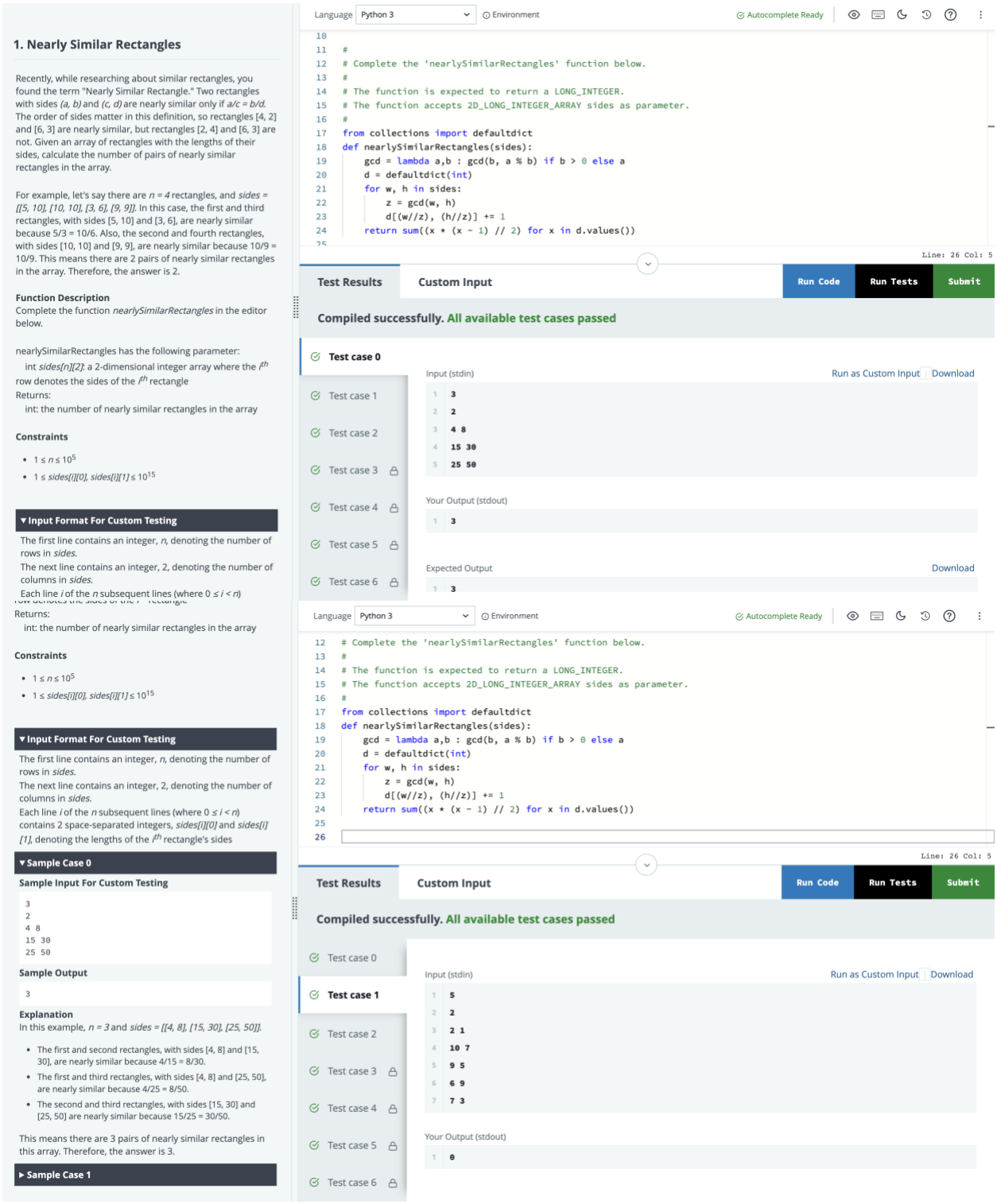
Problem Solving Basic

1. Nearly Similar Rectangles



1. Nearly Similar Rectangles

Recently, while researching about similar rectangles, you found the term "Nearly Similar Rectangle." Two rectangles with sides *(a, b)* and *(c, d)* are nearly similar only if *a/c = b/d*. The order of sides matter in this definition, so rectangles [4, 2] and [6, 3] are nearly similar, but rectangles [2, 4] and [6, 3] are not. Given an array of rectangles with the lengths of their sides, calculate the number of pairs of nearly similar rectangles in the array.

