2.3 Complexity of Integer Operations

Once an algorithm has been specified for an operation, we can consider the amount of time required to perform this algorithm on a computer. We will measure the amount of time in terms of *bit operations*. By a bit operation we mean the addition, subtraction, or multiplication of two binary digits, the division of a two-bit by a one-bit integer (obtaining a quotient and a remainder), or the shifting of a binary integer one place. (The actual amount of time required to carry out a bit operation on a computer varies depending on the computer architecture and capacity.) When we describe the number of bit operations needed to perform an algorithm, we are describing the *computational complexity* of this algorithm.

In describing the number of bit operations needed to perform calculations, we will use *big-O* notation. Big-O notation provides an upper bound on the size of a function in terms of a particular well-known reference function whose size at large values is easily understood.

To motivate the definition of this notation, consider the following situation. Suppose that to perform a specified operation on an integer n requires at most $n^3 + 8n^2 \log n$ bit operations. Since $8n^2 \log n < 8n^3$ for every positive integer, less than $9n^3$ bit operations are required for this operation for every integer n. Since the number of bit operations required is always less than a constant times n^3, namely $9n^3$, we say that $O(n^3)$ bit operations are needed. In general, we have the following definition.

Definition. If f and g are functions taking positive values, defined for all $x \in S$, where S is a specified set of real numbers, then f is $O(g)$ on S if there is a positive constant K such that $f(x) < Kg(x)$ for all sufficiently large $x \in S$. (Normally, we take S to be the set of positive integers, and we drop all reference to S.)

Big-O notation is used extensively throughout number theory and in the analysis of algorithms. *Paul Bachmann* introduced big-O notation in 1892 ([Ba94]). The big-O notation is sometimes called a Landau symbol, after *Edmund Landau*, who used this notation throughout his work in the estimation of various functions in number theory. The use of big-O notation in the analysis of algorithms was popularized by renowned computer scientist *Donald Knuth*.

We illustrate this concept of big-O notation with several examples.

Example 2.10. We can show on the set of positive integers that $n^4 + 2n^3 + 5$ is $O(n^4)$. To do this, note that $n^4 + 2n^3 + 5 \leq n^4 + 2n^4 + 5n^4 = 8n^4$ for all positive integers. (We take $K = 8$ in the definition.) The reader should also note that n^4 is $O(n^4 + 2n^3 + 5)$. ◀

Example 2.11. We can easily give a big-O estimate for $\sum_{j=1}^{n} j$. Noting that each summand is less than n tells us that $\sum_{j=1}^{n} j \leq \sum_{j=1}^{n} n = n \cdot n = n^2$. Note that we could also derive this estimate easily from the formula $\sum_{j=1}^{n} j = n(n+1)/2$. ◀

We now will give some useful results for working with big-O estimates for combinations of functions.

Theorem 2.2. If f is $O(g)$ and c is a positive constant, then cf is $O(g)$.

Proof. If f is $O(g)$, then there is a constant K with $f(x) < Kg(x)$ for all x under consideration. Hence $cf(x) < (cK)g(x)$, so cf is $O(g)$. ∎

Theorem 2.3. If f_1 is $O(g_1)$ and f_2 is $O(g_2)$, then $f_1 + f_2$ is $O(g_1 + g_2)$, and $f_1 f_2$ is $O(g_1 g_2)$.

Proof. If f is $O(g_1)$ and f_2 is $O(g_2)$, then there are constants K_1 and K_2 such that $f_1(x) < K_1 g_1(x)$ and $f_2(x) < K_2 g_2(x)$ for all x under consideration. Hence,

$$\begin{aligned} f_1(x) + f_2(x) &< K_1 g_1(x) + K_2 g_2(x) \\ &\leq K(g_1(x) + g_2(x)), \end{aligned}$$

where K is the maximum of K_1 and K_2. Hence, $f_1 + f_2$ is $O(g_1 + g_2)$.

Also,

$$\begin{aligned} f_1(x) f_2(x) &< K_1 g_1(x) K_2 g_2(x) \\ &= (K_1 K_2)(g_1(x) g_2(x)), \end{aligned}$$

so $f_1 f_2$ is $O(g_1 g_2)$. ∎

PAUL GUSTAV HEINRICH BACHMANN (1837–1920), the son of a pastor, shared his father's pious lifestyle, as well as his love of music. His talent for mathematics was discovered by one of his early teachers. After recovering from tuberculosis, he studied at the University of Berlin and later in Göttingen, where he attended lectures presented by Dirichlet. In 1862, he received his doctorate under the supervision of the number theorist Kummer. Bachmann became a professor at Breslau and later at Münster. After retiring, he continued mathematical research, played the piano, and served as a music critic for newspapers. His writings include a five-volume survey of number theory, a two-volume work on elementary number theory, a book on irrational numbers, and a book on Fermat's last theorem (this theorem is discussed in Chapter 13). Bachmann introduced big-O notation in 1892.

