HIGHEST FEW SUMS AND PRODUCTS OF ONES

HARRY ALTMAN

Abstract. Let $||n||$ be the smallest number of 1's needed to make n using addition and multiplication, called the complexity of n. Let $L(n)$ be $\min_{m \geq n} ||m||$. Here we classify numbers n satisfying $||n|| = L(n)$, using a proof different from the one in our forthcoming paper on integer complexity more generally. (Much of the introduction, etc., is copied from that.)

1. Introduction

We define the *complexity* of a natural number n to be the smallest number of 1's needed to write it using any combination of addition and multiplication, as introduced in [MP]. Following [A], we denote it $||n||$.

Let us define $E(k)$ to be the highest number writable with k ones, using addition and multiplication, and more generally $E_r(k)$ to be the r-th highest number writable with k ones in this way (zero-indexed). Selfridge showed, as mentioned in [R], E is given by $E(3k) = 3^k$, $E(3k+2) = 2 \cdot 3^k$, $E(3k+4) = 4 \cdot 3^k$. (And of course $E(1) = 1$.) Since this is strictly increasing, $E(k)$ does in fact require k ones. If we define $L(n)$ to be $\min_{m \geq n} ||m||$, or equivalently $\min_{E(m) \geq n} m$, then this allows for easy computation of $L(n)$. Note in particular that $L(3n) = L(n) + 3$ for all $n \neq 1$. Also observe that for any n, $3 \log_3 n \leq L(n) < 3 \log_3 n + 5 - 6 \log_3 2$.

Hence it follows that for $m \leq 2$, $||2^m 3^k|| = 2a + 3k$ as long as a and k are not both zero. In fact, though, Selfridge's result is strong enough to show that this is true for $m \leq 10$, though we have not seen this acknowledged elsewhere.

Proposition 1.1. *$||2^m 3^k|| = 2m + 3k$ for $m \leq 10$, m and k not both zero.*

Proof. It suffices to show it for $m = 10$. Note that $E(19+3k) = 4 \cdot 3^{5+k} = 972 \cdot 3^k < 1024 \cdot 3^k$; hence $2^{10} 3^k$ cannot be made with less than $20 + 3k$ ones. □

Furthermore, Rawsthorne shows in [R] that for $k \geq 6$, $E_1(k) = (8/9)E(k)$. From this, we can conclude

Proposition 1.2. *$||2^m 3^k|| = 2m + 3k$ for $m \leq 13$, m and k not both zero.*

Proof. It suffices to show it for $m = 13$. Note that $E_1(25 + 3k) = 32 \cdot 3^{5+k} = 7776 \cdot 3^k < 8192 \cdot 3^k$. Hence $2^{13} 3^k$ cannot be written with less than $26 + 3k$ ones, unless it is equal to $E(25 + 3k) = 4 \cdot 3^{7+k}$, which it is not. □

This raises the question: Can formulae for further $E_r(k)$ be found? And can they prove the above for even higher m? In this paper we find such formulae for $E_r(k)$, for every r, that hold so long as k is sufficiently large (depending on r). As it turns out, all of these formulae can be summed up in the following statement:

Theorem 1.3. *A number n satisfies $||n|| = L(n)$ if and only if it can be written in one of the following forms:*

- $2^a 3^k$ *for some* $a \leq 10$ *(of complexity* $2a + 3k$ *unless* $a = k = 0$*)*
- $2^a(2^b 3^m + 1)3^k$ *for* $a + b \leq 2$ *(of complexity* $2(a + b) + 3(m + k) + 1$ *unless* $a = m = 0$*)*

From this we can conclude

Proposition 1.4. $||2^m 3^k|| = 2m + 3k$ *for* $m \leq 18$*,* m *and* k *not both zero.*

Proof. It suffices to show it for $m = 18$. $n = 2^{18}3^k$ is not of one of the forms listed above, so we must have $||n|| \geq L(n) + 1$. $L(2^{18}3^k) = 3k + L(2^{18}) = 3k + 35$, hence $||2^{18}3^k|| \geq 3k + 36$. □

Unfortunately it is not possible to go further than this based purely on knowledge of E_r; methods for doing so are addressed in our forthcoming paper [AZ], in which we prove the above statement for $m \leq 30$.

Here, though, we will provide our original, less generalizable proof of the above theorem. We do this by first proving the formulae for $E_r(k)$ that hold so long as k is sufficiently large relative to r. The proof will be algorithmic; see the accompanying Haskell code for the implementation of this algorithm that was used to obtain the final results. Once we have these, we will demonstrate the equivalence with Theorem 1.3.

2. Idea: A Computation of $E(k)$

Of course, to determine $E_r(k)$, it suffices to enumerate all possible $+, \cdot, 1$ expressions with k ones, evaluate each, and find the r-th-highest result. But this is both slow and hard to abstract. So, we will present a proof of how to determine $E(k)$, and then use this to find a better way to determine $E_r(k)$. This is far from the simplest proof of this – see e.g. [R] or [SS] for an inductive proof, due to Selfridge – but it is the one that is the basis of the algorithm.

In this section, capital letters will be used to denote $+, \cdot, 1$ expressions; for such an expression A, val A will denote its value.

Theorem 2.1. *For* $k \geq 0, E(3k) = 3^k, E(3k + 2) = 2 \cdot 3^k, E(3k + 4) = 4 \cdot 3^k$.

Proof. Suppose we have any $+, \cdot, 1$ expression with k 1s, and suppose $k > 1$. We beginning by changing the operations to use the best operation at each step. So, if neither of the two operands are 1, we change a $+$ to a $\cdot$, and vice versa if either operand is 1. We will call this an α transformation.

$$\alpha : A + B \mapsto AB \text{ if val } A, \text{val } B \neq 1$$

$$\alpha : AB \mapsto A + B \text{ if val } A = 1 \text{ or val } B = 1$$

Apply α transformations until this becomes impossible. This process can be seen to terminate, as the only way an operation can be changed more than once is if it changes from having a 1 as operand to not, as each α transformation cannot decrease any subexpression, and this can happen at most once by the same reasoning. Thus no operation may be changed more than twice, and the process terminates.