For example, let's say there are *n = 4* rectangles, and *sides = [[5, 10], [10, 10], [3, 6], [9, 9]]*. In this case, the first and third rectangles, with sides [5, 10] and [3, 6], are nearly similar because 5/3 = 10/6. Also, the second and fourth rectangles, with sides [10, 10] and [9, 9], are nearly similar because 10/9 = 10/9. This means there are 2 pairs of nearly similar rectangles in the array. Therefore, the answer is 2.

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

nearlySimilarRectangles has the following parameter:
int *sides[n][2]*: a 2-dimensional integer array where the i^{th} row denotes the sides of the i^{th} rectangle
Returns:
int: the number of nearly similar rectangles in the array

Constraints

- $1 \le n \le 10^5$
- $1 \le sides[i][0], sides[i][1] \le 10^{15}$

▾ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of rows in *sides*.
The next line contains an integer, 2, denoting the number of columns in *sides*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains 2 space-separated integers, *sides[i][0]* and *sides[i][1]*, denoting the lengths of the i^{th} rectangle's sides

▾ Sample Case 0

Sample Input For Custom Testing

```
3
2
4 8
15 30
25 50
```

Sample Output

```
3
```

Explanation
In this example, *n = 3* and *sides = [[4, 8], [15, 30], [25, 50]]*.

- The first and second rectangles, with sides [4, 8] and [15, 30], are nearly similar because 4/15 = 8/30.
- The first and third rectangles, with sides [4, 8] and [25, 50], are nearly similar because 4/25 = 8/50.
- The second and third rectangles, with sides [15, 30] and [25, 50] are nearly similar because 15/25 = 30/50.

This means there are 3 pairs of nearly similar rectangles in this array. Therefore, the answer is 3.

▸ Sample Case 1

Language: Python 3 | Environment | Autocomplete Ready

```
#
# Complete the 'nearlySimilarRectangles' function below.
#
# The function is expected to return a LONG_INTEGER.
# The function accepts 2D_LONG_INTEGER_ARRAY sides as parameter.
#
from collections import defaultdict
def nearlySimilarRectangles(sides):
    gcd = lambda a,b : gcd(b, a % b) if b > 0 else a
    d = defaultdict(int)
    for w, h in sides:
        z = gcd(w, h)
        d[(w//z), (h//z)] += 1
    return sum((x * (x - 1) // 2) for x in d.values())
```

Line: 26 Col: 5

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0 (Input (stdin) | Run as Custom Input | Download)

```
3
2
4 8
15 30
25 50
```

