The ADT operations discussed in this chapter have space complexity O(n). This means that they only take the space required for the data set. Bubble sort and insertion sort have space complexity of O(1). This means they do not require extra memory for sorting larger lists.

Reflection Point:

List the standard algorithms you have met in this chapter. Can you give the essential features of each of these?

Summary

- Standard algorithms include bubble sort, insertion sort, linear search and binary search.
- Abstract data types (ADTs) include stacks, queues, linked lists, binary trees, hash tables, dictionaries and graphs.
- Basic operations required for an ADT include creating an ADT and inserting, finding or deleting an element of an ADT.
- Time taken and space used can be measured using Big O notation.

Exam-style Questions

1 a Complete the algorithm for a binary search function `FindName`.

The data being searched is stored in the array `Names[0 : 50]`.

The name to be searched for is passed as a parameter.

```
FUNCTION FindName(s : STRING) RETURNS INTEGER
    Index ← -1
    First ← 0
    Last ← 50
    WHILE (Last >= First) AND ..................................... DO
        Middle ← (First + Last) DIV 2
        IF Names[Middle] = s
          THEN
            Index ← Middle
          ELSE
            IF ..................................................
              THEN
                Last ← Middle + 1
              ELSE
                ..............................................
            ENDIF
        ENDIF
    ENDWHILE
ENDFUNCTION
```

[3]

b The binary search does not work if the data in the array being searched is ……. [1]

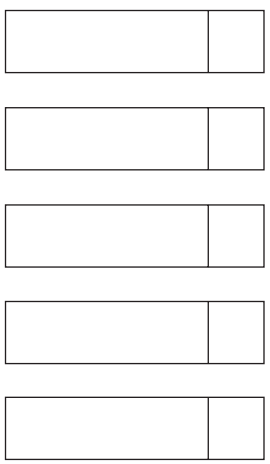
c State the return value of the function `FindName` return when:

i the name searched for exists in the array.

ii the name searched for does not exist in the array. [2]

2 A queue Abstract Data Type (ADT) is to be implemented as a linked list of nodes. Each node is a record, consisting of a data field and a pointer field. The queue ADT also has a `FrontOfQueue` pointer and an `EndOfQueue` pointer associated with it. The possible queue operations are: `JoinQueue` and `LeaveQueue`.

a **i** Write labels on the diagram to show the state of the queue after three data items have been added to the queue in the given order: Apple, Pear, Banana.



[5]

ii Write labels on the diagram to show how the unused nodes are linked to form a list of free nodes. This list has a `StartOfFreeList` pointer associated with it. [2]

b **i** Write program code to declare the record type `Node`. [3]

ii Write program code to create an array `Queue` with 50 records of type `Node`. Your solution should link all nodes and initialise the pointers `FrontOfQueue`, `EndOfQueue` and `StartOfFreeList`. [7]

c The pseudocode algorithm for the queue operation `JoinQueue` is written as a procedure with the header:

```
PROCEDURE JoinQueue(NewItem)
```

where `NewItem` is the new value to be added to the queue. The procedure uses the variables shown in the following identifier table:

Identifier	Data type	Description
`NullPointer`	`INTEGER`	Constant set to –1
		Array to store queue data
	`STRING`	Value to be added
		Pointer to next free node in array
		Pointer to first node in queue
		Pointer to last node in queue
		Pointer to node to be added

i Complete the identifier table. [7]

ii Complete the pseudocode using the identifiers from the table in **part (c) (i)**. [6]

```
PROCEDURE JoinQueue(NewItem : STRING)
    // Report error if no free nodes remaining
    IF StartOfFreeList = ..........................
      THEN
        Report Error
      ELSE
```

```
            // new data item placed in node at start of free list
            NewNodePointer ← StartOfFreeList
            Queue[NewNodePointer].Data ← NewItem
            // adjust free list pointer
            StartOfFreeList ← Queue[NewNodePointer].Pointer
            Queue[NewNodePointer].Pointer ← NullPointer
            // if first item in queue then adjust front of queue pointer
            IF FrontOfQueue = NullPointer
              THEN
                .........................
                .........................
            ENDIF
            // new node is new end of queue
            Queue[.................].Pointer ← .................
            EndOfQueue ← ........................................
        ENDIF
    ENDPROCEDURE
```

3 A program is required that sorts a list of words into alphabetical order. The list of words is supplied as a text file.

a Write a program to declare a string array, `WordList`, that can hold 500 elements. Initialise the array so all elements contain the empty string. [3]

b Write a procedure, `OutputList`, to output all elements in index order. [3]

c Write a procedure, `LoadWords`, that asks the user for a filename and reads the contents of the text file, storing each line of text (word) in a separate array element. The procedure should output a relevant error message if:

- the file doesn't exist
- the array is full. [7]

d Write a procedure, `SortWords`, to perform a bubble sort on all non-empty array elements, so that the words are in alphabetical order. [7]

e Write program code to call `LoadWords`, then `OutputList`, followed by `SortWords` and then `OutputList` again. [3]

f Test your program by running it first with a non-existing file, and then with a text file containing 20 words in random order.

Take screenshots of your test runs that show your code works correctly. [2]