Roblox Physics

Elliot Grafil

May 29, 2019

1 Design and implement an algorithm to find the number that occupies the position 1500 in this sequence

2 Improve your program to find the number at position 100 000. What is the problem with representing the result as a 32- or 64-bit integer? You may use a more tractable number representation.

The problem with using such representations is that the as the position grows the resulting number exceeds the ability for said representations to hold the number causing an overflow. Using just a linear fit to the one data point given, 859 963 392 at 1 500, one could calculate that the number at position 100 000 would be around 57 330 892 800. This is approximately 14 higher then an unsigned 32-bit integer could hold. Just to be clear though that the distribution is in no way linear and is probably many orders of magnitude higher.

3 Discuss the limits of your algorithm (complexity, performance, memory usage

The limits of my algorithm mostly stems from the fact that I am performing an exhaustive search to find number via elimination.

While this leads to a minimal memory footprint of 1 it unfortunately has a time complexity of n^2. The memory usage is what first made me implement this solution since a quick back of the envelope calculation had finding the 4 000 000 000 number take u 240+ gigs of memory.

Due to me not culling the search space this algorithm runs terribly for a high position.

I discuss in the next question potential ways to cull as well as a (potentially) better method of calculation but time is limited so I didn't explore said method.

4 Attempt to optimize your program to find the number at position 4 000 000 000. If the program is unable to finish in few minutes, discuss its limitations.

The limitation on the program is simply due to the fact that I picked an exhaustive search method instead of attempting to calculated the number, α, that holds the position, *pos*, more directly.

Each number must fit the form of:

$$\alpha = 2^l 3^m 5^n \tag{1}$$

We can then put a constraint, C, on l,m,n that:

$$C = l + m + n \tag{2}$$

We then increment C starting at 0, and take the sum of potential combinations, Sum_C for the of values for l,m,n. This will give the total number of numbers that would be generated upto said constraint on l,m,n.

Unfortunately:

$$\alpha_C \not< \alpha_{C+1} \tag{3}$$

Which can be seen by:

$$\alpha_2 \not< \alpha_3 \tag{4}$$

$$2^0 3^0 5^2 \not< 2^3 3^0 5^0 \tag{5}$$

$$25 \not< 8 \tag{6}$$

Ideally, if that for some number, n, that could easily be found/calculated so that the following holds true:

$$\alpha_C < \alpha_{C+n} \tag{7}$$

It could be use that to find what values of C that all possible combinations of α needs to be generated. One would just need to find values for C and n where $Sum_C < pos < Sum_{C+n}$. Then one would directly generate all the α and put it into an ordered container. From there one would pick the, e, element where:

$$e = pos - Sum_C \tag{8}$$

If I had more time to explore the problem I think the α needed to be generated could be further reduced and even potentially find an equation that could directly calculate the α given the position.