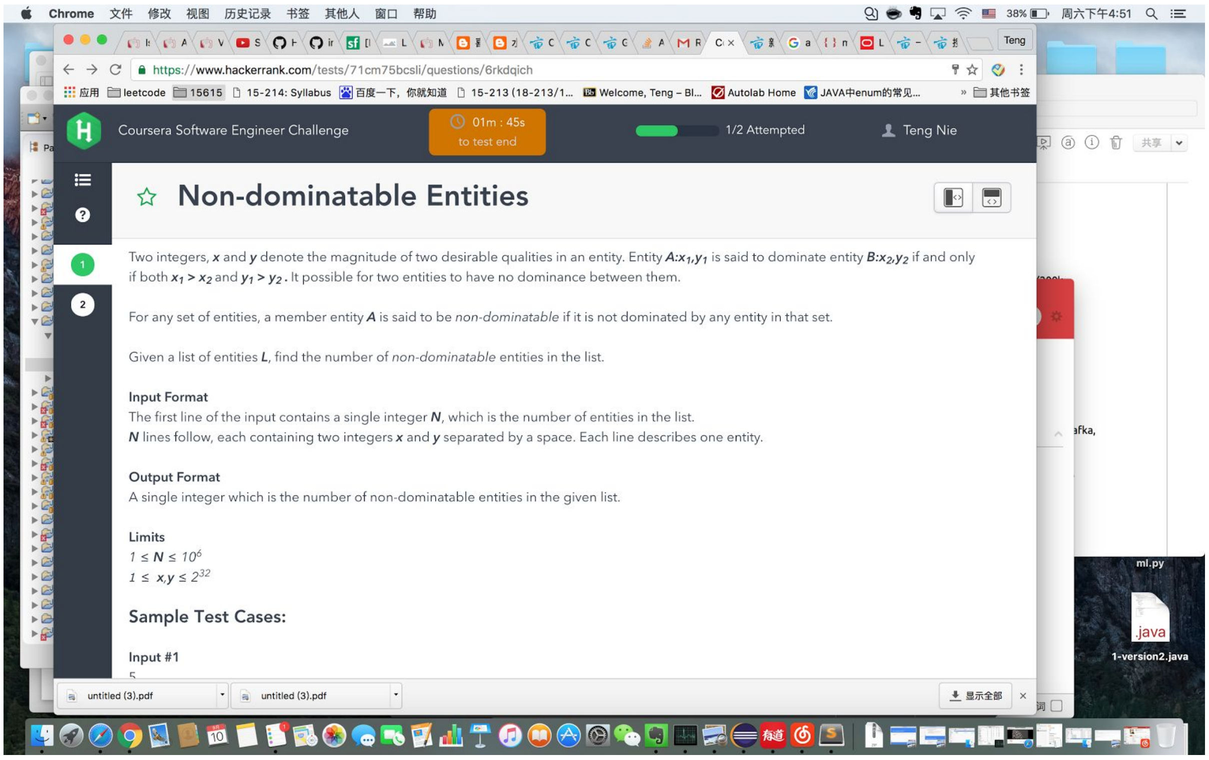


Non-dominatable Entities

Two integers, *x* and *y* denote the magnitude of two desirable qualities in an entity. Entity $A{:}x_1{,}y_1$ is said to dominate entity $B{:}x_2{,}y_2$ if and only if both $x_1 > x_2$ and $y_1 > y_2$. It possible for two entities to have no dominance between them.

For any set of entities, a member entity ***A*** is said to be *non-dominatable* if it is not dominated by any entity in that set.

Given a list of entities ***L***, find the number of *non-dominatable* entities in the list.