Your Output (stdout)

```
3
```

Expected Output | Download

```
3
```

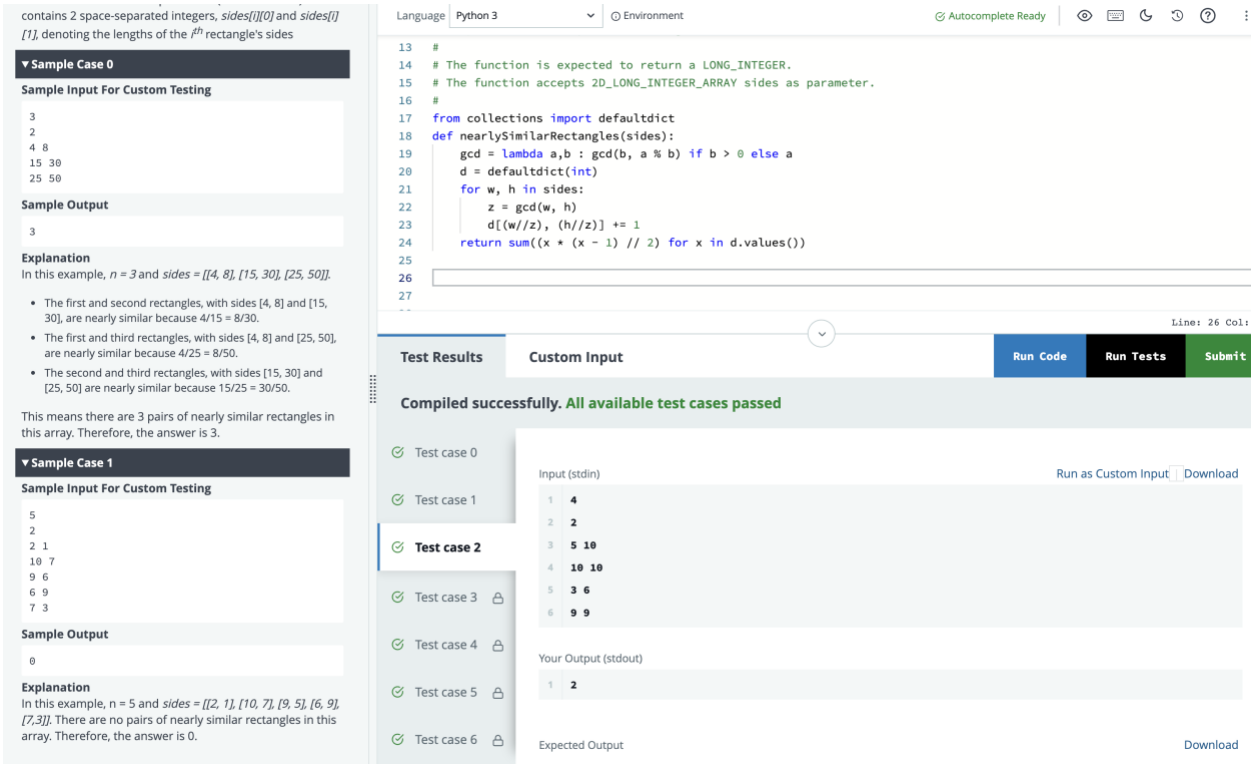
Test case 1 (Input (stdin) | Run as Custom Input | Download)

```
5
2
2 1
10 7
9 5
6 9
7 3
```

Your Output (stdout)

