A note on counting independent terms in asymptotic expressions of computational complexity

Fabiano de S. Oliveira · Valmir C. Barbosa

Abstract The field of computational complexity is concerned both with the intrinsic hardness of computational problems and with the efficiency of algorithms to solve them. Given such a problem, normally one designs an algorithm to solve it and sets about establishing bounds on the algorithm’s performance as functions of the relevant input-size descriptors, particularly upper bounds expressed via the big-oh notation. In some cases, however, especially those arising in the fields of experimental algorithmics and optimization, one may have to resort to performance data on a given set of inputs in order to figure out the algorithm’s big-oh profile. In this note, we are concerned with the question of how many candidate expressions may have to be taken into account in such cases. We show that, even if we only considered upper bounds given by polynomials, the number of possibilities could be arbitrarily large for two or more descriptors. This is unexpected, given the available body of examples on algorithmic efficiency, and underlines the importance of careful and meticulous criteria. It also serves to illustrate the many facets of the big-oh notation, as well as its counter-intuitive twists.

Keywords Analysis of algorithms · Experimental algorithmics and optimization · Computational complexity · Asymptotics · Big-oh notation
1 Introduction

The amount of time or space required by an algorithm is a function of the quantities informing the size of an input to it, henceforth referred to as its variables, and is normally expressed via the well-known big-oh notation in order to highlight the algorithm’s asymptotic behavior. It often happens for such a function to depend on more than one variable. This occurs routinely in the case of graph algorithms, but clearly can also be expected to happen in a variety of other situations, for example when algorithmic building blocks are put together, heuristically or otherwise, and give rise to a complex optimization procedure. In the general case of \( V \) variables, say \( x_1, \ldots, x_V \), we say that \( f(x_1, \ldots, x_V) = O(g(x_1, \ldots, x_V)) \) if there exist positive constants \( c, x_{1,0}, \ldots, x_{V,0} \) such that \( f(x_1, \ldots, x_V) \leq cg(x_1, \ldots, x_V) \) for all \( x_1, \ldots, x_V \geq 0 \) satisfying at least one of the inequalities \( x_1 \geq x_{1,0}, \ldots, x_V \geq x_{V,0} \).

This definition allows for any number of functions to be in the role of \( g(x_1, \ldots, x_V) \) for the same \( f(x_1, \ldots, x_V) \), but a big-oh expression will be as informative to the analysis being carried out as \( g(x_1, \ldots, x_V) \) can be made concise. Conciseness here is to be understood in the sense of irreducibility, which we define as follows. First, let a term be the product of single-variable functions. For example, if \( x \) and \( y \) are variables, then \( x^2, 2xy \), and \( y^2 \) are all terms. Given the \( k \) terms \( T_1, \ldots, T_k \), we say that term \( T_i \) is independent with respect to the sum \( S = T_1 + \cdots + T_k \) if it is not asymptotically bounded from above in proportion to \( S - T_i \), that is, if \( T_i = O(S - T_i) \) does not hold. We say that \( S \) is irreducible if all of \( T_1, \ldots, T_k \) are independent. Continuing with the example, \( x^2 + y^2 \) is irreducible but \( x^2 + 2xy + y^2 \) is not.

In this brief note we concern ourselves with functions that, like \( S \), can be written as a sum of terms. Our goal is to answer a specific question motivated by the following practical application. Suppose we are given the executable code for some program, along with the list of variables affecting its performance, but no further information (no source code, no time or space function, not even their big-oh forms). Suppose further that, before putting such code to use, we are tasked with estimating a bound on its time (or space) function as concisely as possible, via the big-oh notation, over a given range of the variables. If we were allowed to restrict our search to those time functions that can be expressed as a sum of terms, then knowing beforehand how many terms there can be in a concise representation thereof would be of great help. The question that interests us is then, how many terms can an irreducible sum-of-terms function have?

