Some Bounds on Integer Complexity

Katherine Cordwell*, Alyssa Epstein**, Anand Hemmady**, Aaditya Sharma**, Yen Nhi Truong Vu***
*University of Maryland, College Park; **Williams College, ***Amherst College
ktcordwell@gmail.com, ale2@williams.edu, ash6@williams.edu, as17@williams.edu, ytruongvu17@amherst.edu

What is it?

The *complexity* of $n \in \mathbb{N}$, which we denote by $f(n)$, is defined as the least number of 1's needed to represent n using only the operations of addition and multiplication in conjunction with an arbitrary number of parentheses. For example, since we can write $6 = (1+1)(1+1+1)$, we get that $f(6) \leq 5$.

Richard Guy gives the following recursive definition of $f(n)$:

$$\min_{\substack{d \mid n \\ 2 \leq d \leq \sqrt{n} \\ 1 \leq a \leq n/2}} \left\{ f(d) + f\left(\frac{n}{d}\right), f(a) + f(n-a) \right\}.$$

The growth rate of $f(n)$ is quite slow compared to that of n. For example, $f(100) = 14$ and $f(3133160) = 45$.

What's Been Done?

- 1953: Mahler and Popken propose the problem.
- 1986: Richard Guy popularizes the problem in his "Some Suspiciously Simple Sequences". He attributes a sharp lower bound of $f(n) \geq 3\log_3(n)$ to Selfridge.
- 2008: Fuller publishes a C program to compute $f(n)$.
- 2008: Srinivas and Shankar give an algorithm that runs in time $\mathcal{O}(n^{1.58})$.
- 2012: Iradis et al. prove further experimental and analytical results.
- 2014: J. Arias de Reyna and J. van de Lune give an algorithm that runs in time $\mathcal{O}(n^{1.230175})$.

Best Current Algorithm

J. Arias de Reyna and J. van de Lune's algorithm uses Guy's recursive definition to calculate complexities. The rate-limiting step lies in checking the summands $a \leq n$ such that $f(n) = f(n-a) + f(a)$. To optimize the algorithm, J. Arias de Reyna and J. van de Lune bound the number of summands that must be checked by

$$\frac{n}{2}\left(1 - \sqrt{1 - \frac{4}{n^2}3^{f(n-1)/3}}\right).$$

By defining and analyzing the function $D(b,r)$ as the maximum complexity of multiplying by b and adding r, they bound $f(n)$. Then, combining this information with the bound on the number of summands, they obtain their optimal runtime of $\mathcal{O}(n^{1.230175})$ in base $2^{10}3^7$.

C How Fast

J. Arias de Reyna and J. van de Lune wrote code in Python to perform their analysis, which they have generously sent us. Using this, we have developed comparable code in C that calculates the $D(b,r)$. Since C runs much faster than Python, we are able to calculate values and perform analysis for higher bases. In particular, for base $2^{13}3^8$, we obtain a better runtime of $\mathcal{O}(n^{1.222911236})$.

Guy's Method

Given a base b representation of n, say $(d_0 \cdots d_k)_b$, Guy's method (also called Horner's scheme) writes

