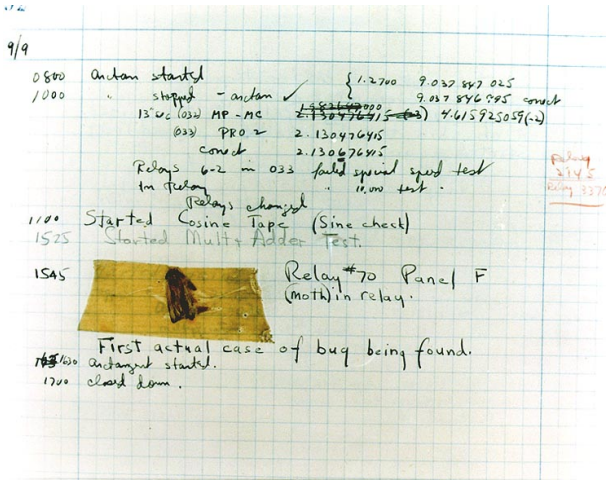
Problem D
Debugging
Time limit: 3 seconds

Your fancy debugger will not help you in this matter. There are many ways in which code can produce different behavior between debug and release builds, and when this happens, one may have to resort to more primitive forms of debugging.



Picture in public domain via Wikimedia Commons

So you and your printf are now on your own in the search for a line of code that causes the release build to crash. Still you are lucky: adding printf statements to this program affects neither the bug (it still crashes at the same original code line) nor the execution time (at least not notably). So even the naive approach of putting a printf statement before each line, running the program until it crashes, and checking the last printed line, would work.

However, it takes some time to add each printf statement to the code, and the program may have a lot of lines. So perhaps a better plan would involve putting a printf statement in the middle of the program, letting it run, seeing whether it crashes before the added line, and then continuing the search in either the first or second half of the code.

But then again, running the program may take a lot of time, so the most time-efficient strategy might be something in between. Write a program that computes the minimum worst-case time to find the crashing line (no matter where it is), assuming you choose an optimal strategy for placing your printf statements.

We're releasing the new version in five hours, so this issue is escalated and needs to be fixed ASAP.

Input

The input consists of one line with three integers:

- n ($1 \leq n \leq 10^6$), the number of code lines;
- r ($1 \leq r \leq 10^9$), the amount of time it takes to compile and run the program until it crashes;
- p ($1 \leq p \leq 10^9$), the time it takes to add a single printf line.

You have already run the program once and therefore already know that it does crash somewhere.

Output

Output the worst-case time to find the crashing line when using an optimal strategy.

Sample Input 1	Sample Output 1
1 100 20	0

Sample Input 2	Sample Output 2
10 10 1	19

Sample Input 3	Sample Output 3
16 1 10	44