

of the terms of the sequence. To see this, note that the initial value of *max* is the first term of the sequence; as successive terms of the sequence are examined, *max* is updated to the value of a term if the term exceeds the maximum of the terms previously examined. This (informal) argument shows that when all the terms have been examined, *max* equals the value of the largest term. (A rigorous proof of this requires techniques developed in Section 5.1.) The algorithm uses a finite number of steps, because it terminates after all the integers in the sequence have been examined. The algorithm can be carried out in a finite amount of time because each step is either a comparison or an assignment, there are a finite number of these steps, and each of these two operations takes a finite amount of time. Finally, Algorithm 1 is general, because it can be used to find the maximum of any finite sequence of integers. ◀

Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts. For instance, a program that checks the spelling of words searches for them in a dictionary, which is just an ordered list of words. Problems of this kind are called **searching problems**. We will discuss several algorithms for searching in this section. We will study the number of steps used by each of these algorithms in Section 3.3.

The general searching problem can be described as follows: Locate an element x in a list of distinct elements a_1, a_2, \dots, a_n , or determine that it is not in the list. The solution to this search problem is the location of the term in the list that equals x (that is, i is the solution if $x = a_i$) and is 0 if x is not in the list.

THE LINEAR SEARCH The first algorithm that we will present is called the **linear search**, or **sequential search**, algorithm. The linear search algorithm begins by comparing x and a_1 . When $x = a_1$, the solution is the location of a_1 , namely, 1. When $x \neq a_1$, compare x with a_2 . If $x = a_2$, the solution is the location of a_2 , namely, 2. When $x \neq a_2$, compare x with a_3 . Continue this process, comparing x successively with each term of the list until a match is found, where the solution is the location of that term, unless no match occurs. If the entire list has been searched without locating x , the solution is 0. The pseudocode for the linear search algorithm is displayed as Algorithm 2.



ALGORITHM 2 The Linear Search Algorithm.

```

procedure linear search( $x$ : integer,  $a_1, a_2, \dots, a_n$ : distinct integers)
 $i := 1$ 
while ( $i \leq n$  and  $x \neq a_i$ )
     $i := i + 1$ 
if  $i \leq n$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript of the term that equals  $x$ , or is 0 if  $x$  is not found}

```

THE BINARY SEARCH We will now consider another searching algorithm. This algorithm can be used when the list has terms occurring in order of increasing size (for instance: if the terms are numbers, they are listed from smallest to largest; if they are words, they are listed in lexicographic, or alphabetic, order). This second searching algorithm is called the **binary search algorithm**. It proceeds by comparing the element to be located to the middle term of the list. The list is then split into two smaller sublists of the same size, or where one of these smaller lists has one fewer term than the other. The search continues by restricting the search to the appropriate sublist based on the comparison of the element to be located and the middle term. In Section 3.3, it will be shown that the binary search algorithm is much more efficient than the linear search algorithm. Example 3 demonstrates how a binary search works.



EXAMPLE 3 To search for 19 in the list

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22,

first split this list, which has 16 terms, into two smaller lists with eight terms each, namely,


1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22.

Then, compare 19 and the largest term in the first list. Because $10 < 19$, the search for 19 can be restricted to the list containing the 9th through the 16th terms of the original list. Next, split this list, which has eight terms, into the two smaller lists of four terms each, namely,

12 13 15 16 18 19 20 22.

Because $16 < 19$ (comparing 19 with the largest term of the first list) the search is restricted to the second of these lists, which contains the 13th through the 16th terms of the original list. The list 18 19 20 22 is split into two lists, namely,

18 19 20 22.

Because 19 is not greater than the largest term of the first of these two lists, which is also 19, the search is restricted to the first list: 18 19, which contains the 13th and 14th terms of the original list. Next, this list of two terms is split into two lists of one term each: 18 and 19. Because $18 < 19$, the search is restricted to the second list: the list containing the 14th term of the list, which is 19. Now that the search has been narrowed down to one term, a comparison is made, and 19 is located as the 14th term in the original list. 

We now specify the steps of the binary search algorithm. To search for the integer x in the list a_1, a_2, \dots, a_n , where $a_1 < a_2 < \dots < a_n$, begin by comparing x with the middle term a_m of the list, where $m = \lfloor (n + 1)/2 \rfloor$. (Recall that $\lfloor x \rfloor$ is the greatest integer not exceeding x .) If $x > a_m$, the search for x is restricted to the second half of the list, which is $a_{m+1}, a_{m+2}, \dots, a_n$. If x is not greater than a_m , the search for x is restricted to the first half of the list, which is a_1, a_2, \dots, a_m .

The search has now been restricted to a list with no more than $\lceil n/2 \rceil$ elements. (Recall that $\lceil x \rceil$ is the smallest integer greater than or equal to x .) Using the same procedure, compare x to the middle term of the restricted list. Then restrict the search to the first or second half of the list. Repeat this process until a list with one term is obtained. Then determine whether this term is x . Pseudocode for the binary search algorithm is displayed as Algorithm 3.

ALGORITHM 3 The Binary Search Algorithm.

```
procedure binary search ( $x$ : integer,  $a_1, a_2, \dots, a_n$ : increasing integers)
 $i := 1$  { $i$  is left endpoint of search interval}
 $j := n$  { $j$  is right endpoint of search interval}
while  $i < j$ 
     $m := \lfloor (i + j)/2 \rfloor$ 
    if  $x > a_m$  then  $i := m + 1$ 
    else  $j := m$ 
if  $x = a_i$  then  $location := i$ 
else  $location := 0$ 
return  $location$  { $location$  is the subscript  $i$  of the term  $a_i$  equal to  $x$ , or 0 if  $x$  is not found}
```