www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 11 November 2017, Page No. 23089-23094
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i11.15

Growth Rates in Algorithm Complexity: the Missing Link

Dana Vrajitoru[1]

[1]Indiana University South Bend, Department of Computer and Information Sciences
1700 Mishawaka Ave, South Bend, IN 46634, USA

Abstract:
In this paper, we propose an explicit measure for the growth rate of an algorithm complexity function. This measure complements the usual time or space complexity analysis of algorithms and can fill a gap in the understanding of the asymptotic notation and thus, provide educational benefits. First, we discuss some properties of the growth measure, such as its behavior with respect to linear operators. Second, we analyze its connection to the asymptotic complexity notations and discuss its implications.

Keywords: algorithm complexity, growth rate, asymptotic notations, education**.**

1. Introduction

The asymptotic algorithm behavior notations of O, Ω, Θ, and corresponding o and ω, are commonly used to describe the time and space complexity of algorithms. They are known as the Landau family of notations [1]. Big O, called the Bachman-Landau notation, was first introduced by P. Bachman in 1892 [2]. Big Ω and Θ, as they are used now, were defined by D. Knuth in 1976 [3], and were based on the Hardy-Littlewood notations [4]. In this paper, we will use the common algorithm complexity definitions for these notations, as found in [5] and [6].

Analysis of algorithms is among the most difficult topics in the computer science curriculum. A typical challenge for the students consists of understanding the need for the constant in the definition of the asymptotic notations and how to choose it. A source of confusion is the fact that even though we use the equality notation, $O(f(n))$ represents a set or family of functions. It is also difficult to grasp that if an algorithm is $O(n)$, then it is also $O(n^2)$, but we only state the smallest such family as the algorithm's order of complexity.

The usual language used with these notations posits that they describe the growth of a function, or the growth of an algorithm effort. We often explain to students that the execution time is not that important in itself, because it can depend on the platform. Thus, in our analysis we want to know how the execution time will change when the problem grows, especially by a given factor such as 2 or 10. It is what we call the *growth* of the function.

In the expert literature as well as in informal complexity discussions, it is often said that if $f(n) = O(g(n))$, then the function f grows at most as fast as the function g for n large enough. It is even fairly common to say that the *growth rate* of the function f is less than or equal to the growth rate of the function g, such as in [7]. Currently, there is no formal definition of the term of "growth rate". The asymptotic notations themselves represent only an indirect measure of the growth of a function. In this study, we propose a direct measure of it.

The term "growth rate" is used in connection to the time or space complexity of algorithms, as well as in other situations related to economics or biology. In some cases, such as [8], it is used to represent $f(n+1)/f(n)$. Thus, for an algorithm of complexity $O(2^n)$, the constant 2 is called the growth rate in this context. It has also been used in other studies [9] to illustrate the evolution of processor speed and Linux kernel size from year to year. Some authors simply use it interchangeably with the term complexity of an algorithm, or of a data structure [10].

Thus, in this paper, we propose to introduce a direct measure of the growth of a function, or the growth of the complexity of an algorithm with the size of the problem. We also analyze the connection between the direct growth rate and the asymptotic behavior notations. Our goal is to give the idea of growth rate a formal definition that can be used both in complexity analysis and as a pedagogical tool.

When the growth rate or the complexity analysis are mentioned, the notion of comparing $f(n)$ with $f(10n)$ or $f(2n)$ is often present [7], [11]. Other authors, such as in [12], inquire about the size of a problem that can be solved by various algorithms in a given amount of time, as for example, in one second. We explore this idea further in our definition of the growth rate function.

The idea is also implicitly present in algorithm scalability discussions [13]. Oftentimes, when an argument for an algorithm being scalable is made, the performance measures are presented on a logarithmic scale. For example, in [14], Figure 9 shows the performance of an image retrieval algorithm for a size of the database going from 10k to 100k and then to 1M. In [15], the run time performance of the algorithm is presented on a logarithmic scale over the number of processors being used. It seems that for scalability purposes, it is important to know what will happen when a parameter in the algorithm is multiplied by a constant.

The paper is organized the following way. Section 2 defines the growth rate function and provides some properties for it. Section 3 analyzes the connection between the comparison of growth rates for two functions and their asymptotic relationship. Section 4 discusses the implications of the theorems in this paper both in terms of algorithms complexity

and in terms of pedagogical uses. The paper ends with some conclusions.

