CHAPTER 8

Algorithms

(Solutions to Review Questions and Problems)

Review Questions

Q8-1. An algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

Q8-3. Universal Modeling Language (UML) is a pictorial representation of an algorithm. It hides all of the details of an algorithm in an attempt to give the big picture; it shows how the algorithm flows from beginning to end.

Q8-5. A sorting algorithm arranges data according to their values.

Q8-7. A searching algorithm finds a particular item (target) among a list of data.

Q8-9. An algorithm is iterative if it uses a loop construct to perform a repetitive task.

Problems

P8-1. The value of ***Sum*** after each iteration is shown in Table 8.1.

Table 8.1 *Solution to P8-1*

Iteration	*Data item*	*Sum* = **0**
1	20	**Sum = 0** + 20 = 20
2	12	**Sum = 20** + 12 = 32
3	70	**Sum = 32** + 70 = 102
4	81	**Sum = 102** + 81 = 183
5	45	**Sum = 183** + 45 = 228
6	13	**Sum = 228** + 13 = 241
7	81	**Sum = 241** + 81 = 322
After exiting the loop		**Sum = 322**

P8-3. The value of ***Largest*** after each iteration is shown in Table 8.2.

Table 8.2 *Solution to P8-3*

Iteration	*Data item*	*Largest* = $-\infty$
1	18	Largest = **18**
2	12	Largest = **18**
3	8	Largest = **18**
4	20	Largest = **20**
5	10	Largest = **10**
6	32	Largest = **32**
7	5	Largest = **32**
After exiting the loop		Largest = **32**

P8-5. The status of the list and the location of the wall after each pass is shown in Table 8.3.

Table 8.3 *Solution to P8-5*

Pass	List							
	14	7	23	31	40	56	78	9	2
1	2	7	23	31	40	56	78	9	14
2	2	7	23	31	40	56	78	9	14
3	2	7	9	31	40	56	78	23	14
4	2	7	9	14	40	56	78	23	31
5	2	7	9	14	23	56	78	40	31
6	2	7	9	14	23	31	78	40	56
7	2	7	9	14	23	78	40	78	56
8	2	7	9	14	23	78	40	56	78

P8-7. The status of the list and the location of the wall after each pass is shown in Table 8.4.

Table 8.4 *Solution to P8-7*

Pass	List								
	14	7	23	31	40	56	78	9	2
1	7	14	23	31	40	56	78	9	2
2	7	14	23	31	40	56	78	9	2
3	7	14	23	31	40	56	78	9	2
4	7	14	23	31	40	56	78	9	2
5	7	14	23	31	40	56	78	9	2
6	7	14	23	31	40	56	78	9	2
7	7	9	14	23	31	40	56	78	2
8	2	7	9	14	23	31	40	56	78

P8-9. The status of the list and the location of the wall after each pass is shown in Table 8.5 .

Table 8.5 *Solution to P8-9*

Pass	List							
	7	8	26	44	13	23	57	98
1	7	8	13	26	44	23	57	98
2	7	8	13	23	26	44	57	98
3	7	8	13	23	26	44	57	98

P8-11. The binary search for this problem follows Table 8.6. The target (88) is found at index $i = 7$.

Table 8.6 *Solution to P8-11*

first	last	mid	1	2	3	4	5	6	7	8	
1	8	4	8	13	17	26	44	56	88	97	target > 44
5	8	6					44	56	88	97	target > 56
7	8	7							88	97	target = 88

P8-13. The sequential search follows Table 8.7. The target (20) is not found.

Table 8.7 *Solution to P8-13*

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	
	4	21	36	14	62	91	8	22	7	81	77	10	
1	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 4
2	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 21
3	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 36
4	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 14
5	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 62
6	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 91
7	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 8
8	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 22
9	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 7
10	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 81
11	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 77
12	4	21	36	14	62	91	8	22	7	81	77	10	target ≠ 10
	Target is not in the list.												

P8-15. Iterative evaluation of (6!) = 720 is shown in Table 8.8.

Table 8.8 *Solution to P8-15*

i	*Factorial*
1	F = **1**
2	F = **1** × 2 = 2
3	F = **2** × 3 = 6
4	F = **6** × 4 = 24
5	F = **24** × 5 = 120
6	F = **120** × 6 = 720
After exiting the loop	F = **720**

P8-17. The algorithm in Table 8.9 shows the pseudocode for evaluating gcd.

