



The 2-Lightbulb, 100 Story Building Puzzle

Version	1.1, 10 January 2006
Language	Maths Puzzles, Python
Author	Craig Mason-Jones (craig@lateral.co.za)
Licence	http://creativecommons.org/licenses/by-nc-sa/2.0/za/

Summary

A simple problem apparently used in recruitment kept me up. I kept thinking of solutions, but wasn't happy they were the correct solutions – I couldn't quite explain why they worked, but a few solutions I found on the web agreed with those I had thought of. Then I realised how one really needs to approach the problem, and the solution was easy.

The Problem

You have two light bulbs and 100 storey building. You must determine the minimal floor such that if you drop the light bulb from that floor it breaks. Once you break a bulb, it can't be reused.

Question: What's the smallest number of drops required in the WORST case to determine the minimal floor.

(<http://www.tranceaddict.com/forums/showthread/t-141475.html>)

The worst case scenario is 14 drops. Here's how I worked it out:

Binary Search

Firstly, I started thinking about a binary search algorithm, for example, start on floor 50, drop the bulb, if it doesn't break, go to floor 75, etc. If it does break, though, you're start on floor 1 and work your way up to 50, one floor at a time. If the bulb breaks on floor 49, then, your worst-case is 50 drops : not great for a 100 floor building.

Big-Step Algorithm

My next idea was to see the situation as a sequence of 'big' steps, which one takes with the first bulb, and 'small' steps, which one takes with the second bulb. Because you've only got two bulbs, those 'small' steps are always single floors, so you can move up in some number of fixed big steps, and when the bulb breaks, work your way up, one floor at a time, from the last good floor, until you find the exact floor on which the bulb breaks. If the bulb breaks on floor F , the number of drops we have is $(F \text{ div } x + 1 + F \text{ mod } x)$ for a big step size of x .

$$F_{\text{worst}} = \begin{cases} (100 \text{ div } x) * x - 1 & , \text{ if } x \nmid 100 \\ 99 & , \text{ if } x | 100 \end{cases}$$

Some simple Python to play with this idea:

```

1 def calcDrops(bigStep):
2     """Returns list of drops required for each floor with a given big step size"""
3     return [i / bigStep + 1 + i % bigStep for i in range(1,101)]
4 def calcWorstDrops():
5     """Returns the worst-case for each big-step type"""
6     return [(i, max(calcDrops(i))) for i in range(1,101)]

```

The best result is with big steps of 8, 9, 10, 11, 12, or 13, for each of which we have a worst-case scenario of 19 drops.

Steps	Worst-Case Drops	Bulb breaks on floor
7	20	97
8	19	95
9	19	98
10	19	99
11	19	98
12	19	95
13	19	90
14	20	97

This looks quite reasonable, and appears to have to do with minimizing the number of 'big' steps in 100, as against the size of each step (i.e. The maximum number of 'small' step drops that would have to be made.) We see that:

$$100 / 7 + 6 = 20$$

$$100 / 8 + 7 = 19$$

$$100 / 9 + 8 = 19$$

...

$$100 / 13 + 12 = 19$$

$$100 / 14 + 13 = 20$$

There is something quite logical in this solution. I thought I had the correct solution until I started thinking how I might improve it, which involved thinking recursively.

If I have a worst-case scenario of 19 drops, there is no point in starting on floor 10. If I start on floor 19, and the bulb breaks, I work with the second bulb from floor 1, one floor at a time, until the bulb breaks (or does not), at floor 18, and my worst-case scenario is still 19 bulbs. Now, I thought, I can use my algorithm above, but not starting on floor 10, rather on floor 19. But I don't need to use the 'big-step' algorithm above, I can continue with my 'worst-case-constant' algorithm, considering the worst case for the 100 – 19 remaining floors...

Worst-Case Constant Algorithm

Assume a worst case of n . Start on floor n . If the first bulb breaks on the first floor (floor n), you will need to try, in the worst-case, the $n-1$ floors beneath floor n before determining that floor n is the 'breaking' floor. If the first bulb doesn't break on floor n , in order to keep the worst-case at n , you now need to try floor $n + (n-1)$, so that if the bulb breaks, you will have (in the worst-case), 2 drops on the first bulb, and $(n-2)$ drops on the second bulb. Again, if the first bulb doesn't break on the 2nd drop, you need to try floor $n + (n-1) + (n-2)$ so that the worst-case becomes 3 drops on the first bulb and $(n-3)$ drops on the second bulb

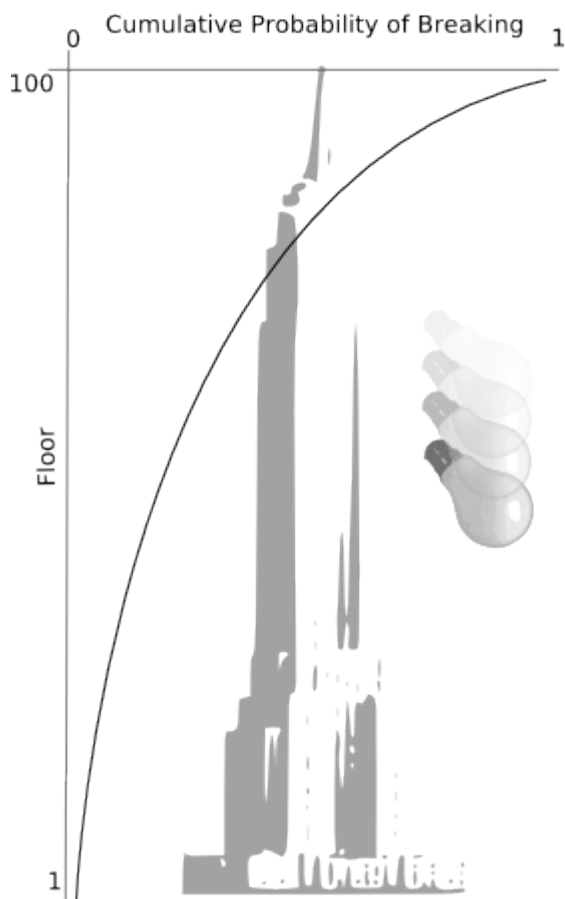
So the solution is to find the smallest n such that $n + (n-1) + (n-2) + \dots + 3 + 2 + 1 \geq 100$. That's a sum of a sequence from 1 to n :

$$\frac{1}{2}n(n+1) \geq 100 \Rightarrow n(n+1) \geq 200 \Rightarrow n = 14, n \in \mathbb{N}$$

The worst-case scenario is 14 drops, and the worst-case dropping sequence occurs whenever the 'breaking floor' is any of the 'first-bulb' floors, or the floor immediately below such a floor – in either case, you won't know until you've dropped the second bulb from the floor below.

Other Thoughts

This solution assumes that there is a uniform probability that the bulbs will break on any floor. Since, in some formulations of the problem, the bulbs are advertised as 'unbreakable', one might want to consider how to alter the algorithm based on an assumption of a non-uniform distribution. The cumulative probability function of the bulb having broken by a given floor might be something like this:



The problem becomes not how to reduce the worst-case scenario (we already have the solution to that problem), but how to reduce the most likely number of drops. The solution would of course depend upon one's estimated parameters, but now we're talking statistics, not algorithms.