

51. When a list of elements is in close to the correct order, would it be better to use an insertion sort or its variation described in Exercise 50?
52. Use the greedy algorithm to make change using quarters, dimes, nickels, and pennies for
- 87 cents.
 - 49 cents.
 - 99 cents.
 - 33 cents.
53. Use the greedy algorithm to make change using quarters, dimes, nickels, and pennies for
- 51 cents.
 - 69 cents.
 - 76 cents.
 - 60 cents.
54. Use the greedy algorithm to make change using quarters, dimes, and pennies (but no nickels) for each of the amounts given in Exercise 52. For which of these amounts does the greedy algorithm use the fewest coins of these denominations possible?
55. Use the greedy algorithm to make change using quarters, dimes, and pennies (but no nickels) for each of the amounts given in Exercise 53. For which of these amounts does the greedy algorithm use the fewest coins of these denominations possible?
56. Show that if there were a coin worth 12 cents, the greedy algorithm using quarters, 12-cent coins, dimes, nickels, and pennies would not always produce change using the fewest coins possible.
57. Use Algorithm 7 to schedule the largest number of talks in a lecture hall from a proposed set of talks, if the starting and ending times of the talks are 9:00 A.M. and 9:45 A.M.; 9:30 A.M. and 10:00 A.M.; 9:50 A.M. and 10:15 A.M.; 10:00 A.M. and 10:30 A.M.; 10:10 A.M. and 10:25 A.M.; 10:30 A.M. and 10:55 A.M.; 10:15 A.M. and 10:45 A.M.; 10:30 A.M. and 11:00 A.M.; 10:45 A.M. and 11:30 A.M.; 10:55 A.M. and 11:25 A.M.; 11:00 A.M. and 11:15 A.M.
58. Show that a greedy algorithm that schedules talks in a lecture hall, as described in Example 7, by selecting at each step the talk that overlaps the fewest other talks, does not always produce an optimal schedule.
- *59. a) Devise a greedy algorithm that determines the fewest lecture halls needed to accommodate n talks given the starting and ending time for each talk.
b) Prove that your algorithm is optimal.

Suppose we have s men m_1, m_2, \dots, m_s and s women w_1, w_2, \dots, w_s . We wish to match each person with a member



of the opposite gender. Furthermore, suppose that each person ranks, in order of preference, with no ties, the people of the opposite gender. We say that a matching of people of opposite genders to form couples is **stable** if we cannot find a man m and a woman w who are not assigned to each other such that m prefers w over his assigned partner and w prefers m to her assigned partner.

60. Suppose we have three men m_1, m_2 , and m_3 and three women w_1, w_2 , and w_3 . Furthermore, suppose that the preference rankings of the men for the three women, from highest to lowest, are $m_1: w_3, w_1, w_2$; $m_2: w_1, w_2, w_3$; $m_3: w_2, w_3, w_1$; and the preference rankings of the women for the three men, from highest to lowest, are $w_1: m_1, m_2, m_3$; $w_2: m_2, m_1, m_3$; $w_3: m_3, m_2, m_1$. For each of the six possible matchings of men and women to form three couples, determine whether this matching is stable.

The **deferred acceptance algorithm**, also known as the **Gale-Shapley algorithm**, can be used to construct a stable matching of men and women. In this algorithm, members of one gender are the **suitors** and members of the other gender the **suitees**. The algorithm uses a sequence of rounds; in each round every suitor whose proposal was rejected in the previous round proposes to his or her highest ranking suitee who has not already rejected a proposal from this suitor. A suitee rejects all proposals except that from the suitor that this suitee ranks highest among all the suitors who have proposed to this suitee in this round or previous rounds. The proposal of this highest ranking suitor remains pending and is rejected in a later round if a more appealing suitor proposes in that round. The series of rounds ends when every suitor has exactly one pending proposal. All pending proposals are then accepted.

- Write the deferred acceptance algorithm in pseudocode.
- Show that the deferred acceptance algorithm terminates.
- * Show that the deferred acceptance always terminates with a stable assignment.
- Show that the problem of determining whether a program with a given input ever prints the digit 1 is unsolvable.
- Show that the following problem is solvable. Given two programs with their inputs and the knowledge that exactly one of them halts, determine which halts.
- Show that the problem of deciding whether a specific program with a specific input halts is solvable.

3.2 The Growth of Functions

Introduction

In Section 3.1 we discussed the concept of an algorithm. We introduced algorithms that solve a variety of problems, including searching for an element in a list and sorting a list. In Section 3.3 we will study the number of operations used by these algorithms. In particular, we will estimate the number of comparisons used by the linear and binary search algorithms to find an element in a sequence of n elements. We will also estimate the number of comparisons used by the

bubble sort and by the insertion sort to sort a list of n elements. The time required to solve a problem depends on more than only the number of operations it uses. The time also depends on the hardware and software used to run the program that implements the algorithm. However, when we change the hardware and software used to implement an algorithm, we can closely approximate the time required to solve a problem of size n by multiplying the previous time required by a constant. For example, on a supercomputer we might be able to solve a problem of size n a million times faster than we can on a PC. However, this factor of one million will not depend on n (except perhaps in some minor ways). One of the advantages of using **big- O notation**, which we introduce in this section, is that we can estimate the growth of a function without worrying about constant multipliers or smaller order terms. This means that, using big- O notation, we do not have to worry about the hardware and software used to implement an algorithm. Furthermore, using big- O notation, we can assume that the different operations used in an algorithm take the same time, which simplifies the analysis considerably.

