Question - 1
Shopper's Delight

A shopaholic has to buy a pair of jeans, a pair of shoes, a skirt, and a top with *budgeted* dollars. Given the quantity of each product and the price per unit, determine how many options of each item are present. If required, all *budgeted* dollars can be spent.

Example

priceOfJeans = [2, 3]

priceOfShoes = [4]

priceOfSkirts = [2, 3]

priceOfTops = [1, 2]

budgeted = 10

The shopper must buy shoes for *4* dollars since there is only one option. This leaves *6* dollars to spend on the other *3* items. Combinations of prices paid for jeans, skirts, and tops respectively that add up to *6* dollars or less are *[2, 2, 2], [2, 2, 1], [3, 2, 1], [2, 3, 1]*. There are *4* ways the shopper can purchase all *4* items.

Function Description

Complete the *getNumberOfOptions* function in the editor below. The function must return a long integer which represents the number of options present to buy the four items.

getNumberOfOptions has *5* parameters:

int priceOfJeans[a]: An integer array, which contains the prices of the pairs of jeans available.

int priceOfShoes[b]: An integer array, which contains the prices of the pairs of shoes available.

int priceOfSkirts[c]: An integer array, which contains the prices of the skirts available.

int priceOfTops[d]: An integer array, which contains the prices of the tops available.

int budgeted: the total number of dollars available to shop with.

Constraints

- $1 \le a, b, c, d \le 10^3$
- $1 \le budgeted \le 10^9$
- $1 \le$ price of each item $\le 10^9$

Note: *a, b, c* and *d* are the sizes of the four price arrays

▼ Input Format For Custom Testing

Locked stub code in the editor reads the following input from stdin and passes it to the function:

The first line contains *a*, the size of *priceOfJeans*.

Each of the subsequent *a* lines contains *priceOfJeans[i]*.

The next line contains *b*, which is the size of *priceOfShoes*.

Each of the subsequent *b* lines contains *priceOfShoes[j]*.

The next line contains *c*, which is the size of *priceOfSkirt*.

Each of the subsequent *c* lines contains *priceOfSkirt[k]*.

The next line contains *d*, which is the size of *priceOfShoes*.

Each of the subsequent *d* lines contains *priceOfJeans[l]*.

The last line contains *budgeted*, the number of dollars to shop with.

▼ Sample Case 0

Sample Input

```
STDIN    Function
-----    --------
2    →   priceOfJeans[] size a = 2
2    →   priceOfJeans = [2, 3]
3
1    →   priceOfShoes[] size b = 1
4    →   priceOfShoes = [4]
1    →   priceOfSkirts[] size c = 1
2    →   priceOfSkirts = [2]
3    →   priceOfTops[] size d = 3
1    →   priceOfTops = [1, 2, 3]
2
3
10   →   budgeted = 10
```

Sample Output

```
3
```

Explanation

With *budgeted = 10* dollars, the shopper can buy:

1. A pair of jeans with price *2*, a pair of shoes with price *4*, a skirt with price *2* and a top with price *1*, OR
2. A pair of jeans with price *2*, a pair of shoes with price *4*, a skirt with price *2* and a top with price *2*, OR
3. A pair of jeans with price *3*, a pair of shoes with price *4*, a skirt with price *2* and a top with price *1*.

Thus, there are *3* options.

▼ Sample Case 1

Sample Input

```
STDIN    Function
-----    --------
1    →   priceOfJeans[] size a = 1
4    →   priceOfJeans = [4]
3    →   priceOfShoes[] size b = 3
3    →   priceOfShoes = [3, 4, 1]
4
1
2    →   priceOfSkirts[] size c = 2
3    →   priceOfSkirts = [3, 2]
2
1    →   priceOfTops[] size d = 1
4    →   priceOfTops = [4]
12   →   budgeted = 12
```

Sample Output

```
2
```

Explanation

With *budgeted = 12* dollars, the shopper can buy:

1. A pair of jeans with price *4*, a pair of shoes with price *1*, a skirt with price *2*, and a top with price *4*, OR
2. A pair of jeans with price *4*, a pair of shoes with price *1*, a skirt with price *3*, and a top with price *4*.

Thus, there are *2* options.

▼ Sample Case 2

Sample Input

```
STDIN    Function
-----    --------
1    →   priceOfJeans[] size a = 1
1    →   priceOfJeans = [1]
1    →   priceOfShoes[] size b = 1
4    →   priceOfShoes = [4]
1    →   priceOfSkirts[] size c = 1
3    →   priceOfSkirts = [3]
1    →   priceOfTops[] size d = 1
1    →   priceOfTops = [1]
3    →   budgeted = 3
```

Sample Output

```
0
```

Explanation

With *budgeted = 3* dollars, the shopper cannot buy all *4* items in any combination.