2. Growth Rate Measure

In this section, we define our growth measure for a complexity function and examine its properties.

Definition. Let $f(n)$ be a positive function defined on positive integers and let $k > 1$ be a real number. Then we define the growth rate function

$$R_k(f,n)=\frac{f(kn)}{f(n)}$$

Thus, we measure the growth of an algorithm by checking how much its complexity grows when the size of the problem increases by a factor. We can call the parameter k in this definition the *growth factor*. In the above definition, we assume that the function f has an extension defined on all real numbers. Practically, if that is not the case, and if k is not an integer, we can apply the floor to kn in the definition. We also mark k with an underscore instead of using it as a normal parameter because we usually consider it a constant.

Table 1 shows the growth rates for a variety of common functions. This table illustrates how the measure can be used as a pedagogical tool. Here, the growth rate can be seen to increase clearly from one family of functions to the next.

Table 1. Growth rate function examples

Function	Growth Rate
$f(n) = c$ (a constant)	$R_k(f,n) = 1$
$f(n) = \lg(n)$	$R_k(f,n) = 1 + \frac{\lg(k)}{\lg(n)} = 1+o(1)$
$f(n) = \sqrt{n}$	$R_k(f,n) = \sqrt{k}$
$f(n) = n$	$R_k(f,n) = k$
$f(n) = 2n + 1$	$R_k(f,n) = k + \frac{1-k}{2n+1} \sim k$
$f(n) = n\sqrt{n}$	$R_k(f,n) = k\sqrt{k}$
$f(n) = n\lg(n)$	$R_k(f,n) = k + \frac{k\lg(k)}{\lg(n)} \sim k$
$f(n) = n^p$	$R_k(f,n) = k^p$
$f(n) = 2^n$	$R_k(f,n) = 2^{(k-1)n}$
$f(n) = n!$	$R_k(f,n) = (n+1)(n+2)\dots(kn) = P_n^{kn}$
$f(n) = n^n$	$R_k(f,n) = k^{kn} \cdot n^{(k-1)n}$

For educational purposes, students can benefit from being introduced to this table for some value of \$k\$ such as 2, 5, or 10. As stated in the introduction, sometimes students struggle to grasp the need and importance of the constant in the \$O\$ definition. This table shows directly how a function grows faster than another without having to select such a constant. It can help them acquire an intuitive sense of how the most common functions are ordered by complexity.

Another important element that easily emerges from this table is the distinction between polynomial functions and the others. Thus, the growth rate of a polynomial is bound by constants if k a constant, while for larger functions, it depends on n.

Theorem 1. If $f(n)$ and $g(n)$ are positive functions defined on positive integers, then for any real $k > 1$

a) If $h(n) = g(n) / f(n)$ is a monotonous ascending function, then

$$R_k(f, n) \le R_k(g, n).$$

b) If $h(n) = g(n) / f(n)$ is a monotonous descending function, then

$$R_k(f, n) \ge R_k(g, n).$$

Proof.

a) If $h(n) = g(n) / f(n)$ is a monotonous ascending function, then for any positive integers p, q such that $p < q$, $h(p) \le h(q)$ or $g(p) / f(p) \le g(q) / f(q)$. Then we can write

$$R_k(f,n) \le R_k(g,n) \Leftrightarrow \frac{f(kn)}{f(n)} \le \frac{g(kn)}{g(n)}$$

$$\Leftrightarrow \frac{g(n)}{f(n)} \le \frac{g(kn)}{f(kn)} \Leftrightarrow h(n) \le h(kn)$$

The last inequality is true because h is a monotonous ascending function and $k>1$ and $n>0$ mean that $n < kn$.

b) The proof is similar to a). ■

Theorem 2. If $f(n)$ and $g(n)$ are positive functions defined on positive integers, then for any real $k > 1$

a) $R_k(cf, n) = R_k(f, n)$, for any real positive constant c,

b) $R_k(fg, n) = R_k(f, n)\, R_k(g, n)$,

c) $R_k(f+g, n) < R_k(f, n) + R_k(g, n)$,

d) $R_{k\,i}(f, n) = R_k(f, n)\, R_i(f, kn)$, for any real constant $i > 1$.