Once this is done, we apply β transformations:

$$\beta : 1 + A_1 A_2 \dots A_r \mapsto A_1 \dots A_{s-1}(1 + A_s)A_{s+1} \dots A_r \text{ for } r \geq 2$$

In the above, we also require that the A_i are not themselves products. Note that a β transformation also cannot decrease the value; in fact, it necessarily increases the value, as none of the A_i can have value 1. Initially none of them can have value 1 as otherwise an α transformation could be applied, and β transformations preserve the property that 1s are never multiplied by anything. Note also they preserve the property that nothing but 1s are ever added to anything.

This, too, terminates: Each β transformation reduces the number of 1s which are added to products. Once it terminates, we must either have an expression of the form $1 + 1 + \ldots + 1$, or an expression of the form $A_1 \ldots A_r$ where each A_i is of the form $1 + \ldots + 1$; we will represent the sum of n 1s by $[n]$. In the former case, we will consider it as the product of one such. So we now apply γ transformations:

$$\gamma : [m + n] \mapsto [m][n] \text{ for } m, n \geq 2$$

Each of these increases the value, unless $m = n = 2$, in which case it remains the same. Again, this terminates, as each $[m]$ can only be split up finitely many times. As only $[2]s$, $[3]s$, and 1s cannot be split up, this then leaves us with something of the form $[2]^a[3]^m$. There can be no 1s as 1s were never multiplied after β transformations were applied, and γ preserves this property; so there can be a 1 only if we started this phase with our entire expression being 1, requiring $k = 1$. So finally we apply δ transformations:

$$\delta : [2]^3 \mapsto [3]^2$$

As the number of [2]s decreases each time, this terminates, leaving us with $[3]^{k/3}$ if $k \equiv 0 \pmod 3$, $[2][3]^{(k-2)/3}$ if $k \equiv 2 \pmod 3$, and $[2]^2[3]^{(k-4)/3}$ if $k \equiv 1 \pmod 3$. Since this can be made with k 1s, but is also a constant upper bound on any number made with k 1s, it is equal to $E(k)$. □

3. Algorithm for Determining $E_r(k)$

To determine $E_r(k)$, we use the transformations described above. We can consider a graph of all possible $+, \cdot$, 1-expressions with k ones, with edges labeled with one of the four types of transformations above, pointing from each expression to its image under such. We then start from the node representing $E(k)$ — specifically, the form of it given above, if $k \equiv 1 \pmod 3$. Consider the of tree of backwards paths from $E(k)$ in this tree, with the restriction that the inverse transformations that make up each path must be of decreasing type (i.e. no δ after γ, etc.) Call this the $E(k)$ tree. Because doing transformations of increasing type eventually reaches $E(k)$, this tree contains every possible expression at least once. We will maintain a portion of this tree; initially, it will contain only the $E(k)$ vertex. We will also maintain a set of "marked" vertices; it will initially consist of this single vertex. The set of marked vertices after i markings have occurred represents a set of i highest-valued vertices.

We will then repeat the following three steps:

(1) Expand the tree at the marked vertices (to be described shortly).
(2) Picked an unmarked vertex of highest value, and mark it.
(3) Check if there are r distinct values among the marked vertices; if there are, return the lowest. If not, repeat these three steps.

Since all vertices will eventually be enumerated this way (as will be described shortly), this will return $E_r(k)$ unless there are not r distinct values that can be made with k ones, in which case eventually there will be no unmarked vertices, and the second step will fail.

Now we must describe the process of expansion. By "expanding" the tree at a vertex, I mean to in some way compute that vertex's children in the $E(k)$ tree and to add them to the tree. The simplest way would be to compute all of that vertex's children, unless this has already been done for that vertex. Because each child of a vertex has value at most that of its parent vertex, we need not consider a vertex for marking as one of the highest until we have considered its parent, so this algorithm will find the r highest values and return $E_r(k)$.

However, we need a way to expand on vertices that we can abstract later, so we must modify this process slightly. Rather than generating all the β^{-1} and α^{-1} children at once, we will generate them in stages. For this, we need to examine β^{-1} and α^{-1} in more detail. (Generating all δ^{-1} and γ^{-1} children is not something there is much to say about.)

β pulls down a 1, so to do a β^{-1} we must pull up a 1; however, a β can also combine two products into one, so in order to consder everything that can β to a given expression, we must not just pull up 1s from products, but rather split up products in all possible ways and pull up 1s from these. Allowing for permutation of the factors, this may be described by

$$\beta^{-1} : (1 + A_1)A_2 \dots A_{s-1}A_s \mapsto (1 + A_1A_2 \dots A_j)A_{j+1} \dots A_s$$

Note that also we may assume A_1 is not 1 and is not a product; in the first case, β would not be applied to the result, and in the second case, a β would not result in the same thing we started with. Define a "stage j β^{-1}" to be a β^{-1} where the number of factors in the resulting product-plus-one is j. Note that the value of the expression above is $A_1 \dots A_s + A_{j+1} \dots A_s$, so if the factor which is having a 1 pulled up is kept constant, any β^{-1} with the set of factors being grouped with it is a subset of the multiset $\{A_2, \dots, A_j\}$ has larger value. In particular, each stage $j + 1$ β^{-1} is less than some stage j β^{-1}. Note also a stage-1 β^{-1} has no effect at all, and so we can also exclude those, as the result would again not truly β to what we started with (none of these transformations can leave an expression the same).

α^{-1} comes in two types, again allowing for permutations of factors:

$$\alpha^{-1} : 1 + A \mapsto A$$
$$\alpha^{-1} : A_1 \dots A_s \mapsto A_1 \dots A_j + A_{j+1} \dots A_s$$

Strictly speaking, the first type does not preserve the number of 1s used, but it is easier than keeping track of multiplication by 1. Note that we do not need to allow the more general $I + A \mapsto A$ where val$I = 1$, as the only way such an I can occur is as a product of 1s, meaning that using the simplification above, it will appear as just a 1. Also strictly speaking, the second sort should deal with products of two things, not arbitrarily many, but this will be necessary for the abstraction later. Call the first sort a "type 0 α^{-1}" and the second sort a "stage j α^{-1}", where we make the assumption that $A_1 \dots A_j \leq A_{j+1} \dots A_s$. (I.e., by symmetry, we can assume the smaller side is on the left, and it is the smaller side we count by; if the two are equal in value, either can go on the left, and we will consider these as distinct possibilities.)