Big- O notation is used extensively to estimate the number of operations an algorithm uses as its input grows. With the help of this notation, we can determine whether it is practical to use a particular algorithm to solve a problem as the size of the input increases. Furthermore, using big- O notation, we can compare two algorithms to determine which is more efficient as the size of the input grows. For instance, if we have two algorithms for solving a problem, one using $100n^2 + 17n + 4$ operations and the other using n^3 operations, big- O notation can help us see that the first algorithm uses far fewer operations when n is large, even though it uses more operations for small values of n , such as $n = 10$.

This section introduces big- O notation and the related big- Ω and big- Θ notations. We will explain how big- O , big- Ω , and big- Θ estimates are constructed and establish estimates for some important functions that are used in the analysis of algorithms.

Big- O Notation

The growth of functions is often described using a special notation. Definition 1 describes this notation.

DEFINITION 1

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. [This is read as “ $f(x)$ is big-oh of $g(x)$.”]

Remark: Intuitively, the definition that $f(x)$ is $O(g(x))$ says that $f(x)$ grows slower than some fixed multiple of $g(x)$ as x grows without bound.

The constants C and k in the definition of big- O notation are called **witnesses** to the relationship $f(x)$ is $O(g(x))$. To establish that $f(x)$ is $O(g(x))$ we need only one pair of witnesses to this relationship. That is, to show that $f(x)$ is $O(g(x))$, we need find only *one* pair of constants C and k , the witnesses, such that $|f(x)| \leq C|g(x)|$ whenever $x > k$.

Note that when there is one pair of witnesses to the relationship $f(x)$ is $O(g(x))$, there are *infinitely many* pairs of witnesses. To see this, note that if C and k are one pair of witnesses, then any pair C' and k' , where $C < C'$ and $k < k'$, is also a pair of witnesses, because $|f(x)| \leq C|g(x)| \leq C'|g(x)|$ whenever $x > k' > k$.



Assessment



Links

THE HISTORY OF BIG- O NOTATION Big- O notation has been used in mathematics for more than a century. In computer science it is widely used in the analysis of algorithms, as will be seen in Section 3.3. The German mathematician Paul Bachmann first introduced big- O notation in 1892 in an important book on number theory. The big- O symbol is sometimes called a **Landau symbol** after the German mathematician Edmund Landau, who used this notation throughout his work. The use of big- O notation in computer science was popularized by Donald Knuth, who also introduced the big- Ω and big- Θ notations defined later in this section.

WORKING WITH THE DEFINITION OF BIG- O NOTATION A useful approach for finding a pair of witnesses is to first select a value of k for which the size of $|f(x)|$ can be readily estimated when $x > k$ and to see whether we can use this estimate to find a value of C for which $|f(x)| \leq C|g(x)|$ for $x > k$. This approach is illustrated in Example 1.

EXAMPLE 1 Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.



Solution: We observe that we can readily estimate the size of $f(x)$ when $x > 1$ because $x < x^2$ and $1 < x^2$ when $x > 1$. It follows that

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

whenever $x > 1$, as shown in Figure 1. Consequently, we can take $C = 4$ and $k = 1$ as witnesses to show that $f(x)$ is $O(x^2)$. That is, $f(x) = x^2 + 2x + 1 < 4x^2$ whenever $x > 1$. (Note that it is not necessary to use absolute values here because all functions in these equalities are positive when x is positive.)

Alternatively, we can estimate the size of $f(x)$ when $x > 2$. When $x > 2$, we have $2x \leq x^2$ and $1 \leq x^2$. Consequently, if $x > 2$, we have

$$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2.$$

It follows that $C = 3$ and $k = 2$ are also witnesses to the relation $f(x)$ is $O(x^2)$.

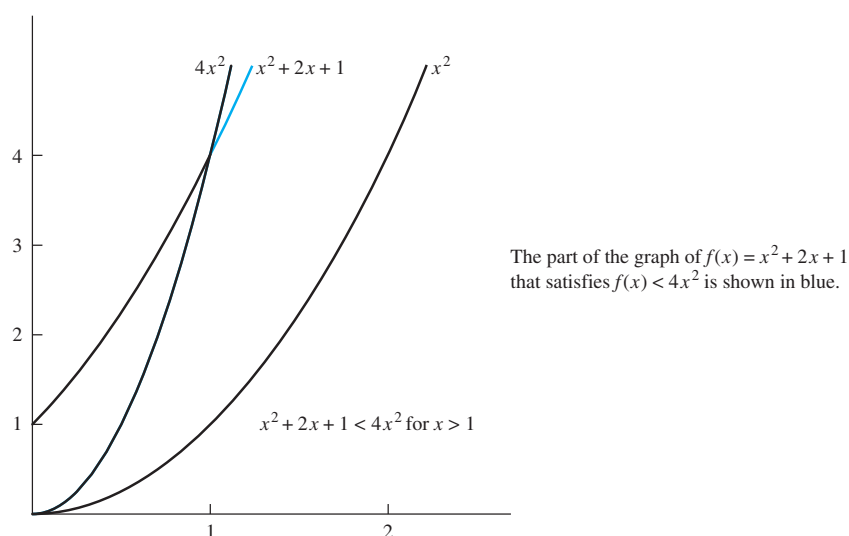


FIGURE 1 The Function $x^2 + 2x + 1$ is $O(x^2)$.

Observe that in the relationship “ $f(x)$ is $O(x^2)$,” x^2 can be replaced by any function with larger values than x^2 . For example, $f(x)$ is $O(x^3)$, $f(x)$ is $O(x^2 + x + 7)$, and so on.

