Amazon Fresh is running a promotion in which customers receive prizes for purchasing a secret combination of fruits. The combination will change each day, and the team running the promotion wants to use a code list to make it easy to change the combination. The code list contains groups of fruits. Both the order of the groups within the code list and the order of the fruits within the groups matter. However, between the groups of fruits, any number, and type of fruit is allowable. The term "anything" is used to allow for any type of fruit to appear in that location within the group.  
Consider the following secret code list: [[apple, apple], [banana, anything, banana]]  
Based on the above secret code list, a customer who made either of the following purchases would win the prize:  
orange, apple, apple, banana, orange, banana  
apple, apple, orange, orange, banana, apple, banana, banana  
Write an algorithm to output 1 if the customer is a winner else output 0.

**Input**  
The input to the function/method consists of two arguments:  
codeList, a list of lists of strings representing the order and grouping of specific fruits that must be purchased in order to win the prize for the day.  
shoppingCart, a list of strings representing the order in which a customer purchases fruit.  
**Output**  
Return an integer 1 if the customer is a winner else return 0.  
**Note**  
'anything' in the codeList represents that any fruit can be ordered in place of 'anything' in the group. 'anything' has to be something, it cannot be "nothing."  
'anything' must represent one and only one fruit.  
If secret code list is empty then it is assumed that the customer is a winner.

**Example 1:**

Input: codeList = [[apple, apple], [banana, anything, banana]] shoppingCart = [orange, apple, apple, banana, orange, banana]

Output: 1

Explanation:

codeList contains two groups - [apple, apple] and [banana, anything, banana].

The second group contains 'anything' so any fruit can be ordered in place of 'anything' in the shoppingCart. The customer is a winner as the customer has added fruits in the order of fruits in the groups and the order of groups in the codeList is also maintained in the shoppingCart.

**Example 2:**

Input: codeList = [[apple, apple], [banana, anything, banana]]

shoppingCart = [banana, orange, banana, apple, apple]

Output: 0

Explanation:

The customer is not a winner as the customer has added the fruits in order of groups but group [banana, orange, banana] is not following the group [apple, apple] in the codeList.

**Example 3:**

Input: codeList = [[apple, apple], [banana, anything, banana]] shoppingCart = [apple, banana, apple, banana, orange, banana]

Output: 0

Explanation:

The customer is not a winner as the customer has added the fruits in an order which is not following the order of fruit names in the first group.

**Example 4:**

Input: codeList = [[apple, apple], [apple, apple, banana]] shoppingCart = [apple, apple, apple, banana]

Output: 0

Explanation:

The customer is not a winner as the first 2 fruits form group 1, all three fruits would form group 2, but can't because it would contain all fruits of group 1.

**Amazon Music Pairs**

Amazon Music is working on a new "community radio station" feature which will allow users to iteratively populate  
the playlist with their desired songs. Considering this radio station will also have other scheduled shows to be  
aired, and to make sure the community soundtrack will not run over other scheduled shows, the user-submitted songs  
will be organized in full-minute blocks. Users can choose any songs they want to add to the community list, but  
only in pairs of songs with durations that add up to a multiple of 60 seconds (e.g. 60, 120, 180).

As an attempt to insist on the highest standards and avoid this additional burden on users, Amazon will let them  
submit any number of songs they want, with any duration, and will handle this 60-second matching internally. Once  
the user submits their list, given a list of song durations, calculate the total number of distinct song pairs that  
can be chosen by Amazon Music.

**Example**

n = 3

songs = [37, 23, 60]

One pair of songs can be chosen whose combined duration is a multiple of a whole minute (37 + 23 = 60) and the

return value would be 1. While the third song is a single minute long, songs must be chosen in pairs.

**Function Description**

Complete the function getSongPairCount in the editor below.

getSongPairCount has the following parameter:

int songs[n]: array of integers representing song durations in seconds

Returns:

long: the total number of songs pairs that add up to a multiple of 60 seconds. If there are no suitable pairs,  
return 0.

**Constraints**

1 ≤ n ≤ 105  
1 ≤ songs[i] ≤ 1000, where 0 ≤ i < n

**Input Format For Custom Testing**  
Input from stdin will be processed as follows and passed to the function.

The first line contains an integer, n, that denotes the number of elements in songs.

