Ancestral Names

Given a list of strings comprised of a name and a Roman numeral, sort the list first by name, then by the decimal value of the Roman numeral.

In Roman numerals, a value is not repeated more than three times. At that point, a smaller value precedes a larger value to indicate subtraction. For example, the letter I represents the number *1,*  and *V* represents *5.*  Reason through the formation of *1* to *10* below, and see how it is applied in the following lines.

* *I*, *II*, *III*, *IV*, *V*, *VI*, *VII*, *VIII*, *IX*, and *X* represent *1* through *10*.
* *XX*, *XXX*, *XL*, and *L*are*20, 30, 40*, and *50*.
* For any other two-digit number *< 50*, concatenate the Roman numeral(s) that represent its multiples of ten with the Roman numeral(s) for its values *< 10*. For example, *43* is *40 + 3* = '*XL*' + '*III*' = '*XLIII*'

**Example**

*names = ['Steven XL', 'Steven XVI', 'David IX', 'Mary XV', 'Mary XIII', 'Mary XX']*

The result with Roman numerals is the expected return value. Written in decimal and sorted, they are *['David 9', 'Mary 13', 'Mary 15', 'Mary 20', 'Steven 16', 'Steven 40'].*  The return array is *['David IX', 'Mary XIII', 'Mary XV', 'Mary XX', 'Steven XVI', 'Steven XL']*.

**Function Description**

Complete the function *sortRoman* in the editor below.

*sortRoman* has the following parameter:

*names[n]:*  an array of strings comprised of names and roman numerals

Returns:

*string[n]:*  an array of strings sorted first by given name, then by ordinal

**Constraints**

* *1 ≤ n ≤ 50*
* Each *names[i]* is a single string composed of *2* space-separated values: *givenName* and *romanNumeral*.
* *romanNumeral* represents a number between *1* and *50*, inclusive.
* *1 ≤ |givenName| ≤ 20*
* Each *givenName* starts with an uppercase letter *ascii[A-Z]* which is followed by lowercase letters *ascii[a-z]*.
* There is a space between *givenName* and *romanNumeral*
* Each *names[i]* is distinct.

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 *names*.

Each of the next *n* lines contains an element *names[i].*

Sample Case 0

**Sample Input**

STDIN     Function

-----     -----

2   →  names[] size n = 2

Louis IX   →  names = ['Louis IX', 'Louis VIII']

Louis VIII

**Sample Output**

Louis VIII

Louis IX

**Explanation**

Sort first by *givenName* then, if *givenName* is not unique, by the value of the Roman numeral. In decimal, the list is sorted *['Louis 8', 'Louis 9'].*

Sample Case 1

**Sample Input**

STDIN     Function

-----     -----

2   →  names[] size n = 2

Philippe I   →  names = ['Philippe I', 'Philip II']

Philip II

**Sample Output**

Philip II

Philippe I

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'sortRoman' function below.

\*

\* The function is expected to return a STRING\_ARRAY.

\* The function accepts STRING\_ARRAY names as parameter.

\*/

public static List<String> sortRoman(List<String> names) {

// Write your code here

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int namesCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> names = IntStream.range(0, namesCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.collect(toList());

List<String> result = Result.sortRoman(names);

bufferedWriter.write(

result.stream()

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*