Proof.

a) $$R_k(cf,n)=\frac{cf(kn)}{cf(n)}=\frac{f(kn)}{f(n)}$$

This makes the measure R_k invariant to the function being multiplied by a constant.

b) $$R_k(fg,n)=\frac{f(kn)g(kn)}{f(n)g(n)}=\frac{f(kn)}{f(n)}\frac{g(kn)}{g(n)}=R_k(f,n)R_k(g,n).$$

This makes the measure R_k distributive with respect to function multiplication.

c) $$R_k(f+g,n)=\frac{f(kn)+g(kn)}{f(n)+g(n)}=\frac{f(kn)}{f(n)+g(n)}+\frac{g(kn)}{f(n)+g(n)}=$$

$$\frac{f(kn)}{f(n)}\frac{f(n)}{f(n)+g(n)}+\frac{g(kn)}{g(n)}\frac{g(n)}{f(n)+g(n)}.$$

Since both $f(n)$ and $g(n)$ are positive functions, we can write that

$$\frac{f(n)}{f(n)+g(n)}<1 \quad \text{and} \quad \frac{g(n)}{f(n)+g(n)}<1.$$

This means that

$$R_k(f+g,n)<\frac{f(kn)}{f(n)}+\frac{g(kn)}{g(n)}=R_k(f,n)+R_k(g,n).$$

Thus, the measure R_k satisfies the triangular property.

d) $$R_{ki}(f,n)=\frac{f(kin)}{f(n)}=\frac{f(kin)}{f(kn)}\frac{f(kn)}{f(n)}=R_i(f,kn)R_k(f,n)$$ ■

Theorem 2 a) can be used in classroom to explain the presence of the constant in the definition of the \$O\$ notation.

This theorem tells us that if we multiply a function by a constant, its growth rate remains the same. Thus, in terms of comparing growth rates, it does not make a difference if we compare a function f with a function g or with cg, where c is a constant.

3. Connection with Asymptotic Notations

In this section, we will examine the connection between the function R_k and the complexity notations of O, Ω, Θ, and o.

Theorem 3. If $f(n)$ and $g(n)$ are two positive functions of positive integers such that $R_k(f, n) \le R_k(g, n)$ for any $k > 1$, $n > 0$, then either $f(n) = o(g(n))$, or $f(n) = \Theta(g(n))$\$. In both cases, $f(n) = O(g(n))$.

Proof.

Let us use the notation $h(n) = g(n) / f(n)$. The first part of the proof consists of showing that the positive function $h(n)$ is monotonous ascending.

Let $n < m$ be two positive integers. Then if we take k to be equal to m / n, then $k > 1$ and $kn = m$. Then it must be true that $R_k(f, n) \le R_k(g, n)$ (f, n) . This means that

$$\frac{f(kn)}{f(n)} \le \frac{g(kn)}{g(n)} \Leftrightarrow \frac{f(m)}{f(n)} \le \frac{g(m)}{g(n)} \Leftrightarrow \frac{g(n)}{f(n)} \le \frac{g(m)}{f(m)} \Leftrightarrow h(n) \le h(m)$$

This proves that the function h is monotonous ascending.

If this is the case, then $h(n) \ge h(1)$\$ for any $n \ge 1$\$. If we denote by $c_1 = h(1)$, then we have that

$$h(n) \ge c_1 \Leftrightarrow \frac{g(n)}{f(n)} \ge c \Leftrightarrow f(n) \le c_1 g(n).$$

In particular, this means that $f(n) = O(g(n))$.

Next, $h(n)$ being a positive function that is monotonous ascending, either $h(n)$ converges to infinity, or there exists a constant c_2 and a number n_0 such that $h(n) \le c_2$ for any $n > n_0$.

If $\lim_{n\to\infty} h(n) = \infty$, then

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \infty \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

which means that \$f(n) = o(g(n))\$.

In the second case, $h(n) \le c_2$ for any $n > n_0$ means that

$$\frac{g(n)}{f(n)} \le c_2 \Leftrightarrow g(n) \le c_2 f(n)$$

which tells us that $g(n) = O(f(n))$ and $f(n) = \Omega(g(n))$. Since we already know that $f(n) = O(g(n))$, this means that $f(n) = \Theta(g(n))$. ■

Theorem 4. If $f(n) = \Theta(g(n))$, then $R_k(f, n) = \Theta(R_k(g, n))$ for any $k > 1$.