Noting that for $x, y > 0, xy = c$, $x+y$ gets larger as x and y get farther apart, i.e. as the smaller one gets smaller, we see that each α^{-1} with smaller side a subset of the multiset $\{A_1, \ldots, A_j\}$ has larger value; hence, we conclude as above that each stage $j+1$ α^{-1} is smaller than some stage j α^{-1}.

Finally, we note that for a given product, any non-type-0 α^{-1} operating on that product yields a value less than any β^{-1} operating on that product, and any type-0 α^{-1} operating on any of the 1s below it but above another product yields a value less than some β^{-1} operating on that product. The latter is obvious, as a if the α^{-1} erases a given 1, some β^{-1} erases that same 1 but also moves it up. As for the former, assume A_1 is the factor with the 1 being pulled up, ignore which side is smaller in the α^{-1} — the left side will simply be whichever side has the factor with the 1 being pulled up — compare $A_1 \ldots A_j + A_{j+1} \ldots A_s$ with $A_1 \ldots A_s - A_2 \ldots A_s$, the latter of which is less than the value of any β^{-1} on that product. It suffices to prove the inequality $ab + c < (a-1)bc$; here, $a = \mathrm{val}A_1, b = \mathrm{val}A_2 \ldots A_j, c = \mathrm{val}A_{j+1} \ldots A_s$. Writing this as $c + ab + bc < abc$, we see from AM-GM that $c+ab+bc \leq 3\sqrt[3]{ab^2c^2}$, and $3\sqrt[3]{ab^2c^2} < abc$ if $a^2bc > 27$. In this case, since none of the factors have value 1, we can say $b, c \geq 2$, and furthermore we can say $a \geq 3$, since if it had value 2, we would not be pulling up a 1 from it — see above. Hence $a^2bc \geq 36$ and the inequality holds.

Using this, we modify the algorithm as follows: Instead of just keeping track at each node of whether or not it is marked and whether or not its children have been generated, we keep track of whether it is marked; a boolean for whether its δ^{-1} and γ^{-1} children, and certain type 0 α^{-1} children, have been generated; a boolean for whether all type 0 α^{-1} children have been generated; an integer for what stage β^{-1} it should do next, which is initially 2, as we do not do stage 1 β^{-1}s as noted earlier; and an integer for what stage (non-type-0) α^{-1} it should do next, which begins at 1.

Then, to expand on a node, we do the following: First, if its δ^{-1} and γ^{-1} children have not been generated, do so, and mark that this has been done. (As always, do not generate children inappropriate to the type of inverse transformation used to reach the node, as these are not actually in the $E(k)$ tree in the first place.) Next, look at its children generated so far via β^{-1}; suppose it is at stage j. If there are no stage $j-1$ children because $j = 2$, or if at least one stage $j-1$ child is marked, generate the stage j children and increment the β^{-1} stage. Otherwise, there is no need to consider these children yet. Then, if the node is not itself a product, generate all type 0 α^{-1} children where the modification is made above the first product; we do not need to mark this separately, the variable marking whether δ^{-1} and γ^{-1} children have been generated, together with the fact that the node is not a product, will suffice to record this. Then, check whether the current β^{-1} stage is more than the minimum length of a product (whose factors are not themselves prodcuts) used in the expression. If so, it is time to start applying α^{-1}; generate all type 0 α^{-1} children if this has not already been done, and mark that this has been done. Finally, assuming the above condition for applyiing α^{-1} is met, look at the non-type-0 α^{-1} children generated so far; suppose it is at stage i. If there are no stage $i-1$ children because $i = 1$, or if at least one stage $i-1$ child is marked, generate the the stage i children and increment the stage.

This completes the description of the algorithm. Since every possible expression is reached by some path, and any path is checked once all greater paths are marked

(as this would require the marking of both its parent, and any other children of that parent that might precede it in order of generation), it follows that the algorithm is correct.

4. Abstracting the Algorithm

Now we abstract the algorithm given above, to produce a new algorithm that will prove results about $E_r(k)$ for more general k. This algorithm will prove:

Proposition 4.1. *For all $r \in \mathbb{N}$ and each congruence class a modulo 3, there exist $h_{a,r}$ and $K_{a,r}$ such that for all $k \geq K_{a,r}$ with $k \equiv a \pmod 3$, $E_r(k) = h_{a,r}E(k)$.*

The algorithm itself will, given r and a, determine $h \geq K_{a,r}, l_{a,r}, and K_{a,r}$. Then, in the next section, we will further modify the algorithm and use that to determine the general form for $K_{a,r}$ and $h_{k,r}$, and in the section after that, we will use those results to prove the main proposition.

Initially, however, our algorithm will only prove

Lemma 4.2. *For all $r \in \mathbb{N}$ and each congruence class a modulo 3, there exist $h_{a,r} \in \mathbb{Q}$, $l_{a,r} \in \mathbb{Z}$ nonnegative and $K_{a,r} \in \mathbb{N}$ such that for all $k \geq K_{a,r}$ with $k \equiv a \pmod 3$, $E_r(k) = h_{a,r}E(k) + l_{a,r}$.*

That $l_{a,r}$ is always 0 will come later.

In order to abstract the algorithm, we make the following observation: For different $k \neq 1$ that are congruent modulo 3, our starting expression looks much the same, differing only in the number of [3]s available. Let us consider, then, expressions of the same sort as before, only now in addition to $+, \cdot$, and 1, we introduce a new sort of object: an abstract clump of 3s. (Note: From hereon, I will simply use n to denote a sum of n 1s, rather than $[n]$ as previously.) I will denote a full clump of 3s by 3^Z. When doing the operations above, we can consider a clump as a "source" of 3s, that can have arbitrarily many 3s pulled out of it; thus, for instance, a γ^{-1} on 3^Z might result in $6 \cdot 3^{Z-2}$, where here 3^{Z-2} represents a clump with 2 3s pulled out of it; its value is one-ninth that of a full clump 3^Z. However, this procedure would quickly become unworkable if we performed such operations as, say, taking an α^{-1} of 3^Z and getting $3^{Z/2} + 3^{Z/2}$, whatever that would mean. Fortunately, our operations — the same ones as above — only ever extract an amount of 3s from a clump not dependent on the (unknown and arbitrary) number of 3s in it; thus, we can always write our clumps in the form 3^{Z-n}, for some n.