$$n = d_k + b(d_{k-1} + b(d_{k-2} + \cdots + bd_0))\cdots)).$$

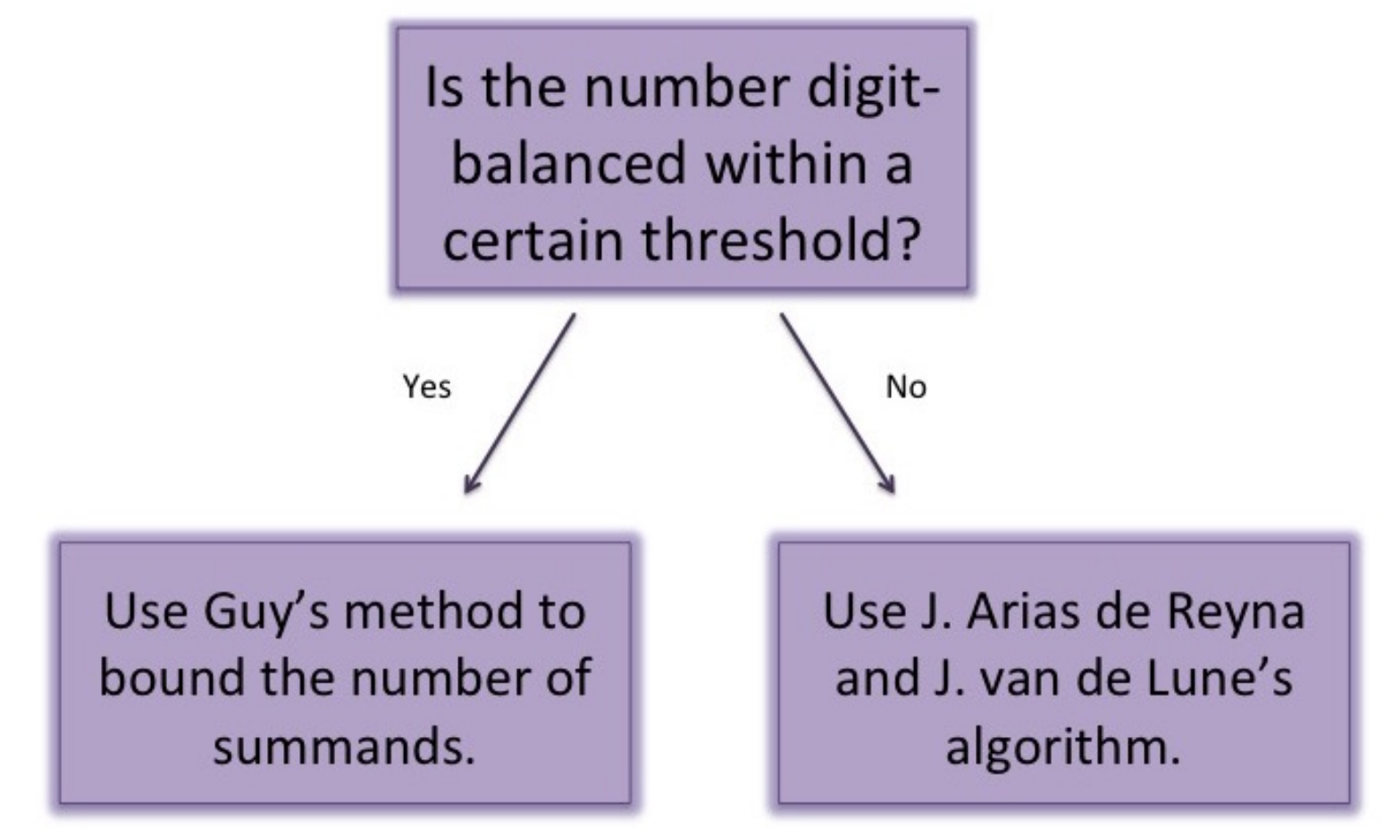
Given base b, define a **digit-balanced number** n as a number with roughly equal proportions of all of the b possible digits. **Digit-unbalanced numbers** are numbers which deviate from this, i.e. digit-unbalanced numbers have some digits appearing significantly more often than other ones. Then, letting $D(b,r)$ be defined as before, using Guy's method, we see that the complexity of n for digit-balanced numbers is then bounded by

$$S = \frac{1}{b\ln(b)}\sum_{r=0}^{b-1} D(b,r).$$

J. Arias de Reyna and J. van de Lune computed this quantity in bases of the form $2^n3^m \leq 2^93^8$ and found that $S \leq 3.6343$ is minimized in base $b = 2^9 \cdot 3^8$. In doing so, they show that a set of numbers of density 1 satisfy $f(n) \leq 3.6343\log_3(n)$. Our computations show that a more optimal base is $2^{11}3^9$, where $f(n) \leq 3.61989\log_3(n)$ for almost all n.

Our Improvements

We propose the following modification of J. Arias de Reyna and J. van de Lune's algorithm:


Is the number digit-balanced within a certain threshold?
Yes
No
Use Guy's method to bound the number of summands.
Use J. Arias de Reyna and J. van de Lune's algorithm.

We need to show that using Guy's method allows us to reduce the number of summands that we must compute. Assume that $f(n) \leq c\log_3(n) = \log_3(n^c)$ for some $c \leq 3.64$ and that $f(n) = f(a) + f(n-a)$. First, using Selfridge's lower bound,

$$\log_3(n^c) \geq 3(\log_3(n-a) + \log_3(a)).$$

Say that $a = kn$, where necessarily $k \leq \frac{1}{2}$. Then we have

$$\log_3(n^{c/3}) \geq \log_3((1-k)n \cdot a).$$

Simplifying gives

$$\frac{n^{c/3-1}}{1-k} \geq a.$$

Using $1 - k \geq \frac{1}{2}$ gives

$$2n^{c/3-1} \geq \frac{n^{c/3-1}}{1-k} \geq a.$$

Then since $c \leq 3.64$, we get $a \leq 2n^{0.214}$. So, what we have is that for almost all numbers, we only need to check summands up to $2n^{0.214}$.

