## 2.6 SUBALGORITHMS

A *subalgorithm* is a complete and independently defined algorithmic module which is used (or *invoked* or *called*) by some main algorithm or by some other subalgorithm. A subalgorithm receives values, called *arguments*, from an originating (calling) algorithm; performs computations; and then sends back the result to the calling algorithm. The subalgorithm is defined independently so that it may be called by many different algorithms or called at different times in the same algorithm. The relationship between an algorithm and a subalgorithm is similar to the relationship between a main program and a subprogram in a programming language.

The main difference between the format of a subalgorithm and that of an algorithm is that the subalgorithm will usually have a heading of the form

$$\text{NAME}(\text{PAR}_1, \text{PAR}_2, \ldots, \text{PAR}_K)$$

Here NAME refers to the name of the subalgorithm which is used when the subalgorithm is called, and $\text{PAR}_1, \text{PAR}_2, \ldots, \text{PAR}_K$ refer to parameters which are used to transmit data between the subalgorithm and the calling algorithm.

Another difference is that the subalgorithm will have a Return statement rather than an Exit statement; this emphasizes that control is transferred back to the calling program when the execution of the subalgorithm is completed.

Subalgorithms fall into two basic categories: *function* subalgorithms and *procedure* subalgorithms. The similarities and differences between these two types of subalgorithms will be examined below by means of examples. One major difference between the subalgorithms is that the function subalgorithm returns only a single value to the calling algorithm, whereas the procedure subalgorithm may send back more than one value.

**EXAMPLE 2.9**

The following function subalgorithm MEAN finds the average AVE of three numbers A, B and C.

**Function 2.5:** MEAN(A, B, C)

1. Set AVE := (A + B + C)/3.
2. Return(AVE).

Note that MEAN is the name of the subalgorithm and A, B and C are the parameters. The Return statement includes, in parentheses, the variable AVE, whose value is returned to the calling program.

The subalgorithm MEAN is invoked by an algorithm in the same way as a function subprogram is invoked by a calling program. For example, suppose an algorithm contains the statement

$$\text{Set TEST} := \text{MEAN}(T_1, T_2, T_3)$$

where $T_1$, $T_2$ and $T_3$ are test scores. The argument values $T_1$, $T_2$ and $T_3$ are transferred to the parameters A, B, C in the subalgorithm, the subalgorithm MEAN is executed, and then the value of AVE is returned to the program and replaces MEAN($T_1$, $T_2$, $T_3$) in the statement. Hence the average of $T_1$, $T_2$ and $T_3$ is assigned to TEST.

Hence,

$$C(n) = MT = O(\log_b n)$$

Accordingly,

2.8 (*a*) Write a procedure FIND(DATA, N, LOC1, LOC2) which finds the location LOC1 of the largest element and the location LOC2 of the second largest element in an array DATA with $n > 1$ elements.

(*b*) Why not let FIND also find the values of the largest and second largest elements?

(*a*) The elements of DATA are examined one by one. During the execution of the procedure, FIRST and SECOND will denote, respectively, the values of the largest and second largest elements that have already been examined. Each new element DATA[K] is tested as follows. If

$$\text{SECOND} \leq \text{FIRST} < \text{DATA[K]}$$

then FIRST becomes the new SECOND element and DATA[K] becomes the new FIRST element. On the other hand, if

$$\text{SECOND} < \text{DATA[K]} \leq \text{FIRST}$$

then DATA[K] becomes the new SECOND element. Initially, set FIRST := DATA[1] and SECOND := DATA[2], and check whether or not they are in the right order. A formal presentation of the procedure follows:

**Procedure P2.8:** FIND(DATA, N, LOC1, LOC2)

1. Set FIRST := DATA[1], SECOND := DATA[2], LOC1 := 1, LOC2 := 2.
2. [Are FIRST and SECOND initially correct?]
   If FIRST < SECOND, then:
   (*a*) Interchange FIRST and SECOND,
   (*b*) Set LOC1 := 2 and LOC2 := 1.
   [End of If structure.]
3. Repeat for K = 3 to N:
   If FIRST < DATA[K], then:
   (*a*) Set SECOND := FIRST and FIRST := DATA[K].
   (*b*) Set LOC2 := LOC1 and LOC1 := K.
   Else if SECOND < DATA[K], then:
   Set SECOND := DATA[K] and LOC2 := K.
   [End of If structure.]
   [End of loop.]
4. Return.