Variations on this question lie at the heart of several of the efforts that have aimed to provide practitioners with software tools to peer into executable code in order to gain insight pertaining to computational complexity. Some of these efforts have targeted the empirical characterization of computational complexity, seeking either to uncover fundamental trends \([1]\) or to come up with worst-case inputs to justify bounds known beforehand \([2]\). Others have concentrated on profiling the code for bottleneck detection (cf. the work of Zaparanuks and Hauswirth \([3]\) and of Coppa, Demetrescu, and Finocchi \([4]\), as well as their references on the broader area of experimental algorithmics and optimization). In all cases, choices have had to be made regarding a set of functional forms to work with.
2 One variable, and beyond

Figuring out how many terms an irreducible sum of terms can have becomes easier if we make further assumptions, now regarding the nature of the single-variable factors that make up a term. If the sum-of-terms function in question is itself a function of a single variable, then the further assumption is quite reasonable and refers to restricting each of those factors to be one of the functions that commonly appear in algorithmic analysis: polynomial, polylogarithmic, logarithmic, and exponential. This given, the question is answered quite simply, because in this case no sum of at least two terms constitutes an irreducible function. More precisely, any function arising naturally when analyzing a single-variable algorithm belongs to the class \( \mathcal{L} \) of the so-called logarithmic-exponential functions \([5,6]\), defined to comprise the following:

- \( f(x) = a \), for any real constant \( a \);
- \( f(x) = x \);
- \( f(x) - g(x) \), if \( f(x), g(x) \in \mathcal{L} \);
- \( e^{f(x)} \), if \( f(x) \in \mathcal{L} \);
- \( \ln f(x) \), if \( f(x) \in \mathcal{L} \) and, for some constant \( x_0 \), \( f(x) > 0 \) for all \( x \geq x_0 \).

As it turns out, any two functions \( f(x), g(x) \in \mathcal{L} \) necessarily satisfy \( f(x) = O(g(x)) \) or \( g(x) = O(f(x)) \), whence it follows that any sum of at least two terms from \( \mathcal{L} \) is not irreducible.

Beyond this simple case of a single variable, and as remarked earlier, multiple variables are a common occurrence in some contexts. The most notable case in point, that of graph algorithms, is such that its time and space functions often depend on both \( n \) and \( m \), the graph’s numbers of vertices and edges, respectively. In fact, some of the best algorithms for numerous graph problems have time functions bounded by sums of two or three terms, often depending on more variables than simply \( n \) and \( m \), as shown in Table 1. What we see in the table are counts of how many algorithms, as reported in a portion of Schrijver’s treatise \([7]\), have time-function bounds with a certain number of terms on a certain number of variables. For example, 65 of the reported bounds on two variables have one single term and 14 have two, but none has three or more terms.

One might then wonder if two is the maximum number of terms whose sum is irreducible in the case of two variables. That, however, is not the case. To see this, consider the function \( f(x, y) = x^2 + y^2 + (xy)^{3/2} \). The first term in this function is

| Number of variables | Number of terms | 1 | 2 | 3 |
|---------------------|----------------|---|---|---|
| 1                   | 52             | 0 | 0 |   |
| 2                   | 65             | 14| 0 |   |
| 3                   | 48             | 33| 2 |   |
| 4                   | 5              | 14| 0 |   |
| 5                   | 3              | 2 | 0 |   |
| Total               | 173            | 63| 2 |   |
independent, since it does not hold that \( x^2 = O(y^2 + (xy)^{3/2}) \), as seen by simply fixing \( y = c \) for any positive constant \( c \). The case of the second term is entirely analogous. As for the third term, set \( x = y \) to conclude that \( (xy)^{3/2} = O(x^2 + y^2) \) does not hold either. So \( f(x, y) \) is irreducible despite having more than two terms.

We may loosen the conjecture a little, and set about testing whether three, not two, is the maximum number of terms in an irreducible sum of terms, but once again with a negative result: the four-term sum \( f(x, y) = x^{485}y + x^{477}y^4 + x^{459}y^8 + x^{243}y^{32} \), for example, is irreducible. This can be seen by setting, for each term in order, \( x = y^{0.7}, x = y^{0.31}, x = y^{0.21}, \) and \( x = y^{0.05} \). So conjecturing further seems to have become a little too daunting and perhaps we should back off and consider the possibility that a function may in fact comprise an arbitrarily large number of terms and still be irreducible. Next we prove that this is the case when all the single-variable factors that go in a term are rising power laws, even if the exponents in these laws are all positive integers (i.e., the sum-of-terms function is a polynomial).

### 3 An arbitrarily large number of terms

Let \( f(x, y) \) be such that

\[
f(x, y) = \sum_{i=1}^{k} x^{a_i} y^{b_i}.
\]

If \( f(x, y) \) is to be an irreducible function, then clearly no two of the \( a_i \)'s may equal each other, and similarly no two of the \( b_i \)'s, since in either case one of the two terms involved would not be independent. Thus, it must be possible to arrange the \( a_i \)'s into an increasing or decreasing sequence, and similarly the \( b_i \)'s, but once again for the sake of irreducibility one of the sequences must be increasing while the other is decreasing. We assume \( 0 < a_1 < \ldots < a_k \) and \( b_1 > \ldots > b_k > 0 \).

