ToA Practice Questions for Week 2

Question One [50 marks]

File names

- Use pricing.c if you are writing your program in C.
- Use pricing.cpp if you are writing your program in C++.
- Use Pricing.java if you are writing your program in Java.

Note that case matters.

Problem Description

A collection of precious gems is being sold. In an attempt to prevent a single buyer from purchasing them all, the cost increases, greater than linearly, with the number of gems bought.

The total cost to buy *K* gems is *f(K)* where *f(K)* is defined as the sum of *j * (K/j)* for all integers *0 < j < K*. (Note that *K/j* here uses integer division, so the result has no fractional part. For example: 8/3 would equal 2.)

As an example, to buy 5 gems, the cost, *f(5)*, would be calculated as:

1 * (5/1) + 2 * (5/2) + 3 * (5/3) + 4 * (5/4) = 1*5 + 2*2 + 3*1 + 4*1 = 16

Write a program that, given an amount of currency, *N*, calculates the greatest value of *K* for which *f(K) ≤ N* i.e., Write a program that calculates the largest number of gems that can be bought for a given amount of money.

Example

Given N = 30:
f(6) = 1 * (6/1) + 2 * (6/2) + 3 * (6/3) + 4 * (6/4) + 5 * (6/5)
= 1*6 + 2*3 + 3*2 + 4*1 + 5*1
= 27
f(7) = 1 * (7/1) + 2 * (7/2) + 3 * (7/3) + 4 * (7/4) + 5 * (7/5) + 6 * (7/6)
= 1*7 + 2*3 + 3*2 + 4*1 + 5*1 + 6*1
= 34

So the maximum number of gems that could be purchased would be 6.

Note that the function *f(K)* is *strictly increasing*.

Note that values used in this question can be larger than the maximum value of a 32-bit integer type, requiring the use of 64-bit integer types (long in Java, long long in C and C++).

Input and Output

Program input and output will make use of stdio streams (System.in and System.out in Java) i.e., not file I/O.

Input consists of a single line containing a single integer value, *N*, the maximum amount of currency that can be spent.

Output consists of a single integer, *K*, the maximum number of gems that can be purchased,followed by a line break — in Java, for example, use System.out.println, not System.out.print. The automatic marker expects this precise form.

Sample Input:

30

Sample output:

6

Constraints

$1 \le K \le 1{,}000{,}000$
This means that the input value *N* is bounded by:
$1 \le N \le f(1{,}000{,}000)$ i.e. $1 \le N \le 822467118437$

Question Two [50 marks]

File names

- Use dividing.c if you are writing your program in C.
- Use dividing.cpp if you are writing your program in C++.
- Use Dividing.java if you are writing your program in Java.

Note that case matters.

Problem Description

A lumber yard has a stock of *N* long wooden planks with lengths $L_1, ..., L_N$. Planks can be divided into shorter planks — a plank of length 12, for example, can be divided into three shorter planks, each of length 4 — but separate pieces can't be joined to form longer planks.

A large order has been placed, requesting that at least *K* planks of equal length, *M*, be delivered.

Write a program that, given *K* and the lengths of planks in stock, $L_1, ..., L_N$, determines the maximum possible value for *M*.

Note that as the length increases, the number of planks that can be made will decrease.

Example

You are given *N* = 4 planks with lengths 10, 14, 15, 11. The order requests a minimum of 6 planks.

The order can be fulfilled with 6 planks of length 7 each:

- The plank of length 10 is divide into one plank of length 7, and one of length 3 (which is discarded).
- The plank of length 14 is divide into two planks of length 7.
- The plank of length 15 is divide into two planks of length 7, and one of length 1 (which is discarded).
- The plank of length 11 is divide into one plank of length 7, and one of length 4 (which is discarded).

The order can't be fulfilled with planks of length 8 or more, so 7 is the maximum length.

Note that although the discarded pieces have a combined length greater than 7, they can't be combined to form a longer plank.

Input and Output

Program input and output will make use of stdio streams (System.in and System.out in Java) i.e., not file I/O.

Input consists of a series of integer values, each on a separate line. The first value is *N*, the number of planks in stock, followed by the lengths of those planks, L_1, ..., L_N, followed by *K*, the minimum number of planks required.

Output consists of a single integer, *M*, the maximum possible length that will allow *K* planks to be delivered, followed by a line break — in Java, for example, use System.out.println, not System.out.print. The automatic marker expects this precise form.

Sample Input:

```
4
10
14
15
11
6
```

Sample output:

```
7
```

Constraints

$1 \leq N \leq 10{,}000$
$1 \leq L_i \leq 1{,}000{,}000{,}000$
$1 \leq K \leq 10{,}000{,}000$

The answer, M, will be bounded by:

$1 \leq M \leq 10{,}000{,}000$