Gems from the world of data structures and algorithms

Dropping a glass jar...

Difficulty level: *moderate*

You are doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still no break. The setup of the experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest from which you can drop a copy of the jar and not have t break. We call this the *highest safe rung.*

It might be natural to try binary search : drop a jar from the middle rung, and see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you need a single jar - at the moment it breaks, you have the correct answer - but you may have to drop it n times (rather than $\log n$ times as in the binary search solution).

So here is a trade-off: it seems you can perform fewer drops if you are willing to break more jars. To understand better how this trade-off works at a quantitative level, let us consider how to run this experiment given a fixed "budget" of $k \geq 1$ jars. In other words, you have to determine the correct answer - the highest rung - and can use at most k jars in doing so.

1. Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} f(n)/n = 0$.)
2. Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, ..., f_k$ should have the property that each grows asymptotically slower than the previous one : $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ for each k.

every art is beautiful and so is the art of algorithm design ...