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 below:

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 below:

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 of the selection sort algorithm is shown below:

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 of the insertion sort algorithm is shown below:

Pass	List								
	14	7	23	31	40	56	78	9	2
1	***7***	***14***	23	31	40	56	78	9	2
2	***7***	***9***	***14***	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 three passes of bubble sort is shown below :

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 the table shown below. The target (88) is found at index $i = 7$.

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 the table shown below. The target (20) is not found.

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 below.:

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. Algorithm P8-17 shows the pseudocode for evaluating *gcd*.

Algorithm P8-17 *Evaluating gcd*

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

P8-19. Algorithm P8-19 shows the pseudocode for evaluating combination.

Algorithm P8-19 *Combination*

```
Algorithm: Combination (n, k)
Purpose: Finds the combination of n objects k at a time
Pre: 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)
}
```

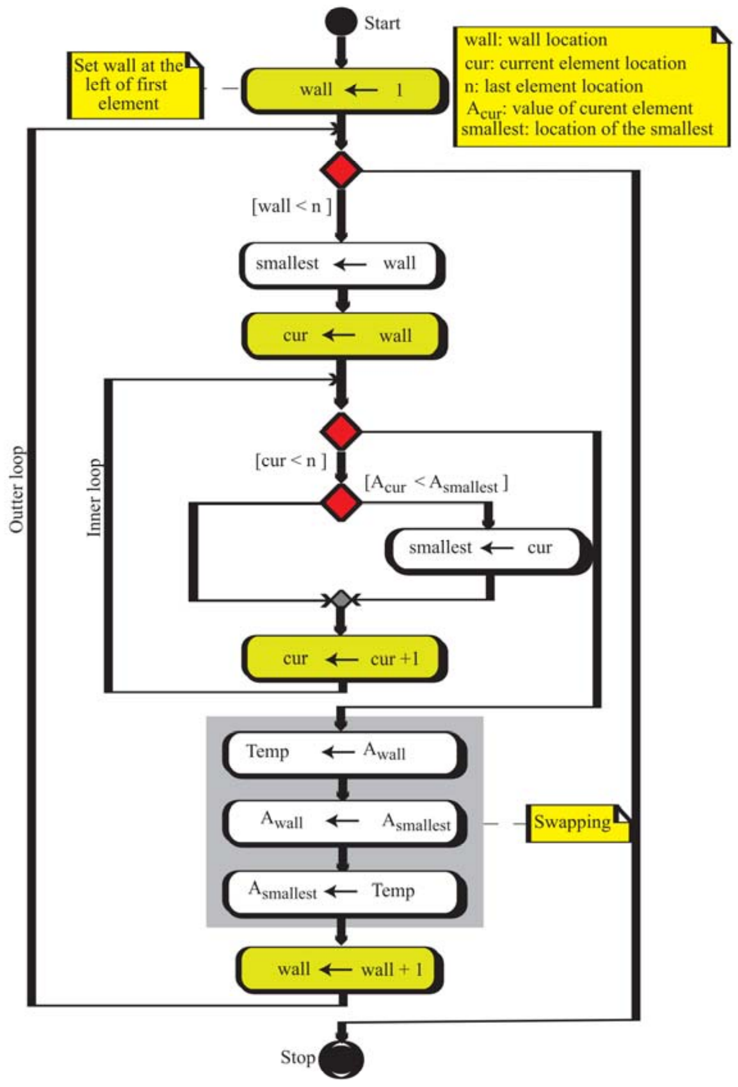
P8-21. Algorithm P8-21 shows the pseudocode for evaluating Fibonacci sequence.