Proof.

If $f(n) = \Theta(g(n))$, then there exists $c_1, c_2 > 0$ such that

$$c_1 g(n) \le f(n) \le c_2 g(n).$$

Then we can write

$$c_1 g(kn) \le f(kn) \le c_2 g(kn) \text{ and}$$

$$\frac{1}{c_2}\frac{1}{g(n)} \le \frac{1}{f(n)} \le \frac{1}{c_1}\frac{1}{g(n)}$$

By multiplying the two inequalities we get

$$\frac{c_1}{c_2}\frac{g(kn)}{g(n)} \le \frac{f(kn)}{f(n)} \le \frac{c_2}{c_1}\frac{g(kn)}{g(n)} \Leftrightarrow$$

$$\frac{c_1}{c_2} R_k(g,n) \le R_k(f,n) \le \frac{c_2}{c_1} R_k(g,n) \Rightarrow$$

$$R_k(f, n) = \Theta(R_k(g, n)) \quad ■$$

Theorem 5. Let $f(n) = A(n) + h(n)$, where $A(n)$ and $h(n)$ are positive functions such that $h(n) = o(A(n))$. Then

$$R_k(f, n) \sim R_k(A, n).$$

Proof.

$$R_k(f,n) = \frac{A(kn)+h(kn)}{A(n)+h(n)} = \frac{A(kn)}{A(n)+h(n)} + \frac{h(kn)}{A(n)+h(n)} =$$

$$\frac{A(kn)}{A(n)}\frac{1}{1+\frac{h(n)}{A(n)}} + \frac{h(kn)}{h(n)}\frac{h(n)}{A(n)}\frac{1}{1+\frac{h(n)}{A(n)}} =$$

$$R_k(A,n)\frac{1}{1+\frac{h(n)}{A(n)}} + R_k(h,n)\frac{h(n)}{A(n)}\frac{1}{1+\frac{h(n)}{A(n)}}.$$

We know that $h(n) = o(A(n))$, which means that $\lim_{n\to\infty} h(n) / A(n) = 0$. From this, we deduce that

$$\lim_{n\to\infty} \frac{1}{1+\frac{h(n)}{A(n)}} = 1$$

which leads to $R_k(f, n) \sim R_k(A, n) + R_k(h, n)\, h(n) / A(n)$.

Now let's compute the limit of $R_k(A, n)$ divided by the expression on the right-hand side when n goes to infinity.

$$\lim_{n\to\infty} \frac{R_k(A,n)}{R_k(A,n)+R_k(h,n)\frac{h(n)}{A(n)}} = \lim_{n\to\infty} \frac{1}{1+\frac{R_k(h,n)}{R_k(A,n)}\frac{h(n)}{A(n)}} =$$

$$\lim_{n\to\infty} \frac{1}{1+\frac{h(kn)}{A(kn)}} = 1$$

because $f(n) = o(A(n))$ implies that $\lim_{n\to\infty} h(kn) / A(kn) = 0$.

Finally, transitivity of the asymptotic notation gives us the conclusion of the theorem. ■

Here is an example. Let $f(n) = 2n^2 + 3n + 5$, where $A(n) = 2n^2$ and $h(n) = 3n + 5$. When we compute the rate functions for f and for A we get

$$R_k(f,n) = k^2 \quad \text{and} \quad R_k(f,n) = k^2 + (1-k)\frac{3kn+5(k+1)}{2n^2+3n+5}$$

Since the second term in $R_k(f, n)$ converges to 0 as n goes to infinity, $R_k(A, n) \sim R_k(f, n)$ indeed.

Here is a second example where $R_k(h, n)$ is not a simple

function of k. Let $f(n) = 3^n + 2^n$, where $A(n) = 3^n$ and $h(n) = 2^n$. When we compute the growth rate function for A and we apply the same reasoning as in the proof for Theorem 5, we get

$$R_k(A, n) = 3^{(k-1)n} \quad \text{and}$$

$$R_k(f, n) \sim 3^{(k-1)n} + 2^{(k-1)n}\,(2/3)^n = 3^{(k-1)n} + (2^k/3)^n$$

We can see that for $k > \log_2(3)$, the second term converges to infinity when n goes to infinity.