Let us formalize this. Define an abstract k-$+, \cdot, 1$-expression to be an expression using the binary operators $+$ and $\cdot$, the constant 1, and constants 3^{Z-n} for all integers n (negatives will come up, though only -1s), which are collectively referred to as "clumps of 3s". Furthermore, we require that such an expression have no more than one clump of 3s. k is not relevant to what can be a k-abstract $+, \cdot, 1$-expression; it is relevant only to their value. The value of such an expression will be an element of $\mathbb{Q}[x]$; it is computed in the same manner as that of a $+, \cdot, 1$-expression, except that 3^{Z-n} has value $3^{-n}x$ if $k \equiv 0 \pmod 3$, $3^{-n}x/2$ if $k \equiv 2 \pmod 3$, and value $3^{-n}x/4$ if $k \equiv 1 \pmod 3$. Here, x represents $E(k)$, as 3^Z is equal to, one half of, or one quarter of $E(k)$ as k is $0, 2$, or 1 modulo 3 respectively. Note the requirement that a given expression contain at most one clump means that the value of a given expression is, in fact, an affine function; furthermore, the constant term must be a nonnegative integer, as any nonintegers must come from the clump, but the clump cannot affect the constant term.

In addition to its value, an abstract k-$+,\cdot,1$-expression also has an associated number of threes. This is the minimum concrete value for Z needed for the expression to make sense, i.e., if the clump is 3^{Z-n} with n nonnegative, the number of threes is n, and the number of threes is 0 if n is negative. However, if the clump of 3s is being added to something rather than multiplied, or if it stands alone as a full expression, we need an additional 3 (unless n is negative) in order for things to have their ordinary value, as we'll need at least 1 three in the clump, as if there were none, we'd evaluate it as 1 when it should be 0, as it's being added. Note that having 3^{Z+n} with n positive would require there to be an additional $3n$ ones beyond those in the initial 3^Z clump, so this case can only occur when $k \equiv 1 \pmod 3$, and in that case 3^{Z+1} is the only way this can occur.

Now, since the values of these expressions are functions rather than constants, we need to say what it means for one value to be greater than another. Naturally, in some cases, one function will always be greater than the other for positive x; more generally, given any two, we can ask which one is eventually greater (if they're not equal), simply by comparing the coefficients. However, it is useful here to introduce another notion. We will say that one expression is "practically greater" than another if it is greater in all concrete cases that can actually arise, i.e., if it is greater for all $x = E(k)$ where k is such that it supplies enough threes to meet the minimum requirements for each. (k that is 0 mod 3 supplies $k/3$ threes; k that is 2 mod 3 supplies $(k-2)/3$ threes; $k \neq 1$ that is 1 mod 3 supplies $(k-4)/3$ threes, though we can consider 1 as supplying -1 threes if we want.) Note that this notion depends on the actual expressions, not just their values. Note being practically greater in particular implies being eventually greater.

We can now define the unpruned abstract $E(k)$ tree as the tree of paths starting from 3^Z, $2 \cdot 3^Z$, or $4 \cdot 3^Z$ as appropriate (the tree will only depend on k modulo 3) via the transformations above with the restrictions above, where 3^{Z-n} is considered equivalent to $3^{Z-n-m}3^m$ for any $m \geq 0$, and where every β^{-1} or non-type-0 α^{-1} is stage j for some j, where value for the purposes of α^{-1} is compared by eventual value, and where any clump is not considered as one factor, but as infinitely many. We will define the pruned abstract $E(k)$ tree to be the same, but with never applying α^{-1}s at a product containing a clump. (Note this makes irrelevant some of the requirements about α^{-1} in the unpruned tree.)

The unpruned tree has two important properties: Firstly, if we pick any $k \neq 1$ of the appropriate congruence class mod 3, we can extract its number of 3s and plug this in for Z, simply dropping the nodes where this is less than the minimum number of 3s required, or less than the minimum for some ancestor, and this will result in the $E(k)$ tree. Secondly, each node is practically greater than all its children, each stage $j+1$ β^{-1} is practically less than some stage j β^{-1}, each stage $j+1$ α^{-1} is practically less than some stage j α^{-1}, and each non-type-0 α^{-1} at a product and each type-0 α^{-1} immediately below it is practically less than each β^{-1} at that product. In short, the inequalities that make the algorithm above work, still hold, and if we were to specify Z, the algorithm would go exactly the same except possibly for some nodes being excluded, and comparisons not necessarily going the same way.

Note the finiteness requirement made above - each β^{-1} is stage j for some j, similarly for α^{-1}; in other words, there is no splitting of 3^Z into $3^{Z/2} + 3^{Z/2}$ or

anything similar. Of course, such a thing does not even make sense in this framework, as we only allow for clumps of the form 3^{Z-n}. However, it is important to note that such possibilities are, in fact, irrelevant. This follows from the fact that a stage j transformation is always practically more than a stage $j+1$ transformation, so we always want to do stage 1, then stage 2, then stage 3, etc., i.e., the stage when instantiated never depends on the concrete Z, and so we can always assume a constant finite stage in the abstract case. Furthermore, we need never apply an α^{-1} at a product containing a clump, as there, we can apply β^{-1}s of arbitrarily high stage, and we will never reach the point of applying an α^{-1}, so it suffices to use the pruned tree.

Suppose, then, that we apply this same algorithm, using eventual greatness for comparison. When the algorithm terminates, all nodes not in the tree we have built are each practically less than some node in this tree, and so are all possibilities with clumps of other forms, which are not even in our abstract $E(k)$ tree at all. Hence our final result, the rth-greatest node in the tree by eventual greatness, is, in fact, eventually the rth-greatest possible number constructible with k ones (for k in the appropriate congruence class), and we have our $h_{k,r}$ and $l_{k,r}$.

In fact, each $l_{k,r}$ will be zero. This is because applying the transformations above will not change the fact that, firstly, the expression as a whole is a product, and secondly, that the unique clump is in this top-level product. This is because the clump starts out in the top-level product, and the only way to make it otherwise would be either through a β^{-1} that pulled the whole thing out of the top-level product — impossible as we only do finite β^{-1}s — or through an α^{-1} at that product. Even if we do an α^{-1} at the product containing the clump due to the crudeness of our algorithm, it cannot be relevant and will never be marked, as we can do β^{-1}s of arbitrarily large stage at such a product.

