Ch23

Algorithms

CHAPTER ASSIGNMENT · 54 MARKS

Name _____ Class _____ No. _____ Date _____

Answer every question in the space given. Show your working. You may use the chapter page to help you.

Questions from *Computer Science for Cambridge International AS & A Level Coursebook*, 2nd edition, Sylvia Langfield and Dave Duddell (Cambridge University Press), for classroom use.

Tasks and questions from the chapter

DP **Discussion Point:** What were the essential features of a linear search?

T23.01 **Task 23.01:** Write program code for the linear search algorithm. Assume that the items to be searched are stored in a 1D array with n elements.

DP **Discussion Point:** What were the essential features of a bubble sort?

T23.02 **Task 23.02:** Write program code for the most efficient bubble sort algorithm. Assume that the items to be sorted are stored in a 1D array with n elements.

T23.03 **Task 23.03:**

(a) Dry-run the insertion sort algorithm using a trace table. Assume the list consists of the following six items in the order given: 53, 21, 60, 18, 42, 19.

(b) Write program code for the insertion sort algorithm. Assume that the items to be sorted are stored in a 1D array with n elements.

EQ23.01 **Extension Question 23.01:** Investigate the performances of the insertion sort and the bubble sort by:

- varying the initial order of the items
- increasing the number of items to be sorted.

T23.04 **Task 23.04:**

(a) Dry-run the binary search algorithm using a trace table. Assume the list consists of the following 20 items in the order given: 7, 12, 19, 23, 27, 33, 37, 41, 45, 56, 59, 60, 62, 71, 75, 80, 84, 88, 92, 99.

(b) Search for the value 60. How many times did you have to execute the `While` loop?

(c) Dry-run the algorithm again, this time searching for the value 34. How many times did you have to execute the `While` loop?

DP **Discussion Point:** Compare the binary-search algorithm with the linear-search algorithm. If the array contains n items, how many times on average do you need to test a value when using a binary search and how many times on average do you need to test a value when using a linear search? Can you describe how the search time varies with increasing n?

T23.05 **Task 23.05:** Convert the pseudocode for the linked-list handling subroutines to program code. Incorporate the subroutines into a program and test them.

T23.06 **Task 23.06:** Write program code to implement a binary search tree.

T23.07 **Task 23.07:**

(a) Write a program to implement the above pseudocode subroutines. Add a menu to test the subroutines.

(b) Write a program to implement a stack as a linked list. Note that the adding and removing of nodes is much simpler than for an ordered linked list.

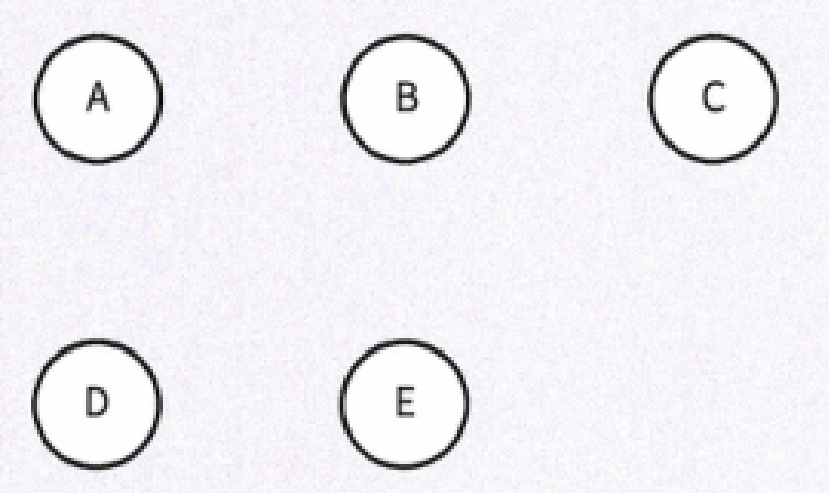
T23.08 **Task 23.08:**

(a) Write a program to implement the above pseudocode subroutines. Add a menu to test the subroutines.

(b) Write program code to implement a queue as a linked list. You may find it helpful to introduce another pointer that always points to the end of the queue. You will need to update it when you add a new node to the queue.

T23.09 **Task 23.09:** Draw the labelled edges for a directed graph of a road system where:

- A to B is 2 km, and it is a one-way system
- D to A is 1 km, and is one-way
- B to D is 3 km and is one-way
- B to C is 4 km
- C to E is 7 km
- D to E is 5 km.



T23.10 **Task 23.10:** Construct an adjacency matrix and an adjacency list to represent your graph from Task 23.09.

