Integer Factorization

Fernando Javier López Cerezo

October 8, 2021

1 Objectives

The objective of this project is to estimate the complexity of an algorithm that factorizes an integer and adds its prime divisors into a list. In particular we want to find the complexity of an algorithm implementing the Brute Force factorization method in the average case.

2 Experimental Setup

For this project we used:

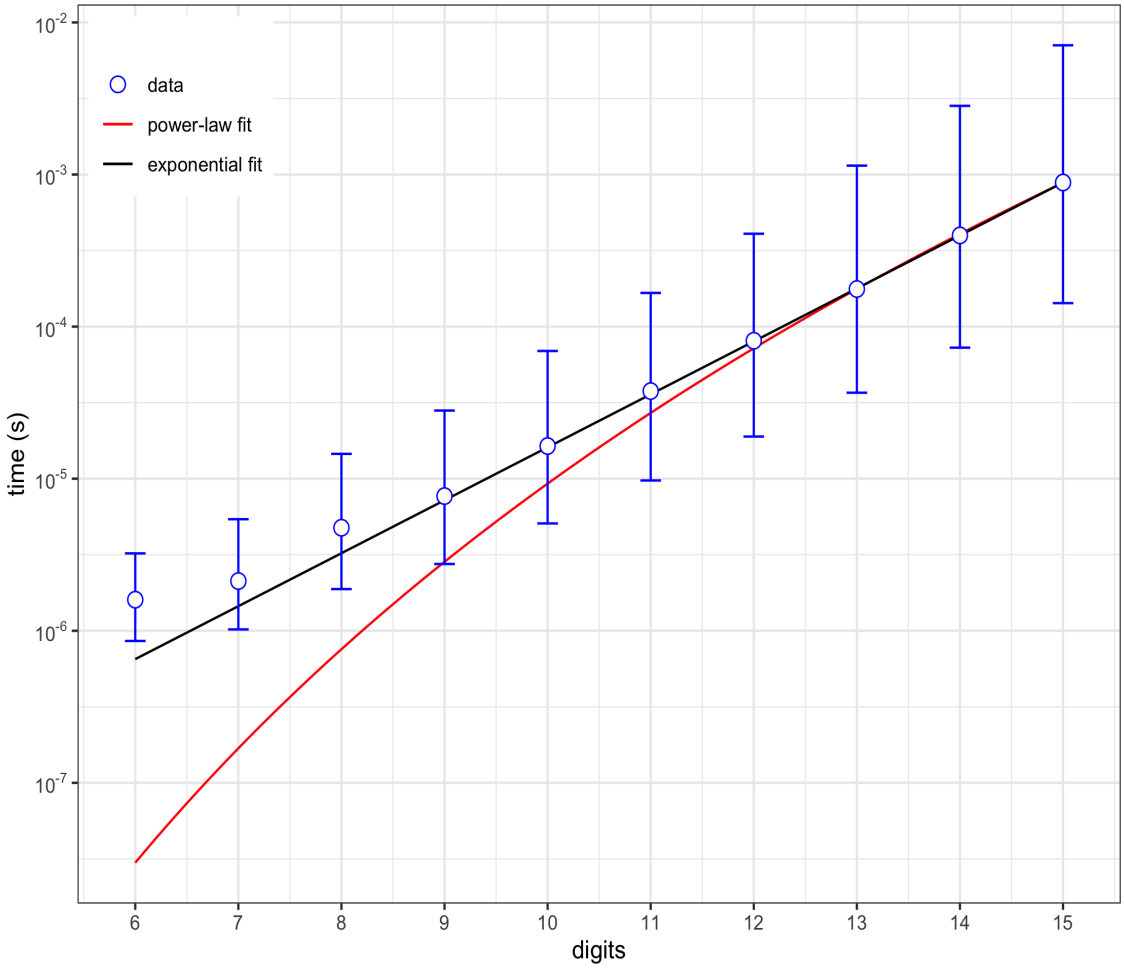
- MacBook Air 13 (1,8 GHz Dual-Core Intel Core i5 CPU, 8 GB 1600 MHz DDR3 RAM and macOS Catalina 10.15.5 OS)
- Java 15.0.2 and R 4.1.1

The method we used to obtain our data was the following:
We implemented the Brute Force factorization algorithm using Java. To make our algorithm as fast as possible given an integer n we first added 2 to the list of prime divisors of n as many times as we could and then checked all uneven numbers from 3 to $\sqrt{n}$ for the rest of prime divisors. Everytime a prime divisor p of n was found we changed n to n/p and recalculated $\sqrt{n}$. This made the algorithm a lot faster especially when the integer we had to factorize was uneven from the start.

To test our algorithm we used different lists of 10,000 random integers. The first list had integers with 6 digits, the second list had integers with 7 digits and so on all the way to the final list which had integers with 15 digits. We ran a Test program which factorized all of these integers and calculated the time taken to go over each list. Finally all of this data was analyzed using R and studied.

3 Empirical Results

Using R we plotted the following graph showing how the time the algorithm takes to execute increases with the number of digits the integer which is factorized has.



R approximated our data to an exponential fit and a power-law fit. In other words, it plotted the functions of the type ad^b (polynomial) and ab^d(exponential) that best fitted our data. R also gave us estimated values for this parameters. If our data was to follow a polynomial pattern of growth the closest polynomial to it would be $5.391^{-17}d^{11.24}$. If, on the other hand, our algorithm followed an exponential pattern of growth that estimated exponential function would be $5.298^{-9}2.23^d$. In order to estimate the order of growth of our algorithm we can actually discard the constants (a) as it won't affect its order of growth (which is what we are loooking for).

4 Discussion

We are interested in estimating the order of growth of the computational cost when the input size grows arbitrarily large. In this case the input size would be the number of digits that the integer to be factorized has. Let's call this order of growth $t(d)$ where d is the number of digits.

In the above graph we see the pattern that an algorithm with a polynomial order of growth would follow (red line). This algorithm would have an order $O(d^b), b \in \mathbb{N}$. Taking a look at the pattern that our algorithm follows (blue circles) we can see that for a low number of digits (< 11) the time taken for our algorithm to be executed is larger than the time taken for an algorithm with a polynomial order of growth. However as d increases the time taken to be executed of both algorithms are more similar to each other. From this data we can conclude that $t(d) \in \Omega(d^b)$.

If we now look at how an algorithm with an exponential order of growth ($O(d) = b^d, b \in \mathbb{N}$) would behave and compare it to our data we can see that our algorithm behaves in a very similar way for all integers with the number of digits that we used. We can therefore conclude that the order of growth of our algorithm is exponential, $t(d) \in \Theta(b^d)$. This means that $t(d) = c^d, c \in \mathbb{N}$

Thanks to the data analysed by R we can approximate this constant c to 2.23. However it should be taken into account that our analysis of the algorithm shows an estimate of the complexity in the average case. Normally we would be interested in finding the complexity in the worst case which for Brute Force factorization is estimated to be $O(3.1623^d)$. It would have also been very beneficial to analyse data with a larger range of input (100 digits, 1000 digits) as we are interested in wthat happens when the input size is very large. However this would have been unpractical as it would have greatly increased the computational cost, making the experiment unfeasible.

Lastly it should also be taken into account that we implemented what probably is the worst algorithm for factorising integers. Alternative algorithms implementing the Fermat method or the Lehman method would have possibly given different results of a lower complexity.

In conclusion the average complexity of the brute force approach is $O(2.23^d)$ where d is the number of digits that the integer has.