To find a concrete $K_{k,r}$, we want two conditions. Firstly, K should be large enough to supply enough 3s for at least one node of value $h_{k,r}x + l_{k,r}$, and for at least one node of each of the $r-1$ values eventually greater than it. (The value must itself be constructible; and all those above it must be, or else it would be higher than rth-highest. Note, we will define $T_{k,r}$ to be the minimum number of threes to make that value, and $T'_{k,r}$ to be the smallest K of the appropriate congruence class supplying that many threes. If we really want to find the minimum, we can go out to $r+1$; this will ensure that we have found all the nodes of rth-highest value, and we can check all of them for number of threes required. In the next section, though, we will see that it turns out that the number of threes required never varies among different expressions for the same value for those values that can actually be marked at some point.) Note that none of the marked nodes will have a clump added to anything, so the only case where we will have to add 1 to the number of 3s taken out of the clump is when the expression is of the form 3^{Z-n}.

Secondly, we want K to be large enough that this value is, in fact, less than the $r-1$ that are eventually greater than it; and also large enough that it is greater than the remaining nodes in the tree we have generated which are eventually less than it. We do not need to worry about nodes not in the tree we have generated, as such a node A can be compared to its parent B, which it is practically at most. If A was made by any method other than a type 0 α^{-1}, it requires at least as many 3s as B; so if there are enough 3s for B, A is irrelevant, and if there are not enough 3s for B, A is irrelevant. If A was made by a type 0 α^{-1}, it may require fewer threes

than its parent, but it is always less than B, not just practically at most B, so it is still irrelevant. However, since any node A appearing in the tree which has nonzero constant term will never be marked, and the same comparison to its parent applies, these too are irrelevant. This means we only need to compare against nodes with zero constant term - which means that this restriction in fact imposes no restriction at all.

Therefore, we can take for $K_{a,r}$ the minimum K of the appropriate congruence class needed to make the r eventually-highest values. And because all of these values have constant term 0 and the ordering between them is always strict for $x > 0$, only that particular function can make that number (anything with nonzero constant term will be strictly less at this point, remember) at that point and so this is always the smallest K of that congruence class we can pick.

This proves that the algorithm is correct, and proves Proposition 4.1.

5. Results of the Algorithm

There are, in fact, infinitely many nodes of the each of the abstract E_k trees with value always greater than $(2/3)x$. This can be seen by considering the following families of nodes: For $k \equiv 0 \pmod 3$, $3^{Z-n-1}(1 + 2 \cdot 3^n)$; for other k, multiply by 2 or by 2^2 as appopriate. This yields a value of $(2 \cdot 3^{n+1} + 1)x/3^{n+1} > 2x/3$. Therefore, for all r and k, $h_{r,k} > 2/3$.

Furthermore, for $k \equiv 1 \pmod 3$, we can find infinitely many nodes with value greater than $(3/4)x$; consider the infinite family $3^{Z-n}(1 + 3^{n+1})$, of value $(3^{n+1} + 1)x/4 \cdot 3^n$.

So the question becomes: How low does $h_{r,k}$ go? As it turns out, $l_{r,k}$ is always zero, and $h_{r,k}$ tends to a limit of $2/3$ if $k \equiv 0, 2 \pmod 3$ and $3/4$ if $k \equiv 1 \pmod 3$. We show this by slightly modifying (again) the algorithm described above. Define λ_k to be $2/3$ if $k \equiv 0, 2 \pmod 3$ and $3/4$ if $k \equiv 1 \pmod 3$.

First, here is a list of a few infinite families of nodes that always have value greater than $\lambda_k x$:

(a) $k \equiv 0 \pmod 3$:
  (1) For $n \geq 1, 3^{Z-n-1}2(1+3^n)$ of value $(2 \cdot 3^n + 2)x/3^{n+1}$ requiring $n+1$ threes
  (2) For $n \geq 0, 3^{Z-n-1}(1 + 2 \cdot 3^n)$ of value $(2 \cdot 3^n + 1)x/3^{n+1}$ requiring $n + 1$ threes (including when $n = 0$)

(b) $k \equiv 2 \pmod 3$:
  (1) For $n \geq 1, 3^{Z-n-1}2^2(1 + 3^n)$ of value $(2 \cdot 3^n + 2)x/3^{n+1}$ requiring $n + 1$ threes
  (2) For $n \geq 1, 3^{Z-n-1}4(1+3^n)$ of value $(2 \cdot 3^n + 2)x/3^{n+1}$ requiring $n+1$ threes
  (3) For $n \geq 0, 3^{Z-n-1}2(1 + 2 \cdot 3^n)$ of value $(2 \cdot 3^n + 1)x/3^{n+1}$ requiring $n + 1$ threes (unless $n = 0$, then only 0 are required)
  (4) For $n \geq 0, 3^{Z-n-1}(1 + 2^2 3^n)$ of value $(4 \cdot 3^n + 1)x/(2 \cdot 3^{n+1})$ requiring $n + 1$ threes
  (5) For $n \geq 0, 3^{Z-n-1}(1 + 4 \cdot 3^n)$ of value $(4 \cdot 3^n + 1)x/(2 \cdot 3^(n + 1))$ requring $n + 1$ threes

(c) $k \equiv 1 \pmod 3$:
  (1) For $n \geq 0, 3^{Z-n}(1 + 3^{n+1})$ of value $(3^{n+1} + 1)x/(4 \cdot 3^n)$ requiring n threes

I claim that, in fact, for each k, there are only finitely many nodes of the abstract E_k tree not falling into one of the above families. We prove this algorithmically. We can modify our earlier algorithm as follows. Firstly, when computing children of a

node, we exclude any falling into the infinite families above, except for those that simplify further. (Note that if certain strange infinite families were to be excluded, we might have to further modify the algorithm to ensure that there is nothing lying outside these families or their descendants that it fails to hit because it lies it is at a higher stage than something in the appropriate family, and β_m^{-1} at some stage returns nothing, preventing us from continuing; however, this problem does not actually come up, as can be seen by looking at how the infinite families above can actually be generated, as we will do shortly.) Secondly, instead of marking the node of highest value, we mark all nodes of value greater than $\lambda_k x$. Thirdly, instead of stopping when a given number of values have been generated, we stop when we find no leaves to mark, i.e., when the last expansion generated nothing of value greater than $\lambda_k x$.