However, if we compute the limit of $R_k(3^n, n)$ divided by this last expression, we get

$$\lim_{n\to\infty} \frac{3^{(k-1)n}}{3^{(k-1)n} + \left(\frac{2^{k-1}}{3}\right)^n} = \lim_{n\to\infty} \frac{1}{1 + \left(2/3\right)^{(k-1)n}} = 1$$

so even in this case, $R_k(A, n) \sim R_k(f, n)$.

So far, the theorems that we have proved seem to show that our measure R_k behaves pretty well and might be a contender to the asymptotic notations that can describe the complexity of an algorithm in a simpler way. The remaining of this section will show some arguments to the contrary.

Preliminary Observation. Let f and g be positive functions of positive integers such that $f(n) < g(n)$ for n large enough. It is possible that $R_k(f, n_a) > R_k(g, n_a)$ for some particular value n_a.

Let us develop this inequality:

$$R_k(f, n_a) > R_k(g, n_a) \Leftrightarrow \frac{f(kn_a)}{f(n_a)} > \frac{g(kn_a)}{g(n_a)} \Leftrightarrow$$

$$f(kn_a)\frac{g(n_a)}{f(n_a)} > g(kn_a)$$

From $f(n) < g(n)$ we know that $g(n_a) / f(n_a)\} > 1$.

Let us assume that the two functions can be extended to real numbers and that $f(x) < g(x)$ for any real number x large enough. Figure 1 illustrates visually the value of $f(kn_a)\, g(n_a) / f(n_a)$ in connection to the other involved values.

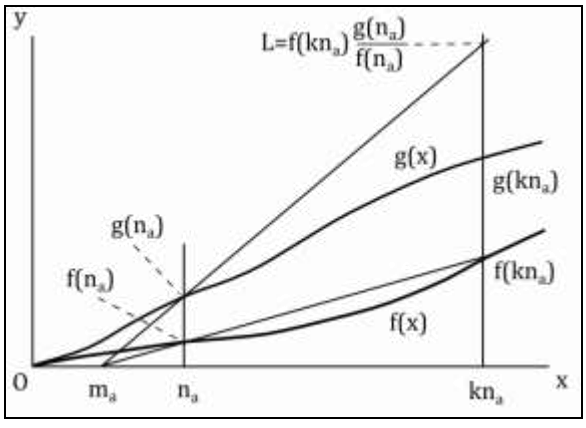

Figure 1. Growth rates for the functions f and g

If we draw a straight line through $f(n_a)$ and $f(kn_a)$, as long as these values are not equal to each other, it will intersect the Ox axis in a point m_a. Then if we draw another line connecting m_a and $g(n_a)$, it intersects the vertical line going up from kn_a at the vertical value $g(n_a) / f(n_a)$. This comes from the triangle similarity: since the two vertical lines are parallel, if we denote by L the vertical coordinate of the top corner, the following equalities must be true:

$$\frac{f(n_a)}{f(kn_a)} = \frac{f(n_a - m_a)}{kn_a - m_a} = \frac{g(n_a)}{L}$$

which we can solve for L to find the value mentioned above.

Incidentally, this also allows us to compute the value $\displaystyle

$$m_a = n_a - \frac{n_a(k-1)f(n_a)}{f(kn_a) - f(n_a)}$$

From this image we can see that there is a whole interval of possible values for $g(kn_a)$\$ that keep it larger than $f(kn_a)$ without going over L. For all of these values, $R_k(g, n_a) < R_k(f, n_a)$.

This observation suggests that the function f could be consistently less than the function g, but its growth rate could be larger than that of g in a good number of points.

Theorem 6. If $f(n)$ and $g(n)$ are positive functions of positive integers, then $f(n) = O(g(n))$ does not imply that $R_k(f, n) \le R_k(g, n)$ for n large enough and for any positive real number k. Thus,

a) there exist pairs of such functions f and g such that for some values of k, $R_k(f, n) > R_k(g, n)$ for an infinite number of values of n;

b) there exist pairs of such functions f and g such that both f and g are monotonous ascending, and such that for some values of k, $R_k(f, n) > R_k(g, n)$ for an infinite number of values of n;

c) there exist pairs of such functions f and g such that $f(n) = o(g(n))$ and that for some values of k, $R_k(f, n) > R_k(g, n)$ for an infinite number of values of n.