For each \( i \), we concentrate on valuations of \( x \) and \( y \) such that \( x = y^{z_i} \) for some \( z_i > 0 \), so the \( i \)th term of \( f(x, y) \) is independent if and only if \( a_i z_i + b_i > a_j z_i + b_j \) for all \( j \neq i \). So arguing for the irreducibility of \( f(x, y) \) requires that we find \( a_1, \ldots, a_k, b_1, \ldots, b_k, \) and \( z_1, \ldots, z_k \) such that

\[
a_i z_i + b_i > a_j z_i + b_j
\]

for all \( i \) and all \( j \neq i \). Our approach will be to determine the \( a_i \)'s and the \( b_i \)'s in such a way as to automatically establish an interval within which to choose the value of each \( z_i \). For \( 1 \leq i, j \leq k \), the constraints on this choice are

\[
z_i < (b_i - b_j)/(a_j - a_i) \text{ for } i < j,
\]

\[
z_i > (b_i - b_j)/(a_j - a_i) \text{ for } j < i.
\]

A convenient, alternative way to view this condition is to define the ratio

\[
r(i, j) = \frac{b_i - b_j}{a_j - a_i},
\]
for which it holds that \( r(i, j) = r(j, i) \). Using this equivalence whenever \( j < i \) allows the condition to be written as

\[
\max_{1 \leq j < i} r(j, i) < z_i < \min_{i < j \leq k} r(i, j) \text{ for } 1 < i < k,
\]

in addition to \( z_1 < \min_{1 < j \leq k} r(1, j) \) and \( z_k > \max_{1 \leq j < k} r(j, k) \). So in order for \( f(x, y) \) to be irreducible, we must ensure that

\[
\max_{1 \leq j < i} r(j, i) < z_i < \min_{i < j \leq k} r(i, j) \text{ for } 1 < i < k.
\]

**Theorem 1** Given any positive integer \( k \), \( f(x, y) \) is irreducible with

\[
a_i = a_1(2 - \alpha^{i-1}) \quad \text{and} \quad b_i = b_1\beta^{i-1},
\]

where \( 0 < \alpha < \beta < 1 - \alpha < 1 \).

**Proof** We first write \( r(i, j) \) as

\[
r(i, j) = \frac{b_1}{a_1} \left( \frac{\beta}{\alpha} \right)^{i-1} \frac{1 - \beta^{j-i}}{1 - \alpha^{j-i}}
\]

and note that

\[
r(i, j + 1) = r(i, j) \frac{h_{i,j}(\beta)}{h_{i,j}(\alpha)},
\]

where

\[
h_{i,j}(t) = \frac{1 - t^{j-i+1}}{1 - t^{j-i}},
\]

with first derivative given by

\[
h_{i,j}'(t) = \frac{t^{j-i-1}((j-i)(1-t) - t(1-t^{j-i}))}{(1-t^{j-i})^2}.
\]

For \( i < j < k \), we have \( h_{i,j}'(t) > 0 \) for \( t \in (0, 1) \), since using

\[
u_{i,j}(t) = (j-i)(1-t) - t(1-t^{j-i}),
\]

we have \( u_{i,j}'(t) = (j-i+1)(t^{j-i} - 1) < 0 \) and \( u_{i,j}(1) = 0 \), hence \( u_{i,j}(t) > 0 \). It follows that \( h_{i,j}(\beta)/h_{i,j}(\alpha) > 1 \), and therefore, \( r(i, j) < r(i, j + 1) \).

Moreover, we also have \( r(i - 1, k) < r(i, i + 1) \) for \( 1 < i < k \), which can be seen by noting that

\[
r(i - 1, k) < \lim_{j \to \infty} r(i - 1, j) < r(i, i + 1),
\]
where
\[ \lim_{j \to \infty} r(i - 1, j) = \frac{b_1}{a_1} \left( \frac{\beta}{\alpha} \right)^{i-2} \]
and
\[ r(i, i + 1) = \frac{b_1}{a_1} \left( \frac{\beta}{\alpha} \right)^{i-1} \frac{1 - \beta}{1 - \alpha}, \]
since \( \beta(1 - \beta)/\alpha(1 - \alpha) > 1 \) for \( \alpha < \beta < 1 - \alpha \).