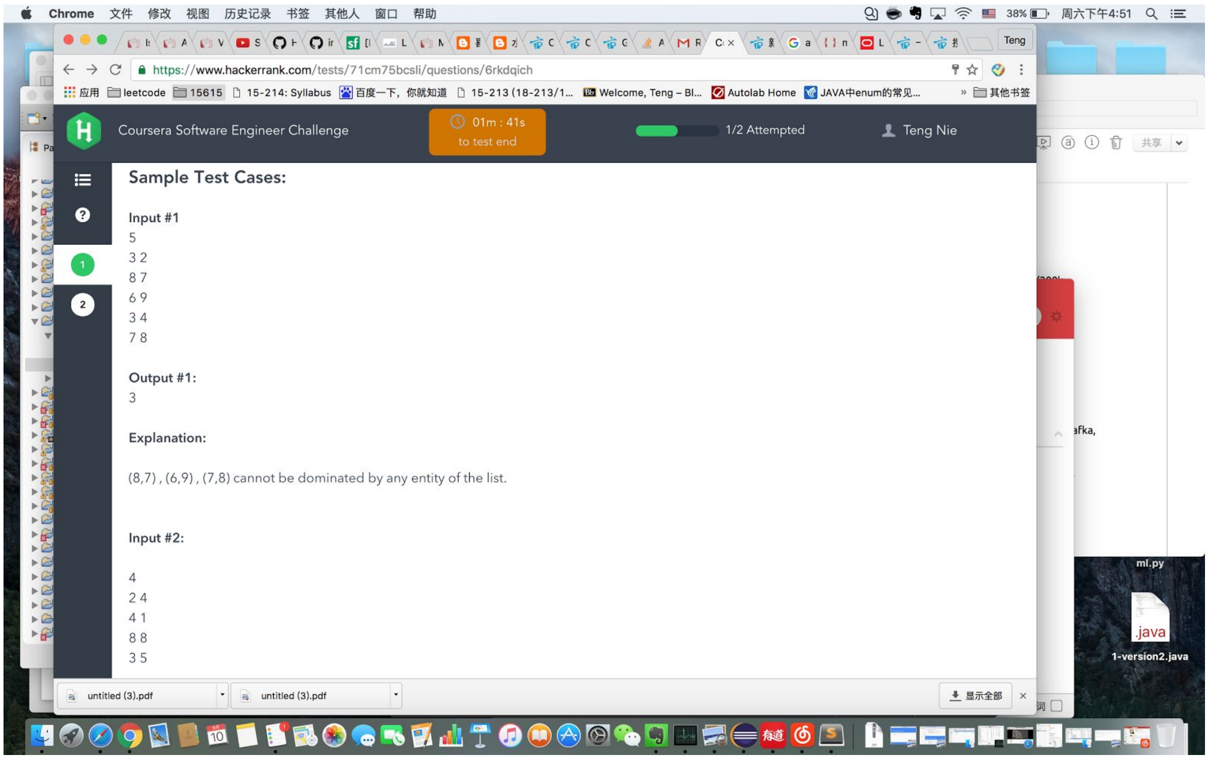
Input Format
The first line of the input contains a single integer ***N***, which is the number of entities in the list.
N lines follow, each containing two integers ***x*** and ***y*** separated by a space. Each line describes one entity.

Output Format
A single integer which is the number of non-dominatable entities in the given list.

Limits
$1 \le N \le 10^6$
$1 \le x,y \le 2^{32}$

Sample Test Cases:

Input #1



Sample Test Cases:

Input #1

```
5
3 2
8 7
6 9
3 4
7 8
```

Output #1:

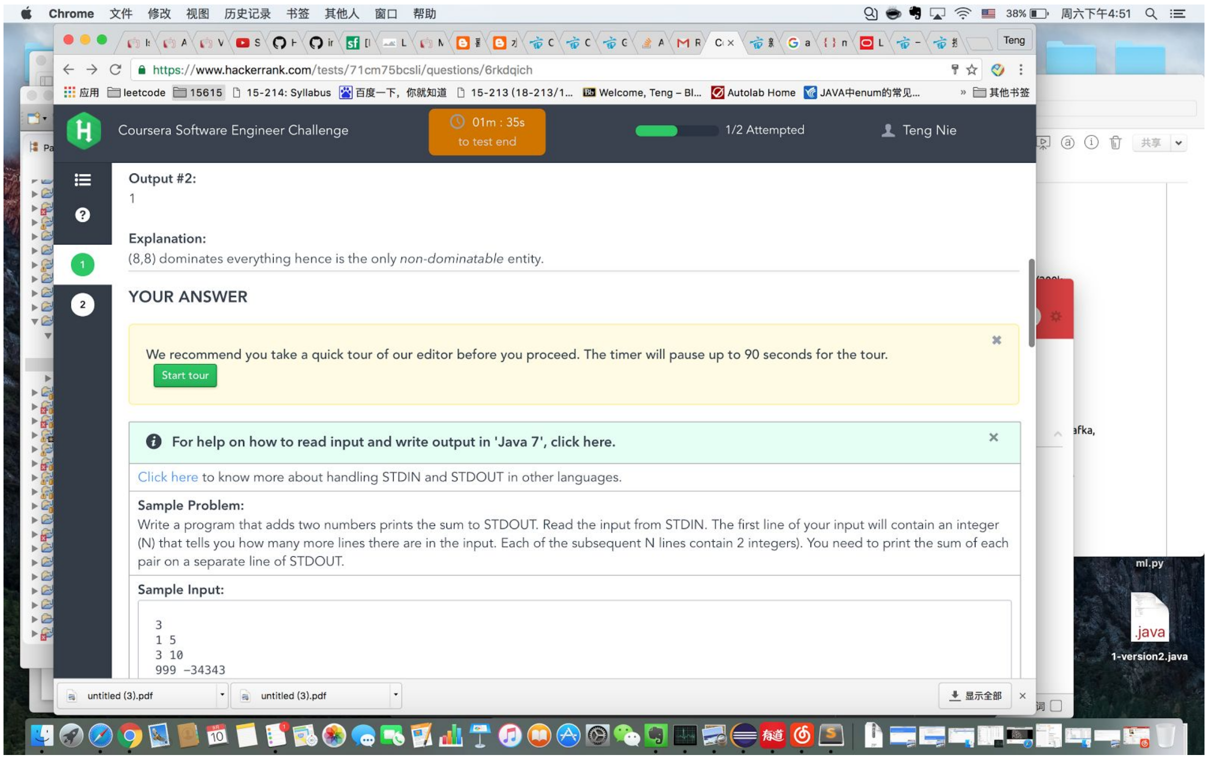
```
3
```

Explanation:

(8,7) , (6,9) , (7,8) cannot be dominated by any entity of the list.

Input #2:

```
4
2 4
4 1
8 8
3 5
```



Output #2:
1

Explanation:
(8,8) dominates everything hence is the only *non-dominatable* entity.