Proof.

a) Let

$$f(n) = \begin{cases} n & \text{if } n = 2m+1 \\ n^2 & \text{if } n = 2m \end{cases}$$

and $g(n) = n^2$.

The function f is linear for odd numbers and quadratic for even numbers. It is easily clear that $f(n) = O(g(n))$. The growth rate for the function g is $R_2(g, n) = k^2$ for any $k > 0$, as shown in Table 1.

Let $k = 2$. Then the growth rate for the function f for odd numbers is

$$R_2(f, 2m+1) = \frac{f(4m+2)}{f(2m+1)} = \frac{(4m+2)^2}{2m+1} = 4(2m+1)$$

Thus, $R_2(f, 2m+1) > R_2(g, 2m+1) = 4$ for any $m \ge 1$. Furthermore, $R_2(f, 2m+1)$ is not bound by any constant times $R_2(g, 2m+1)$, nor by any constant in general.

As a note, the function $f(n)$ is neither $\Theta(g(n))$, nor $o(g(n))$.

b) Let us denote by $t_m = 4^{2^{m+1}}$ the sequence of powers of 4 where the exponent is a power of 2. Note that $t_m^2 = t_{m+1}$.

$$f(n)=\begin{cases} n^2 & \text{if } n = t_m \\ {t_m}^2 + n = t_{m+1} + n & \text{if } t_m < n < t_{m+1} \end{cases}$$

and $g(n) = n^2$. The function is quadratic in the numbers of the sequence t_m. In between them, the function grows linearly. Again, it's quite easy to see that $f(n)= O(g(n))$.

Thus, the function f is quadratic for numbers that increase in a quadratic fashion, and it grows linearly in between these values. The function is monotonic ascending.

Let $k = 2$. We are going to examine the growth rate for $n = ½\ t_{m+1}$.

$$R_2\left(f, ½ t_{m+1}\right) = \frac{{t_{m+1}}^2}{{t_m}^2 + ½ t_{m+1}} = \frac{t_{m+2}}{\frac{3}{2} t_{m+1}} = \frac{2}{3} t_{m+1}$$

Just like for the function f at point a), $R_2(f, ½\ t_{m+1}) > R_2(g, ½\ t_{m+1}) = 4$ for any $m \geq 1$. The growth rate of this function is also not bound by any constant.

c) Let $f(n)$ be either the function from the proof of a) or of b). If we take $g(n) = n^3$, then $f(n) = o(g(n))$ in both cases. $R_2(g, n) = 8$, while for both of these functions, the value of R_2 for the set of numbers in the previous proofs grows with n, so eventually it will be larger than $R_2(g, n)$ and than any constant times $R_2(g, n)$. ■

Theorems 3 and 6 show us that the big O notation is more general than the growth rate function. Thus, if the growth rate of f is less than the growth rate of g, then $f = O(g)$, but the other way around is not always true. Even the condition $f = o(g)$ is not sufficient for the growth rates to be in order.

4. Discussion and Future Work

An observation we can draw from the theorems presented here is that saying that if $f(n) = O(g(n))$, then the function g grows faster than the function f is fine, but not that it grows at a faster *rate*.

In a vast majority of the functions observed from analyzing the complexity of algorithms, when $f(n) = O(g(n))$, the growth rate of f is lower than the growth rate of g. Theorem 6 shows that this is not necessarily the case because the functions are monotonous. In many cases, the fraction $f(n)/g(n)$ can be shown to be monotonous, which connects to the grows rates through Theorem 1. A direction of future research would be to find more sufficient conditions for the growth rate of a function to be larger than that of another one.

Furthermore, the function used in the proof for Theorem 6 a) can be used pedagogically as an example where the growth rate simply fails to describe the behavior of a function. This function is clearly bound by two polynomials, but its growth rate in some circumstances is much larger than that of either polynomial. Even more, it is not a constant, which makes it larger than any polynomial. This can be used in classroom as argument explaining why the growth rate is not sufficient.

Another educational argument in favor of the asymptotic notations is the fact that some algorithms do not always have a single function describing their number of operations. For example, the linear search of a value in an array can require only a constant number of operations if the target is at the beginning of the array and a linear function of the array size if the target is not in the array. It is hard to express the growth rate of this algorithm as a whole. The O notation is more appropriate in this case.