The next n lines each contain an integer that describes songs[i] and denotes the duration of the ith song  
(in seconds).

**Sample Case 0  
Sample Input For Custom Testing**

STDIN Function

4 -> songs[] size n = 4  
10 -> songs = [10, 50, 90, 30]  
50  
90  
30  
**Sample Output**

2  
**Explanation**

The first and second songs pair to 60 seconds. The third and fourth songs pair to 120 seconds. No other pairs  
will satisfy the requirement.

**Sample Case 1  
Sample Input For Custom Testing**

STDIN Function

5 -> songs[] size n = 5  
30 -> songs = [30, 20, 150, 100, 40]  
20  
150  
100  
40  
**Sample Output**

3  
**Explanation**

There are three pairs of songs whose whole duration is a multiple of a whole minute. They are 20 + 100, 30 + 150,  
and 20 + 40 corresponding to (1, 3), (0, 2) and (1, 4).

**Sample Case 2  
Sample Input For Custom Testing**

STDIN Function

3 -> songs[] size n = 3  
60 -> songs = [60, 60, 60]  
60  
60  
**Sample Output**

3  
**Explanation**

There are three pairs of songs that end in whole minutes. They are (0, 1), (1, 2) and (0, 2).

Imagine a small Amazon Go store that has exactly one turnstile. It can be used by customers either as an entrance or an exit. Sometimes multiple customers want to pass through the turnstile and their directions can be different. The ith customer comes to the turnstile at time[i] and wants to either exit the store if direction [i] = 1 or enter the store if direction[i] = 0. Customers form 2 queues, one to exit and one to enter. They are ordered by the time when they came to the turnstile and, if the times are equal, by their indices.

If one customer wants to enter the store and another customer wants to exit at the same moment, there are three cases:

If in the previous second the turnstile was not used (maybe it was used before, but not at the previous second), then the customer who wants to exit goes first.  
If in the previous second the turnstile was used as an exit, then the customer who wants to leave goes first.  
If in the previous second the turnstile was used as an entrance, then the customer who wants to enter goes first.  
Passing through the turnstile takes 1 second.

Write an algorithm to find the time for each customer when they will pass through the turnstile.

Input

The function/method consists of three arguments:

numcustomers, an integer representing the number of customers (n);  
arrTime, a list of integers where the value at indexi is the time in seconds when the ith customer will come to the turnstile;  
direction, a list of integers where the value at indexi is the direction of the ith customer.

Output

Return a list of integers where the value at index i is the time when the ith customer will pass the turnstile.

Constraints

1 <= numCustomers <= 10^5  
0 <= arrTime[i] <= arrTime[i + 1] <= 10^9  
0 <= i <= numCustomers - 2  
0 <= direction[i] <= 1  
0 <= j <= numCustomers - 1

Examples

Example 1:

Input:  
numCustomers = 4  
arrTime = [0, 0, 1,5]  
direction = [0, 1, 1, 0]  
Output:  
[2,0,1,5]

Explanation:  
At time 0, customer 0 and 1 want to pass through the turnstile. Customer 0 wants to enter the store and customer 1 wants to leave the store. The turnstile was not used in the previous second, so the priority is on the side of the customer 1  
At time 1, customers 0 and 2 want to pass through the turnstile. Customer 2 wants to leave the store and at the previous second the turnstile was used as an exit, so the customer 2 passes through the turnstile.  
At time 2, customer 0 passes through the turnstile.  
At time 5, customer 3 passes through the turnstile.

Example 2

Input:  
numCustomers = 5  
arrTime = [0,1,1,3,3]  
direction = [0, 1, 0, 0, 1]  
Output: [0, 2, 1, 4, 3]

Explanation:

At time 0, customer 0 passes through the turnstile (enters).  
At time 1, customers 1 (exit) and 2 (enter) want to pass through the turnstile, and customer 2 passes through the turnstile because their direction is equal to the direction at the previous second.  
At time 2. customer 1 passes through the turnstile (exit).  
At time 3, customers 3 (enter) and 4 (exit) want to pass through the turnstile. Customer 4 passes through the turnstile because at the previous second the turnstile was used to exit.  
At time 4, customer 3 passes through the turnstile.