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour.
Start tour

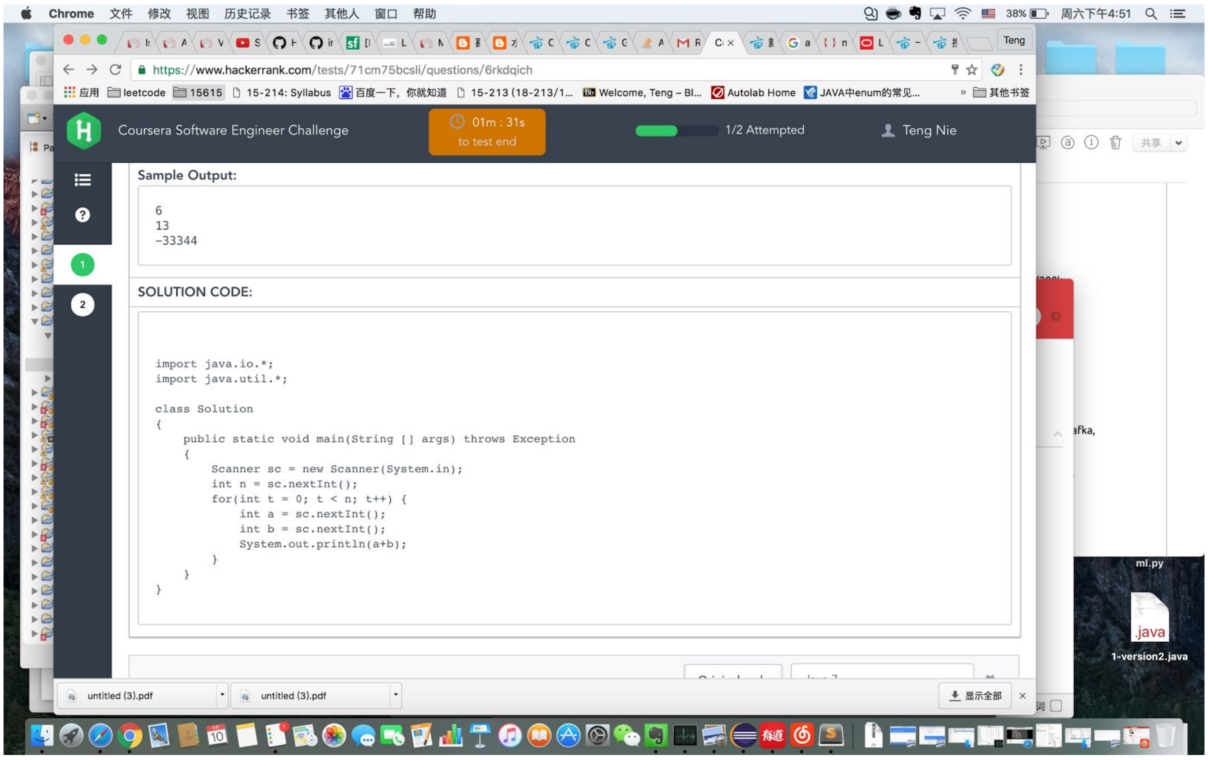
For help on how to read input and write output in 'Java 7', click here.

Click here to know more about handling STDIN and STDOUT in other languages.

Sample Problem:
Write a program that adds two numbers prints the sum to STDOUT. Read the input from STDIN. The first line of your input will contain an integer (N) that tells you how many more lines there are in the input. Each of the subsequent N lines contain 2 integers). You need to print the sum of each pair on a separate line of STDOUT.