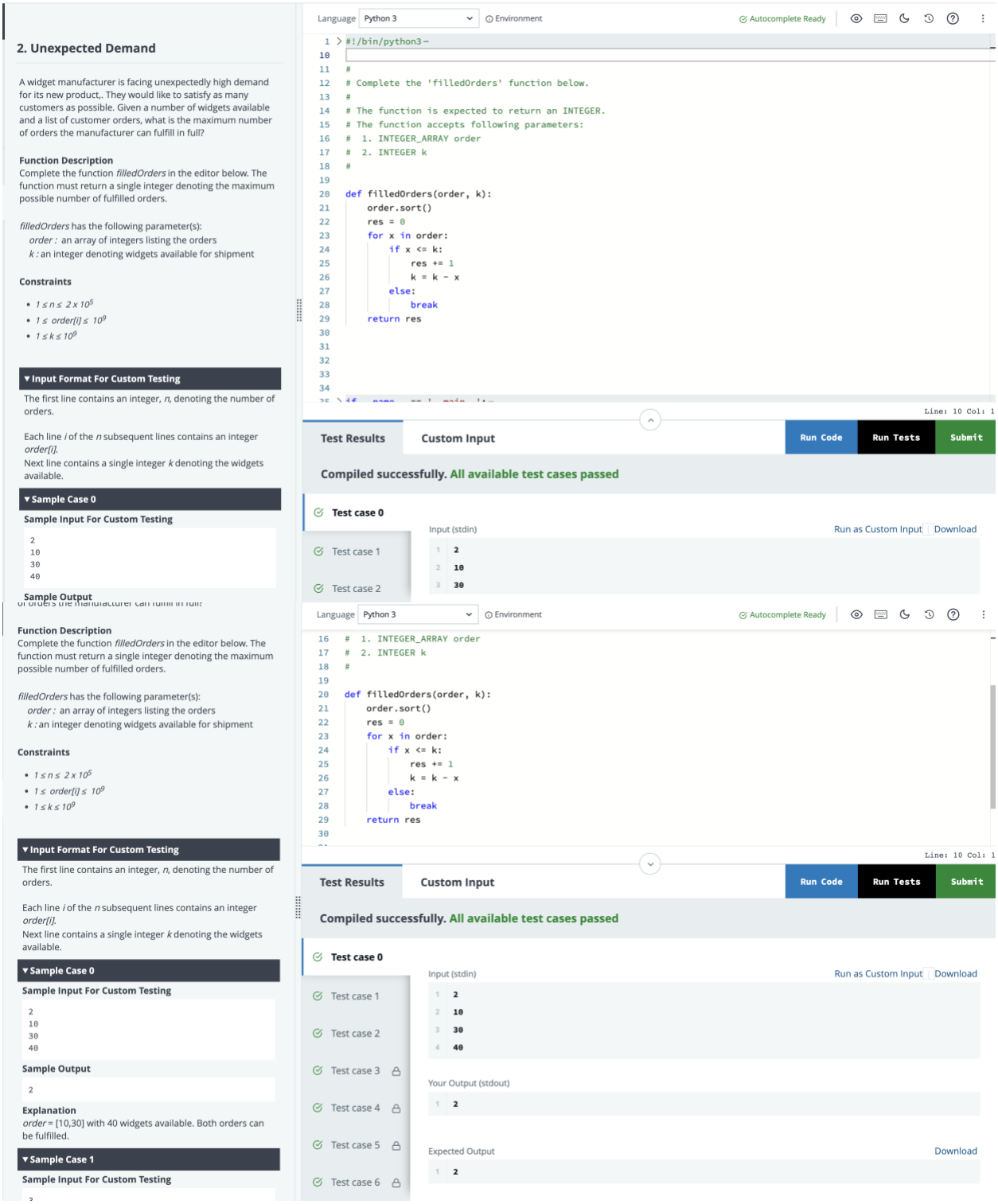
```
0
```

Test case 2 | Test case 3 | Test case 4 | Test case 5 | Test case 6




```
#
# The function is expected to return a LONG_INTEGER.
# The function accepts 2D_LONG_INTEGER_ARRAY sides as parameter.
#
from collections import defaultdict
def nearlySimilarRectangles(sides):
    gcd = lambda a,b : gcd(b, a % b) if b > 0 else a
    d = defaultdict(int)
    for w, h in sides:
        z = gcd(w, h)
        d[(w//z), (h//z)] += 1
    return sum((x * (x - 1) // 2) for x in d.values())
```

2.Unexpected Demand



2. Unexpected Demand

A widget manufacturer is facing unexpectedly high demand for its new product,. They would like to satisfy as many customers as possible. Given a number of widgets available and a list of customer orders, what is the maximum number of orders the manufacturer can fulfill in full?

Function Description
Complete the function *filledOrders* in the editor below. The function must return a single integer denoting the maximum possible number of fulfilled orders.

filledOrders has the following parameter(s):
order : an array of integers listing the orders
k : an integer denoting widgets available for shipment

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le order[i] \le 10^9$
- $1 \le k \le 10^9$

▼ Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines contains an integer *order[i]*.
Next line contains a single integer *k* denoting the widgets available.

