Problem C. Insertion Sort

Input file: `standard input`
Output file: `standard output`

Insertion sort is a simple sorting algorithm that builds the final sorted array one item at an iteration.

More precisely, insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

This type of sorting is typically done in-place, by iterating up the array, growing the sorted array behind it. At each array-position, it checks the value there against the largest value in the sorted array (which happens to be next to it, in the previous array-position checked). If larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted array, shifts all the larger values up to make a space, and inserts into that correct position.

The resulting array after k iterations has the property where the first k entries are sorted. In each iteration the first remaining entry of the input is removed, and inserted into the result at the correct position, thus extending the result.

Knuth is an ACM-ICPC master and provides a modified pseudocode implementation about the insertion sort for you. His modified algorithm for an array of sortable items A (1-based array) can be expressed as:

```
1: function InsertionSort(A, k)                    ▷ Elements in A would be modified
2:     n ← the length of A
3:     i ← 1
4:     while i ≤ n and i ≤ k do                    ▷ the i-th iteration
5:         j ← i
6:         while j > 1 and A[j − 1] > A[j] do
7:             t ← A[j]
8:             A[j] ← A[j − 1]
9:             A[j − 1] ← t
10:            j ← j − 1
11:        end while
12:        i ← i + 1
13:    end while
14: end function
```

Given the parameter k, you are asked to count the number of distinct permutations of 1 to n meeting the condition that, after his modified insertion sort, each permutation would become an almost sorted permutation. Here he notes that a permutation of 1 to n is almost sorted if the length of its longest increasing subsequence is at least $(n - 1)$.

Input

The input contains several test cases, and the first line contains a positive integer T indicating the number of test cases which is up to 5000.

For each test case, the only line contains three integers n, k and q indicating the length of the permutations, the parameter in his implementation and a prime number required for the output respectively, where $1 \le n, k \le 50$ and $10^8 \le q \le 10^9$.

Output

For each test case, output a line containing "`Case #x: y`" (without quotes), where `x` is the test case number starting from 1, and `y` is the remainder of the number of permutations which meet the requirement divided by q.

Example

standard input	standard output
4 4 1 998244353 4 2 998244353 4 3 998244353 4 4 998244353	Case #1: 10 Case #2: 14 Case #3: 24 Case #4: 24

Note

In the first sample case, we can discover 10 permutations which meet the condition, and they are listed as follows:

- $[1, 2, 3, 4]$;
- $[1, 2, 4, 3]$;
- $[1, 3, 2, 4]$;
- $[1, 3, 4, 2]$;
- $[1, 4, 2, 3]$;
- $[2, 1, 3, 4]$;
- $[2, 3, 1, 4]$;
- $[2, 3, 4, 1]$;
- $[3, 1, 2, 4]$;
- $[4, 1, 2, 3]$.