It is also true that x^2 is $O(x^2 + 2x + 1)$, because $x^2 < x^2 + 2x + 1$ whenever $x > 1$. This means that $C = 1$ and $k = 1$ are witnesses to the relationship x^2 is $O(x^2 + 2x + 1)$. ◀

Note that in Example 1 we have two functions, $f(x) = x^2 + 2x + 1$ and $g(x) = x^2$, such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$ —the latter fact following from the inequality $x^2 \leq x^2 + 2x + 1$, which holds for all nonnegative real numbers x . We say that two functions $f(x)$ and $g(x)$ that satisfy both of these big- O relationships are of the **same order**. We will return to this notion later in this section.

Remark: The fact that $f(x)$ is $O(g(x))$ is sometimes written $f(x) = O(g(x))$. However, the equals sign in this notation does *not* represent a genuine equality. Rather, this notation tells us that an inequality holds relating the values of the functions f and g for sufficiently large numbers in the domains of these functions. However, it is acceptable to write $f(x) \in O(g(x))$ because $O(g(x))$ represents the set of functions that are $O(g(x))$.

When $f(x)$ is $O(g(x))$, and $h(x)$ is a function that has larger absolute values than $g(x)$ does for sufficiently large values of x , it follows that $f(x)$ is $O(h(x))$. In other words, the function $g(x)$ in the relationship $f(x)$ is $O(g(x))$ can be replaced by a function with larger absolute values. To see this, note that if

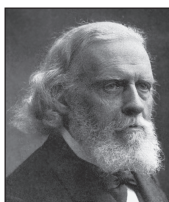
$$|f(x)| \leq C|g(x)| \quad \text{if } x > k,$$

and if $|h(x)| > |g(x)|$ for all $x > k$, then

$$|f(x)| \leq C|h(x)| \quad \text{if } x > k.$$

Hence, $f(x)$ is $O(h(x))$.

When big- O notation is used, the function g in the relationship $f(x)$ is $O(g(x))$ is chosen to be as small as possible (sometimes from a set of reference functions, such as functions of the form x^n , where n is a positive integer).



PAUL GUSTAV HEINRICH BACHMANN (1837–1920) Paul Bachmann, the son of a Lutheran pastor, shared his father's pious lifestyle and love of music. His mathematical talent was discovered by one of his teachers, even though he had difficulties with some of his early mathematical studies. After recuperating from tuberculosis in Switzerland, Bachmann studied mathematics, first at the University of Berlin and later at Göttingen, where he attended lectures presented by the famous number theorist Dirichlet. He received his doctorate under the German number theorist Kummer in 1862; his thesis was on group theory. Bachmann was a professor at Breslau and later at Münster. After he retired from his professorship, he continued his mathematical writing, played the piano, and served as a music critic for newspapers. Bachmann's mathematical writings include a five-volume survey of results and methods in number theory, a two-volume work on elementary number theory, a book on irrational numbers, and a book on the famous conjecture known as Fermat's Last Theorem. He introduced big- O notation in his 1892 book *Analytische Zahlentheorie*.



EDMUND LANDAU (1877–1938) Edmund Landau, the son of a Berlin gynecologist, attended high school and university in Berlin. He received his doctorate in 1899, under the direction of Frobenius. Landau first taught at the University of Berlin and then moved to Göttingen, where he was a full professor until the Nazis forced him to stop teaching. Landau's main contributions to mathematics were in the field of analytic number theory. In particular, he established several important results concerning the distribution of primes. He authored a three-volume exposition on number theory as well as other books on number theory and mathematical analysis.

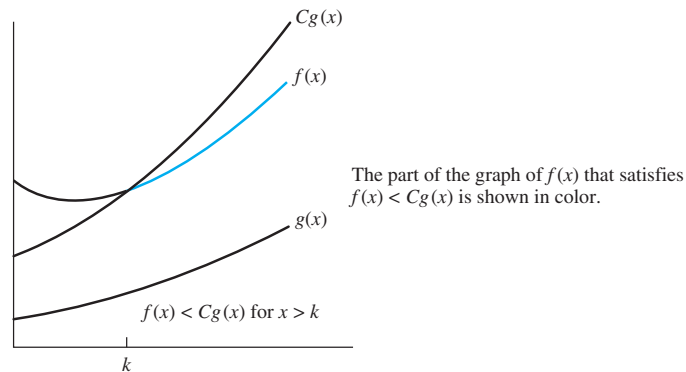


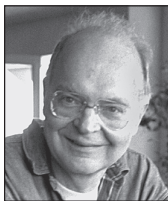
FIGURE 2 The Function $f(x)$ is $O(g(x))$.

In subsequent discussions, we will almost always deal with functions that take on only positive values. All references to absolute values can be dropped when working with big- O estimates for such functions. Figure 2 illustrates the relationship $f(x)$ is $O(g(x))$.

Example 2 illustrates how big- O notation is used to estimate the growth of functions.

EXAMPLE 2 Show that $7x^2$ is $O(x^3)$.

Solution: Note that when $x > 7$, we have $7x^2 < x^3$. (We can obtain this inequality by multiplying both sides of $x > 7$ by x^2 .) Consequently, we can take $C = 1$ and $k = 7$ as witnesses to establish



DONALD E. KNUTH (BORN 1938) Knuth grew up in Milwaukee, where his father taught bookkeeping at a Lutheran high school and owned a small printing business. He was an excellent student, earning academic achievement awards. He applied his intelligence in unconventional ways, winning a contest when he was in the eighth grade by finding over 4500 words that could be formed from the letters in “Ziegler’s Giant Bar.” This won a television set for his school and a candy bar for everyone in his class.