Binary Analysis

As a special case, let us consider the binary base. First, we wish to better characterize how many numbers are not covered by Guy's method. In base 2, J. Arias de Reyna and J. van de Lune's algorithm runs in $\mathcal{O}(n^{1.345})$.

Binary digit-unbalanced numbers are simply those that have a larger proportion of 1's than 0's. For numbers where $p\%$ of the digits are 0's and $(1-p)\%$ are 1's, the constant we obtain from Guy's method is:

$$\frac{1}{\ln(2)}\left(\frac{p}{100}D(2,0) + \frac{p-1}{100}D(2,1)\right)\ln(3).$$

When $p = 1-p = 50\%$, this gives 3.9625. Some other values are:

Percent 0's	Percent 1's	Constant
45	55	4.041655
46	54	4.02581
49	51	3.97826

In particular, the constant 3.97826 would improve the runtime of numbers where at most 49% of the digits are 1's to $\mathcal{O}(n^{1.326})$, and the constant 4.02581 is enough to improve the runtime of numbers where at most 54% are 1's to $\mathcal{O}(n^{1.342})$.

So, we focus on bounding the number of n where at most 46% are 0's. This comes down to bounding

$$B(n,0) + \cdots + B(n,n-d),$$

where $n - d = \frac{46N}{100}$. We use the following bound,

$$B(n,0) + \cdots + B(n,n-d) \leq e^{-nD\left(\frac{n-d}{n},||\frac{1}{2}\right)}$$

where $D\left(\frac{n-d}{n} \,||\, \frac{1}{2}\right)$ is

$$\frac{n-d}{n}\log\left(2\left(\frac{n-d}{n}\right)\right) + \left(1 - \frac{n-d}{n}\right)\log\left(2\left(1 - \frac{n-d}{n}\right)\right).$$

In particular, $D(\frac{46}{100}||\frac{1}{2}) < 0.0032034$, and so the number of $n \leq N$ where at most 46% are 0's is $\leq N^{1-0.0032034}$, and so we can improve the algorithm on a set of $N - N^{1-0.0032034}$ numbers, or in the limit as $N \to \infty$, almost all numbers.

An Unconditional Upper Bound?

Something that we would like to explore more in the future is the idea of using division to improve the upper bound on complexity. The current upper bound is

$$f(n) \leq 3\log_2(n),$$

which comes from applying Guy's method in base 2, where the worst numbers have a binary expansion that contains only 1's. Experimentally, if we start with a number n that has a binary representation with a lot of 1's, then usually $(n - (n \mod 3))/3$ has a binary representation with close to 50% 1's, and so Guy's method provides a much better upper bound for this set of numbers. We would like to characterize the set of numbers where division by 3 does not afford a nice binary representation and to perform an alternate analysis on this set–for example, dividing by 5.

A Generalization

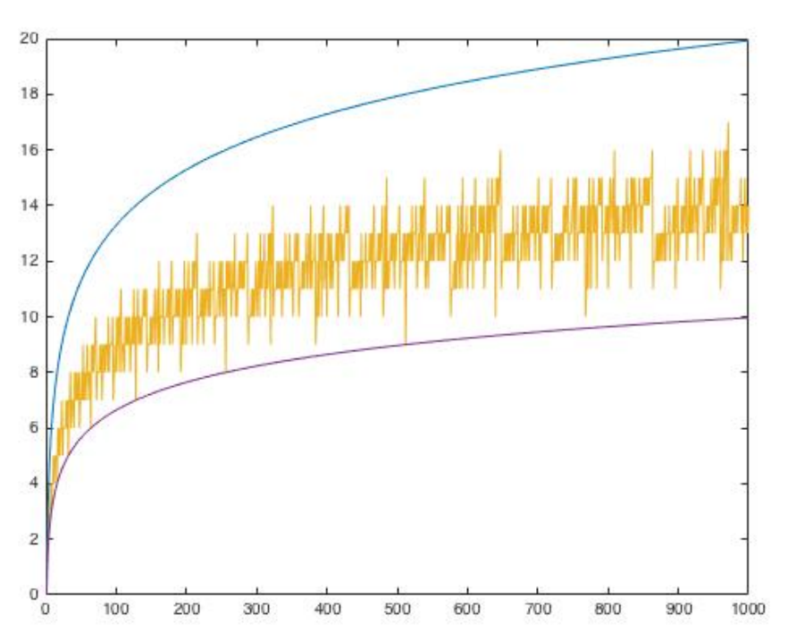
Given $n \in \mathbb{Z}$, define $f_{\{1,x\}}(n)$ as the minimum number of 1's and x's needed to represent n with addition, multiplication, and parentheses. For example, $f_{\{1,5\}}(6) = 2$, since $6 = 5+1$, and $f_{\{1,3\}}(9) = 2$, because $9 = 3 \cdot 3$.
A sharp lower bound on $f_{\{1,x\}}(n)$ follows from an inductive argument in the style of Selfridge.