Algorithm P8-21 *Fibonacci*

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

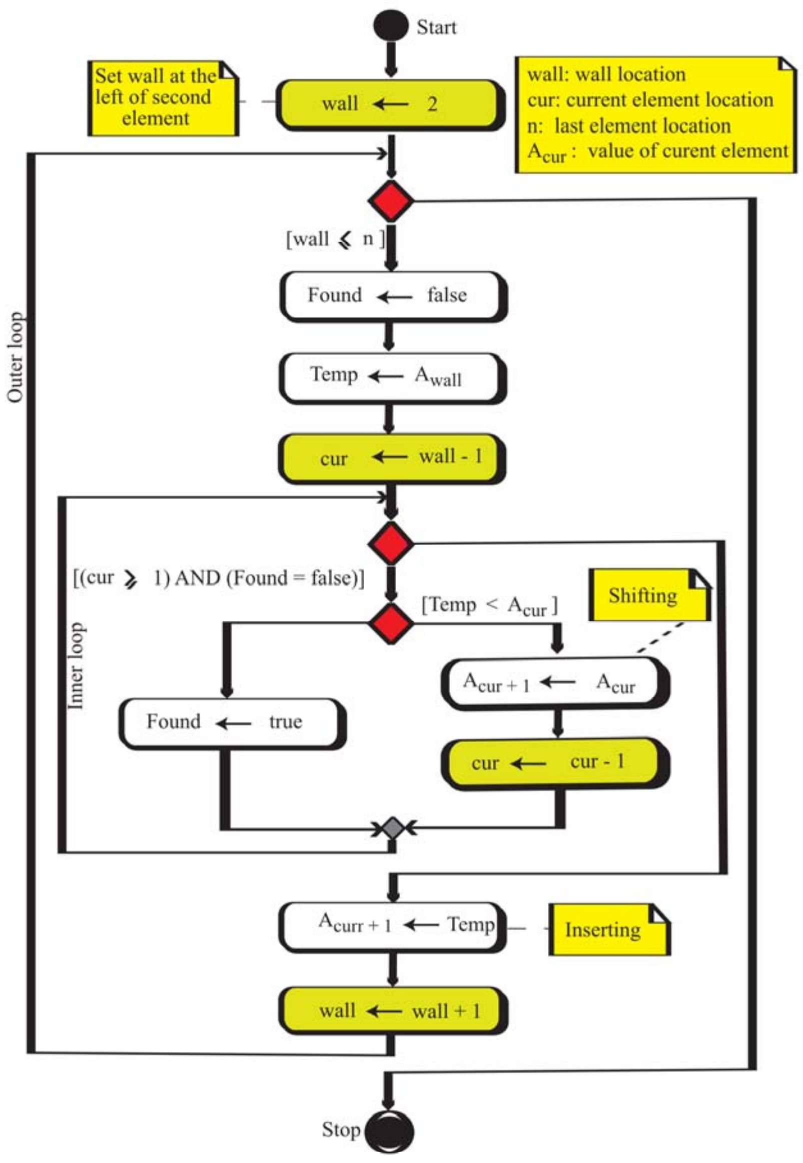
P8-23. The UML for the selection sort is shown in Figure P8-23. The inner loop finds the location of the smallest element in the unsorted list. The three instructions after the inner loop swap this element with the first element in the unsorted list. Before searching for the smallest element, we assume that the first element in the unsorted list is the smallest one.

Figure P8-23 *Selection sort*



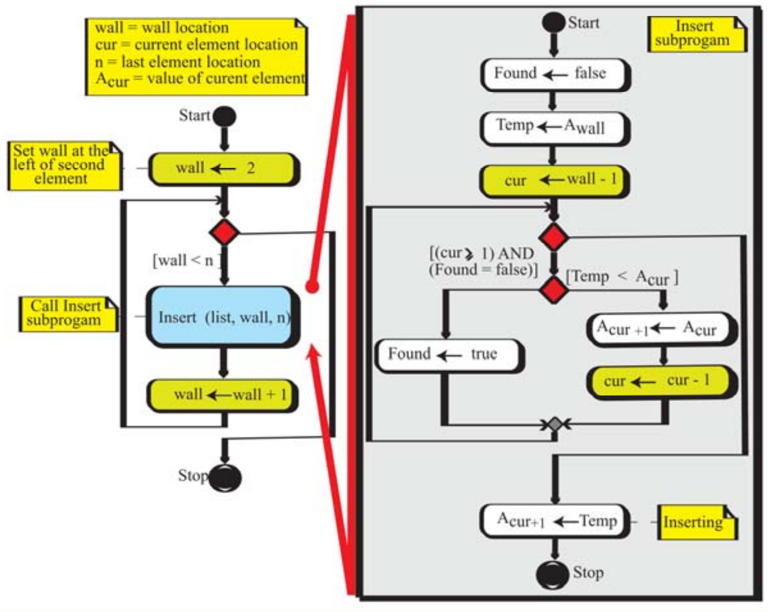
P8-25. The UML for insertion sort is shown in Figure P8-25. The inner loop finds the location of the insertion. We shift the elements in the sorted sublist until we find the appropriate location to insert the element. When the algorithm exits the inner loop, insertion can be done. We have used a true/false value, **Found**, to stop shifting when the location of the insertion is found.