Knuth had a difficult time choosing physics over music as his major at the Case Institute of Technology. He then switched from physics to mathematics, and in 1960 he received his bachelor of science degree, simultaneously receiving a master of science degree by a special award of the faculty who considered his work outstanding. At Case, he managed the basketball team and applied his talents by constructing a formula for the value of each player. This novel approach was covered by *Newsweek* and by Walter Cronkite on the CBS television network. Knuth began graduate work at the California Institute of Technology in 1960 and received his Ph.D. there in 1963. During this time he worked as a consultant, writing compilers for different computers.

Knuth joined the staff of the California Institute of Technology in 1963, where he remained until 1968, when he took a job as a full professor at Stanford University. He retired as Professor Emeritus in 1992 to concentrate on writing. He is especially interested in updating and completing new volumes of his series *The Art of Computer Programming*, a work that has had a profound influence on the development of computer science, which he began writing as a graduate student in 1962, focusing on compilers. In common jargon, “Knuth,” referring to *The Art of Computer Programming*, has come to mean the reference that answers all questions about such topics as data structures and algorithms.

Knuth is the founder of the modern study of computational complexity. He has made fundamental contributions to the subject of compilers. His dissatisfaction with mathematics typography sparked him to invent the now widely used TeX and Metafont systems. TeX has become a standard language for computer typography. Two of the many awards Knuth has received are the 1974 Turing Award and the 1979 National Medal of Technology, awarded to him by President Carter.

Knuth has written for a wide range of professional journals in computer science and in mathematics. However, his first publication, in 1957, when he was a college freshman, was a parody of the metric system called “The Potrzebie Systems of Weights and Measures,” which appeared in *MAD Magazine* and has been in reprint several times. He is a church organist, as his father was. He is also a composer of music for the organ. Knuth believes that writing computer programs can be an aesthetic experience, much like writing poetry or composing music.

Knuth pays \$2.56 for the first person to find each error in his books and \$0.32 for significant suggestions. If you send him a letter with an error (you will need to use regular mail, because he has given up reading e-mail), he will eventually inform you whether you were the first person to tell him about this error. Be prepared for a long wait, because he receives an overwhelming amount of mail. (The author received a letter years after sending an error report to Knuth, noting that this report arrived several months after the first report of this error.)

the relationship $7x^2$ is $O(x^3)$. Alternatively, when $x > 1$, we have $7x^2 < 7x^3$, so that $C = 7$ and $k = 1$ are also witnesses to the relationship $7x^2$ is $O(x^3)$. ◀

Example 3 illustrates how to show that a big- O relationship does not hold.

EXAMPLE 3 Show that n^2 is not $O(n)$.

Solution: To show that n^2 is not $O(n)$, we must show that no pair of witnesses C and k exist such that $n^2 \leq Cn$ whenever $n > k$. We will use a proof by contradiction to show this.

Suppose that there are constants C and k for which $n^2 \leq Cn$ whenever $n > k$. Observe that when $n > 0$ we can divide both sides of the inequality $n^2 \leq Cn$ by n to obtain the equivalent inequality $n \leq C$. However, no matter what C and k are, the inequality $n \leq C$ cannot hold for all n with $n > k$. In particular, once we set a value of k , we see that when n is larger than the maximum of k and C , it is not true that $n \leq C$ even though $n > k$. This contradiction shows that n^2 is not $O(n)$. ◀

EXAMPLE 4 Example 2 shows that $7x^2$ is $O(x^3)$. Is it also true that x^3 is $O(7x^2)$?

Solution: To determine whether x^3 is $O(7x^2)$, we need to determine whether witnesses C and k exist, so that $x^3 \leq C(7x^2)$ whenever $x > k$. We will show that no such witnesses exist using a proof by contradiction.

If C and k are witnesses, the inequality $x^3 \leq C(7x^2)$ holds for all $x > k$. Observe that the inequality $x^3 \leq C(7x^2)$ is equivalent to the inequality $x \leq 7C$, which follows by dividing both sides by the positive quantity x^2 . However, no matter what C is, it is not the case that $x \leq 7C$ for all $x > k$ no matter what k is, because x can be made arbitrarily large. It follows that no witnesses C and k exist for this proposed big- O relationship. Hence, x^3 is not $O(7x^2)$. ◀

Big- O Estimates for Some Important Functions

Polynomials can often be used to estimate the growth of functions. Instead of analyzing the growth of polynomials each time they occur, we would like a result that can always be used to estimate the growth of a polynomial. Theorem 1 does this. It shows that the leading term of a polynomial dominates its growth by asserting that a polynomial of degree n or less is $O(x^n)$.


THEOREM 1 Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, where $a_0, a_1, \dots, a_{n-1}, a_n$ are real numbers. Then $f(x)$ is $O(x^n)$.

Proof: Using the triangle inequality (see Exercise 7 in Section 1.8), if $x > 1$ we have

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0| \\ &\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x + |a_0| \\ &= x^n (|a_n| + |a_{n-1}|/x + \cdots + |a_1|/x^{n-1} + |a_0|/x^n) \\ &\leq x^n (|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|). \end{aligned}$$

This shows that

$$|f(x)| \leq Cx^n,$$


where $C = |a_n| + |a_{n-1}| + \cdots + |a_0|$ whenever $x > 1$. Hence, the witnesses $C = |a_n| + |a_{n-1}| + \cdots + |a_0|$ and $k = 1$ show that $f(x)$ is $O(x^n)$. 