- Given $n \in \mathbb{Z}$, if $f(n) = k$, then $n \leq x^k$.
- For any $n \in \mathbb{Z}$, $\log_x(n) \leq f(n)$.

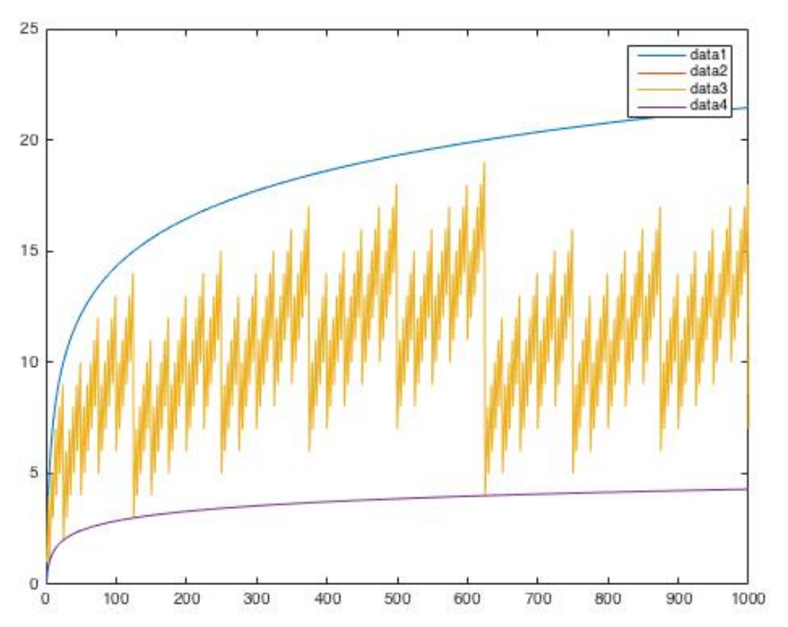
We can define a greedy algorithm, which we aptly name "greedy", to obtain a rough upper bound on $f_{\{1,x\}}(n)$. This algorithm works on $n \equiv t \pmod{x}$ in the following manner:

- If $n < x$, then $\text{greedy}(n) = \underbrace{1 + \cdots + 1}_{\text{t times}}$.
- If $n \geq x$ and $t = 0$, then $\text{greedy}(n) = x \cdot \text{greedy}(\frac{n}{x})$.
- If $n \geq x$ and $t > 0$, then $\text{greedy}(n) = \underbrace{1 + \cdots + 1}_{\text{t times}} + x \cdot \text{greedy}(\frac{n}{x})$.

Here is a plot of the bounds compared to the complexity of "greedy" for $f_{1,2}$:



Here is a plot of the bounds compared to the complexity of "greedy" for $f_{1,5}$:



Of course, there is no reason to stop here. We can continue to generalize $f_{\{1,x\}}$ to $f_{\{1,x_1,\ldots,x_t\}}$ where $x_1 < x_2 < \cdots < x_t$. Again we can obtain a sharp lower bound inductively: $f_{\{1,x_1,\ldots,x_t\}}(n) \leq \log_{x_t}(n)$. Upper bounds, however, become more difficult. We suspect that the upper bound depends on the density of $\{x_1, \ldots, x_t\} \in \mathbb{Z}$.

Acknowledgments

We would like to thank our mentors, Professors Steven J. Miller (Williams College), Eyvindur Palsson (Virginia Tech), and Stefan Steinerberger (Yale University). Thank you to the SMALL REU program, Williams College, and the Williams College Science Center. We would also like to thank Professor Amanda Folsom for funding as well as NSF Grants DMS1265673, DMS1561945, DMS1347804, DMS1449679, the Williams College Finnerty Fund, and the Clare Boothe Luce Program.