

# Factorial modulo $p$ in $O(p \log n)$

## Table of Contents

- [Algorithm](#)
- [Implementation](#)

In some cases it is necessary to consider complex formulas modulo  $p$ , containing factorials in both numerator and denominator. We consider the case when  $p$  is relatively small. This problem makes sense only when factorials are included in both numerator and denominator of fractions. Otherwise  $p!$  and subsequent terms will reduce to zero, but in fractions all multipliers containing  $p$  can be reduced, and the resulting expression will be non-zero modulo  $p$ .

Thus, formally the task is: You want to calculate  $n! \bmod p$ , without taking all the multiple factors of  $p$  into account that appear in the factorial. Imaging you write down the prime factorization of  $n!$ , remove all factors  $p$ , and compute the product modulo  $p$ . We will denote this modified factorial with  $n!_{\%p}$ .

Learning how to effectively calculate this modified factorial allows us to quickly calculate the value of the various combinatorial formulae (for example, [Binomial coefficients](#)).

## Algorithm

Let's write this modified factorial explicitly.

$$\begin{aligned}
 n!_{\%p} &= 1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-2) \cdot (p-1) \cdot \underbrace{1}_{p^1} \cdot (p+1) \cdot (p+2) \cdot \dots \cdot (2p-1) \cdot \underbrace{1}_{p^2} \cdot (p^2+1) \cdot \dots \cdot n \\
 &= 1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-2) \cdot (p-1) \cdot \underbrace{1}_{p^1} \cdot 2 \cdot \dots \cdot (p-1) \\
 &\quad \cdot \dots \cdot (p-1) \cdot \underbrace{1}_{p^2} \cdot 1 \cdot 2 \cdot \dots \cdot (n \bmod p) \quad (\text{mod } p)
 \end{aligned}$$

It can be clearly seen that factorial is divided into several blocks of same length expect for the last one.

$$\begin{aligned}
 n!_{\%p} &= \underbrace{1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-2) \cdot (p-1) \cdot 1}_{1\text{st}} \cdot \underbrace{1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-2) \cdot (p-1) \cdot 1}_{2\text{nd}} \cdot \dots \cdot \underbrace{1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-2) \cdot (p-1) \cdot 1}_{p\text{th}} \cdot \dots \cdot \underbrace{1 \cdot 2 \cdot \dots}_{\text{last}}
 \end{aligned}$$

The general part of the blocks it is easy to count — it's just  $(p-1)! \bmod p$  that you can calculate programmatically or via Wilson theorem, according to which  $(p-1)! \bmod p = p-1$ . To multiply these common parts of all blocks, we can raise the value to the higher power modulo  $p$ , which can be done in  $O(\log n)$  operations using [Binary Exponentiation](#); however, you may notice that the result will always be either 1 or  $p-1$ , depending on the parity of the index. The value of the last partial block can be calculated separately in  $O(p)$ . Leaving only the last elements of the blocks, we can examine that:

$$n!_{\%p} = \underbrace{\dots \cdot 1}_{p^1} \cdot \underbrace{\dots \cdot 2}_{p^2} \cdot \dots \cdot \underbrace{\dots \cdot (p-1)}_{p^{p-1}} \cdot \underbrace{\dots \cdot 1}_{p^p} \cdot \underbrace{\dots \cdot 1}_{p^{p+1}} \cdot \dots$$

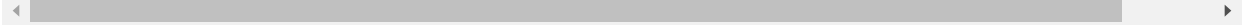
And again, by removing the blocks that we already computed, we receive a "modified" factorial but with smaller dimension ( $\lfloor n/p \rfloor$  blocks remain). Thus, in the calculation of "modified" the factorial  $n!_{\%p}$  we did  $O(p)$

operations and are left with the calculation of  $(n/p)!_{\%p}$ . Revealing this recursive dependence, we obtain that the recursion depth is  $O(\log_p n)$ , the total asymptotic behavior of the algorithm is thus  $O(p \log_p n)$ .

## Implementation

We don't need recursion because this is a case of tail recursion and thus can be easily implemented using iteration.

```
int factmod(int n, int p) {  
    int res = 1;  
    while (n > 1) {  
        res = (res * ((n/p) % 2 ? p-1 : 1)) %  
        for (int i = 2; i <= n%p; ++i)  
            res = (res * i) % p;  
        n /= p;  
    }  
    return res % p;  
}
```



This implementation works in  $O(p \log_p n)$ .