Problem

Rogelio Guerrero Reyes

September 2025

J. Just an integer

You are given a directed graph $G = (V, E)$ with n nodes. The set of vertices is

$$V = \{1, 2, \dots, n\}.$$

For every node $u \in V$, there is a directed edge from u to each of its *proper divisors*. Formally:

$$(u, v) \in E \iff v \mid u \quad \text{and} \quad 1 \leq v < u.$$

For each node u, define:

- $I(u)$: the length of the longest directed path in G that ends at u.
- $O(u)$: the length of the longest directed path in G that starts at u.

The length of a path is the number of edges it contains.
Your task is to compute the following sum:

$$S(n) = \sum_{u=1}^{n} \max\big(I(u), O(u)\big).$$

Input

The input consists of a single integer n ($1 \leq n \leq 10^{10}$).

Output

Output a single integer — the value of $S(n)$.

1 Solution

First we need to see that:

$$I(u) = \Omega(u)$$

$$O(u) = \left\lfloor \log_2\left(\frac{n}{u}\right)\right\rfloor$$

Where $\Omega(u)$ is the prime omega function.

A naive solution is to compute the value of $\Omega(u)$ and compare it with $O(u)$ for $u = 1, 2, \dots, n$, with a sieve the time complexity is about $n \log\log n$.
This is not suitable for $n > 10^8$. But we can see one thing, $\Omega(u)$ is more likely to be bigger than $O(u)$.

1.1 Omega Sum

We can precalculate the value of

$$\sum_{i=n-10^7}^{n} \Omega(i)$$

For $n = \{10^7, 2\cdot 10^7, \dots, 10^{10}\}$, with a sieve for each range of 10^7 size. With the next code we can get the values we search in an expected time of 20 minutes.

```cpp
const long long N=1e7,C=1e5;

int main(){

    for(int i=2;i<C;i++){
        if(sieve[i])continue;
        primes.pb(i);
        for(int k=2;k*i<N/100;k++)sieve[k*i]=1;
    }

    for(int i=0;i<1000;i++){
        for(int j=0;j<N;j++){
            number[j]=N*i+j+1;
            omega[j]=0;
        }

        for(auto p:primes)
            for(ll j=p-(N*i)%p-1;j<N;j+=p)
                while(number[j]%p==0)number[j]/=p, omega[j]++;

        long long omega_sum=0;

        for(int j=0;j<N;j++){
            if(number[j]!=1)omega[j]++;
            omega_sum+=omega[j];
        }
        cout<<omega_sum<<" ";
    }
}
```

So to compute the value of:

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

For any n, we can just add all the ranges that are fully contain between 1 an n and the compute then values of $\Omega(u)$ for the remaining numbers. This is fast because the remaining numbers are at most 10^7 of them, and we can use a sieve.

1.2 Omega Count

Now we need to add the terms were $O(u) > \Omega(u)$ note that if $\frac{n}{2^{k+1}} < u \leq \frac{n}{2^k}$ then $O(u) = k$ so there is no value such that $u > \frac{n}{4}$ and $O(u) > \Omega(u)$ in general we need:

- The number of primes between 1 and $\frac{n}{4}$
- The number of two prime factors between 1 and $\frac{n}{8}$
- The number of three prime factors between 1 and $\frac{n}{16}$
- etc.

So we can compute in ranges of 10^6 how many numbers with $1, 2, \ldots, 10$ primes factors there are. An in a similar way that in the omega sum, we can compute how many there are up to any number.

Observe that $\frac{n}{2^{12}} < 2.5 \cdot 10^6$ so we can do a naive approach for the rest of them.
With this code we can compute the needed information in 5 minutes

```
const ll N=1e6,C=1e5;

int bound[10]={2500,1250,625,313,157,80,40,20,10,5};

int main(){
    for(int i=2;i<C;i++){
        if(sieve[i])continue;
        primes.pb(i);
        for(int k=2;k*i<N/100;k++)sieve[k*i]=1;
    }

    for(int i=0;i<2500;i++){
        for(int j=0;j<N;j++){
            number[j]=N*i+j+1;
            omega[j]=0;
        }

        for(auto p:primes)
            for(ll j=p-(N*i)%p-1;j<N;j+=p)
                while(number[j]%p==0)number[j]/=p, omega[j]++;

        for(int j=0;j<34;j++)sizes[j]=0;

        for(int j=0;j<N;j++){
            if(number[j]!=1)omega[j]++;
            if(omega[j])sizes[omega[j]-1]++;
        }

        for(int j=0;j<10;j++){
            if(i<bound[j])cout<<sizes[j]<<" ";
            else cout<<"0 ";
        }
        cout<<"\n";
    }
}
```

1.3 Full solution

So finally we can transform the information into strings to save characters and then recover the values, and then compute the total sum as mentioned.