Question - 1

SCORE: **75 points**

Passing Your Exam

Data Structures | Medium | Algorithms

In one of your programming courses, you received a low score on your final exam, which is too low for you to pass the course. Fortunately, your professor is very generous, and they have agreed to give you a better score on the exam if you spend some more time studying the materials. You want to spend the least number of hours needed to raise your score enough to pass the course.

On the exam, there are a total of n tasks, each having equal weight. For each task, you're provided with a_i (the score you received on the i^{th} task) and b_i (the maximum score of the i^{th} task). You are also given the minimum percentage r that you must score on the exam in order to pass the course.

According to your professor, you can alter your score by choosing any task and studying its topic for an hour. Doing this will increase your score on that task by 1, but it will also increase the maximum score for the task by 1. So, if you earned a score of p out of the maximum score q in a task, you would change your score to $p + 1$ and the maximum score of the task to $q + 1$. You must determine the minimum number of hours you must study in order to raise your score to the percentage needed to pass the course.

For example, let's say $n = 1$ and $r = 60$. This means there's one task and the passing level is 60%. On the task, you scored 2 points out of a total of 5. You could do the following to raise your score:

- Study for one hour, thus changing your score from 2 to 3 (2+1). This would increase the maximum score from 5 to 6. Therefore, the total percentage would be 50% (3 out of 6).
- Study for two hours, thus changing your score from 2 to 4 (2+1+1). This would increase the maximum score from 5 to 7. Therefore, the total percentage would be 57.14% (4 out of 7).
- Study for three hours, thus changing your score from 2 to 5 (2+1+1+1). This would increase the maximum score from 5 to 8. Therefore, the total percentage would be 62.5% (5 out of 8).

Therefore, you would need to study for 3 hours to raise your score to passing level, so the answer is 3. (It is guaranteed that the minimum number of hours you need to study will be less than 10^6.)

Note**:** Each task has equal weight, so they must first be normalized to a common total score before calculating the percentage.

Function Description

Help

Complete the function *passExam* in the editor below. The function must return an integer denoting the minimum number of hours you must study in order to pass the course.

passExam has the following parameters:

scores$_{ij}$, a 2d array of non-negative integers where the i^{th} element contains two values, the first one denoting a_i and the second one denoting b_i

r, an integer

Constraints

- $1 \le r < 100$
- $1 \le n \le 200$
- $0 \le a < b \le 100$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of tasks you performed on the exam.

The second line contains the fixed integer 2, denoting the number of columns in the 2d array *scores*.

The next *p* lines consist of two space-separated integers, a_i and b_i, denoting the scores you earned and the total possible scores of the i^{th} task.

The last line contains an integer, *r*, denoting the minimum percentage you must have on the exam in order to pass the course.

▼ Sample Case 0

Sample Input For Custom Testing

```
2
2
1 2
1 3
50
```

Sample Output

```
1
```

Explanation

Studying the second task would raise your score to be exactly 50%—the first task would remain 1/2, and the second task would become 2/4. Therefore, you can pass the course by studying for just 1 hour.

▼ Sample Case 1

Sample Input For Custom Testing

```
2
2
1 2
2 3
75
```

Sample Output

```
3
```

Explanation

For this, you could study the first task for 2 hours and the second task for 1 hours.

- 1^{st} hour spent on the first task: 1/2 -> 2/3

- 2nd hour spent on the first task: 2/3 -> 3/4
- 3rd hour spend on the second task: 2/3 -> 3/4

Therefore, you need to study a total of 3 hours to raise your score to 75%, which is high enough to pass the course.

Question - 2

SCORE: **75 points**

Computing Cluster Quality

Implementation | Algorithms | Medium

When building a computing cluster consisting of several machines, two parameters are most important: *speed* and *reliability*. The *quality* of a computing cluster is the sum of its machines' speeds multiplied by the minimum reliability of its machines.

Given information about several available machines, select machines to create a cluster of less than or equal to a particular size. Determine the maximum quality of cluster that can be created.

Example

n = 5
speed = [4, 3, 15, 5, 6]
reliability = [7, 6, 1, 2, 8]
maxMachines = 3

The maximum number of machines to use is *maxMachines* = 3 chosen from n=5 available machines. A *machine[i]*'s speed and reliability are *speed[i]* and *reliability[i]*.
Select the first, second, and fifth machines. The quality of the cluster is:
*(speed[0] + speed[1] + speed[4]) * min(reliability[0], reliability[1], reliability[4]) = (4 + 3 + 6) * min(7, 6, 8) = 13 * 6 = 78*.

