(Analysis by Benjamin Qi, Sanjeev Murty)

Call the number of steps that a permutation requires the **period** of the permutation. Every permutation can be partitioned into cycles of sizes $c_1, c_2, c_3, \ldots, c_k$ such that $c_1 + c_2 + \ldots + c_k = N$. Then the period is equal to $\operatorname{lcm}(c_1, c_2, \ldots, c_k)$.

Subtask $N \le 50$:

Maintain the number of possible permutations for each possible LCM of a permutation with n elements for each $1 \le n \le N$. This number is quite small for $N = 50$ (it ends up being 1056) but grows quite rapidly. The number of permutations for a single LCM should be stored $\pmod{M-1}$ because $a^{M-1} \equiv 1 \pmod{M}$ for all $0 < a < M$ (Fermat's Little Theorem).

Subtask $N \le 500$:

In general, we can calculate the period of a permutation as follows:

- Start with the period equal to one.
- Let p be a prime and k be a positive integer.
- If $D = p^k$ divides one of the cycle lengths then multiply the period by p.

So we can essentially solve the problem independently for each distinct prime power D. Doing this in $O(N^2)$ for a single prime power is sufficient for this subtask.

Subtask $N \le 3000$:

If we can count the number of ways to create a permutation of length n for each $n \in [1, N]$ such that no cycle length is divisible by D in $O\left(\frac{N^2}{D}\right)$ time, then the solution runs in $O\left(N^2 \cdot \sum_{D=1}^{N} \frac{1}{D}\right) = O\left(N^2 \log N\right)$ time.

Subtask $N \le 7500$:

The high bound on N (hopefully) ensures that the above solution does not receive full credit. How can we do better?

Note that if $2D > N$ then we can compute the number of permutations containing a cycle with length divisible by D in $O(1)$ time (assuming that we have precomputed some quantities in $O(N^2)$). This is true because if there is a cycle with length divisible by p^k, then there must be exactly one cycle with length equal to p^k (and the rest can have arbitrary lengths).

Let's try to generalize. Define $D = p^k$. Any permutation has between 0 and $\left\lfloor \frac{N}{D} \right\rfloor$ cycles with length divisible by D. So it suffices to count each of the following quantities for each $k \in \left[0, \left\lfloor \frac{N}{D} \right\rfloor\right]$.

- The number of permutations of length Dk such that every cycle has length divisible by D.
- The number of permutations of length $N - Dk$ such that no cycle has length divisible by D.

If we can count both of these in $O\left(\frac{N^2}{D^2}\right)$ time, then this solution runs in $O\left(N^2 \cdot \sum_{D=1}^{\infty} \frac{1}{D^2}\right) = O\left(N^2 \cdot \frac{\pi^2}{6}\right) = O\left(N^2\right)$ time.

Mark Chen's code:

```
#include <bits/stdc++.h>
using namespace std;

typedef long long LL;

int n; LL m;

typedef unsigned long long ull;
typedef __uint128_t L;
```

```
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
    ull reduce(ull a) {
        ull q = (ull)((L(m) * a) >> 64);
        ull r = a - q * b; // can be proven that 0 <= r < 2*b
        return r >= b ? r - b : r;
    }
};
FastMod f(2);

LL mul(LL x, LL y) {
    return f.reduce(x * y);
}

LL add(LL x, LL y) {
    x += y;
    if (x >= m) x -= m;
    return x;
}

LL sub(LL x, LL y) {
    x -= y;
    if (x < 0) x += m;
    return x;
}

LL powmod(LL a, LL b) {LL res=1; a %= (m+1); for(;b;b>>=1) {if (b&1) res = res*a % (m+1); a = a*a % (m+1);} return res;}

const int MAXN = 7505;
LL factorial[MAXN], c[MAXN][MAXN];

int main() {
    freopen("exercise.in","r",stdin);
    freopen("exercise.out","w",stdout);
    cin >> n >> m;

    m--;
    f = FastMod(m);

    factorial[0] = 1;
    for (int i = 1; i < MAXN; ++i) factorial[i] = mul(factorial[i-1], i);

    c[1][0] = c[1][1] = 1;
    for (int i = 2; i < MAXN; i++) {
        c[i][0] = c[i][i] = 1;
        for (int j = 1; j < i; j++) c[i][j] = add(c[i-1][j-1], c[i-1][j]);
    }

    vector<int> composite(MAXN);

    LL ans = 1;

    for (int i = 2; i <= n; i++) {
        if (!composite[i]) {
            for (int j = i; j <= n; j *= i) {
                // count permutations of length j*k where ALL cycles are divisible by j
                vector<LL> aj(n/j+1);
                aj[0] = 1;

                for (int k = 1; k < n/j+1; k++) {
                    for (int l = 1; l <= k; l++) {
                        aj[k] = add(aj[k], mul(mul(c[j*k-1][j*l-1], factorial[j*l-1]), aj[k-l]));
                    }
                }

                // count permutations of length n-j*k where NO cycle is divisible by j
                vector<LL> nj(n/j+1);

                for (int k = n/j; k >= 0; k--) {
                    nj[k] = factorial[n-j*k];
                    for (int l = k+1; l <= n/j; l++) {
                        nj[k] = sub(nj[k], mul(c[n-j*k][n-l*j], mul(aj[l-k], nj[l])));
                    }
                }

                ans = (ans * powmod(i, sub(factorial[n], nj[0]))) % (m+1);
            }

            for (int j = 2*i; j <= n; j += i) {
                composite[j] = 1;
            }
        }
    }
```

```
    printf("%lld\n", ans);
}
```