EDMUND LANDAU (1877–1938) was the son of a Berlin gynecologist, and attended high school 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 full professor until the Nazis forced him to stop teaching. His main contributions to mathematics were in the field of analytic number theory; he established several important results concerning the distribution of primes. He authored a three-volume work on number theory and many other books on mathematical analysis and analytic number theory.

Corollary 2.3.1. If f_1 and f_2 are $O(g)$, then $f_1 + f_2$ is $O(g)$.

Proof. Theorem 2.3 tells us that $f_1 + f_2$ is $O(2g)$. But if $f_1 + f_2 < K(2g)$, then $f_1 + f_2 < (2K)g$, so $f_1 + f_2$ is $O(g)$. ∎

The goal in using big-O estimates is to give the best big-O estimate possible while using the simplest reference function possible. Well-known reference functions used in big-O estimates include 1, $\log n$, n, $n \log n$, $n \log n \log \log n$, n^2, and 2^n, as well as some other important functions. Calculus can be used to show that each function in this list is smaller than the next function in the list, in the sense that the ratio of the function and the next function tends to 0 as n grows without bound. Note that more complicated functions than these occur in big-O estimates, as you will see in later chapters.

We illustrate how to use theorems for working with big-O estimates with the following example.

Example 2.12. To give a big-O estimate for $(n + 8 \log n)\,(10n \log n + 17n^2)$, first note that $n + 8 \log n$ is $O(n)$ and $10n \log n + 17n^2$ is $O(n^2)$ (because $\log n$ is $O(n)$ and $n \log n$ is $O(n^2)$) by Theorems 2.2 and 2.3 and Corollary 2.3.1. By Theorem 2.3, we see that $(n + 8 \log n)(10n \log n + 17n^2)$ is $O(n^3)$. ◀

Using big-O notation, we can see that to add or subtract two n-bit integers takes $O(n)$ bit operations, whereas to multiply two n-bit integers in the conventional way takes $O(n^2)$ bit operations (see Exercises 12 and 13 at the end of this section). Surprisingly,

DONALD KNUTH (b. 1938) grew up in Milwaukee where his father owned a small printing business and taught bookkeeping. He was an excellent student who also applied his intelligence in unconventional ways, such as finding more than 4500 words that could be spelled from the letters in "Ziegler's Giant Bar," winning a television set for his school and candy bars for everyone in his class.

Knuth graduated from Case Institute of Technology in 1960 with B.S. and M.S. degrees in mathematics, by special award of the faculty who considered his work outstanding. At Case he managed the basketball team and applied his mathematical talents by evaluating each player using a formula he developed (receiving coverage on CBS television and in *Newsweek*). Knuth received his doctorate in 1963 from the California Institute of Technology.

Knuth taught at the California Institute of Technology and Stanford University, retiring in 1992 to concentrate on writing. He is especially interested in updating and adding to his famous series, *The Art of Computer Programming*. This series has had a profound influence on the development of computer science. Knuth is the founder of the modern study of computational complexity and has made fundamental contributions to the theory of compilers. Knuth has also invented the widely used TeX and Metafont systems used for mathematical (and general) typography. TeX played an important role in the development of HTML and the Internet. He popularized the big-O notation in his work on the analysis of algorithms.

Knuth has written for a wide range of professional journals in computer science and mathematics. However, his first publication, in 1957, when he was a college freshman, was the "The Potrzebie System of Weights and Measures," a parody of the metric system, which appeared in *MAD Magazine*.

there are faster algorithms for multiplying large integers. To develop one such algorithm, we first consider the multiplication of two $2n$-bit integers, say $a = (a_{2n-1}a_{2n-2}\ldots a_1a_0)_2$ and $b = (b_{2n-1}b_{2n-2}\ldots b_1b_0)_2$. We write

$$a = 2^nA_1 + A_0 \quad b = 2^nB_1 + B_0,$$

where

$$A_1 = (a_{2n-1}a_{2n-2}\ldots a_{n+1}a_n)_2 \quad A_0 = (a_{n-1}a_{n-2}\ldots a_1a_0)_2$$
$$B_1 = (b_{2n-1}b_{2n-2}\ldots b_{n+1}b_n)_2 \quad B_0 = (b_{n-1}b_{n-2}\ldots b_1b_0)_2.$$

We will use the identity

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0. \tag{2.2}$$

To find the product of a and b using (2.2) requires that we perform three multiplications of n-bit integers (namely, A_1B_1, $(A_1 - A_0)(B_0 - B_1)$, and A_0B_0), as well as a number of additions and shifts. This is illustrated by the following example.