T23.11 **Task 23.11:** Write pseudocode to:

- insert a key–value pair into a dictionary
- look up a value in a dictionary.

Use the hashing function from Worked Example 23.01.

RP **Reflection Point:** List the standard algorithms you have met in this chapter. Can you give the essential features of each of these?

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. [3]

```
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
```

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

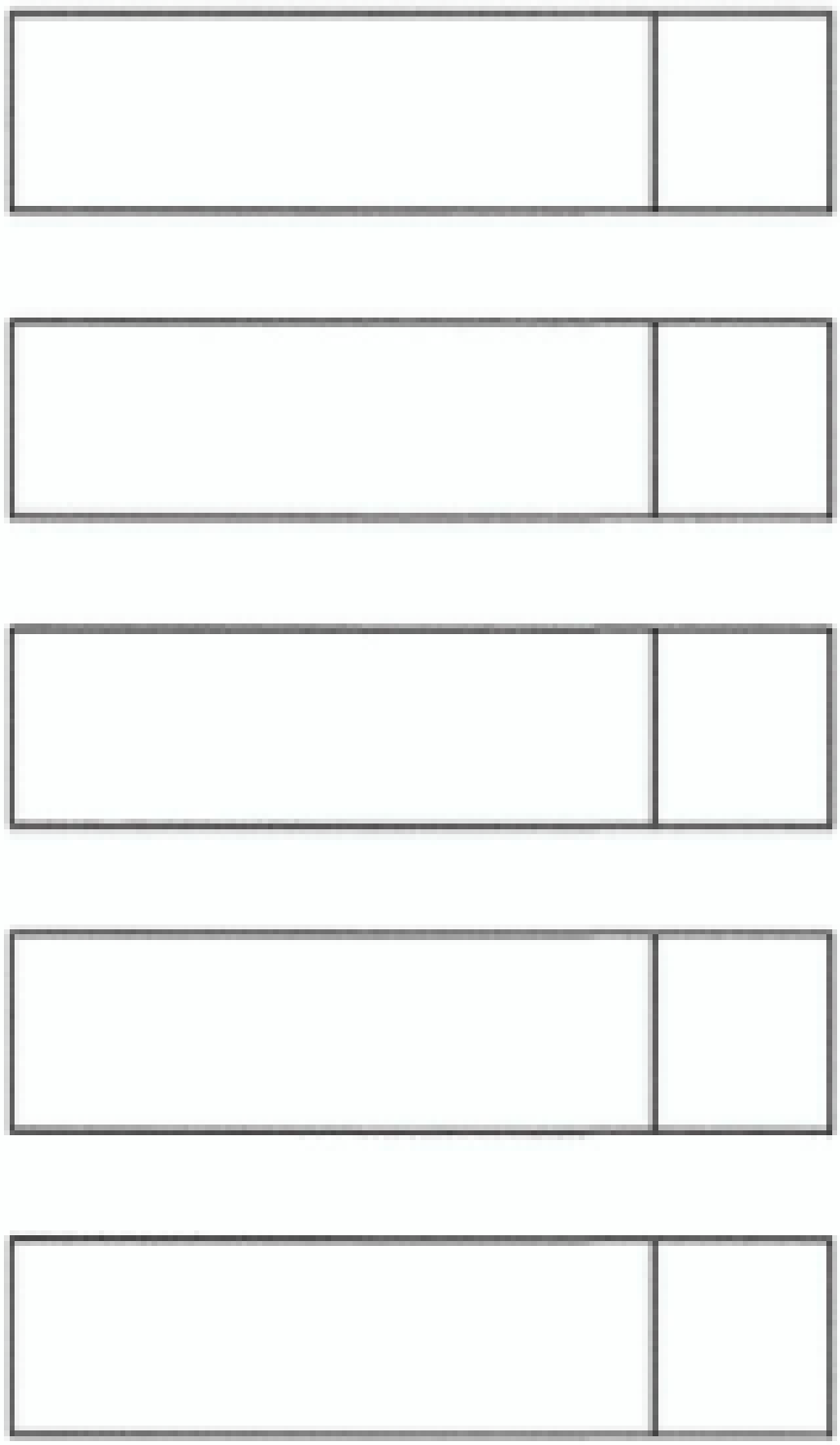
(c) State the return value of the function `FindName` when:

(i) the name searched for exists in the array. [1]

(ii) the name searched for does not exist in the array. [1]

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`.

(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]

(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]

End of assignment · Check your answers against the chapter at gurujazzy.com/subjects/computer-science/as-a-level/algorithms/