Question - 1
Freelancing Platform

A customer has posted several web development projects on a freelancing platform, and various web developers have put in bids for these projects. Given the bid amounts and their corresponding projects, what is the minimum amount the customer can pay to have all the projects completed?

Note: If any project has no bids, return -1.

Example
numProjects = 3 projects.
projectId = [2, 0, 1, 2]
bid = [8, 7, 6, 9].

- *projectId[i]* is aligned with *bid[i]*
- The first web developer bid 8 currency units for project 2.
- The second web developer bid 7 currency units for project 0.
- The third web developer bid 6 currency units for project 1.
- The fourth web developer bid 9 currency units for project 2.

There is only one choice of who to hire for project *0*, and it will cost *7*. Likewise, there is only one choice for project *1*, which will cost *6*. For project *2*, it is optimal to hire the first web developer, instead of the fourth, and doing so will cost *8*. So the final answer is *7 + 6 + 8 = 21*.

If instead there were *n = 4* projects, the answer would be *-1* since there were no bids received on the fourth project.

Function Description
Complete the function *minCost* in the editor below.

minCost has the following parameters:
int numProjects: the total number of projects posted by the client (labeled from *0* to *n*)
int projectId[n]: an array of integers denoting the projects that the freelancers bid on
int bid[n]: an array of integers denoting the bid amounts posted by the freelancers

Returns:
long: the minimum cost the client can spend to complete all projects, or *-1* if any project has no bids.

Constraints
- $1 \le numProjects, n \le 5 \times 10^5$
- $0 \le projectId[i] < n$
- $1 \le bid[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *numProjects*.
The next line contains an integer, *n*, which denotes the number of elements in array *projectId[]*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *projectId[i]*.
The next line contains an integer, *n*, which denotes the number of elements in array *bid*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer, *bid[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       -----
2       →   numProjects = 2
5       →   projectId[] size n = 5
0       →   projectId = [0, 1, 0, 1, 1]
1
0
1
1
5       →   bid[] size n = 5
4       →   bid = [4, 74, 47, 744, 7]
74
47
744
7
```

Sample Output

```
11
```

Explanation

The bids are as follows:

- The first web developer bid *4* currency units for project *0*.
- The second web developer bid *74* currency units for project *1*.
- The third web developer bid *47* currency units for project *0*.
- The fourth web developer bid *744* currency units for project *1*.
- The fifth web developer bid *7* currency units for project *1*.

The optimal solution is to hire the first web developer to complete project *0* (which costs *4*) and to hire the fifth web developer to complete project *1* (which costs *7*). This brings the total cost to *11*.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       -----
2       →   numProjects = 2
2       →   projectId[] size n = 2
1       →   projectId = [1, 1]
1
2       →   bid[] size n = 2
4       →   bid = [4, 7]
7
```

Sample Output

```
-1
```

Explanation

Because there are no bids for project *0*, the function should return *-1*.

Question - 2

Listening Fun

There are two integer arrays, *singer[n]* and *length[n]*. Each song is described by *singer[i]* and *length[i]*. The *fun* amount for a song is the product of its length and the total number of different singers heard in all prior songs plus the singer of the current song. The songs can be played in any order.

Arrange the songs to produce the maximum possible fun listening to the entire playlist. Return that maximum fun value.

Example

Given *n* = 3, *singer* = [1,2,2] and *length* = [2,3,2], the optimal order is

- the first song, 1 * 2 = 2.
- the third song, 2 * 2 = 4.
- the second song, 2 * 3 = 6.

The maximum fun possible is 2 + 4 + 6 = 12.

Function Description

Complete the function *getMaxFun* in the editor below. The function *getMaxFun* has the following parameters:

int singer[n]: each singer's unique id

int length[n]: the lengths of each song

Return

long int: the maximum possible fun

Constraints

- $1 \le n \le 10^5$
- $1 \le singer[i] \le 10^9$
- $1 \le length[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in the *singer*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *singer[i]*.

The next line contains an integer, *n,* the number of elements in *length*.

Further, each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *length[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       ----------
3       →   singer[] size n = 3
1       →   singer = [1,1,2]
1
1
2
3       →   length[] size n = 3
5       →   length = [5,4,3]
4
3
```

Sample Output

```
21
```

Explanation

It is optimal to listen to the second song, the third song, and finally, the first song. 1*3 + 2*4 + 2*5 = 21

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       ----------
3       →   singer[] size n = 3
```

```
1       →   singer = [1,2,3]
2
3
3       →   length size n = 3
1       →   length = [1,2,3]
2
3
```

Sample Output

```
14
```

Explanation

The optimal order is third, second then the first song. 1*1 + 2*2 + 3*3 = 14

Question - 3
Purchase Optimization

Alex is shopping at Ozone Galleria Mall. There is a dedicated cubicle for a type of product at the shopping center. All the products sold at the i^{th} cubicle are priced the same, denoted by *prices[i]*. The cubicles are arranged such that the price of the products sold at each cubicle are in non-decreasing order.

