3.1.2.2. Step-Count Method:

As noted in some of the examples on operations count, the operation-count method of estimating time complexity omits accounting for the time spent on all but the chosen operations. In the step-count method, we attempt to account for the time spent in all parts of the program/function. The step-count is a function of the instance characteristics (e.g., the number of inputs, the number of outputs, the magnitudes of inputs and outputs). The number of steps is computed as a function of some subset of these.

The step-count method is a measure to estimate the time complexity where the number of steps any program statement is to be assigned depends on the nature of that statement. The following discussion considers the various statement types that can appear in a Pascal program and states the complexity of each in terms of the number of steps.

- Comments: comments are nonexecutable statements and have a step count of zero.
- Declarative statements: this includes all statements of type c**onst**, **label**, **type**, and **var**. these counts as zero steps as these are either nonexecutable or their cost may be lumped into the cost of invoking the procedure/function they are associated with.
- Expressions and assignment statements: most expressions have a step count of one. The exceptions are expressions that contain function calls. In this case, we need to determine the cost of invoking the functions. This cost can be large if the function employ many-element value parameters as the values of all actual parameters need to be assigned to the formal parameters. This is discussed future under procedure and function invocation. When the expression contains functions, the step count is the sum of the step counts assignable to each function invocation.

The assignment statement <variable> := <expr> has a step count equal to that of <expr> unless the size of <variable> is a function of the instance characteristics. In this later case, the step count is the size of <variable> plus the step count of <expr>. For example, the assignment a := b where a and b are of type ElementList has a step count equal to the size of ElementList.

- Iteration statements: this class of statements includes the For, While, and Until statements. We shall consider the step counts only for the control part of these statements. These have the form:

```
For i:= <expr> to <expr> do
For i:= <expr> downto <exprl> do
While <expr> do
Until <expr>;
```

Each execution of the control part of a while and until statement will be given a step count equal to the number of step counts assignable to <expr>. The step count for each execution of the control part of a For statement is one, unless the counts attributable to <expr> and <exprl> are a function of the instance characteristics. In this latter case, the first execution of the control part of the For has a step count equal to the sum of the counts for <expr> and <exprl> (note that these expressions are computed only when the loop is started). Remaining executions of the For have a step counts of one.

- Case statement: this statement consists of a header followed by one or more sets of condition and statement pairs.

```
Case <expr> of
  Cond1: <statement1>
  Cond2: <statement2>
      .
      .
      .
  Else: <statement>
End;
```

The cost of the header Case <expr> of is given a cost equal to that assignable to <expr>. The cost of each following condition-statement pair is the cost of this condition plus that of all preceding conditions plus that of this statement.

- If-Then-Else statements: the if-then-else statement consists of three parts:

  If <expr>
  Then <statement 1>
  Else <statement 2>

  Each part is assigned the number of steps corresponding to <expr>, <statement 1>, and <statement 2> respectively, note that if the else clause is absent, then no cost is assigned to it.
- Procedure and Function invocation: all invocations of procedures and functions count as one step unless the invocation involves value parameters whose size depends on the instance characteristics. In this latter case, the count is the sum of the size of these value parameters. In case the procedure/ function being invoked is recursive, then we must also consider the local variable in the procedure or function being invoked. The sizes of local variables that are characteristic dependent need to be added into the step count.
- Begin, End, With, and Repeat statements: each With statement counts as one step. Each Begin, End, and Repeat statement counts as zero steps.
- Procedure and function statements: these count as zero steps as their cost has already been assigned to the invoking statements.
- Goto statement: this has a step count of one.

Example 1: Find the space and the time complexities for the following function using step count method.

```
Function Abc(a, b, c : real) : real;
Begin
```

```
    Abc := a + b + b * c + (a + b – c) / (a + b) + 4;
End;
```

Solution:

Space complexity: One word of type real is a adequate to store the values of each of a, b, c, 4, and Abc. We see that the space needed by function Abc is independent of the instance characteristics. Consequently,

$S_{Abc}(a, b, c) = 0$

Time complexity:

$T_{Abc}(a, b, c) = 1$

Example 2: Find the space and the time complexities for the following function using step count method.

```
Function sum1(n : integer) : integer;
var  k, s : integer;
Begin
    s := 0;
    For  k := 1  to  n  do
        s  :=  s + k;
    sum1  :=  s;
End;
```

Solution:

Space complexity: The function sum1 requires six words of type integer to store the values of each (n, k, s, sum1, and constants 0, 1). We see that the space needed by function sum1 (12 Bytes) is independent of the instance characteristics. Consequently,

$S_{sum1}(n) = 0$

Time complexity:

$T_{sum1}(n) = 1 + (n+1) + n + 1 = 2n + 3$

Example 3: Find the space and the time complexities for the following function using step count method.

```
Function sum2( a: ElemList ; n : integer) : real;
var  k : integer;
       s: real;
Begin
    s := 0;
    For  k := 1  to  n  do
         s  :=  s + a[k];
    sum1  :=  s;
End;
```

Solution:

Space complexity: The function sum2 requires six spaces (four of type integer and two of type real) to store the values of each (n, k, s, sum2,and constants 0, 1). The space needed by a is the space needed by variables of type ElemList. This is equal to MaxSize. We see that the space needed by function sum2 is

$S_{sum2}(n)$ = 16 Bytes + Maxsize

Or $S_{sum2}(n) \geq n$

Note: if we change the formal parameter a from value to reference (or Var), only the address of the actual parameter gets transferred to the function and the space needed by the function is independent of instance characteristics (n), in this case $S_{sum2}(n) = 0$.

Time complexity:

$T_{sum2}(n) = 1 + (n+1) + n + 1 = 2n + 3$