▼ Sample Case 0

Sample Input For Custom Testing

```
2
10
30
40
```

Sample Output

```
2
```

Explanation
order = [10,30] with 40 widgets available. Both orders can be fulfilled.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
```

```
#!/bin/python3 -

#
# Complete the 'filledOrders' function below.
#
# The function is expected to return an INTEGER.
# The function accepts following parameters:
#  1. INTEGER_ARRAY order
#  2. INTEGER k
#

def filledOrders(order, k):
    order.sort()
    res = 0
    for x in order:
        if x <= k:
            res += 1
            k = k - x
        else:
            break
    return res
```

Language: Python 3 | Environment | Autocomplete Ready | Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0 | Test case 1 | Test case 2

Input (stdin) | Run as Custom Input | Download

```
2
10
30
```


Language: Python 3 | Environment | Autocomplete Ready | Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0 | Test case 1 | Test case 2 | Test case 3 | Test case 4 | Test case 5 | Test case 6

Input (stdin) | Run as Custom Input | Download

```
2
10
30
40
```

Your Output (stdout)

```
2
```

Expected Output | Download

```
2
```

- $1 \le n \le 2 \times 10^5$
- $1 \le order[i] \le 10^9$
- $1 \le k \le 10^9$

▾ Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines contains an integer *order[i]*.
Next line contains a single integer *k* denoting the widgets available.

▾ Sample Case 0

Sample Input For Custom Testing

```
2
10
30
40
```

Sample Output

```
2
```

Explanation
order = [10,30] with 40 widgets available. Both orders can be fulfilled.

▾ Sample Case 1

Sample Input For Custom Testing

```
3
5
4
6
3
```

Sample Output

```
0
```

Explanation
order = [5,4,6] with 3 widgets available
None of the orders can be fulfilled.

Language: Python 3 | Environment | Autocomplete Ready

```
#  1. INTEGER_ARRAY order
#  2. INTEGER k
#

def filledOrders(order, k):
    order.sort()
    res = 0
    for x in order:
        if x <= k:
            res += 1
            k = k - x
        else:
            break
    return res
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0 | **Test case 1** | Test case 2 | Test case 3 | Test case 4 | Test case 5 | Test case 6

Input (stdin) — Run as Custom Input | Download

```
3
5
4
6
3
```

Your Output (stdout)

```
0
```

Expected Output — Download

```
0
```

2. Unexpected Demand

A widget manufacturer is facing unexpectedly high demand for its new product,. They would like to satisfy as many customers as possible. Given a number of widgets available and a list of customer orders, what is the maximum number of orders the manufacturer can fulfill in full?

Function Description
Complete the function *filledOrders* in the editor below. The function must return a single integer denoting the maximum possible number of fulfilled orders.

filledOrders has the following parameter(s):
order : an array of integers listing the orders
k : an integer denoting widgets available for shipment

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le order[i] \le 10^9$
- $1 \le k \le 10^9$

▾ Input Format For Custom Testing

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

Each line *i* of the *n* subsequent lines contains an integer *order[i]*.
Next line contains a single integer *k* denoting the widgets available.

▾ Sample Case 0

Sample Input For Custom Testing

```
2
10
30
40
```

Sample Output

Language: Python 3 | Environment | Autocomplete Ready

```
#  1. INTEGER_ARRAY order
#  2. INTEGER k
#

def filledOrders(order, k):
    order.sort()
    res = 0
    for x in order:
        if x <= k:
            res += 1
            k = k - x
        else:
            break
    return res
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0 | Test case 1 | **Test case 2** | Test case 3 | Test case 4 | Test case 5 | Test case 6

Input (stdin) — Run as Custom Input | Download

```
2
21
24
83178701
```

Your Output (stdout)

```
2
```

Expected Output — Download

```
2
```