Allocating Shares

Time limit: 4 seconds

The newest huge maths company, South Pacific Computation Corporation (SPPC), is here! SPPC has shares that they need to distribute to their employees. Assume that you are the CEO for SPPC, and you will receive your shares over n days. On day 1, you receive n shares. On day two, you receive $^{n}/_{2}$ shares (rounded down). On day three, you receive $^{n}/_{3}$ shares (rounded down). On day i, you receive $^{n}/_{i}$ shares (rounded down). On the final day (day n), you receive $^{n}/_{n} = 1$ share. For example, if $n = 3$, then you would receive 3+1+1 = 5 shares.

How many shares in total do you receive?

Input

The input consists of a single line containing one integer $\boldsymbol{n}$ **(**$\boldsymbol{1 \leq n \leq 10^{12}}$**)**, which is the number of days over which you receive your shares.

Output

Display the number of shares in total that you receive.

Sample Input`	Output
1	1
3	5
4	8
5	10
10	27

Submission guidelines: You need to upload two files in the system.

(1) Write one page document (upload pdf version of the doc) describing your algorithm or pseudocode. You should describe ***why and how*** your algorithm design should be ***efficient*** (the corresponding program should run fast).
(2) One program file – actual C/C++ or Python code file. Make sure your **code run for n=10^{12} and be aware of the time limit (4 seconds)**.
(3) Take a screenshot of your code execution and the time elapsed.
(4) Zip the above files and submit your zip file in the system.

Marking criteria:

- If your code runs more than 4 seconds for n=10^{12}, you only get at most 80% of the total mark.

Hints:

- Use long long or long long int for C++
- Use numpy.int64 if using Python 2. In Python 3 using int will automatically give you long integer.
- Look at example below. Instead of iterating i from 1 to 16, we can jump i from 6 to 8 because we know all the quotients are the same. Similarly we can jump I from 9 to 16 because we know all the quotients are the same.

To illustrate this, let's take n = 16 as an example and compute ⌊16/i⌋ for i = 1 to 16:

1. ⌊16/1⌋ = 16
2. ⌊16/2⌋ = 8
3. ⌊16/3⌋ = 5
4. ⌊16/4⌋ = 4
5. ⌊16/5⌋ = 3
6. ⌊16/6⌋ = 2
7. ⌊16/7⌋ = 2
8. ⌊16/8⌋ = 2
9. ⌊16/9⌋ = 1
10. ⌊16/10⌋ = 1
11. ⌊16/11⌋ = 1
12. ⌊16/12⌋ = 1
13. ⌊16/13⌋ = 1
14. ⌊16/14⌋ = 1
15. ⌊16/15⌋ = 1
16. ⌊16/16⌋ = 1

You can observe that the values of ⌊16/i⌋ change less frequently as i increases:

- For i = 1 to 3, the values of ⌊16/i⌋ decrease by 1 for each increment of i.
- For i = 4 to 6, the values of ⌊16/i⌋ decrease by 1 every two increments of i.
- For i = 7 to 16, the values of ⌊16/i⌋ remain constant at 1.

- To analyze the time complexity concretely, you need to observe that n/sqrt(n) = sqrt(n)