Big-O Notation

Computer science uses a short form called **Big-O notation** to specify an algorithm's efficiency. Big-O notation syntax uses a capital letter O followed by a set of parentheses containing the equation specifying the algorithm's speed in relation to the size of the input set. For example, in Big-O notation a linear search is O(n) (pronounced **Order *n***). This means that execution speed is proportional to the number of elements.

In Big-O notation, the equation has no factors ($2n$) or constants (+ 5). For example, the expression O(n/2) is illegal. As well, all smaller elements of the speed are removed. For example, if the speed is proportional to $n^2 + n$, then we ignore the "+ n" because it becomes negligible for large enough values of n. The algorithm is O(n^2).

Here are a number of common Big-O equations.

- O(n^2): doubling the number of elements takes four times as long to execute.
- O(log n): execution time increases much more slowly than the number of elements. (For example, the algorithm would take only a third longer to process 1,000 elements than it took to process 100 elements).
- O(1): execution time does not depend upon the number of elements. An O(1) algorithm takes as long to operate on 100,000 elements as it takes to operate on 100 elements.

While Big-O notation does not allow you to decisively compare algorithms (for example, you cannot say that a O(log n) algorithm is always faster than an O(n) algorithm when performing a task on 100 elements), looking at a table to see how the algorithms compare with themselves for differing number of elements is useful.

	With 100 elements	**With 10,000 elements**	**Ratio**
O(1)	1	1	1
O(log *n*)	4.6	9.2	2
O(*n*)	100	10,000	100
O(*n* log *n*)	460	920,000	920
O(n^2)	10,000	100,000,000	10,000

As you can see, there are significant reasons to select an O(n log n) algorithm over an O(n^2) algorithm when the number of element being acted upon grows large enough.

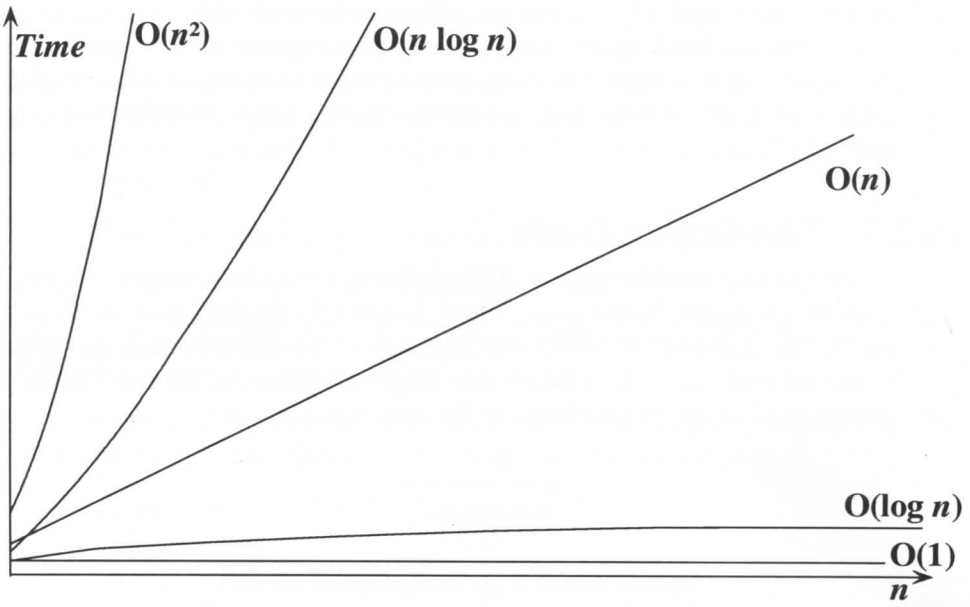

Figure 16.1 Different Algorithm Efficiencies

Note that expressing the efficiency of the algorithm in Big-O notation requires that there be some set of input to measure the speed against. For example, you cannot speak of `System.out.println` having an efficiency that is expressible in Big-O notation.

Big-O notation gives us a means for comparing algorithms for different uses, but it is not an exact measure of speed. For small enough values of n, other factors come into play.

Maximum Time

As well as comparing the average time an algorithm takes to perform a given task, it is also useful to obtain the maximum time that an algorithm takes. Often the speed of an algorithm is expressed as both the average time and the maximum time.

While discussing the average time a program takes to execute a task is useful, in some circumstances, it is more important to minimize the maximum time an algorithm could take, than to minimize the average time. For example, in an airport traffic control program, it is more important that no algorithm in the program take too long to execute. Otherwise, the plane may not receive orders in time.

16.2 Genericity and the Object Class

In the previous implementations of the search algorithms you had to create a new version of the search method every time you wanted to search through an array of a