As stated in several places, the growth rate measure can be used as an educational tool, to help students understand various aspects of algorithms complexity more easily. A direction of future development will be to study the impact that the idea of growth rate can have on teaching complexity analysis. It is important, though, to establish the mathematical foundations of this measure first, which is the purpose of this paper.

5. Conclusions

In this paper, we introduced an explicit measure for the growth rate for algorithm complexity functions, $R_k(f, n)$, defined as the rate between $f(kn)$ and $f(n)$, where n is the size of the problem and k the growth factor.

In Section 2, we formally defined the measure R_k and showed that it is invariable with respect to constant multiplication, similarly to the asymptotic notations. The growth measure is distributive with respect to function multiplication, and satisfies the triangular property with respect to function addition.

In Section 3, we showed that if the growth rate of a function is less than or equal to the growth rate of another, then the first function is O of the second. The opposite connection is not as straightforward, though. If a function is Θ of another one, then their growth rates are also Θ of each other. More generally, the growth rate of a function is asymptotic to the growth rate of its largest term. This echoes the similar property of the Θ notation.

However, a function can be O of another one, but its growth rate can be larger than the growth rate of the second function for some value of the growth factor k and for an infinite number of values of n. In some of these cases, the first function is neither o of the second, nor Θ of the second. However, even the o notation does not guarantee a smaller growth rate.

For most of the functions observed in analysis of algorithms, the O notation is tied to a smaller growth rate. Theorem 6 cautions us that this is not true for any pair of functions. It is better to use the o or Θ notations instead of O when possible.

Finally, the growth rate shows potential of use as a pedagogical tool in teaching complexity of algorithms and the paper has highlighted several ideas that can improve the teaching effectiveness of this difficult topic.

References

1. E. Landau, “Handbuch der Lehre von der Verteilung der Primzahlen”, Leipzig: B. G. Teubner, 1909.
2. P. Bachman, “Analytische Zahlentheorie”, Leipzig: B. G. Teubner, 1892.
3. D. Knuth, “Big Omicron and Big Omega and Big Theta”, in SIGACT News, pp. 18–24, 1976.
4. G. H. Hardy and J. E. Littlewood, “Some Problems of Diophantine approximation”, in Acta Mathematica, p. 225, 1914.
5. D. Vrajitoru and W. Knight, “Practical Analysis of Algorithms”, Theoretical Computer Science Series, Springer, 2014.
6. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, “Introduction to Algorithms”, MIT Press, 3 ed., 2009.
7. M. A. Weiss, “Data Structures and Problem Solving Using C++”, Pearson, 2nd ed., 1999.

8. M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom, "On Problems as Hard as CNF-SAT", in ACM Transactions on Algorithms, 12, 2016.
9. L. Yu, "Coevolution of Information Ecosystems: a Study of the Statistical Relations Among the Growth Rates of Hardware, System Software, and Application Software", in ACM SIGSOFT Software Engineering Notes, 36, 2011.
10. L. Babai, "The Growth Rate of Vertex-transitive Planar Graphs", in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, 11, 5-7 January 1997, pp. 564–573.
11. A. Drozdek, "Data Structures and Algorithms in C++", Cengage, 4th ed., 2012.
12. M. Goodrich, R. Tamassia, and D. Mount, "Data Structures and Algorithms in C++", Wiley, 2nd ed., 2011.
13. W. Pak and Y. Choi, "High Performance and High Scalable Packet Classification Algorithm for Network Security Systems", in IEEE Trans. Dependable Security Computing, 14, pp. 37–49, 2017.
14. D. Nistér and H. Stewénius, "Scalable Recognition with a Vocabulary Tree", in Proceedings of CVPR, 2006, pp. 2161–2168.
15. J. Strassburg and V. Alexandrov, "On Scalability Behaviour of Monte Carlo Sparse Approximate Inverse for Matrix Computations", in Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '13, New York, NY, USA, 2013, ACM, pp. 6:1–6:8.

Author Profile

Dr. Vrajitoru has obtained her title of Doctor of Sciences from the University of Neuchatel, Switzerland, in 1997. She is currently an associate professor of computer science at Indiana University South Bend. Her research interest include intelligent systems, evolutionary computation, computer graphics, and scientific visualization.