Central European Olympiad in Informatics
Tîrgu Mureş, România
July 8 – 14, 2009
Contest Day 2

sorting 100 points

Input files: **0-sorting.in, 1-sorting.in, …, 9-sorting.in**
Output files: **0-sorting.out, 1-sorting.out, …, 9-sorting.out**
Time limit: **5 hours**
Source code: **None**

Task

For given **N** and **X**, determine the number of permutations of **{1, 2, …, N}** on which Insertion Sort makes at most **X** times the number of comparisons that Quick Sort makes. Since the number can be pretty big, we ask you to output it modulo **1234567**.

The following is our implementation of Insertion Sort, which also counts the number of comparisons it makes:

```
procedure insertionSort(int N, array A[1..N]) defined as:
    A[0] := -Infinity
    for i := 2 to N do:
        j := i
        Increment(comparison_count)
        while A[j - 1] > A[j] do:
            SWAP(A[j - 1], A[j])
            j := j - 1
            Increment(comparison_count)
         end while
    end for
```

The following is our implementation of Quick Sort. If **L** is the length of the list we are sorting in a recursive step, the partition algorithm makes **L-1** comparisons.

```
procedure quickSort(list A) defined as:
    list less, greater
    if length(A) <= 1  then
        return A

    pivot := A[1]
    for i := 2 to length(A) do:
        Increment(comparison_count)
        if A[i] < pivot then append A[i] to less
                        else append A[i] to greater
        end if
    end for
    return concatenate(quickSort(less), pivot, quickSort(greater))
```

For example, let us consider the permutation **(3, 1, 4, 2)**.
The number of comparisons for Insertion Sort is **6**: **2** comparisons for i=2, **1** for i=3, and **3** for i=4.

The number of comparisons for Quick Sort is **4**. In the first iteration 3 is the pivot. It takes **3** comparisons to partition **(1,4,2)** into **(1,2)** and **(4)**. To further sort **(1,2)** it takes **1** more comparison.

Input

The first line contains 2 integers, separated by a space: **N** and **X**.

Output

The number of permutations for which Insertion Sort is at most **X** times slower than Quick Sort. The number should be printed modulo **1234567**.

Constraints

- In all inputs files, $1 < N < 32$.
- In all inputs files, $1 \leq X \leq N^2$
- The official solution computes the answers to all **10** test cases in less than **6** minutes.

Examples

x-sorting.in	x-sorting.out
3 1	2

For the 6 possible permutations we have **NI** and **NQ**, the number of comparisons for Insertion Sort and Quick Sort:

```
1 2 3 - NI = 2, NQ = 3
1 3 2 - NI = 3, NQ = 3
2 1 3 - NI = 3, NQ = 2
2 3 1 - NI = 4, NQ = 2
3 1 2 - NI = 4, NQ = 3
3 2 1 - NI = 5, NQ = 3
```

x-sorting.in	x-sorting.out
6 2	719

x-sorting.in	x-sorting.out
21 3	660773

Description of input

You are not supposed to submit any program to solve the described task. Instead, in the archive you can download from the grading system, you will find the files **`0-sorting.in`**, **`1-sorting.in`**, …, **`9-sorting.in`**. The files are the input data of each of the 10 test cases. You can use the command "**`unzip sorting.zip`**" to extract files from zip archives.

Each of the input files **`0-sorting.in`**, **`1-sorting.in`**, …, **`9-sorting.in`** describes a single test case. The first and only line contains two integers **`N`** and **`X`**, separated by a single space.

Description of output

For each input file, you should create a corresponding output file **`0-sorting.out`**, **`1-sorting.out`**, …, **`9-sorting.out`**. Write all these files to a directory called **`sorting-out`** and create a zip archive containing this directory. You should submit this archive as your solution.

You can use the command "**`zip –r sorting-out.zip sorting-out`**" to create an archive called **`sorting-out.zip`**.

The output files **`0-sorting.out`**, **`1-sorting.out`**, …, **`9-sorting.out`** should consist of exactly one line containing the requested number.

Submission feedback

When you submit your zip archive, the grading system will test whether your files have correct format, but won't test the correctness of the answers. For instance, if you receive a "score" of **`50`** points, it means your archive had **`5`** files that are named correctly, and each contains exactly one number. It **does not** mean that your answers were correct.