Loan Repayment Analysis

July 30, 2020

1 Proof

First off, to summarize the structure of the problem, you start with a total number: n, as well as a fixed x, that you binary search on. While $n > 0$ you compute $max(\lfloor \frac{n}{x} \rfloor, m)$, and subtract that amount from n. Throughout we will refer to $\lfloor \frac{n}{x} \rfloor$ as y.

At every time step, we will subtract y from n. Since n is reduced to 0 through a series of subtractions of all the y's. We can write:

$$n = y_1 + y_2 + y_3 ... y_k$$

In this case, k represents the total number of operations we do using the naive algorithm. k can be very large here if n is large, and x is large, which will lead to y being small. This is why the naive algorithm TLE's.

Next let's look at a smarter algorithm that groups together all y_i of the same value into one solution. I'm going to change the notation here and say that y_i is the ith distinct value of y, instead of the ith y value used. We will introduce a_i which corresponds to how many times y_i is subtracted from our total loan n. we have now that:

$$n = a_1 * y_1 + a_2 * y_2 + a_3 * y_3 ... + a_k * y_k$$

In this new formulation, we only do k operations where this k is obviously less than the original k, since we bundle together operations that have the same value for y. The next question for us is how much better is this? Can we put a bound on k in terms of n?

Let's reword this problem. Thinking about how big k can be written as the following. We have a bag of size n, and we have many rocks of different integer sizes. We can put many rocks of the same size in the bag. How many different types of rock can we fit in the bag? Using this analogy, n is the size of the bag, a_i is the number of type i rocks you put in the bag, y_i is the size of rock type i. We have that all y_i are distinct. In order to maximize k, or the number of different types of rocks given a fixed bag size constraint, we would never want to put the same type of rock in multiple times. What this translates to is that in worst case, all the $a_i = 1$. Additionally, we want to put the smallest size

rocks in the bag, starting at size 1, then size 2 and so on. Our y_i need to be as small as possible, starting from 1 and going up to k

This means that the worst case for our problem is when $a_i = 1$ and $y_i = i$. We have that in worst case.

$$n = (1)(1) + (1)(2) + (1)(3) + (1)(4)...(1)(k)$$

In this case, we can apply the fact that $1 + 2 + 3 + 4..k = \frac{(k)(k+1)}{2}$ to get that:

$$n \approx \frac{k^2}{2}$$

$$k \approx \sqrt{2 * n}$$

What we have shown is that in the worst possible case, where we try to maximize number of distinct y, we still have that there are around $\sqrt{2 * n}$ distinct values of y meaning that if we do $O(1)$ work per distinct value of y, we can achieve $O(\sqrt{n}log(n))$ as our final time complexity. (Remember the extra $log(n)$ from binary search!