CHAPTER 8

Algorithms

(Solutions to Odd-Numbered Problems)

Review Questions

1. An algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.
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.
5. A sorting algorithm arranges data according to their values.
7. A searching algorithm finds a particular item (target) among a list of data.
9. An algorithm is iterative if it uses a loop construct to perform a repetitive task.

Multiple-Choice Questions

11. d 13. b 15. c 17. c 19. b 21. d
23. a 25. b 27. b

Exercises

29. 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**

31. 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**

33. The status of the list and the location of the wall after each pass 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

35. The status of the list and the location of the wall after each pass 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	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

37. The status of the list and the location of the wall after each pass 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

39. 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

41. 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.												

43. 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**

45. Algorithm S8.45 shows the pseudocode for evaluating gcd.

Algorithm S8.45 *Exercise 45*

```
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)
}
```

47. Algorithm S8.47 shows the pseudocode for evaluating combination.

Algorithm S8.47 *Exercise 47*

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 $(k = 0 \text{ or } n = k)$ **return** 1

else **return** $C(n - 1, k) + C(n - 1, k - 1)$

}

49. Algorithm S8.49 shows the pseudocode for evaluating Fibonacci sequence.

Algorithm S8.49 *Exercise 49*

Algorithm: **Fibonacci**(n)

Purpose: It finds the elements of Fibonacci sequence

Pre: Given: n

Post: None

Return: **Fibonacci** (n)

{

If $(n = 0)$ **return** 0

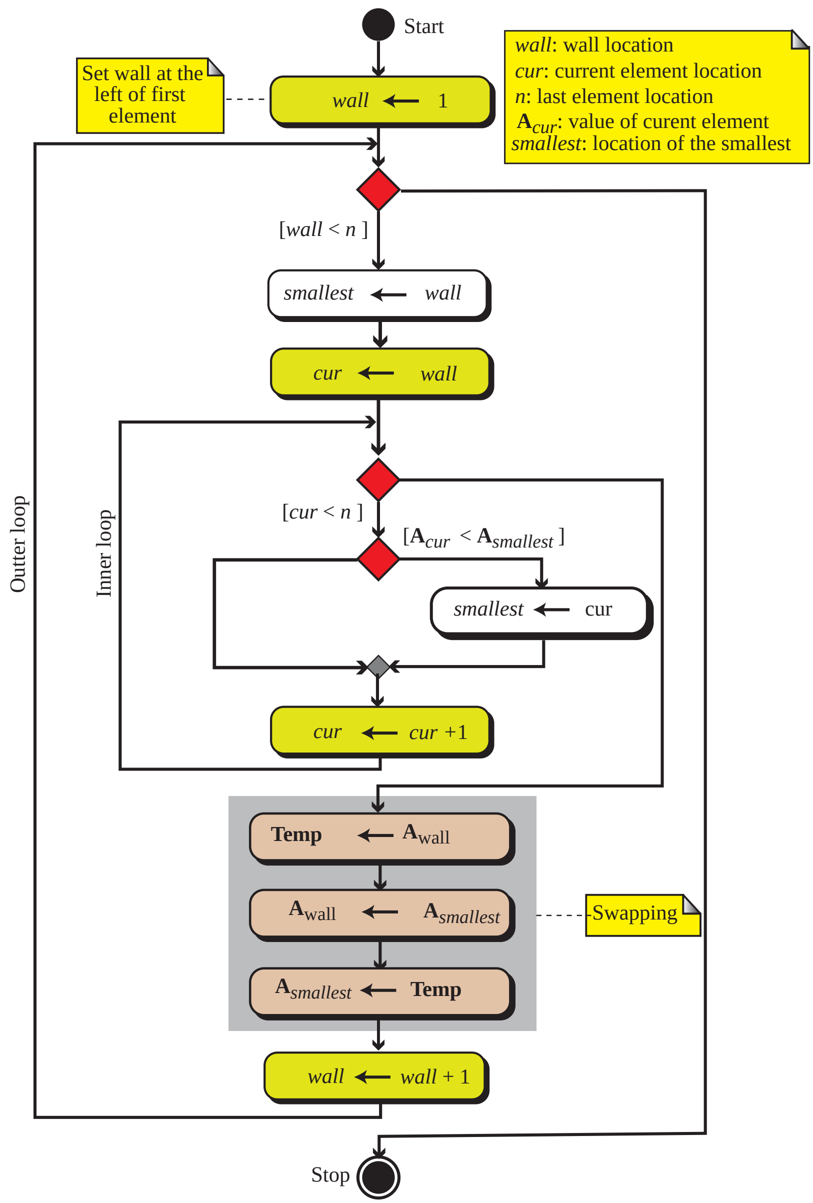
If $(n = 1)$ **return** 1

else **return** (**Fibonacci** $(n - 1)$ + **Fibonacci**$(n - 2)$)

