CHAPTER 11

Data Structures

(Solutions to Odd-Numbered Problems)

Review Questions

1. Arrays, records, and linked lists are three types of data structures discussed in this chapter.
2. Elements of an array are contiguous in memory and can be accessed by use of an index. Elements of a linked list are stored in nodes that may be scattered throughout memory and can only be accessed via the access functions for the list (i.e., the address of a specific node returned by a search function).
3. An array is stored contiguously in memory. Most computers use row-major storage to store a two-dimension array.
4. The fields of a node in a linked list are the data and a pointer (address of) the next node.
5. We use the head pointer to point to the first node in the linked list.

Multiple-Choice Questions

11. d 13. c 15. c 17. a 19. d

Exercises

21. Algorithm S11.21 shows a routine in pseudocode that compares two arrays.

Algorithm S11.21  Exercise 21

Algorithm: CompareArrays(A, B)
Purpose: Test if every element in array A equals to its corresponding element in array B
Pre: Arrays A and B of 10 integers
Post: None
Return: true or false

{
    i ← 1
    while (i ≤ 10)
Algorithm S11.21 Exercise 21

```plaintext
{  
    if A[i] ≠ B[i]  return false  // A is not equal to B  
    i ← i + 1  
}  
return true  // A is equal to B
```
Algorithm S11.25  Exercise 25

\[
\{ \\
\quad \text{mid} = (\text{first} + \text{last}) / 2 \\
\quad \text{if} (x < A[\text{mid}]) \quad \text{Last} \leftarrow \text{mid} - 1 \\
\quad \text{if} (x > A[\text{mid}]) \quad \text{first} \leftarrow \text{mid} + 1 \\
\quad \text{if} (x = A[\text{mid}]) \quad \text{first} \leftarrow \text{Last} + 1 \quad // x \text{ is found} \\
\}\ \\
\text{if} (x > A[\text{mid}]) \quad i \leftarrow \text{mid} + 1 \\
\text{if} (x \leq A[\text{mid}]) \quad i \leftarrow \text{mid} \\
\text{if} (x = A[\text{mid}]) \quad \text{flag} \leftarrow \text{true} \\
\text{return} \text{(flag, } i \text{)} \\
\} \\
// \text{If flag is true, it means } x \text{ is found and } i \text{ is its location.} \\
// \text{If flag is false, it means } x \text{ is not found; } i \text{ is the location where the target supposed to be.}
\]

27. Algorithm S11.27a shows a delete routine in pseudocode. Note that this algorithm calls BinarySearch algorithm (Algorithm S11.25) and ShiftUp algorithm (Algorithm S11.27b).

Algorithm S11.27a  Exercise 27

\begin{algorithm}
\caption{DeleteSortedArray($A$, $n$, $x$)}
\textbf{Purpose:} Delete an element from a sorted array
\textbf{Pre:} $A$, $n$, $x$  \hspace{1cm} // $x$ is the value we want to delete
\textbf{Post:} None
\textbf{Return:}
\{
\{flag, $i$\} \leftarrow \text{BinarySearch} (A, n, x)  \quad // \text{Call binary search algorithm}
\text{if} (\text{flag} = \text{false}) \quad // x \text{ is not in A}
\{
\quad \text{print} (x \text{ is not in the array})
\quad \text{return}
\}
\text{ShiftUp} (A, n, i)  \quad // \text{call shift up algorithm}
\text{return}  \quad // \text{call shift up algorithm}
\}
\end{algorithm}

Algorithm S11.27b  Exercise 27

\begin{algorithm}
\caption{ShiftUp($A$, $n$, $i$)}
\textbf{Purpose:} Shift up all elements one place from the last element up to element with index $i$. 
\textbf{Pre:} $A$, $n$, $i$
\textbf{Post:} None 
\textbf{Return:}
\}
\end{algorithm}
Algorithm S11.29 \textit{Exercise 29}

\begin{algorithm}
\hspace{1em}\textbf{Algorithm: AddFraction}\hspace{1em}(Fr1, Fr2) \\
\textbf{Purpose:} Add two fractions \\
\textbf{Pre:} Fr1, Fr2 \hspace{1em} // Assume denominators have nonzero values \\
\textbf{Post:} None \\
\textbf{Return:} The resulting fraction (Fr3) \\
\begin{algorithmic}
\State $x \leftarrow \gcd(Fr1\text{.denom}, Fr2\text{.denom})$ \hspace{1em} // Call gcd (see Exercise 8.57)
\State $y \leftarrow (Fr1\text{.denom} \times Fr2\text{.denom}) / x$ \hspace{1em} // $y$ is the least common denominator
\State Fr3.num $\leftarrow (y / Fr1\text{.denom}) \times Fr1\text{.num} + (y / Fr2\text{.denom}) \times Fr2\text{.num}$
\State Fr3.denom $\leftarrow y$
\State $z \leftarrow \gcd(Fr3\text{.num}, Fr3\text{.denom})$ \hspace{1em} // Simplifying the fraction
\State Fr3.num $\leftarrow Fr3\text{.num} / z$
\State Fr3.denom $\leftarrow Fr3\text{.denom} / z$
\State \textbf{return} (Fr3)
\end{algorithmic}
\end{algorithm}

Algorithm S11.31 \textit{Exercise 31}

\begin{algorithm}
\hspace{1em}\textbf{Algorithm: MultiplyFraction}\hspace{1em}(Fr1, Fr2) \\
\textbf{Purpose:} Multiply two fractions \\
\textbf{Pre:} Fr1, Fr2 \hspace{1em} // Assume denominators with nonzero values \\
\textbf{Post:} None \\
\textbf{Return:} Fr3 \\
\begin{algorithmic}
\State Fr3.num $\leftarrow Fr1\text{.num} \times Fr2\text{.num}$
\State Fr3.denom $\leftarrow Fr1\text{.denom} \times Fr2\text{.denom}$
\State $z \leftarrow \gcd(Fr3\text{.num}, Fr3\text{.denom})$ \hspace{1em} // Simplifying the fraction
\end{algorithmic}
\end{algorithm}

29. Algorithm S11.29 shows a routine in pseudocode that adds two fractions.

31. Algorithm S11.31 shows a routine in pseudocode that multiplies two fractions.
33. Figure S11.33 shows a linked list of records.

Algorithm S11.31  Exercise 31

```plaintext
Fr3.num ← Fr3.num / z
Fr3.denom ← Fr3.denom / z
return (Fr3)
```

35. Since list = null, the SearchLinkedList algorithm performs new ← list. This creates a list with a single node.

37. Algorithm S11.37 shows a routine for finding the average data in a linked list.

Algorithm S11.37  Exercise 37

```plaintext
Algorithm: LinkedListAverage (list)
Purpose: Evaluate average of numbers in a linked list
Pre: list
Post: None
Return: Average value
{
    counter ← 1
    sum ← 0
    walker ← list
    while (walker ≠ null)
    {
        sum ← sum + (*walker).data
        walker ← (*walker).link
        counter ← counter + 1
    }
    average ← sum / counter
    return average
}
```

39. Figure S11.39a shows that if pre is not null, the two statements cur ← (*cur).link and pre ← (*pre).link move the two pointers together to the right. In this case the two statements are equivalent to the ones we discussed in the text. However, the statement pre ← (*pre).link does not work when pre is null.
because, in this case, \((\ast \text{pre}).\text{link}\) does not exist (Figure S11.39b). For this reason, we should avoid using this method.

**Figure S11.39a  Exercise 39**

---

**Figure S11.39b  Exercise 39**