We now give some examples involving functions that have the set of positive integers as their domains.

EXAMPLE 5 How can big- O notation be used to estimate the sum of the first n positive integers?

Solution: Because each of the integers in the sum of the first n positive integers does not exceed n , it follows that

$$1 + 2 + \cdots + n \leq n + n + \cdots + n = n^2.$$

From this inequality it follows that $1 + 2 + 3 + \cdots + n$ is $O(n^2)$, taking $C = 1$ and $k = 1$ as witnesses. (In this example the domains of the functions in the big- O relationship are the set of positive integers.) 

In Example 6 big- O estimates will be developed for the factorial function and its logarithm. These estimates will be important in the analysis of the number of steps used in sorting procedures.

EXAMPLE 6 Give big- O estimates for the factorial function and the logarithm of the factorial function, where the factorial function $f(n) = n!$ is defined by

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

whenever n is a positive integer, and $0! = 1$. For example,

$$1! = 1, \quad 2! = 1 \cdot 2 = 2, \quad 3! = 1 \cdot 2 \cdot 3 = 6, \quad 4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24.$$

Note that the function $n!$ grows rapidly. For instance,

$$20! = 2,432,902,008,176,640,000.$$

Solution: A big- O estimate for $n!$ can be obtained by noting that each term in the product does not exceed n . Hence,

$$\begin{aligned} n! &= 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n \\ &\leq n \cdot n \cdot n \cdot \cdots \cdot n \\ &= n^n. \end{aligned}$$

This inequality shows that $n!$ is $O(n^n)$, taking $C = 1$ and $k = 1$ as witnesses. Taking logarithms of both sides of the inequality established for $n!$, we obtain

$$\log n! \leq \log n^n = n \log n.$$

This implies that $\log n!$ is $O(n \log n)$, again taking $C = 1$ and $k = 1$ as witnesses. 

EXAMPLE 7 In Section 4.1, we will show that $n < 2^n$ whenever n is a positive integer. Show that this inequality implies that n is $O(2^n)$, and use this inequality to show that $\log n$ is $O(n)$.

Solution: Using the inequality $n < 2^n$, we quickly can conclude that n is $O(2^n)$ by taking $k = C = 1$ as witnesses. Note that because the logarithm function is increasing, taking logarithms (base 2) of both sides of this inequality shows that

$$\log n < n.$$

It follows that

$$\log n \text{ is } O(n).$$

(Again we take $C = k = 1$ as witnesses.)

If we have logarithms to a base b , where b is different from 2, we still have $\log_b n$ is $O(n)$ because

$$\log_b n = \frac{\log n}{\log b} < \frac{n}{\log b}$$

whenever n is a positive integer. We take $C = 1/\log b$ and $k = 1$ as witnesses. (We have used Theorem 3 in Appendix 2 to see that $\log_b n = \log n / \log b$.)

As mentioned before, big- O notation is used to estimate the number of operations needed to solve a problem using a specified procedure or algorithm. The functions used in these estimates often include the following:

$$1, \log n, n, n \log n, n^2, 2^n, n!$$

Using calculus it can be shown that each function in the list is smaller than the succeeding function, in the sense that the ratio of a function and the succeeding function tends to zero as n grows without bound. Figure 3 displays the graphs of these functions, using a scale for the values of the functions that doubles for each successive marking on the graph. That is, the vertical scale in this graph is logarithmic.

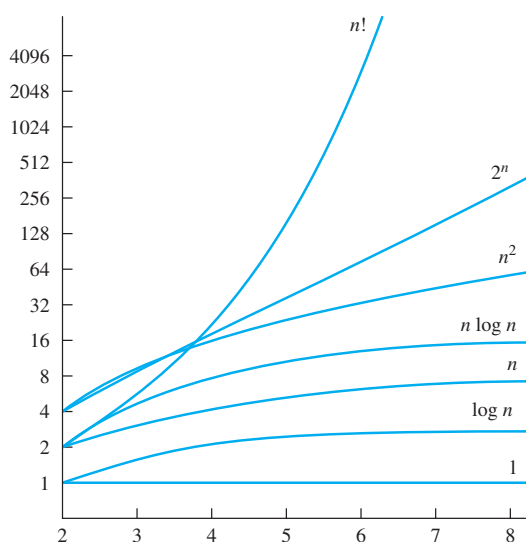


FIGURE 3 A Display of the Growth of Functions Commonly Used in Big- O Estimates.

USEFUL BIG- O ESTIMATES INVOLVING LOGARITHMS, POWERS, AND EXPONENTIAL FUNCTIONS We now give some useful facts that help us determine whether big- O relationships hold between pairs of functions when each of the functions is a power of a logarithm, a power, or an exponential function of the form b^n where $b > 1$. Their proofs are left as Exercises 57–60 for readers skilled with calculus.

Theorem 1 shows that if $f(n)$ is a polynomial of degree d , then $f(n)$ is $O(n^d)$. Applying this theorem, we see that if $d > c > 1$, then n^c is $O(n^d)$. We leave it to the reader to show that the reverse of this relationship does not hold. Putting these facts together, we see that if $d > c > 1$, then

$$n^c \text{ is } O(n^d), \text{ but } n^d \text{ is not } O(n^c).$$

In Example 7 we showed that $\log_b n$ is $O(n)$ whenever $b > 1$. More generally, whenever $b > 1$ and c and d are positive, we have

$$(\log_b n)^c \text{ is } O(n^d), \text{ but } n^d \text{ is not } O((\log_b n)^c).$$

This tells us that every positive power of the logarithm of n to the base b , where $b > 1$, is big- O of every positive power of n , but the reverse relationship never holds.