}

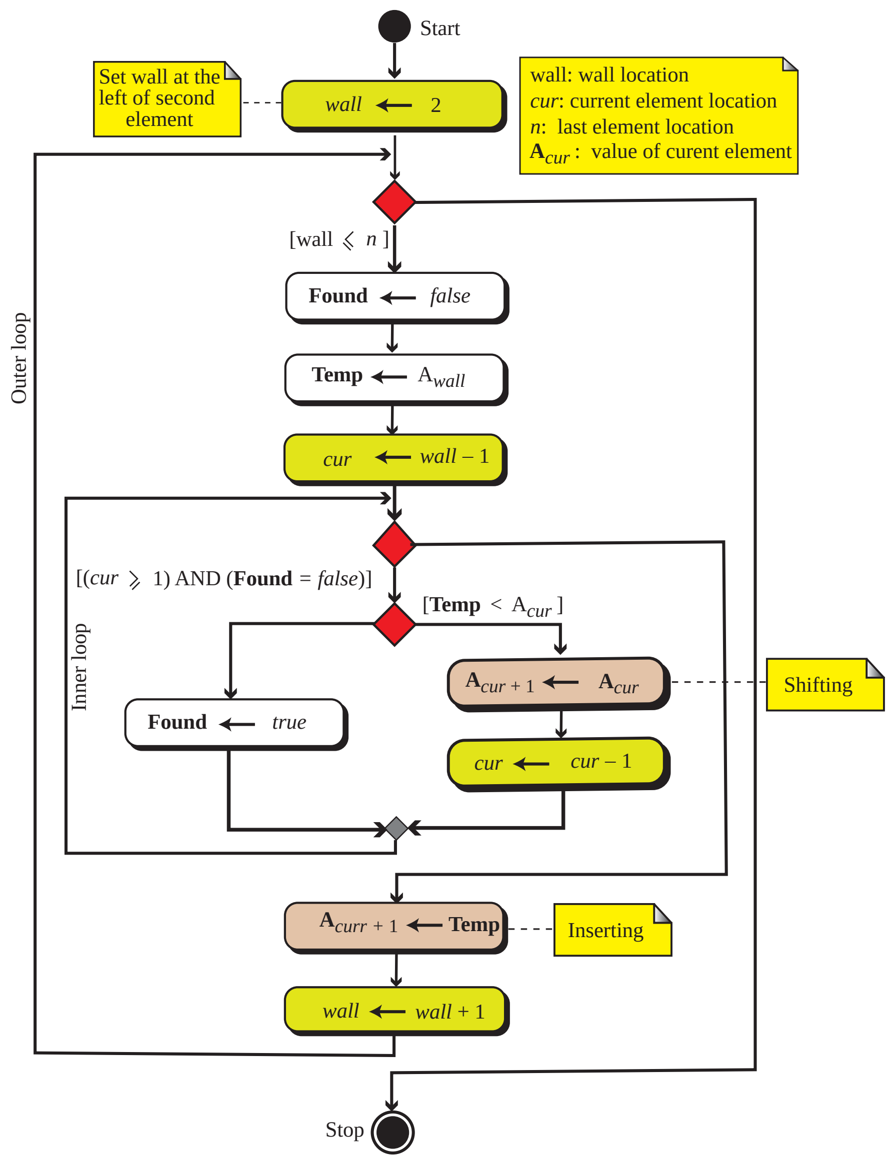
51. The UML for the selection sort is shown in Figure S8.51. The inner loop finds the location of the smallest element in the unsorted list. The three instructions after the inner loop swap this element with the first element in the unsorted list. Before searching for the smallest element, we assume that the first element in the unsorted list is the smallest one.