As for just how each infinite family can be generated, the first element of each infinite family (except $3^{Z-n-1}(1+2^2 3^n)$) simplifies further, but the rest can only be generated by a β^{-1}. Applying β to these expressions in all possible ways, we see that the ways of generating these are as follows:

(1) For $k \equiv 0 \pmod 3$:
   (a) $3^{Z-n-1}2(1+3^n)$ comes from $3^{Z-2}4 \cdot 2$
   (b) $3^{Z-n-1}(1+2\cdot 3^n)$ comes from $3^{Z-2}4 \cdot 2$ or 3^Z

(2) For $k \equiv 2 \pmod 3$:
   (a) $3^{Z-n-1}2^2(1+3^n)$ comes from $3^{Z-2}4 \cdot 2^2$
   (b) $3^{Z-n-1}4(1+3^n)$ comes from $3^{Z-2}4^2$
   (c) $3^{Z-n-1}2(1+2\cdot 3^n)$ comes from $3^{Z-2}4 \cdot 2^2$ or $3^Z 2$
   (d) $3^{Z-n-1}(1+2^2 3^n)$ comes from $3^{Z-2}4 \cdot 2^2$ or $3^Z 2$
   (e) $3^{Z-n-1}(1+4\cdot 3^n)$ comes from $3^{Z-2}4^2$ or $3^{Z-1}5$

(3) For $k \equiv 1 \pmod 3$:
   (a) $3^{Z-n}(1+3^{n+1})$ comes from $3^Z 4$

And so when expanding on any of these latter expressions, we do not generate anything of the former form. (Note that those that "simplify further" are precisely those that would only be generated by a stage-1 β^{-1}, so we do not have to specifically exclude those cases, as the algorithm already excludes them for us.)

It is not clear a priori that this algorithm will terminate; however, if it does, then that means the expressions it has output of value greater than $\lambda_k x$ are all such expressions aside from those in the excluded families, and their descendants. As it happens, the algorithm, when run, does terminate, for each of the three possible values of k – see the accompanying implementation of this algorithm in Haskell. Furthermore, we can check by hand that all children of the given infinite families, other than those of members which simplify further, are too low in value. Therefore, the only nodes of value greater than $\lambda_k x$ are the infinite families listed above, and the finitely many exceptions returned by the algorithm. Furthermore, as the infinite families above all have $\lambda_k x$ as the infimum of their values, and there are only finitely many others, it follows that as r goes to infinity, $h_{r,k}$ approaches λ_k.

The actual results of the algorithm are as follows:

(a) $k \equiv 0 \pmod 3$:
   (1) 3^Z of value $1x$ requiring 1 three
   (2) $3^{Z-2}2^3$ of value $(8/9)x$ requiring 2 threes
   (3) $3^{Z-2}4 \cdot 2$ of value $(8/9)x$ requiring 2 threes

(4) $3^{Z-4}2^6$ of value $(64/81)x$ requiring 4 threes
(5) $3^{Z-4}4 \cdot 2^4$ of value $(64/81)x$ requiring 4 threes
(6) $3^{Z-4}4^2 2^2$ of value $(64/81)x$ requiring 4 threes
(7) $3^{Z-4}4^3$ of value $(64/81)x$ requiring 4 threes
(8) $3^{Z-3}2^2(1+2^2)$ of value $(20/27)x$ requiring 3 threes
(9) $3^{Z-3}4(1+2^2)$ of value $(20/27)x$ requiring 3 threes
(10) $3^{Z-3}5 \cdot 2^2$ of value $(20/27)x$ requiring 3 threes
(11) $3^{Z-3}5 \cdot 4$ of value $(20/27)x$ requiring 3 threes
(12) $3^{Z-6}2^9$ of value $(512/729)x$ requiring 6 threes
(13) $3^{Z-6}4 \cdot 2^7$ of value $(512/729)x$ requiring 6 threes
(14) $3^{Z-6}4^2 2^5$ of value $(512/729)x$ requiring 6 threes
(15) $3^{Z-6}4^3 2^3$ of value $(512/729)x$ requiring 6 threes
(16) $3^{Z-6}4^4 2$ of value $(512/729)x$ requiring 6 threes
(17) $3^{Z-4}2^3(1+3 \cdot 2)$ of value $(56/81)x$ requiring 4 threes
(18) $3^{Z-4}4 \cdot 2(1+3 \cdot 2)$ of value $(56/81)x$ requiring 4 threes

(b) $k \equiv 2 \pmod 3$:
(1) $3^Z 2$ of value $1x$ requiring 0 threes
(2) $3^{Z-2}2^4$ of value $(8/9)x$ requiring 2 threes
(3) $3^{Z-2}4 \cdot 2^2$ of value $(8/9)x$ requiring 2 threes
(4) $3^{Z-2}4^2$ of value $(8/9)x$ requiring 2 threes
(5) $3^{Z-1}5$ of value $(5/6)x$ requiring 1 threes
(6) $3^{Z-4}2^7$ of value $(64/81)x$ requiring 4 threes
(7) $3^{Z-4}4 \cdot 2^5$ of value $(64/81)x$ requiring 4 threes
(8) $3^{Z-4}4^2 2^3$ of value $(64/81)x$ requiring 4 threes
(9) $3^{Z-4}4^3 2$ of value $(64/81)x$ requiring 4 threes
(10) $3^{Z-3}2^3(1+2^2)$ of value $(20/27)x$ requiring 3 threes
(11) $3^{Z-3}4 \cdot 2(1+2^2)$ of value $(20/27)x$ requiring 3 threes
(12) $3^{Z-3}5 \cdot 2^3$ of value $(20/27)x$ requiring 3 threes
(13) $3^{Z-3}5 \cdot 4 \cdot 2$ of value $(20/27)x$ requiring 3 threes
(14) $3^{Z-6}2^{10}$ of value $(512/729)x$ requiring 6 threes
(15) $3^{Z-6}4 \cdot 2^8$ of value $(512/729)x$ requiring 6 threes
(16) $3^{Z-6}4^2 2^6$ of value $(512/729)x$ requiring 6 threes
(17) $3^{Z-6}4^3 2^4$ of value $(512/729)x$ requiring 6 threes
(18) $3^{Z-6}4^4 2^2$ of value $(512/729)x$ requiring 6 threes
(19) $3^{Z-6}4^5$ of value $(512/729)x$ requiring 6 threes
(20) $3^{Z-4}2^4(1+3 \cdot 2)$ of value $(56/81)x$ requiring 4 threes
(21) $3^{Z-4}4 \cdot 2^2(1+3 \cdot 2)$ of value $(56/81)x$ requiring 4 threes
(22) $3^{Z-4}4^2(1+3 \cdot 2)$ of value $(56/81)x$ requiring 4 threes