This is the highest quality that can be achieved, so the answer is 78.

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

maxClusterQuality has the following parameter(s):
int speed[n]: an integer array of size *n*, such that *speed[i]* is the speed of the i^{th} machine
int reliability[n]: an integer array of size *n*, such that *reliability[i]* is the reliability of the i^{th} machine
int maxMachines: an integer denoting the maximum number of machines you want in a cluster
Returns:
int: integer denoting the maximum quality of a computing cluster that can be built

Constraints

- $1 \le n \le 10^5$
- $1 \le speed[i] \le 10^5$
- $1 \le reliability[i] \le 10^5$
- $1 \le maxMachines \le n$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of available machines.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, *speed[i]*, the speed of i^{th} machine.
The next line contains an integer, *n*, the number of available machines.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, *reliability[i]*, the reliability of i^{th} machine.
The last line contains a single integer, *maxMachines*, the maximum number of machines in a cluster.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
5      →  speed[] size n = 5
12     →  speed[] = [12, 112, 100, 13, 55]
112
100
13
55
5      →  reliability[] size n = 5
31     →  reliability[] = [31, 4, 100, 55, 50]
4
100
55
50
3      →  maxMachines = 3
```

Sample Output

```
10000
```

Explanation
There are 5 machines available. Their speeds are 12, 112, 100, 13, and 55 respectively, while their reliabilities are 31, 4, 100, 55, and 50 respectively. The maximum number of machines allowed in a cluster is 3.

The best quality of a cluster can be achieved by selecting only the third machine. The quality of a cluster will be 100 * 100 = 10000.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
3      →   speed[] size n = 3
11     →   speed[] = [11, 10, 7]
10
7
3      →   reliability[] size n = 3
6      →   reliability[] = [6, 4, 8]
4
8
2      →   maxMachines = 2
```

Sample Output

```
108
```

Explanation

There are 3 machines available. Their speeds are 11, 10, and 7 respectively, while their reliabilities are 6, 4, and 8 respectively. The maximum number of machines allowed in a cluster is 2.

The best quality of a cluster can be achieved by selecting the first and third machines. The quality of a cluster will be (11 + 7) * min(6, 8) = 18 * 6 = 108.

Question - 3

SCORE: **75 points**

Minimum Start Value

Search | Binary Search | Medium | Algorithms | Problem Solving | Core CS

Start with a given array of integers and an arbitrary initial value *x*. Calculate the running sum of *x* plus each array element, from left to right. The running sum must never get below *1*. Determine the minimum value of *x*.

Example

arr = [-2, 3, 1, -5].

If *x = 4*, the following results are obtained:

```
Running
sum       arr[i]
-----     -----
4          -2
2           3
5           1
6          -5
1
```

The final value is *1*, and the running sum has never dropped below *1*. The minimum starting value for *x* is 4.

Function Description

Complete the function *minX* in the editor below.

minX has the following parameter(s):

int arr[n]: an array of integers

Returns

int: the minimum integer value for *x*

Constraints

- $1 \le n \le 10^5$
- $-100 \le arr[i] \le 100$

▼ 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*, the size of the array *arr*.
Each of the next *n* lines contains an integer *arr[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       -----
10      →   arr[i] size n = 10
-5      →   arr = [-5, 4, -2, 3, 1, -1, -6, -1, 0,
5]
4
-2
3
1
-1
-6
-1
0
5
```

Sample Output

```
8
```

Explanation

```
Running
sum         arr[i]
-----       -----
8            -5
3             4
7            -2
5             3
8             1
9            -1
8            -6
2            -1
1             0
1             5
6
```

The minimum starting value for *x* is *8*.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       -----
5       →   arr[i] size n = 5
-5      →   arr = [-5, 4, -2, 3, 1]
4
-2
3
1
```

Sample Output

```
6
```

Explanation

```
Running
sum       arr[i]
-----     -----
6          -5
1           4
5          -2
```

```
3           3
6           1
7
```

The minimum starting value for *x* is *6*.

▼ Sample Case 2

Sample Input

```
STDIN       Function
-----       -----
10      →   arr[i] size n = 10
-5      →   arr = [-5, 4, -2, 3, 1, -1, -6, -1, 0,
-5]
4
-2
3
1
-1
-6
-1
0
-5
```

Sample Output

```
13
```

Explanation

```
Running
sum         arr[i]
-----       -----
13           -5
 8            4
12           -2
10            3
13            1
14           -1
13           -6
 7           -1
 6            0
 6           -5
 1
```

The minimum starting value for *x* is *13*.