Figure S8.51 *Exercise 51*



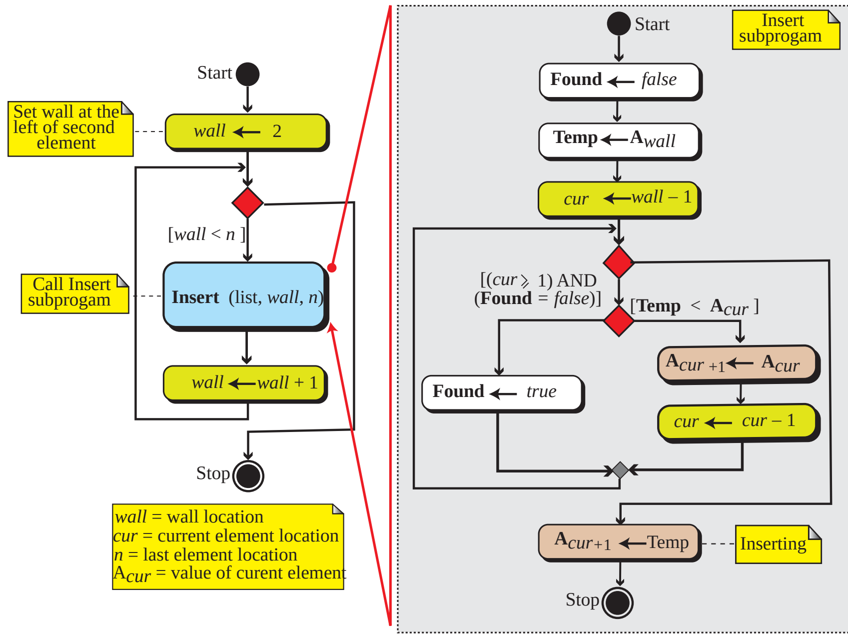
53. The UML for insertion sort is shown in Figure S8.53. The inner loop finds the location of the insertion. We shift the elements in the sorted sublist until we find the appropriate location to insert the element. When the algorithm exits the inner loop, insertion can be done. We have used a true/false value, **Found**, to stop shifting when the location of the insertion is found.

Figure S8.53 *Exercise 53*



55. The UML is shown in Figure S8.55. The program calls the Insert subprogram

Figure S8.55 *Exercise 55*



57. Algorithm S8.57 shows the pseudocode for finding the product of integers.

Algorithm S8.57 *Exercise 57*

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
}
```

59. Algorithm S8.59a 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 (Algorithm 8.59b).

Algorithm S8.59a *Exercise 59*

```
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
}
```

Algorithm 8.59b *Exercise 59*

```
Algorithm: FindSmallest(list, wall, n)
Purpose: To find the smallest number in an unsorted list
Pre: Given: 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
}
```

61. Algorithm S8.61a 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 (Algorithm 8.61b).

Algorithm S8.61a *Exercise 61*

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

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

Algorithm 8.61b *Exercise 61*

Algorithm: **Bubble**(list, *wall*, *n*)
Purpose: to bubble an unsorted list
Pre: Given: 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 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
    }
}
```

63. Algorithm S8.63a shows the pseudocode for the insertion sort routine that uses a subprogram (Algorithm S8.63b).

Algorithm S8.63a *Exercise 63*

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
    }
}
```

Algorithm S8.63b *Exercise 63*

Algorithm: **Insert**(list, *wall*, *n*)
Purpose: Insert a number in a sorted list
Pre: Given: 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
}
```

65. Algorithm S8.65 shows the pseudocode for binary search.

Algorithm S8.65 *Exercise 65*

```
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 (true if target found, false if not)

{
        flag ← false
        first ← 1
        last ← n
        while (first ≤ last)
        {
                mid = (first + last) / 2
                if (target < A_mid)        Last ← mid − 1     // A_A 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 number larger than the target
 // If flag is true, i is the location of the target
}
```

67. Algorithm S8.67 shows the pseudocode for finding the integer power of an integer.

Algorithm S8.67 *Exercise 67*

```
Algorithm: Power (x, n)
Purpose: Find x^n where x and n are integers
Pre: x, n
Post: None
Return: x^n
{
        z ← 1
        while (n ≠ 1)
        {
                z ← z × x
                n ← n − 1
        }
        return z
}
```