In Example 7, we also showed that n is $O(2^n)$. More generally, whenever d is positive and $b > 1$, we have

$$n^d \text{ is } O(b^n), \text{ but } b^n \text{ is not } O(n^d).$$

This tells us that every power of n is big- O of every exponential function of n with a base that is greater than one, but the reverse relationship never holds. Furthermore, we have when $c > b > 1$,

$$b^n \text{ is } O(c^n) \text{ but } c^n \text{ is not } O(b^n).$$

This tells us that if we have two exponential functions with different bases greater than one, one of these functions is big- O of the other if and only if its base is smaller or equal.

The Growth of Combinations of Functions

Many algorithms are made up of two or more separate subprocedures. The number of steps used by a computer to solve a problem with input of a specified size using such an algorithm is the sum of the number of steps used by these subprocedures. To give a big- O estimate for the number of steps needed, it is necessary to find big- O estimates for the number of steps used by each subprocedure and then combine these estimates.

Big- O estimates of combinations of functions can be provided if care is taken when different big- O estimates are combined. In particular, it is often necessary to estimate the growth of the sum and the product of two functions. What can be said if big- O estimates for each of two functions are known? To see what sort of estimates hold for the sum and the product of two functions, suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$.

From the definition of big- O notation, there are constants C_1 , C_2 , k_1 , and k_2 such that

$$|f_1(x)| \leq C_1 |g_1(x)|$$

when $x > k_1$, and

$$|f_2(x)| \leq C_2 |g_2(x)|$$

when $x > k_2$. To estimate the sum of $f_1(x)$ and $f_2(x)$, note that

$$\begin{aligned} |(f_1 + f_2)(x)| &= |f_1(x) + f_2(x)| \\ &\leq |f_1(x)| + |f_2(x)| \quad \text{using the triangle inequality } |a + b| \leq |a| + |b|. \end{aligned}$$

When x is greater than both k_1 and k_2 , it follows from the inequalities for $|f_1(x)|$ and $|f_2(x)|$ that

$$\begin{aligned} |f_1(x)| + |f_2(x)| &\leq C_1|g_1(x)| + C_2|g_2(x)| \\ &\leq C_1|g(x)| + C_2|g(x)| \\ &= (C_1 + C_2)|g(x)| \\ &= C|g(x)|, \end{aligned}$$

where $C = C_1 + C_2$ and $g(x) = \max(|g_1(x)|, |g_2(x)|)$. [Here $\max(a, b)$ denotes the maximum, or larger, of a and b .]

This inequality shows that $|(f_1 + f_2)(x)| \leq C|g(x)|$ whenever $x > k$, where $k = \max(k_1, k_2)$. We state this useful result as Theorem 2.

THEOREM 2

Suppose that $f_1(x)$ is $O(g_1(x))$ and that $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$.

We often have big- O estimates for f_1 and f_2 in terms of the same function g . In this situation, Theorem 2 can be used to show that $(f_1 + f_2)(x)$ is also $O(g(x))$, because $\max(g(x), g(x)) = g(x)$. This result is stated in Corollary 1.

COROLLARY 1

Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$.

In a similar way big- O estimates can be derived for the product of the functions f_1 and f_2 . When x is greater than $\max(k_1, k_2)$ it follows that

$$\begin{aligned} |(f_1 f_2)(x)| &= |f_1(x)| |f_2(x)| \\ &\leq C_1|g_1(x)| C_2|g_2(x)| \\ &\leq C_1 C_2 |(g_1 g_2)(x)| \\ &\leq C |(g_1 g_2)(x)|, \end{aligned}$$

where $C = C_1 C_2$. From this inequality, it follows that $f_1(x)f_2(x)$ is $O(g_1 g_2(x))$, because there are constants C and k , namely, $C = C_1 C_2$ and $k = \max(k_1, k_2)$, such that $|(f_1 f_2)(x)| \leq C|g_1(x)g_2(x)|$ whenever $x > k$. This result is stated in Theorem 3.

THEOREM 3

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.

The goal in using big- O notation to estimate functions is to choose a function $g(x)$ as simple as possible, that grows relatively slowly so that $f(x)$ is $O(g(x))$. Examples 8 and 9 illustrate how to use Theorems 2 and 3 to do this. The type of analysis given in these examples is often used in the analysis of the time used to solve problems using computer programs.

EXAMPLE 8 Give a big- O estimate for $f(n) = 3n \log(n!) + (n^2 + 3) \log n$, where n is a positive integer.

Solution: First, the product $3n \log(n!)$ will be estimated. From Example 6 we know that $\log(n!)$ is $O(n \log n)$. Using this estimate and the fact that $3n$ is $O(n)$, Theorem 3 gives the estimate that $3n \log(n!)$ is $O(n^2 \log n)$.

Next, the product $(n^2 + 3) \log n$ will be estimated. Because $(n^2 + 3) < 2n^2$ when $n > 2$, it follows that $n^2 + 3$ is $O(n^2)$. Thus, from Theorem 3 it follows that $(n^2 + 3) \log n$ is $O(n^2 \log n)$. Using Theorem 2 to combine the two big- O estimates for the products shows that $f(n) = 3n \log(n!) + (n^2 + 3) \log n$ is $O(n^2 \log n)$. ◀

EXAMPLE 9 Give a big- O estimate for $f(x) = (x + 1) \log(x^2 + 1) + 3x^2$.