My code (which uses the principle of inclusion and exclusion):

```
#include <bits/stdc++.h>
using namespace std;

void setIO(string s) {
    ios_base::sync_with_stdio(0); cin.tie(0);
    freopen((s+".in").c_str(),"r",stdin);
    freopen((s+".out").c_str(),"w",stdout);
}

typedef long long ll;
const int MX = 7501;

typedef unsigned long long ul;
typedef __uint128_t L;
struct ModFast {
    ul b, m; ModFast(ul b) : b(b), m(ul((L(1)<<64)/b)) {}
    ul reduce(ul a) {
        ul q = (ul)((L(m)*a)>>64), r = a-q*b;
        return r>=b?r-b:r; }
};

ModFast MF(1);

int M,MOD,n;
int ad(int a, int b) {
    a += b; if (a >= M) a -= M;
    return a;
}
int sub(int a, int b) {
    a -= b; if (a < 0) a += M;
    return a;
}
int mul(int a, int b) { return MF.reduce((ul)a*b); }

int choose[MX][MX];
int with(int z) { // # of permutations with z dividing some cycle length
    int res = 0;
    vector<int> dp(n/z+1); dp[0] = sub(0,1);
    for (int i = 1; i <= n/z; ++i) for (int j = 1; j <= i; ++j)
        dp[i] = sub(dp[i],mul(choose[i*z-1][j*z-1],dp[i-j]));
    for (int i = 1; i <= n/z; ++i)
        res = ad(res,mul(choose[n][n-i*z],dp[i]));
    return res;
}

ll mpow(ll a, ll b) { return !b?1:mpow(a*a%MOD,b/2)*(b&1?a:1)%MOD; }

int main() {
    setIO("exercise");
    cin >> n >> MOD; M = MOD-1; MF = ModFast(M);
    for (int i = 0; i <= n; ++i) {
        choose[i][0] = 1;
        for (int j = 0; j < i; ++j) choose[i][j+1] = mul(choose[i][j],i-j);
    }
    vector<bool> comp(n+1); ll ans = 1;
    for (int i = 2; i <= n; ++i) if (!comp[i]) {
        for (int j = 2*i; j <= n; j += i) comp[j] = 1;
        for (int j = i; j <= n; j *= i) ans = ans*mpow(i,with(j))%MOD;
    }
    cout << ans << "\n";
}
```

It is possible to solve this problem more quickly if you are familiar with exponential generating functions (EGF). [This comment]() gives an explicit formula, which I'll try to explain here. From KACTL, we have the following fact. Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Letting $S(x) = \sum_{n \in S} \frac{x^n}{n}$, it follows that

$$\sum_{n=0} g_S(n) \frac{x^n}{n!} = \exp(S(x)) = \sum_{k=0}^{\infty} \frac{S(x)^k}{k!}.$$

Essentially, the $\frac{S(x)^k}{k!}$ term corresponds to the number of ways to form permutations with exactly k cycles. When all cycle lengths are valid,

$$S(x) = \sum_{n=1}^{\infty} \frac{x^n}{n} = -\ln(1-x)$$

and

$$\exp(S(x)) = \frac{1}{1-x} = 1 + x + x^2 + \cdots,$$

which is clearly correct. If we want to exclude cycles with lengths that are multiples of z, then

$$S(x) = -\ln(1-x) - \frac{1}{z} \cdot \sum_{k=1}^{\infty} \frac{x^{zk}}{k} = -\ln(1-x) + \frac{1}{z} \cdot \ln(1-x^z).$$

It follows that

$$\exp(S(x)) = \frac{(1-x^z)^{1/z}}{1-x}.$$

By the binomial theorem, the numerator has only terms with degree divisible by z. Letting $d = \lfloor n/z \rfloor$ and $[x^n]P(x)$ denote the coefficient of x^n in $P(x)$, it follows that

$$[x^n]\frac{(1-x^z)^{1/z}}{1-x} = [x^{zd}]\frac{(1-x^z)^{1/z}}{1-x^z} = [x^{zd}](1-x^z)^{1/z-1}.$$

By the binomial theorem,

$$g_S(n) = n! \cdot (-1)^d \binom{1/z-1}{d} = \frac{n!}{d!}\prod_{i=1}^{d}(i-1/z) = \frac{n!}{d!z^d} \cdot \prod_{i=1}^{d}(zi-1).$$

So it turns out that we can replace part of the above code with

```
int without(int z) {
    int res = 1;
    for (int i = 1; i <= n; ++i) {
        if (i%z != 0) res = mul(res,i);
        else res = mul(res,i-1);
    }
    return res;
}
int with(int z) { // # of permutations with z dividing some cycle length
    return sub(choose[n][n],without(z));
}
```

although this isn't actually faster since it still runs in $O(N\pi(N)) = O(N^2/\log N)$ time (where $\pi(N)$ denotes the number of primes that are at most N).