Sample Input:

```
3
1 5
3 10
999 -34343
```



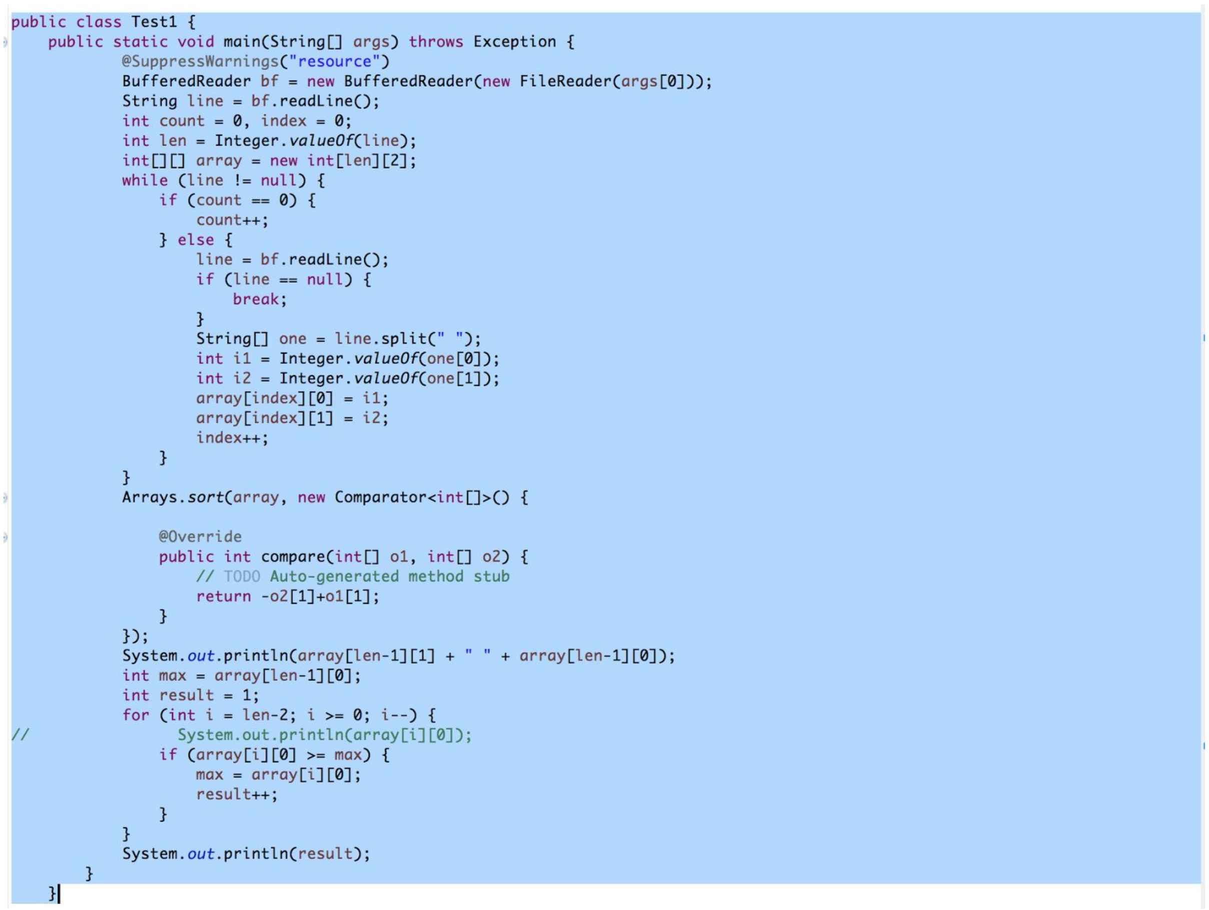
Sample Output:

```
6
13
-33344
```

SOLUTION CODE:

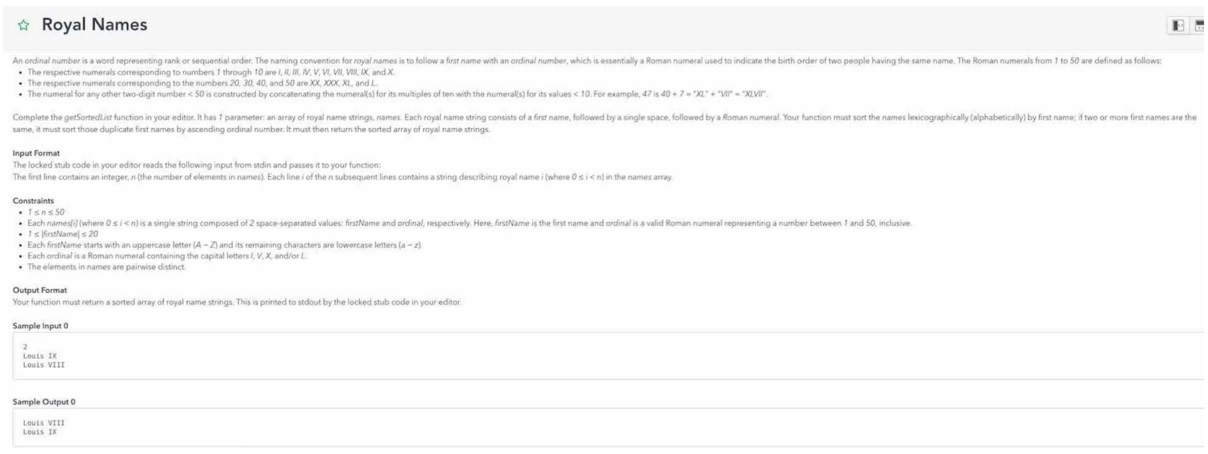
```
import java.io.*;
import java.util.*;

class Solution
{
    public static void main(String [] args) throws Exception
    {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for(int t = 0; t < n; t++) {
            int a = sc.nextInt();
            int b = sc.nextInt();
            System.out.println(a+b);
        }
    }
}
```



```java
public class Test1 {
    public static void main(String[] args) throws Exception {
            @SuppressWarnings("resource")
            BufferedReader bf = new BufferedReader(new FileReader(args[0]));
            String line = bf.readLine();
            int count = 0, index = 0;
            int len = Integer.valueOf(line);
            int[][] array = new int[len][2];
            while (line != null) {
                if (count == 0) {
                    count++;
                } else {
                    line = bf.readLine();
                    if (line == null) {
                        break;
                    }
                    String[] one = line.split(" ");
                    int i1 = Integer.valueOf(one[0]);
                    int i2 = Integer.valueOf(one[1]);
                    array[index][0] = i1;
                    array[index][1] = i2;
                    index++;
                }
            }
            Arrays.sort(array, new Comparator<int[]>() {

                @Override
                public int compare(int[] o1, int[] o2) {
                    // TODO Auto-generated method stub
                    return -o2[1]+o1[1];
                }
            });
            System.out.println(array[len-1][1] + " " + array[len-1][0]);
            int max = array[len-1][0];
            int result = 1;
            for (int i = len-2; i >= 0; i--) {
//                System.out.println(array[i][0]);
                if (array[i][0] >= max) {
                    max = array[i][0];
                    result++;
                }
            }
            System.out.println(result);
        }
    }
```