Example 2.13. We can use (2.2) to multiply $(1101)_2$ and $(1011)_2$. We have $(1101)_2 = 2^2(11)_2 + (01)_2$ and $(1011)_2 = 2^2(10)_2 + (11)_2$. Using (2.2), we find that

$$\begin{aligned}(1101)_2(1011)_2 &= (2^4 + 2^2)(11)_2(10)_2 + 2^2((11)_2 - (01)_2)\cdot((11)_2 - (10)_2) + \\ &\qquad (2^2 + 1)(01)_2(11)_2 \\ &= (2^4 + 2^2)(110)_2 + 2^2(10)_2(01)_2 + (2^2 + 1)(11)_2 \\ &= (1100000)_2 + (11000)_2 + (1000)_2 + (1100)_2 + (11)_2 \\ &= (10001111)_2.\end{aligned}$$

◀

We will now estimate the number of bit operations required to multiply two n-bit integers by using (2.2) repeatedly. If we let $M(n)$ denote the number of bit operations needed to multiply two n-bit integers, we find from (2.2) that

$$M(2n) \leq 3M(n) + Cn, \tag{2.3}$$

where C is a constant, because each of the three multiplications of n-bit integers takes $M(n)$ bit operations, whereas the number of additions and shifts needed to compute ab via (2.2) does not depend on n, and each of these operations takes $O(n)$ bit operations.

From (2.3), using mathematical induction, we can show that

$$M(2^k) \leq c(3^k - 2^k), \tag{2.4}$$

where c is the maximum of the quantities $M(2)$ and C (the constant in (2.3)). To carry out the induction argument, we first note that with $k = 1$, we have $M(2) \leq c(3^1 - 2^1) = c$, because c is the maximum of $M(2)$ and C.

As the induction hypothesis, we assume that

$$M(2^k) \leq c(3^k - 2^k).$$

Then, using (2.3), we have

$$\begin{aligned} M(2^{k+1}) &\leq 3M(2^k) + C2^k \\ &\leq 3c(3^k - 2^k) + C2^k \\ &\leq c3^{k+1} - c \cdot 3 \cdot 2^k + c2^k \\ &\leq c(3^{k+1} - 2^{k+1}). \end{aligned}$$

This establishes that (2.4) is valid for all positive integers k.

Using inequality (2.4), we can prove the following theorem.

Theorem 2.4. Multiplication of two n-bit integers can be performed using $O(n^{\log_2 3})$ bit operations. (*Note:* $\log_2 3$ is approximately 1.585, which is considerably less than the exponent 2 that occurs in the estimate of the number of bit operations needed for the conventional multiplication algorithm.)

Proof. From (2.4), we have

$$\begin{aligned} M(n) = M(2^{\log_2 n}) &\leq M(2^{\lceil \log_2 n \rceil + 1}) \\ &\leq c(3^{\lceil \log_2 n \rceil + 1} - 2^{\lceil \log_2 n \rceil + 1}) \\ &\leq 3c \cdot 3^{\lceil \log_2 n \rceil} \leq 3c \cdot 3^{\log_2 n} = 3cn^{\log_2 3} \quad (\textit{because } 3^{\log_2 n} = n^{\log_2 3}). \end{aligned}$$

Hence, $M(n)$ is $O(n^{\log_2 3})$. ∎

We now state, without proof, two pertinent theorems. Proofs may be found in [Kn97] or [Kr79].

Theorem 2.5. Given a positive number $\epsilon > 0$, there is an algorithm for multiplication of two n-bit integers using $O(n^{1+\epsilon})$ bit operations.

Note that Theorem 2.4 is a special case of Theorem 2.5 with $\epsilon = \log_2 3 - 1$, which is approximately 0.585.

Theorem 2.6. There is an algorithm to multiply two n-bit integers using $O(n \log_2 n \log_2 \log_2 n)$ bit operations.

Since $\log_2 n$ and $\log_2 \log_2 n$ are much smaller than n^ϵ for large numbers n, Theorem 2.6 is an improvement over Theorem 2.5. Although we know that $M(n)$ is $O(n \log_2 n \log_2 \log_2 n)$, for simplicity we will use the obvious fact that $M(n)$ is $O(n^2)$ in our subsequent discussions.

The conventional algorithm described in Section 2.2 performs a division of a $2n$-bit integer by an n-bit integer with $O(n^2)$ bit operations. However, the number of bit operations needed for integer division can be related to the number of bit operations needed for integer multiplication. We state the following theorem, which is based on an algorithm discussed in [Kn97].