The desired inequality follows, since \( \max_{1 \leq j < i} r(j, i) = r(i - 1, i) < r(i - 1, k) < r(i, i + 1) = \min_{i < j \leq k} r(i, j). \)

For the \( a_i \)'s and \( b_i \)'s of Theorem 1, following the proof reveals a clear recipe to determine the \( z_i \)'s: choose \( z_1 < r(1, 2), z_k > r(k - 1, k) \), and the remaining ones to satisfy
\[ r(i - 1, k) < z_i < r(i, i + 1) \]
for \( 1 < i < k \).

4 Integral exponents and constrained valuations

Our argument in the previous section for the irreducibility of the function \( f(x, y) = \sum_{i=1}^{k} x^{a_i} y^{b_i} \) relied on the particular valuation for \( x \) and \( y \) that sets \( x = y^{z_i} \). All we have required of the exponents \( a_i, b_i, \) and \( z_i \) is that they be positive, which leaves plenty of room for them to be nonintegers. This is not a problem in itself, and in fact there exist landmark algorithms that run in time bounded by the input size raised to an irrational power.\(^1\) However, when it comes to the analysis of practical computer algorithms, in most cases we expect the exponents \( a_i \) and \( b_i \) to be positive integers.

Additionally, depending on the domain at hand, it is often the case that the valuation tying the \( x \) and \( y \) variables together should only employ values for \( z_i \) that are bounded from above by a constant. This is the case of the already noted domain of graph algorithms, in which a graph’s numbers \( n \) of vertices and \( m \) of edges are such that \( m = O(n^2) \). In this section we show that there continue to exist exponents for which \( f(x, y) \) is irreducible even if we constrain \( a_i \) and \( b_i \) to be positive integers (hence \( f(x, y) \) to be a polynomial), and likewise if \( z_i \) is constrained to be no greater than a constant.

**Theorem 2** Given any positive integer \( k \), \( f(x, y) \) is an irreducible polynomial with \( a_i = a_1(2 - \alpha^{i-1}) \) and \( b_i = b_1 \beta^{i-1} \), where \( \alpha = p_\alpha/q_\alpha \) and \( \beta = p_\beta/q_\beta \) are rational constants such that \( 0 < \alpha < \beta < 1 \), \( \alpha < 1 \), \( a_1 = q_\alpha^{k-1} \), and \( b_1 = q_\beta^{k-1} \).

**Proof** Proceed exactly as in the proof of Theorem 1, then observe that all \( a_i \)'s and \( b_i \)'s are positive integers. \(\Box\)

---

\(^1\) One of the well-known examples of this in the single-variable case is Strassen’s algorithm for multiplying two \( n \times n \) matrices, whose running time is \( O(n^{\log_2 7}) \) [8].
Counting independent terms

Table 2 For \( f(x, y) \) as in Theorem 2, with \( k = 6 \), \( p_\alpha = 1 \), \( q_\alpha = 3 \), \( p_\beta = 1 \), and \( q_\beta = 2 \), the cell at position \( j, i \) gives the value of \( a_j z_i + b_j \)

| \( j \) | \( a_j \) | \( b_j \) | \( i: z_i \) |
|-------|-------|-------|-------|
| 1     | 243   | 32    | 1:0.05  |
| 2     | 405   | 16    | 2:0.14  |
| 3     | 459   | 8     | 3:0.21  |
| 4     | 477   | 4     | 4:0.31  |
| 5     | 483   | 2     | 5:0.47  |
| 6     | 485   | 1     | 6:0.70  |

Values are highlighted in a bold typeface whenever \( j = i \), indicating the maximum on each column (i.e., for fixed \( i \) and all \( j \))

Fig. 1 \( z_i \) values for \( 1 \leq i \leq k \) and \( r(i, j) \) values for \( i < j \leq k \), in the same setting as in Table 2

A detailed example illustrating Theorem 2 is given in Table 2 and Fig. 1 for \( k = 6 \). Table and figure provide different takes on the exact same setting, the former highlighting the integral nature of the exponents in \( f(x, y) \) as well as each \( a_j z_i + b_j \) as a maximum over all \( j \), the latter highlighting each \( z_i \) and \( r(i, j) \) for \( j > i \). That is, each entry set in a bold typeface in Table 2 occupies a diagonal cell, say at position \( i, i \), and gives the value of \( a_i z_i + b_i \). This value, in turn, can be seen to be strictly greater than \( a_j z_i + b_j \) for all \( j \neq i \), which is precisely the condition given in Sect. 3 for the \( i \)th term of \( f(x, y) \) to be independent, and therefore for \( f(x, y) \) to be irreducible if true of every \( i \). This, as we have seen, can be restated in terms of there being nonempty intervals bounded by certain \( r(i, j) \)’s inside which the \( z_i \)’s are to be found. This alternative is spelled out at the end of Sect. 3 and is illustrated in Fig. 1 for the example in question.

