Sum of Divisors

Let $\sigma(n)$ denote the sum of divisors of an integer n.
For example, $\sigma(12)$ = 1 + 2 + 3 + 4 + 6 + 12 = 28.

Your task is to calculate the sum

$\sum_{i=1}^{n} \sigma(i)$ modulo 10^9+7.

Input

The only input line has an integer n.

Output

$Print \sum_{i=1}^{n} \sigma(i) modulo 10^9 + 7.$

Constraints

$1 \le n \le 10^{12}$

Goal

Compute

$$S(n) = \sum_{i=1}^{n} \sigma(i)$$

where

σ(i) = sum of all divisors of i

Example:

σ(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28

But n can be as big as 10^12
So we cannot iterate all 1...n.

Rewrite the Problem

Instead of thinking:

For each number i, find its divisors.

Think the opposite:

Each divisor contributes to some numbers.

If a number d is a divisor of a number i, that means:

i = d · k

So d contributes to all numbers:

d, 2d, 3d, ...

Up to n.
Count how many such multiples exist:

⌊n/d⌋

So each divisor d is added exactly
⌊n/d⌋ times in the entire sigma-sum.

Therefore:

$$\sum_{i=1}^{n} \sigma(i) = \sum_{d=1}^{n} d \left\lfloor \frac{n}{d} \right\rfloor$$

Still impossible to loop to n = 10^12 ...
Unless we exploit structure.

Key Observation (Harmonic Trick)

Even though d goes to 10^12,
the value ⌊n/d⌋ does NOT take 10^12 distinct values.

It changes only about 2√n times.

Example (n = 20)

d	floor(20/d)
1	20
2	10
3	6
4	5
5	4
6	3
7..10	2
11..20	1

Code

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

static const long long MOD = 1000000007;

// Computes sum of integers from L to R modulo MOD
// sum = (R-L+1)*(L+R)/2
long long range_sum(long long L, long long R) {
    long long cnt = (R - L + 1) % MOD;
    long long sumLR = ( (L % MOD) + (R % MOD) ) % MOD;

    // multiply safely, handle /2 under mod
    long long res = (cnt * sumLR) % MOD;

    // divide by 2 modulo MOD
    if (res % 2 == 0) res /= 2;
    else res = (res + MOD) / 2;

    return res % MOD;
}

int main() {
    long long n;
    cin >> n;

    long long ans = 0;
    long long d = 1;

    while (d <= n) {
        long long q = n / d;          // floor(n/d)
        long long r = n / q;          // last d giving same q

        long long sumD = range_sum(d, r);
        ans = (ans + (sumD * (q % MOD)) % MOD) % MOD;

        d = r + 1;
    }

    cout << ans % MOD;
}
```