Table 8.9 *Solution to P8-17*

```
Algorithm: gcd (x, y)
Purpose: Find the greatest common devisor of two numbers
Pre: x, y
Post: None
Return: gcd (x, y)
{
     If (y = 0)   return x
     else         return gcd (y, x mod y)
}
```

P8-19. Table 8.10 shows the pseudocode for evaluating combination.

Table 8.10 *Solution to P8-19*

```
Algorithm: Combination (n, k)
Purpose: It finds the combination of n objects k at a time
Pre: Given: n and k
Post: None
Return: C (n, k)
{
     If (k = 0 or n = k)    return 1
     else                        return C (n − 1, k) + C(n − 1, k − 1)
}
```

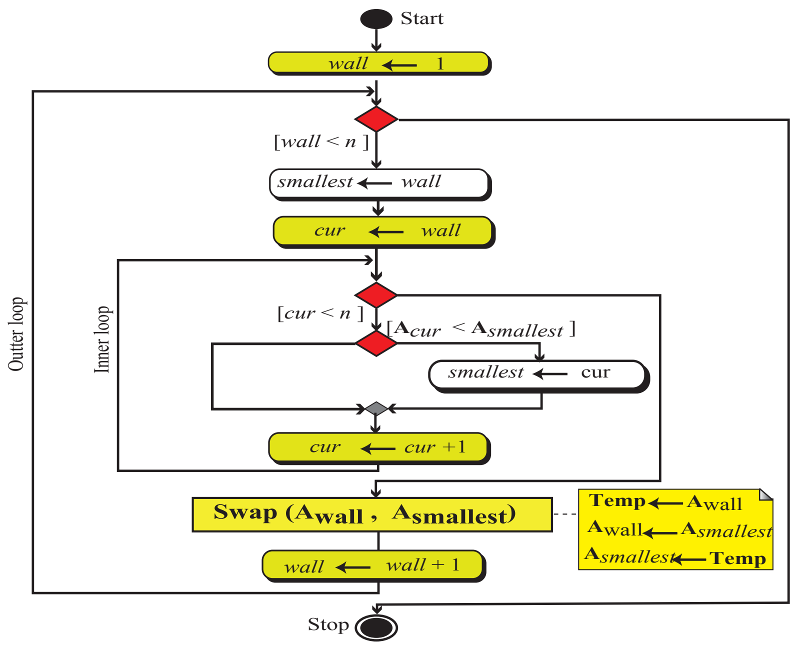
P8-21. Table 8.11 shows the pseudocode for evaluating Fibonacci sequence.

Table 8.11 *Solution to P8-21*

```
Algorithm: Fibonacci (n)
Purpose: It finds the elements of Fibonacci sequence
Pre: Given: n
Post: None
Return: Fibonacci (n)
{
    If (n = 0)    return 0
    If (n = 1)    return 1
    else          return (Fibonacci (n − 1) + Fibonacci (n − 2))
}
```