Theorem 3 Given any positive integer \( k \) and a constant \( c > 0 \), there exist constants \( \alpha \) and \( \beta \) such that \( 0 < \alpha < \beta < 1 - \alpha < 1 \) for which \( f(x, y) \) is irreducible with \( z_i < c \), \( a_i = a_1(2 - \alpha^{i-1}) \), and \( b_i = b_1 \beta^{i-1} \), where \( b_1/a_1 < c \).
Proof Proceed exactly as in the proof of Theorem 1, then impose \( r(k - 1, k) < c \) to obtain an upper bound on the allowable values of \( k \):

\[
k < 2 + \log_{\beta/\alpha} \frac{a_1}{b_1} \left( \frac{1 - \alpha}{1 - \beta} \right) c.
\]

The result follows from noting that, for \( b_1 < c a_1 \),

\[
\lim_{\beta/\alpha \to 1^+} \log_{\beta/\alpha} \frac{a_1}{b_1} \left( \frac{1 - \alpha}{1 - \beta} \right) c = \infty.
\]

Therefore, choosing \( \alpha \) and \( \beta \) to be arbitrarily close to each other accommodates any desired \( k \). \( \square \)

5 Concluding remarks

The analysis of computer algorithms via the big-oh notation is an essential part of most activities within computer science and optimization, including both theoretical studies and the myriad of applications to which people working in these fields devote themselves. In the great majority of situations the algorithm that is being considered is known at some level of detail, so that obtaining big-oh expressions for how much time or space it consumes, though far from being a simple task, is at least a well-defined one. In this paper, by contrast, we started out with a “black-box” version of an algorithm, that is, a version that we can only analyze by running it on a given set of inputs to make measurements of how much of the necessary resources the algorithm spends.

Faced with the task of discovering big-oh expressions bounding such resource usage, and limiting our search to polynomial-like functions of the relevant variables, we found that, in principle, an automated procedure to carry out the task might have to consider functions comprising an unbounded number of terms. This is surprising, given all the accumulated knowledge on so many algorithms to solve so many different problems, but we feel that it sheds additional light on the big-oh notation itself, especially when we consider the subtle pitfalls that sometimes motivate a deeper examination of its use [9].

We close with two final remarks. The first is that our conclusions can be easily extended to the case of more variables, recursively by simply fusing together all current variables through appropriate valuations whenever a new variable is added to the pool. The second remark is that, even though for this work we found motivation in the analysis of computer algorithms, the big-oh notation is in fact of much wider interest and applicability, providing a crucial tool whenever it is asymptotics, not exact figures, that matter. This occurs in several other fields within mathematics, as well as in science and engineering.

Acknowledgements The authors acknowledge partial support from Conselho Nacional de Desenvolvimento Científico e Tecnológico, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ), and a FAPERJ BBP grant.
References

1. Goldsmith, S.F., Aiken, A.S., Wilkerson, D.S.: In Proc. ESEC-FSE 2007, pp. 395–404 (2007)
2. Burnim, J., Juvekar, S., Sen, K.: In Proc. ICSE 2009, pp. 463–473 (2009)
3. Zaparanuks, D., Hauswirth, M.: In Proc. PLDI 2012, pp. 67–76 (2012)
4. Coppa, E., Demetrescu, C., Finocchi, I.: IEEE T. Software Eng. 40:1185 (2014)
5. Hardy, G.H.: Orders of infinity: The ‘infinitärrechn’ of Paul du Bois-Reymond, 2nd edn. Cambridge University Press, Cambridge (1924)
6. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete mathematics: a foundation for computer science, 2nd edn. Addison-Wesley, Reading (1989)
7. Schrijver, A.: Combinatorial optimization: polyhedra and efficiency, vol A–C. Springer, Berlin (2003)
8. Strassen, V.: Numer. Math. 13:354 (1969)
9. Regan, K.W.: A polynomial growth puzzle (2015). Gödel’s Lost Letter and P = NP. https://rjlipton.wordpress.com/2015/09/12/a-polynomial-growth-puzzle/