Solution: First, a big- O estimate for $(x + 1) \log(x^2 + 1)$ will be found. Note that $(x + 1)$ is $O(x)$. Furthermore, $x^2 + 1 \leq 2x^2$ when $x > 1$. Hence,

$$\log(x^2 + 1) \leq \log(2x^2) = \log 2 + \log x^2 = \log 2 + 2 \log x \leq 3 \log x,$$

if $x > 2$. This shows that $\log(x^2 + 1)$ is $O(\log x)$.

From Theorem 3 it follows that $(x + 1) \log(x^2 + 1)$ is $O(x \log x)$. Because $3x^2$ is $O(x^2)$, Theorem 2 tells us that $f(x)$ is $O(\max(x \log x, x^2))$. Because $x \log x \leq x^2$, for $x > 1$, it follows that $f(x)$ is $O(x^2)$. ◀

Big-Omega and Big-Theta Notation

Ω and Θ are the Greek uppercase letters omega and theta, respectively.

Big- O notation is used extensively to describe the growth of functions, but it has limitations. In particular, when $f(x)$ is $O(g(x))$, we have an upper bound, in terms of $g(x)$, for the size of $f(x)$ for large values of x . However, big- O notation does not provide a lower bound for the size of $f(x)$ for large x . For this, we use **big-Omega (big- Ω) notation**. When we want to give both an upper and a lower bound on the size of a function $f(x)$, relative to a reference function $g(x)$, we use **big-Theta (big- Θ) notation**. Both big-Omega and big-Theta notation were introduced by Donald Knuth in the 1970s. His motivation for introducing these notations was the common misuse of big- O notation when both an upper and a lower bound on the size of a function are needed.

We now define big-Omega notation and illustrate its use. After doing so, we will do the same for big-Theta notation.

DEFINITION 2 Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are positive constants C and k such that

$$|f(x)| \geq C|g(x)|$$

whenever $x > k$. [This is read as “ $f(x)$ is big-Omega of $g(x)$.”]

There is a strong connection between big- O and big-Omega notation. In particular, $f(x)$ is $\Omega(g(x))$ if and only if $g(x)$ is $O(f(x))$. We leave the verification of this fact as a straightforward exercise for the reader.

EXAMPLE 10 The function $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$, where $g(x)$ is the function $g(x) = x^3$. This is easy to see because $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ for all positive real numbers x . This is equivalent to saying that $g(x) = x^3$ is $O(8x^3 + 5x^2 + 7)$, which can be established directly by turning the inequality around. ◀

Often, it is important to know the order of growth of a function in terms of some relatively simple reference function such as x^n when n is a positive integer or c^x , where $c > 1$. Knowing the order of growth requires that we have both an upper bound and a lower bound for the size of the function. That is, given a function $f(x)$, we want a reference function $g(x)$ such that $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. Big-Theta notation, defined as follows, is used to express both of these relationships, providing both an upper and a lower bound on the size of a function.

DEFINITION 3

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. When $f(x)$ is $\Theta(g(x))$ we say that f is big-Theta of $g(x)$, that $f(x)$ is of *order* $g(x)$, and that $f(x)$ and $g(x)$ are of the *same order*.

When $f(x)$ is $\Theta(g(x))$, it is also the case that $g(x)$ is $\Theta(f(x))$. Also note that $f(x)$ is $\Theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$ (see Exercise 31). Furthermore, note that $f(x)$ is $\Theta(g(x))$ if and only if there are real numbers C_1 and C_2 and a positive real number k such that

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

whenever $x > k$. The existence of the constants C_1 , C_2 , and k tells us that $f(x)$ is $\Omega(g(x))$ and that $f(x)$ is $O(g(x))$, respectively.

Usually, when big-Theta notation is used, the function $g(x)$ in $\Theta(g(x))$ is a relatively simple reference function, such as x^n , c^x , $\log x$, and so on, while $f(x)$ can be relatively complicated.


EXAMPLE 11

We showed (in Example 5) that the sum of the first n positive integers is $O(n^2)$. Is this sum of order n^2 ?



Solution: Let $f(n) = 1 + 2 + 3 + \cdots + n$. Because we already know that $f(n)$ is $O(n^2)$, to show that $f(n)$ is of order n^2 we need to find a positive constant C such that $f(n) > Cn^2$ for sufficiently large integers n . To obtain a lower bound for this sum, we can ignore the first half of the terms. Summing only the terms greater than $\lceil n/2 \rceil$, we find that


$$\begin{aligned} 1 + 2 + \cdots + n &\geq \lceil n/2 \rceil + (\lceil n/2 \rceil + 1) + \cdots + n \\ &\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \cdots + \lceil n/2 \rceil \\ &= (n - \lceil n/2 \rceil + 1) \lceil n/2 \rceil \\ &\geq (n/2)(n/2) \\ &= n^2/4. \end{aligned}$$

This shows that $f(n)$ is $\Omega(n^2)$. We conclude that $f(n)$ is of order n^2 , or in symbols, $f(n)$ is $\Theta(n^2)$. 

EXAMPLE 12

Show that $3x^2 + 8x \log x$ is $\Theta(x^2)$.




Solution: Because $0 \leq 8x \log x \leq 8x^2$, it follows that $3x^2 + 8x \log x \leq 11x^2$ for $x > 1$. Consequently, $3x^2 + 8x \log x$ is $O(x^2)$. Clearly, x^2 is $O(3x^2 + 8x \log x)$. Consequently, $3x^2 + 8x \log x$ is $\Theta(x^2)$. 

