- line 2:  $order_0 \ order_1 \ \dots \ order_{N-1}$ , where
  - the liquid with label  $order_0$  has the highest density.
  - the liquid with label  $order_1$  has the second highest density.
  - ...
  - the liquid with label  $order_{N-1}$  has the lowest density.

The labels should be distinct integers between 1 to  $N$  (inclusive).

If your program correctly finds out the order of the density, the sample grader outputs **Correct**, followed by the value of  $C$  and your score on that case. Otherwise, it outputs **Wrong Answer**, followed by a message suggesting what might have been done incorrectly. The sample grader also prints the function calls in the standard error stream.

To read from file, you may use: `./experiment < input.txt`

To print the function calls to file, you may use: `./experiment 2> calls.txt`

To read from file and print the function calls to file, you may use: `./experiment < input.txt 2> calls.txt`

## SAMPLE TESTS

	Input	Output
1	<pre>3 2 1 3</pre>	<pre>Correct. C = 8, score = 100.000</pre>

## T194 - ROULETTE

Time Limit: 1.000 s / Memory Limit: 256 MB

You are the manager of a casino, running a training course for the dealers. The topic today is roulette manipulation.

The roulette consists of a full circle divided into  $C$  equal parts (cells), conveniently numbered from 1 to  $C$ , in clockwise order. There are  $N$  "cell groups" that one can bet on. Group  $i$  occupies  $A_i$  contiguous cells. Group 1 occupies cells  $[1, A_1]$ , group 2 occupies cells  $[A_1 + 1, A_1 + A_2]$ , and so on.

When all bets are made, the dealer will roll the ball around the roulette. After a while, the ball will fall into exactly one cell, and everyone who bets on the group that contains that cell will receive a payoff equal to the amount they have bet.

There are  $M$  dealers and they have not mastered the craft of controlling ball movement yet. The dealers can choose a starting cell for the ball, then let it roll clockwise through some number of cells, until the ball stops. When the  $i$ -th dealer rolls the ball, the ball will roll through  $LOW_i$  to  $UP_i$  cells (inclusive). Each possibility has a nonzero probability of occurrence.

Given the amount bet on each group ( $B_i$  for group  $i$ ), determine, for each dealer, an optimal starting position so that the maximum payoff is minimized.

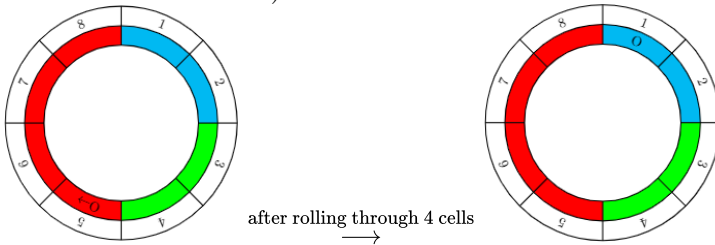
Here is an example with  $N = 3$  cell groups and  $M = 3$  dealers. We use cyan, green and red to represent groups 1, 2 and 3 respectively. The cyan region consists of the first  $A_1 = 2$  cells, the green region consists of the next  $A_2 = 2$  cells, and the red region consists of the last  $A_3 = 4$  cells.

If the ball is inside the cyan region, the payoff will be  $B_1 = 2$ .

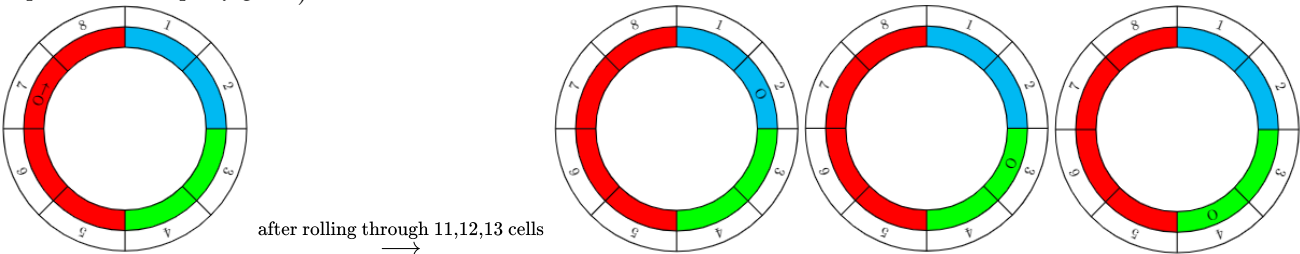
If the ball is inside the green region, the payoff will be  $B_2 = 3$ .

If the ball is inside the red region, the payoff will be  $B_3 = 5$ .

For dealer 1,  $LOW_1 = 4$  and  $UP_1 = 4$ . Starting at position 5 is optimal; the maximum payoff will be 2. (Note that starting at position 6 is equally good.)



For dealer 2,  $LOW_2 = 11$  and  $UP_2 = 13$ . Starting at position 7 is optimal; the maximum payoff will be 3. (Note that starting at position 6 is equally good.)



For dealer 3,  $LOW_3 = 1$  and  $UP_3 = 1000000000$ . Dealer 3 has such dreadful skills, it does not matter where to start rolling. (It is perhaps best not to let him handle the roulette.)

## INPUT

The first line contains an integer  $N$ , the number of groups.

The second line contains  $N$  integers  $A_i$ , denoting the size of the  $i$ -th group.

The third line contains  $N$  integers  $B_i$ , denoting the bet on the  $i$ -th group.

The forth line contains an integer  $M$ , the number of dealers.

$M$  lines follow, each line contains two integers  $LOW_i$  and  $UP_i$ , describing the  $i$ -th dealer.

## OUTPUT

Output  $M$  lines. On the  $i$ -th line, output an optimal starting position for the  $i$ -th dealer so that the maximum payoff is minimized.

If there is more than one optimal solution, you can output any of them.

## SAMPLE TESTS

	Input	Output
<b>1</b>	3 2 2 4 2 3 5 3 4 4 11 13 1 1000000000	5 7 1

Sample 1 is exactly the example in the problem statement.

<b>2</b>	2 1000000000 1000000000 1 1000000000 2 1 1 1000000000 1000000000	1 2000000000
----------	---	-----------------

This case satisfies the constraints of subtask 1.

## SUBTASKS

For all cases:

$$1 \leq N, M \leq 10^5,$$

$$1 \leq A_i \leq 10^9,$$

$$0 \leq B_i \leq 10^9,$$

$$1 \leq LOW_i \leq UP_i \leq 10^9$$

	Points	Constraints
<b>1</b>	8	$LOW_i = UP_i$ for all $1 \leq i \leq M$
<b>2</b>	9	$1 \leq N, M \leq 500$ Sum of $A_i \leq 500$
<b>3</b>	12	$1 \leq N, M \leq 5000$ Sum of $A_i \leq 5000$
<b>4</b>	34	$A_i = 1$ for all $1 \leq i \leq N$
<b>5</b>	37	No additional constraints