Veyor Coding Challenge

Introduction

Thank you for accepting this coding challenge. Whilst the example code is in Java you may also answer the questions in C# or C++.

Please write production quality code. This implies implementing unit tests and the right level of error checking.

Q2 can be solved with a brute force algorithm. But if time permits then you are encouraged to address it with a more optimal solution.

Upload your source code and associated build instructions to a cloud storage repository and share the link with Veyor via email.

Questions

Q1. An array of integers *arr*, of size *n* is defined as [*a[0], a[1], ..., a[n-1]*]. You will be given an array of integers to sort. Sorting must first be by frequency of occurrence, then by value. For instance, given an array [4, 5, 6, 5, 4, 3], there is one each of 6's and 3's, and there are two 4's, two 5's. The sorted list is [3, 6, 4, 4, 5, 5].

Function Description

Complete the function *customSort* in the editor below. The function must print the array each element on a separate line, sorted ascending first by frequency of occurrence, then by value within frequency.

customSort has the following parameter(s):

$arr[arr_0,...arr_{n-1}]$: an array of integers to sort

Constraints

- $1 \le n \le 2 \times 10^5$

- *$1 \leq arr[i] \leq 10^{6}$*

Input Constraints

Input from stdin will be processed as follows and passed to the function.

- The first line contains an integer *n*, the size of the integer array *arr*.
- The next *n* lines each contain an element *arr[i]*.

Sample Input 0

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

Sample Output 0

```
1
3
4
2
2
```

```
class Result {

    /*
     * Complete the 'customSort' function below.
     * The function accepts INTEGER_ARRAY arr as parameter.
     */
    public static void customSort(List<Integer> arr) {

    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));

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

        List<Integer> arr = IntStream.range(0, arrCount).mapToObj(i -> {
            try {
                return bufferedReader.readLine().replaceAll("\\s+$", "");
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
        })
            .map(String::trim)
            .map(Integer::parseInt)
            .collect(toList());

        Result.customSort(arr);

        bufferedReader.close();
    }
}
```

Q2. Julia is collecting money from her classmates for a trip. Each classmate has a unique ID number where ID numbers start at *1* and increment by *1* until all classmates have a number. Each classmate is prepared to donate the number of dollars that matches their ID, so classmate *1* can give *1* dollar, classmate *2* can give *2* dollars and so on. Julia is superstitious, though, and does not ever want to have a sum that matches her unlucky number. To avoid this, she may refuse a donation from any classmate. If she visits all of her classmates in ID order, what is the maximum amount of money she can collect without ever having a number of dollars that matches her unlucky number? Since the result may be very large, return the result modulo *1000000007*.

For example, there are $n = 4$ classmates, and her unlucky number $k = 6$. If she collects from each of her first three classmates, she will have *6* dollars, which is her unlucky number. To avoid this, she will not collect from at least one classmate, either *1, 2* or *3*. If she skips classmate *1*, she collects $2 + 3 + 4 = 9$ dollars. If she skips number *2*, she collects $1 + 3 + 4 = 8$ dollars, and if she skips classmate *3*, she only collects $1 + 2 + 4 = 7$ dollars. The maximum amount she can collect is *9* dollars.

Function Description

Complete the *maxMoney* function in the editor below. It must return an integer that represents the maximum amount she can collect, modulo *1000000007* $(10^9 + 7)$.

maxMoney has the following parameters:

n: an integer that denotes the number of classmates

k: an integer that denotes Julia's unlucky number

Constraints

- $1 \leq n \leq 2 \times 10^9$

- $1 \le k \le 4 \times 10^{15}$

Input Constraints

The first line contains an integer, *n*, that denotes the number of classmates.

The second line contains an integer, *k*, that denotes Julia's unlucky number.

Sample Input For Custom Testing

Sample Input 0

```
2

2
```

Sample Output 0

```
3
```

Explanation 0

Julia visits the following sequence of *n* = *2* classmates:

1. Julia collects *1* dollar from classmate *1* to get *sum* = *1*.
2. Julia collects *2* dollars from classmate *2* to get *sum* = *1* + *2* = *3*; observe that she collected a maximal amount of money and avoided having exactly *k* = *2* dollars.

Sample Input For Custom Testing

Sample Input 1

```
2

1
```

Sample Output 1

```
2
```

Explanation 1

Julia visits the following sequence of $n = 2$ classmates:

1. Julia will not collect *1* dollar from classmate *1* because $k = 1$ and she refuses to have a $\mathit{sum} \equiv k$ at any time.
2. Julia moves on and collects *2* dollars from classmate *2* to get $\mathit{sum} = 0 + 2 = 2$.

Sample Input For Custom Testing

Sample Input 2

```
3
```

```
3
```

Sample Output 2

```
 5
```

Explanation 2

Julia must skip some classmate because collecting from all her classmates will result in a $\mathit{sum} \equiv k = 3$ when she collects from the second classmate. There are two ways for her to visit all $n = 3$ classmates:

- She can collect *1* dollar from classmate *1* to get $\mathit{sum} = 1$. Next, she can refuse to collect *2* dollars from classmate *2* to avoid having a *sum* equal to *k*. Next, she can collect *3* dollars from classmate *3* to get $\mathit{sum} = 1 + 3 = 4$.
- She can refuse to collect *1* dollar from classmate *1*, meaning that $\mathit{sum} = 0$. Next, she can collect *2* dollars from classmate *2* to get $\mathit{sum} = 0 + 2 = 2$. Next, she can collect *3* dollars from classmate *3* to get $\mathit{sum} = 2 + 3 = 5$.

Because we want the maximum amount of money that Julia can collect from her sequentially-numbered classmates without ever having a *sum* equal to *k*, we return *5* as our answer.

```
public class Solution {

    // Complete the maxMoney function below.
    static int maxMoney(int n, long k) {

    }

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 n = Integer.parseInt(bufferedReader.readLine().trim());

        long k = Long.parseLong(bufferedReader.readLine().trim());

        int res = maxMoney(n, k);

        bufferedWriter.write(String.valueOf(res));
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```