(c) $k \equiv 1 \pmod 3$:
(1) $3^Z 2^2$ of value $1x$ requiring 0 threes
(2) $3^Z 4$ of value $1x$ requiring 0 threes
(3) $3^{Z-2}2^5$ of value $(8/9)x$ requiring 2 threes
(4) $3^{Z-2}4 \cdot 2^3$ of value $(8/9)x$ requiring 2 threes
(5) $3^{Z-2}4^2 2$ of value $(8/9)x$ requiring 2 threes
(6) $3^{Z-1}5 \cdot 2$ of value $(5/6)x$ requiring 1 threes
(7) $3^{Z-1}2(1+2 \cdot 2)$ of value $(5/6)x$ requiring 1 threes
(8) $3^{Z-4}2^8$ of value $(64/81)x$ requiring 4 threes
(9) $3^{Z-4}4 \cdot 2^6$ of value $(64/81)x$ requiring 4 threes

(10) $3^{Z-4}4^2 2^4$ of value $(64/81)x$ requiring 4 threes
(11) $3^{Z-4}4^3 2^2$ of value $(64/81)x$ requiring 4 threes
(12) $3^{Z-4}4^4$ of value $(64/81)x$ requiring 4 threes
(13) $3^{Z-2}2^2(1+3\cdot 2)$ of value $(7/9)x$ requiring 2 threes
(14) $3^{Z-2}4(1+3\cdot 2)$ of value $(7/9)x$ requiring 2 threes

We now have our full list of $h_{k,r}$, being simply the $h_{k,r}$ that have come up above, sorted in descending order. Finally, though not relevant to what will follow, we can determine $K_{k,r}$. This requires just the information on numbers of threes needed listed above, as noted earlier. (Note, also, that if we knew the above table of exceptions in advance, we could verify it by hand by checking that the children of each entry were either too small, or already listed (either as an exception or in one of the infinite families).)

To summarize, then, the list $h_{k,r}$ that occur and $T_{k,r}$ for each is:

(A) For $k \equiv 0 \pmod 3$:
  (a) For $n \geq 1$, $(2\cdot 3^n+2)/3^{n+1}$ requiring $n+1$ threes
  (b) For $n \geq 0$, $(2\cdot 3^n+1)/3^{n+1}$ requiring $n+1$ threes
  (c) 64/81 requiring 4 threes
  (d) 512/729 requiring 6 threes
(B) For $k \equiv 2 \pmod 3$:
  (a) For $n \geq 1$, $(2\cdot 3^n+2)/3^{n+1}$ requiring $n+1$ threes
  (b) For $n \geq 0$, $(2\cdot 3^n+1)/3^{n+1}$ requiring $n+1$ threes
  (c) For $n \geq 0$, $(4\cdot 3^n+1)/(2\cdot 3^{n+1})$ requiring $n+1$ threes if $n \neq 0$ and 0 threes if $n = 0$
  (d) 64/81 requiring 4 threes
  (e) 512/729 requiring 6 threes
(C) For $k \equiv 1 \pmod 3, k \neq 1$:
  (a) For $n \geq 0$, $(3^{n+1}+1)/(4\cdot 3^n)$ requiring n threes
  (b) 8/9 requiring 2 threes
  (c) 64/81 requiring 4 threes

Note that, happily, the number of threes required for a given fraction is, with the exception of 1 when $k \equiv 0 \pmod 3$, always exactly the same as the number of threes appearing in its denominator. This also shows that the $K_{k,r}$ resulting from numbers of threes is the smallest such we can pick.

This gives us:

Proposition 5.1. *The table of $h_{k,r}$ and their corresponding $K_{k,r}$ is as follows:*

(1) For $k \equiv 0 \pmod 3$:

r	$h_{k,r}$	$K_{k,r}$
0	1	3
1	8/9	6
2	64/81	12
3	7/9	12
4	20/27	12
5	19/27	12
6	512/729	18
7	56/81	18
8	55/81	18
9	164/243	18
10	163/243	18
$n \geq 6, 2n-1$	$2/3 + 2/3^n$	$3n$
$n \geq 6, 2n$	$2/3 + 1/3^n$	$3n$

(2) For $k \equiv 2 \pmod 3$:

r	$h_{k,r}$	$K_{k,r}$
0	1	2
1	8/9	8
2	5/6	8
3	64/81	14
4	7/9	14
5	20/27	14
6	13/18	14
7	19/27	14
8	512/729	20
9	56/81	20
10	37/54	20
11	55/81	20
12	164/243	20
13	109/162	20
14	163/243	20
$n \geq 6, 3n-3$	$2/3 + 2/3^n$	$3n+2$
$n \geq 6, 3n-2$	$2/3 + 1/(2 \cdot 3^{n-1})$	$3n+2$
$n \geq 6, 3n-1$	$2/3 + 1/3^n$	$3n+2$

(3) For $k \equiv 1 \pmod 3$ (note here, unlike before, we are allowing $k = 1$):

r	$h_{k,r}$	$K_{k,r}$
0	1	1
1	8/9	10
2	5/6	10
3	64/81	16
4	7/9	16
5	41/54	16
$n \geq 4, n+2$	$3/4 + 1/(4 \cdot 3^n)$	$3n+4$