Several queries would be asked in the problem. In each query, the cubicle number Alex is initially standing at, and the amount of money Alex has is given, and Alex can travel in the right direction visiting from the current cubicle to the last cubicle. Alex may buy at most one item from any cubicle visited, but the total cost of the purchase must not exceed the amount Alex has. Report the maximum number of products that can be purchased for each query.

More formally, given an array of *n* integers, *prices*, where *prices[i]* denotes the price of the product sold in the i^{th} cubicle. The array *prices* are in non-decreasing order (i.e., *price[i] ≤ price[i+1]), and q* queries need to be processed. For each query, two integers are given:

- *pos:* Alex's initial position
- *amount:* the amount of money Alex has

Alex needs to visit each cubicle from number *pos* to *n*, purchasing at most one product from each cubicle. For each query, the goal is to find the maximum number of products that Alex can buy.

Example

For example, let:

n = 5

prices = [3, 4, 5, 5, 7]

q = 3

The cubicle at the convention would be ordered like this:

The arrows between the cubicles denote the next cubicle that Alex can visit, the cubicle to the right.

Let the following queries be:

Query 1: *pos* = 2, *amount* = 10

Alex needs to visit the cubicles from cubicle no. 2 to cubicle no. 5. Alex can purchase at most two products from these cubicles either by making one of these possible purchases:

- Cubicle No. 2 and cubicle No. 3. The total cost is 4 + 5 = 9 (≤ 10).
- Cubicle No. 2 and cubicle No. 4. The total cost is 4 + 5 = 9 (≤ 10).
- Cubicle No. 3 and cubicle No. 4. The total cost is 5 + 5 = 10 (≤ 10).

Hence, the answer is 2.

Query 2: *pos* = 1, *amount* = 24

Alex needs to visit the cubicles from cubicle no. 1 to cubicle no. 5. With the given amount of money, Alex can make purchases at every cubicle visited since the total cost is 3 + 4 + 5 + 5 + 7 = 24 (≤ 24).

Hence, the answer is 5.

Query 3: *pos = 5, amount = 5*

Alex can only visit cubicle no. 5, but cannot make any purchase there since the total amount Alex has is less than the product price.

Hence, the answer is 0.

Function Description

Complete the function *findMaximumValue* in the editor below.

findMaximumValue has the following parameters:

prices[prices[0],...prices[n-1]]: an array of integers

pos[pos[0],...pos[q-1]]: an array of integers

amount[amount[0],...amount[q-1]]: an array of integers

Returns

int[n]: an integer array denoting the answer for each query.

Constraints

- $1 \le n \le 5 \cdot 10^5$
- $1 \le q \le 2 \cdot 10^5$
- $1 \le prices[i] \le 10^6$
- $prices[i] \le prices[i+1]$
- $1 \le pos[i] \le n$
- $0 \le amount[i] \le 10^{12}$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *prices*.

The next *n* lines contain one integer each, denoting *prices[i]*.

The next line contains an integer, *q*, denoting the number of queries.

The next *n* lines contain one integer each, denoting *pos[i]*.

The next line contains an integer, *q*, denoting the number of queries.

The next *n* lines contain one integer each, denoting *amount[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN    FUNCTION
-----    ---------
5    →   prices size is 5
1    →   prices = [1, 2, 3, 4, 5]
2
3
4
5
2    →   there are 2 queries, which is the size of pos
1    →   pos = [1, 3]
3
2    →   amount size is equal to pos size
4    →   amount = [4, 14]
14
```

Sample Output

```
2
3
```

Explanation

Given that *prices* = [1, 2, 3, 4, 5]:

Query 1:

Alex needs to visit cubicles no. 1 to no. 5. Alex can purchase at most 2 products from these cubicles making one of these possible purchases:

1. Cubicle No. 1 and cubicle No. 2. The total cost is 1 + 2 = 3 (≤ 4).
2. Cubicle No. 1 and cubicle No. 3. The total cost is 1 + 3 = 4 (≤ 4).

Query 2:

Alex needs to visit cubicle no. 3 to cubicle no. 5. With the given amount of money, Alex can make a purchase at every cubicle visited since the total cost is 3 + 4 + 5 = 12 (≤ 14).

Hence, after running all the queries, return [2, 3].

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       ---------
5     →     prices size is 5
1     →     prices = [1, 2, 2, 2, 3]
2
2
2
3
2     →     there are 2 queries, which is the size of pos
2     →     pos = [2, 5]
5
2     →     amount size is equal to pos size
5     →     amount = [5, 2]
2
```

Sample Output

```
2
0
```

Explanation

Given *prices* = [1, 2, 2, 2, 3]:

Query 1:

Alex needs to visit cubicle no. 2 to cubicle no. 5. Alex can purchase at most 2 products from these cubicles, such as cubicle no. 2 and cubicle no. 5. The total cost would then be 2 + 3 = 5 (≤ 5).

Query 2:

Alex can only visit cubicle no. 5, but cannot make any purchase there since the total amount Alex has is less than the product price.

Hence, after running all the queries, return [2, 0].