☆ Royal Names

An *ordinal number* is a word representing rank or sequential order. The naming convention for *royal names* is to follow a *first name* with an *ordinal number*, which is essentially a Roman numeral used to indicate the birth order of two people having the same name. The Roman numerals from *1* to *50* are defined as follows:

- The respective numerals corresponding to numbers *1* through *10* are *I, II, III, IV, V, VI, VII, VIII, IX*, and *X*.
- The respective numerals corresponding to the numbers *20, 30, 40*, and *50* are *XX, XXX, XL*, and *L*.
- The numeral for any other two-digit number < *50* is constructed by concatenating the numeral(s) for its multiples of ten with the numeral(s) for its values < *10*. For example, *47* is *40 + 7 = "XL" + "VII" = "XLVII"*.

Complete the *getSortedList* function in your editor. It has *1* parameter: an array of royal name strings, *names*. Each royal name string consists of a *first name*, followed by a single space, followed by a *Roman numeral*. Your function must sort the *names* lexicographically (alphabetically) by first name; if two or more first names are the same, it must sort those duplicate first names by ascending ordinal number. It must then return the sorted array of royal name strings.

Input Format

The locked stub code in your editor reads the following input from stdin and passes it to your function:
The first line contains an integer, *n* (the number of elements in *names*). Each line *i* of the *n* subsequent lines contains a string describing royal name *i* (where $0 \le i < n$) in the *names* array.

Constraints

- $1 \le n \le 50$
- Each *names[i]* (where $0 \le i < n$) is a single string composed of *2* space-separated values: *firstName* and *ordinal*, respectively. Here, *firstName* is the first name and *ordinal* is a valid Roman numeral representing a number between *1* and *50*, inclusive.
- $1 \le |firstName| \le 20$
- Each *firstName* starts with an uppercase letter (*A − Z*) and its remaining characters are lowercase letters (*a − z*).
- Each *ordinal* is a Roman numeral containing the capital letters *I, V, X*, and/or *L*.
- The elements in *names* are pairwise distinct.

Output Format

Your function must return a sorted array of royal name strings. This is printed to stdout by the locked stub code in your editor.

Sample Input 0

```
2
Louis IX
Louis VIII
```

Sample Output 0

```
Louis VIII
Louis IX
```