6. Equivalence with the Main Theorem

We can now conclude Theorem 1.3. Observe that for any k (save 1 and 2, which are easily handled) we have $\lambda_k = E(k-1)/E(k)$. (Hence for $k \geq K_{k,r}$, we have $||E_r(k)|| = k$, and $E_r(k)$ is the also r-th highest number with complexity

exactly k.) Now suppose n is of one of the forms listed in Theorem 1.3; checking that $L(n) = ||n||$ is then straightforward. So we just need the converse; suppose $L(n) = ||n|| = k$. Then $n > E(k-1)$ and so $R(n) > \lambda$. Hence there exists some r with $R(n) \geq h_{k,r} > \lambda$. Thus if we let $N = n3^l$ be such that $k + 3l \geq K_{k,r}$, we must have $R(N) = R(n) \in \{h_0, \ldots, h_{r-1}\}$. ($R(N) = R(n)$ because $||n|| = L(n)$, so otherwise we would have $||N|| < L(N)$, which is impossible.) And given this fact it then follows from the above tables that n is of one of the forms listed in Theorem 1.3.

Observe that a consequence of Theorem 1.3 is that if $||3n|| = L(3n)$, then $||n|| = L(n)$.

Suppose now that we already knew Theorem 1.3, and wanted to deduce the above formulae. Taking the $n \neq 1$ satisfying the classification and splitting them up by $||n||$ modulo 3, we get that they are:

- For $||n|| \equiv 0 \pmod 3$:
  - $(2 \cdot 3^m + 1)3^l$ with $R(n) = \frac{2 \cdot 3^m + 1}{3^{m+1}}$
  - $2(3^m + 1)3^l$ $(m \geq 1)$ with $R(n) = \frac{2(3^m+1)}{3^{m+1}}$
  - $64 \cdot 3^l$ with $R(n) = 64/81$
  - $512 \cdot 3^l$ with $R(n) = 512/729$
- For $||n|| \equiv 2 \pmod 3$:
  - $(4 \cdot 3^m + 1)3^l$ with $R(n) = \frac{(4 \cdot 3^m + 1)}{2 \cdot 3^{m+1}}$
  - $2(2 \cdot 3^m + 1)3^l$ with $R(n) = \frac{2(2 \cdot 3^m + 1)}{2 \cdot 3^{m+1}}$
  - $4(3^m + 1)3^l$ $(m \geq 1)$ with $R(n) = \frac{4(3^m+1)}{2 \cdot 3^{m+1}}$
  - $2 \cdot 3^l$ with $R(n) = 1$
  - $128 \cdot 3^l$ with $R(n) = 64/81$
  - $1024 \cdot 3^l$ with $R(n) = 512/729$
- For $||n|| \equiv 1 \pmod 3$, $n \neq 1$:
  - $(3^m + 1)3^l$ $(m \geq 1)$ with $R(n) = \frac{(3^m+1)}{4 \cdot 3^{m-1}}$
  - $32 \cdot 3^l$ with $R(n) = 8/9$
  - $256 \cdot 3^l$ of $R(n) = 64/81$

Now suppose we want to determine the r-th largest number with complexity k. From the above tables, we can determine the r-th largest value of R that occurs for numbers with complexity congruent to k modulo 3. Call this value h; then $hE(k)$ is indeed the r-th largest number with complexity k if and only if all $r+1$ values of R that are at least h and occur for numbers with complexity congruent to k modulo 3, do indeed occur for numbers with complexity k. However we also know that if $L(3n) = ||3n||$, then $L(n) = ||n||$, or in other words, that if $L(n) = ||n||$ and $k \equiv ||n|| \pmod 3$, then n can be written with k ones if and only if $R(n)E(k)$ is an integer. So this simply requires taking k to be large enough for all of these to be integers; hence we can use this to get the get a table of resulting formulae for the r-th largest number with complexity k – and this table precisely matches the ones above. Furthermore, when these formulae are valid, we have that they are also equal to $E_r(k)$, the r-th largest number writable with k ones, because each of them is greater than $E(k-1)$; hence any n greater than such a number which was writable with k ones would also have to have complexity exactly k.

Hence Theorem 1.3 really is just a summary of the above tables.

7. Consequences and Conclusions

Having concluded Theorem 1.3, we can now also conclude Proposition 1.4.

Unfortunately, this method does not seem to generalize beyond finite r, in particular because it's not even clear from this perspective what $E_r(k)$ could even mean for r an infinite ordinal. In order to do that we need to instead look at values of $n/E(||n||)$, as we do in [AZ].

We can also use this to classify the numbers n with $||n|| < 3\log_3 n + 1$; since $3\log_3 n \leq L(n) < 3\log_3 n + 5 - 6\log_3 2$, if $||n|| < 3\log_3 n + 1$, we must have $||n|| = L(n)$. Since we know the complexities of all the numbers n with $||n = L(n)$, it is easy to then check that

Proposition 7.1. *A number n satisfies $||n|| < 3\log_3 n + 1$ if and only if it can be written in one of the following forms:*

- 3^k *for* $k \geq 1$
- $2^a 3^k$ *for* $a \leq 9$
- $5 \cdot 2^a 3^k$ *for* $a \leq 3$
- $7 \cdot 2^a 3^k$ *for* $a \leq 2$
- $19 \cdot 3^k$
- $13 \cdot 3^k$
- $(3^n + 1)3^k$

This can then be used directly as a base case for the main lemma from [AZ]. (As opposed to how it is actually done in [AZ], where Rawsthorne's formula is first used as a base case for the main lemma to bootstrap up to the above list.)

References

[A] J. Arias de Reyna, Complejidad de los números naturales, *Gaceta R. Soc. Mat. Esp.*, **3**(2000) 230–250.

[AZ] H. Altman & J. Zelinsky, Some New Results in Integer Complexity, forthcoming

[MP] K. Mahler & J. Popken, On a maximum problem in arithmetic (Dutch), *Nieuw Arch. Wiskunde*, (3) **1**(1953) 1–15; *MR* **14**, 852e.

[R] Daniel A. Rawsthorne, How many 1's are needed? *Fibonacci Quart.*, **27**(1989) 14–17; *MR* **90b**:11008.

[SS] Srinivas Vivek V. & Shankar B. R., Integer Complexity: Breaking the $\Theta(n^2)$ barrier, *World Academy of Science*, **41**(2008) 690–691