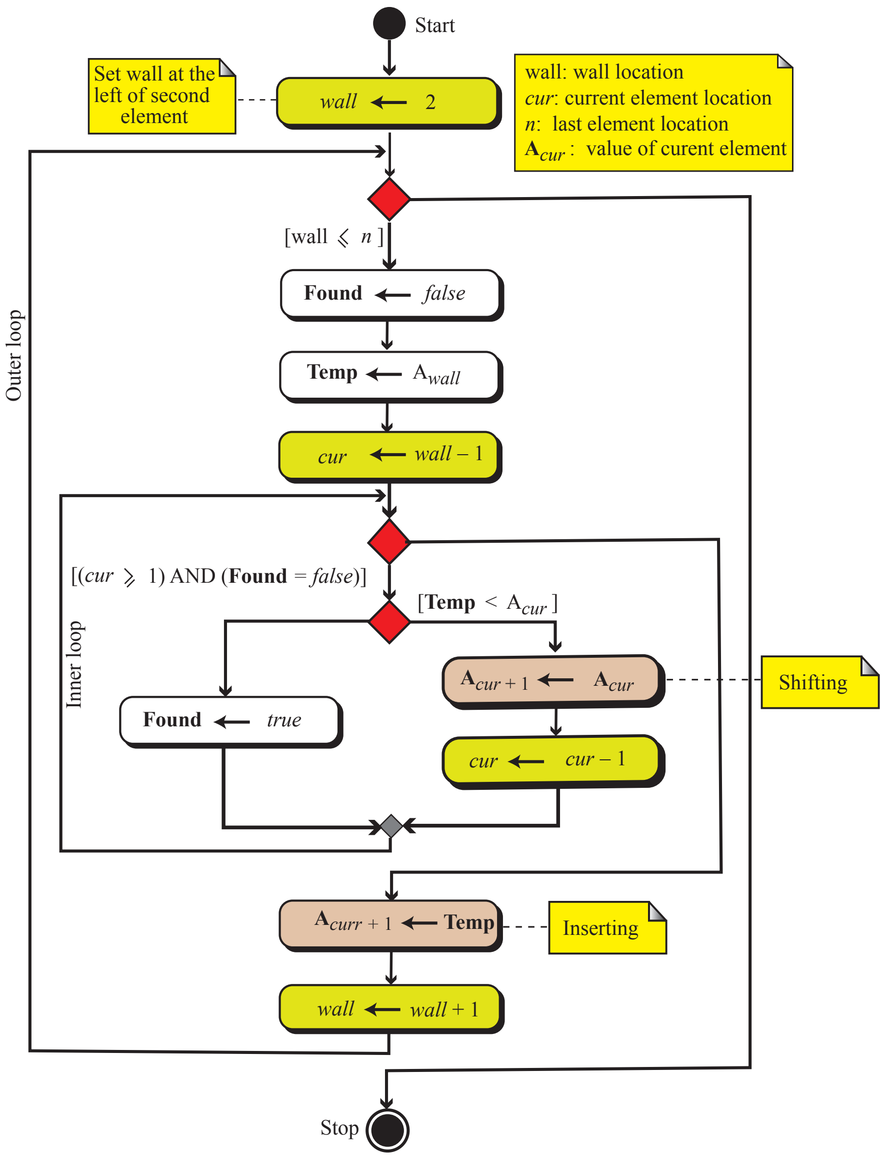
P8-23. The UML for the selection sort is shown in Figure 8.1.

Figure 8.1 *Solution to P8-23*



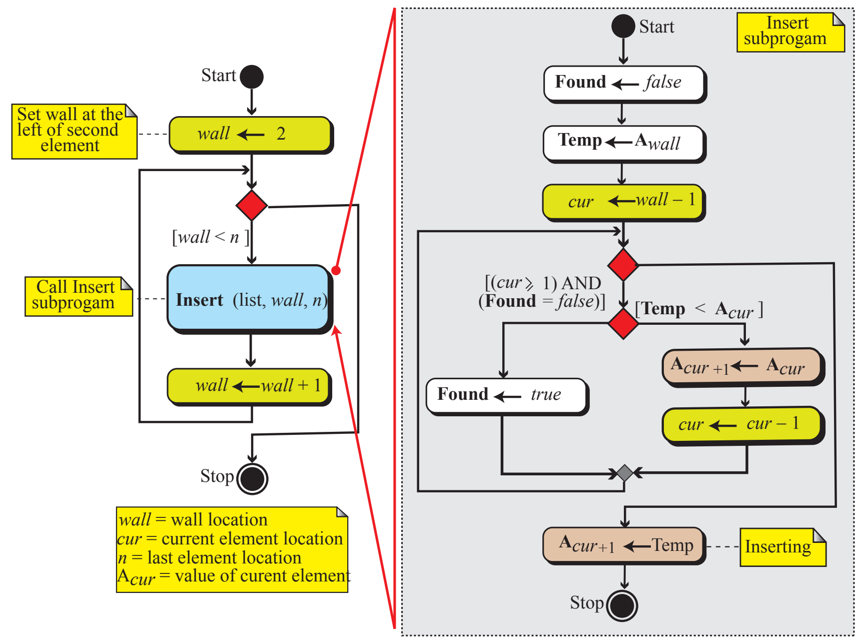
P8-25. The UML for insertion sort is shown in Figure 8.2. We have used a true/false value, **Found**, to stop shifting when the location of the insertion is found.

Figure 8.2 *Solution to P8-25*



P8-27. The UML is shown in Figure 8.3. The program calls the Insert subprogram

Figure 8.3 *Solution to P8-27*



P8-29. Table 8.12 shows the pseudocode for finding the product of integers.