One useful fact is that the leading term of a polynomial determines its order. For example, if $f(x) = 3x^5 + x^4 + 17x^3 + 2$, then $f(x)$ is of order x^5 . This is stated in Theorem 4, whose proof is left as Exercise 50.

THEOREM 4

Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, where a_0, a_1, \dots, a_n are real numbers with $a_n \neq 0$. Then $f(x)$ is of order x^n .

EXAMPLE 13

The polynomials $3x^8 + 10x^7 + 221x^2 + 1444$, $x^{19} - 18x^4 - 10,112$, and $-x^{99} + 40,001x^{98} + 100,003x$ are of orders x^8 , x^{19} , and x^{99} , respectively. 

Unfortunately, as Knuth observed, big- O notation is often used by careless writers and speakers as if it had the same meaning as big- Θ notation. Keep this in mind when you see big- O notation used. The recent trend has been to use big- Θ notation whenever both upper and lower bounds on the size of a function are needed.

Exercises

In Exercises 1–14, to establish a big- O relationship, find witnesses C and k such that $|f(x)| \leq C|g(x)|$ whenever $x > k$.

- Determine whether each of these functions is $O(x)$.
 - $f(x) = 10$
 - $f(x) = 3x + 7$
 - $f(x) = x^2 + x + 1$
 - $f(x) = 5 \log x$
 - $f(x) = \lfloor x \rfloor$
 - $f(x) = \lceil x/2 \rceil$
- Determine whether each of these functions is $O(x^2)$.
 - $f(x) = 17x + 11$
 - $f(x) = x^2 + 1000$
 - $f(x) = x \log x$
 - $f(x) = x^4/2$
 - $f(x) = 2^x$
 - $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil$
- Use the definition of “ $f(x)$ is $O(g(x))$ ” to show that $x^4 + 9x^3 + 4x + 7$ is $O(x^4)$.
- Use the definition of “ $f(x)$ is $O(g(x))$ ” to show that $2^x + 17$ is $O(3^x)$.
- Show that $(x^2 + 1)/(x + 1)$ is $O(x)$.
- Show that $(x^3 + 2x)/(2x + 1)$ is $O(x^2)$.
- Find the least integer n such that $f(x)$ is $O(x^n)$ for each of these functions.
 - $f(x) = 2x^3 + x^2 \log x$
 - $f(x) = 3x^3 + (\log x)^4$
 - $f(x) = (x^4 + x^2 + 1)/(x^3 + 1)$
 - $f(x) = (x^4 + 5 \log x)/(x^4 + 1)$
- Find the least integer n such that $f(x)$ is $O(x^n)$ for each of these functions.
 - $f(x) = 2x^2 + x^3 \log x$
 - $f(x) = 3x^5 + (\log x)^4$
 - $f(x) = (x^4 + x^2 + 1)/(x^4 + 1)$
 - $f(x) = (x^3 + 5 \log x)/(x^4 + 1)$
- Show that $x^2 + 4x + 17$ is $O(x^3)$ but that x^3 is not $O(x^2 + 4x + 17)$.
- Show that x^3 is $O(x^4)$ but that x^4 is not $O(x^3)$.
- Show that $3x^4 + 1$ is $O(x^4/2)$ and $x^4/2$ is $O(3x^4 + 1)$.
- Show that $x \log x$ is $O(x^2)$ but that x^2 is not $O(x \log x)$.
- Show that 2^n is $O(3^n)$ but that 3^n is not $O(2^n)$. (Note that this is a special case of Exercise 60.)
- Determine whether x^3 is $O(g(x))$ for each of these functions $g(x)$.
 - $g(x) = x^2$
 - $g(x) = x^3$
 - $g(x) = x^2 + x^3$
 - $g(x) = x^2 + x^4$
 - $g(x) = 3^x$
 - $g(x) = x^3/2$
- Explain what it means for a function to be $O(1)$.
- Show that if $f(x)$ is $O(x)$, then $f(x)$ is $O(x^2)$.
- Suppose that $f(x)$, $g(x)$, and $h(x)$ are functions such that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(h(x))$. Show that $f(x)$ is $O(h(x))$.
- Let k be a positive integer. Show that $1^k + 2^k + \cdots + n^k$ is $O(n^{k+1})$.
- Determine whether each of the functions 2^{n+1} and 2^{2n} is $O(2^n)$.
- Determine whether each of the functions $\log(n + 1)$ and $\log(n^2 + 1)$ is $O(\log n)$.
- Arrange the functions \sqrt{n} , $1000 \log n$, $n \log n$, $2n!$, 2^n , 3^n , and $n^2/1,000,000$ in a list so that each function is big- O of the next function.
- Arrange the function $(1.5)^n$, n^{100} , $(\log n)^3$, $\sqrt{n} \log n$, 10^n , $(n!)^2$, and $n^{99} + n^{98}$ in a list so that each function is big- O of the next function.
- Suppose that you have two different algorithms for solving a problem. To solve a problem of size n , the first algorithm uses exactly $n(\log n)$ operations and the second algorithm uses exactly $n^{3/2}$ operations. As n grows, which algorithm uses fewer operations?
- Suppose that you have two different algorithms for solving a problem. To solve a problem of size n , the first algorithm uses exactly $n^2 2^n$ operations and the second algorithm uses exactly $n!$ operations. As n grows, which algorithm uses fewer operations?