Question - 2
Employee Profile

Implement the following classes:

1. abstract class Employee with the following methods:
 * abstract void setSalary(int salary) method
 * abstract int getSalary() method
 * abstract void setGrade(String grade) method (grade of the employee in the organization)
 * abstract String getGrade() method
 * void label() method which prints "Employee's data:\n" (Concrete method, implementation is hidden from end user)

2. class Engineer extending class Employee:
 * private attribute int salary
 * private attribute String grade
 * implement the setter and getter methods from the parent class to set and get attributes (salary and grade)

3. class Manager extending class Employee:
 * private attribute int salary
 * private attribute String grade
 * implement the setter and getter methods from the parent class to set and get attributes (salary and grade)

Note: The code stub handles input and calls the methods.

▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of employees to be instantiated.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains 3 variables.
TYPE_OF_EMPLOYEE GRADE SALARY

▼ Sample Case 0

Sample Input For Custom Testing

```
2
ENGINEER B 50000
MANAGER A 70000
```

Sample Output

```
Employee's data:
GRADE : B
SALARY : 50000
Employee's data:
GRADE : A
SALARY : 70000
```

▼ Sample Case 1

Sample Input For Custom Testing

```
3
ENGINEER B 50000
MANAGER A 70000
MANAGER A 90000
```

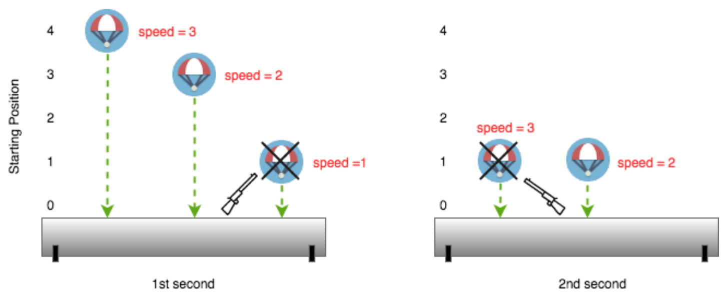
Sample Output

```
Employee's data:
GRADE : B
SALARY : 50000
Employee's data:
GRADE : A
SALARY : 70000
Employee's data:
GRADE : A
SALARY : 90000
```

Question - 3
Shooting Bogeys

Multiple bogeys, or enemy aircraft, are coming down to destroy a military base. Each bogey has a certain starting position above the ground level and speed at which it is descending. Alex is in charge of the protection of military base and has a gun which can shoot one bogey each second. How long can Alex protect the base?

For example, *n = 3* bogeys are coming down, with starting positions given by *startingPos = [4, 3, 1]* and *speed = [3, 2, 1]*. Alex can shoot the bogey with starting position *1* in the 1^{st} second and the bogey with starting position *4* in the *2nd* second, but cannot shoot down the third bogey. Alex can protect the base for *2* seconds.



Function Description

Complete the function *protectionTime* in the editor below. The function must return an integer, the maximum time up to which Alex can protect the base.

protectionTime has the following parameter(s):
    startingPos[startingPos[0],...startingPos[n-1]]: an array of integers, denoting the starting positions of all the bogeys.
    speed[speed[0],...speed[n-1]]: an array of integers, denoting the speed at which the bogeys are coming down.

Constraints

- $1 \le n \le 10^5$
- $1 \le startingPos[i] \le 10^9$ $(0 \le i < n)$
- $1 \le speed[i] \le 10^5$ $(0 \le i < n)$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of bogeys

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer describing *startingPos[i]*

The next line contains the integer *n*

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer describing *speed[i]*

▼ Sample Case 0

Sample Input For Custom Testing

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

Sample Output

```
2
```

Explanation

Alex can shoot the last bogey in the 1st second, then shoot the first bogey in the 2nd second and then the military base is destroyed by the second bogey. Refer to the image in the problem statement.

▼ Sample Case 1

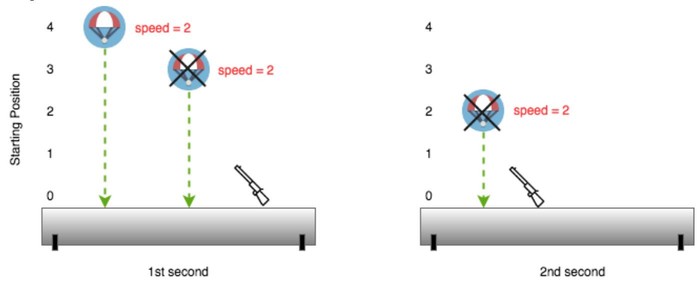
Sample Input For Custom Testing

```
2
4
3
2
2
2
```

Sample Output

```
2
```

Explanation



Alex can shoot both bogeys down, choosing any bogey to be shot first.