Table 8.12 *Solution to P8-29*

Algorithm: Product (list)
Purpose: It finds the product of integers
Pre: Given: A list of integers
Post: None
Return: Product of the integers

```
{
    product ← 1
    while (more integer to multiply)
    {
        get next integer
        product ← product × (next integer)
    }
    return product
}
```

P8-31. Table 8.13 shows the pseudocode for the selection sort routine that uses a subprogram. Finding the smallest numbers in the unsorted side is performed by a subalgorithm called FindSmallest.

Table 8.13 *Solution to P8-31*

```
Algorithm: SelectionSort (list, n)
Purpose: to sort a list using selection sort method
Pre: Given: A list of numbers
Post: None
Return:
{
    wall ← 1            // Set wall at the left of first element
    while (wall < n)
    {
        smallest ← FindSmallest (list, wall, n)   // Call the FindSmallest
        Temp ← A_wall          // The next three lines perform swapping
        A_wall ← A_smallest
        A_smallest ← Temp
        wall ← wall + 1        // Move wall one element to the right
    }
    return SortedList
}

FindSmallest (list, wall, n)
{
    smallest ← wall   // Assume the first element is the smallest one
    cur ← wall        // The current item is the one left to the wall
    while (cur < n)
    {
        if (A_cur < A_smallest)
        smallest ← cur
        cur ← cur + 1    // Move the current element
    }
    return smallest
}
```

P8-33. Table 8.14 shows the pseudocode for the bubble sort routine that uses a subprogram. The bubbling of the numbers in the unsorted side is performed by a subalgorithm called Bubble.

Table 8.14 *Solution to P8-33*

```
Algorithm: BubbleSort (list, n)
Purpose: to sort a list using bubble sort
Pre: Given: A list of N numbers
Post: None
Return:
{
```

Table 8.14 *Solution to P8-33*

```
    wall ← 1        // Place the wall at the leftmost end of the list
    while (wall < n)
    {
        Bubble (list, wall, n)     // Move the wall one place to the right
        wall ← wall + 1
    }
    return SortedList
}

Bubble (list, wall, n)
{
    cur ← n       // Start from the end of the list
    while (cur > wall))      // Bubble the smallest to the left of unsorted list
    {
        if (A_cur < A_cur - 1)     // Bubble one location to the left
        {
            Temp ← A_cur
            A_cur ← A_cur-1
            A_cur-1 ← Temp
        }
        cur ← cur − 1
    }
}
```

P8-35. Table 8.15 shows the pseudocode for the insertion sort routine that uses a subprogram named Insert.

Table 8.15 *Solution to P8-35*

Algorithm: InsertionSort(list, *n*)
Purpose: to sort a list using insertion sort
Pre: Given: A list of *N* numbers
Post: None
Return: Sorted list

```
{
    wall ← 2
    while (wall < n)
    {
        Insert (list, wall, n)
        wall ← wall + 1
    }
}

Insert (list, wall, n)
{
    Found ← false
    Temp ← A_wall
```

Table 8.15 *Solution to P8-35*

```
    cur ← wall − 1
    while ((cur ≥ 1) AND Found = false))
    {
        if (Temp < A_cur)
        {
            A_cur + 1 ← A_cur
            cur ← cur − 1
        }
        else    Found ← true
    }
    A_cur +1 ← Temp
}
```

P8-37. Table 8.16 shows the pseudocode for binary search.

Table 8.16 *Solution to P8-37*

```
Algorithm: BinarySearch (list, target, n)
Purpose: Apply a binary search a list of n sorted numbers
Pre: list, target, n
Post: None
Return: flag, i
{
    flag ← false
    first ← 1
    last ← n
    while (first ≤ last)
    {
        mid = (first + last) / 2
        if (target < A_mid)    Last ← mid − 1    // A_i is the ith number in the list
        if (target > A_mid)    first ← mid + 1
        if (target = A_mid)    first ← Last + 1    // target is found
    }
    if (target > A_mid)    i ← mid + 1
    if (x ≤ A_mid)    i ← mid
    if (x = A_mid)    flag ← true
    return (flag, i)
    // If flag is false, i is the location of the smallest
    // If flag is true, i is the location of the target
}
```

P8-39. Table 8.17 shows the pseudocode.

Table 8.17 *Solution to P8-39*

```
Algorithm: Power (x, n)
Purpose: Find x^n where x and n are integers
Pre: x, n
```

Table 8.17 *Solution to P8-39*

```
Post: None
Return: x^n
{
    z ← 1
    while (n ≠ 1)
    {
        z ← z × x
        n ← n − 1
    }
    return z
}
```