Figure P8-25 *Insertion sort*



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

Figure P8-27 *Insertion sort that calls insert subprogram*



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

Algorithm P8-29 *Product*

```
Algorithm: Product (list)
Purpose: It finds the product of integers
Pre: 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. We use two algorithms for this problem.

a. Algorithm P8-31a is the selection sort algorithm. To do its task, it needs another algorithm defined in part b.

Algorithm P8-31a *Selection sort algorithm calling smallest subroutine*

```
Algorithm: SelectionSort (list, n)
Purpose: to sort a list using selection sort method
Pre: 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 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
}
```

b. Algorithm P8-31b is find-smallest routing that is called by the selection sort algorithm.

Algorithm P8-31b *Smallest subroutine*

```
Algorithm: FindSmallest (list, wall, n)
Purpose: To find the smallest number in an unsorted list
Pre: A list of numbers
Post: None
Return: The location of the smallest element in the unsorted list
{
    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. The solution is made of two parts: the main routine and a subroutine.

a. Algorithm P8-33a shows the main routine: bubble sort.

Algorithm P8-33a *Bubble sort algorithm calling bubble subroutine*

```
Algorithm: BubbleSort (list, n)
Purpose: to sort a list using bubble sort
Pre: A list of N numbers
Post: None
Return:
{
    wall ← 1            // Place the wall at the left-most end of the list
    while (wall < n)
    {
        Bubble (list, wall, n)
        wall ← wall + 1     // Move the wall one place to the right
    }
    return SortedList
}
```

b. Algorithm P8-33b shows the subroutine, bubble, called by the main routine.

Algorithm P8-33b *Bubble subroutine*

```
Algorithm: Bubble (list, wall, n)
Purpose: to bubble an unsorted list
Pre: A list, N and location of the wall
Post: None
Return:
{
    cur ← n                     // Start from the end of the list
    while (cur > wall))         // Bubble the smallest to the left of 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.

a. Algorithm P8-35a shows the pseudocode for the insertion sort routine that uses the insert routine

Algorithm P8-35a *Insertion Sort Algorithm calling Insert routine*

```
Algorithm: InsertionSort(list, n)
Purpose: to sort a list using insertion sort
Pre: A list of N numbers
Post: None
Return: Sorted list
{
    wall ← 2
    while (wall < n)
    {
        Insert (list, wall, n)        // Calls the insert routine
        wall ← wall + 1
    }
}
```

b. Algorithm P8-35b shows the subroutine insert used by the insertion sort algorithm.

Algorithm P8-35b *Insert subroutine*

```
Algorithm: Insert(list, wall, n)
Purpose: Insert a number in a sorted list
Pre: A of numbers and location of wall
Post: None
Return:
{
    Found ← false
    Temp ← $A_{wall}$
    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. Algorithm P8-37 shows the pseudocode for binary search.

Algorithm P8-37 *Binary search*

```
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 shows the status of the search
{
    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 = $A_{mid}$)        // target is found
            first ← Last + 1
    }
    if (target > $A_{mid}$)
        i = mid + 1
    if (x ≤ $A_{mid}$)
        i = mid
    if (x = $A_{mid}$)
        flag ← true
    return (flag, i)
}
```

P8-39. Algorithm P8-39 shows the pseudocode for finding the power of an integer.

Algorithm P8-39 *Power of an integer*

```
Algorithm: Power ($x$, $n$)
Purpose: Find $x^n$ where $x$ and $n$ are integers
Pre: $x$, $n$
Post: None
Return: $x^n$
{
    $z \leftarrow 1$
    while ($n \neq 1$)
    {
        $z \leftarrow z \times x$
        $n \leftarrow n - 1$
    }
    return $z$
}
```