Here is a way to derive this formula without generating functions (by Sanjeev):

(BEGIN)

We use $(n)_k$ to denote $n \cdot (n-1) \cdots (n-k+1)$, the falling factorial.

Let a_n be the number of permutations that have no cycles with length dividing z. Then, if we imagine choosing the rest of the cycle that 1 belongs to then recursing, we have

$$a_n = \sum_{\substack{k=1 \\ z\nmid k}}^{n} (n-1)_{k-1} a_{n-k}.$$

Expressing this in terms of a_{n-z}, we have

$$a_n = (n-1)_z a_{n-z} + \sum_{k=1}^{z-1} (n-1)_{k-1} a_{n-k}.$$

You can think of the above as two cases: we choose a cycle of length greater than z or less than z.

Now consider the corresponding expression for a_{n-1}:

$$a_{n-1} = (n-2)_z a_{n-z-1} + \sum_{k=1}^{z-1} (n-2)_{k-1} a_{n-k-1}.$$

If we subtract $n-1$ times this expression from the expression for a_n, we get

$$\begin{aligned} a_n &= (n-1)_z a_{n-z} + (n-1)\Big[a_{n-1} - (n-2)_z a_{n-z-1} - (n-2)_{z-2} a_{n-z}\Big] + (n-1)_0 a_{n-1} \\ &= n a_{n-1} + (n-1)_{z-1}(n-z-1) a_{n-z} - (n-1)_{z+1} a_{n-z-1}. \end{aligned}$$

This already implies an $O(n\pi(n))$ algorithm for the original problem after pre-computation, but we can do better. Manipulating the above, we see

$$\begin{aligned} b_n \triangleq n a_{n-1} - a_n &= (n-1)_{z-1}(n-z-1)\Big[(n-z)a_{n-z-1} - a_{n-z}\Big] \\ &= (n-1)_{z-1}(n-z-1) b_{n-z}. \end{aligned}$$

From the initial conditions, we see that b_n is only nonzero when $z \mid n$. It is then straightforward by induction that $b_n = a_{n-1}$ when $z \mid n$, so we have

$$a_n = \begin{cases} n a_{n-1} & \text{if } z \nmid n \\ (n-1) a_{n-1} & \text{else} \end{cases} = \frac{n!}{z^{\lfloor n/z \rfloor} \lfloor n/z \rfloor !} \prod_{i=1}^{\lfloor n/z \rfloor} (zi - 1)$$

If we precompute $\frac{n!}{z^{\lfloor n/z \rfloor} \lfloor n/z \rfloor !}$ for all prime powers $z \le n$ (noting that it is an integer), then after that we have an

$$O\left(\sum_{\substack{1 \le z \le n, \\ z = p^k}} \frac{n}{z} \right) = O(n \log\log n)$$

algorithm for the problem. First factorize $M-1$ in $O(\sqrt{M})$ time (or $O(n)$, since prime factors of $M-1$ greater than n do not affect our answer). Then we can do this precomputation in $O(n \log M)$ time by looping in increasing order of z and keeping track of the powers of the various prime factors of $M-1$ in $\frac{n!}{\lfloor n/z \rfloor !}$, in addition to the part of it sharing no prime factors with $M-1$. For each prime power $z = p^k$, if it is coprime to $M-1$, we simply multiply $\frac{n!}{\lfloor n/z \rfloor !}$ by $z^{-\lfloor n/z \rfloor} \mod M-1$. Otherwise, we subtract $k\lfloor n/z \rfloor$ from the exponent of p we have been keeping track of. Our final time complexity (assuming a word size of $\Omega(\max(\log n, \log M))$) is then

$$O\left(n(\log\log n + \log M)\right),$$

and we require $O(n)$ space. The space can probably be improved to $O(n/\log n)$. Note that the constant factor for $\log M$ is favorable, since it comes from the maximum number of primes dividing $M-1$ (e.g. 9 for $M \le 10^9$).

(END)

Another solution that runs in $O(N \log N)$ time is to use divide and conquer to initialize a data structure that allows you to query any range product $l \cdot (l+1) \cdots (r-1) \cdot r$ modulo $M-1$ in constant time (where $1 \le l \le r \le N$). This avoids the need to factorize $M-1$.