4.1.2 Algorithms

Monday, 15 February 2021 9:19 AM

4.1.2 Algorithms

4.1.2 Algorithms

May/June 2003

2 . The data Nile, Zambesi, Amazon, Indus, Thames, Volga, Danube and Mississippi are to be entered into a binary tree in that order so that later these names can be extracted in alphabetical order.

(a) Draw the representation of the binary tree with this data held in it. [3]

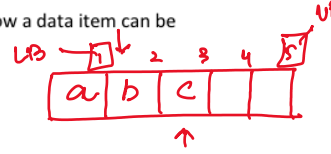
(b) Describe how to insert a new value correctly into this tree. [4]

Oct/NOV 2003

10. (a) By using diagrams, or otherwise, explain how a data item can be

(i) inserted into

(ii) read from (deleted from)



a queue held in an array. [4]

(b) Explain a problem that could arise when storing a queue in an array and state a possible solution. [2]

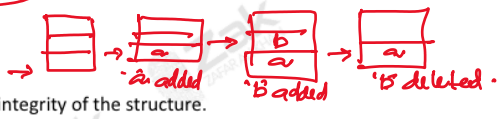
Both pointers move forward. Array is static. Once pointers reach UB, they can't move forward. Solution is circular Q.

May/June 2004

6. (c) A stack is to be held in an array. With the aid of a diagram, explain how an item may be

(i) added to,

(ii) deleted from



the stack, while maintaining the integrity of the structure. [6]

May/June 2005

3. Two lists of numbers need to be combined into a single list which will be in numerical order, smallest first.

List A: 2 3 8 11 17

List B: 7 10 5 6 9

(a) By showing each of the stages, describe how list B can be sorted into numerical order using an insertion sort. [4]

(b) Explain how list A and the sorted list B can be merged to give a complete, sorted, set of numbers. [6]

Oct/NOV 2005

6. (a) Explain the difference between static and dynamic data structures. [2]

(b) Give an example of a

(i) static, array data structure that can't change its size during the execution

(ii) dynamic " " " can " " " " " "

Linked list.

Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

data structure, giving an advantage of each. [4]

(c) The details of a car part are stored in a binary tree according to this algorithm

```
READ VALUE NEW_PART
START AT ROOT NODE
WHILE NODE NOT EMPTY, DO
   IF NEW_PART < VALUE AT NODE
      THEN FOLLOW LEFT SUBTREE
      ELSE FOLLOW RIGHT SUBTREE
   ENDIF
ENDWHILE
INSERT NEW_PART AT NODE
END
```

(i) Show the binary tree after the following values have been input
Radio Visor Brakes Tyres Alternator Windscreen [3]

(ii) Explain how Clutch is added to the tree in (i). [5]

(iii) Describe an algorithm that can be applied to the binary tree of car parts, so that the tree is read in alphabetic order. [3]

May/June 2006

8 (a) Explain the difference between a dynamic data structure and a static data structure, giving an example of each.

[3]

May/June 2007

3. Jobs that require printing, by a network printer, are stored until the printer is ready. Their addresses are placed in a queue to await their turn for printing. Addresses of new jobs are placed at one end of the queue. These job addresses are taken from the other end when the printer is ready.

(a) State two reasons why it would be preferable to store the queue in a linked list rather than an array. [2]

(b) If the queue is held in a linked list, describe an algorithm for
(i) inserting an address into the queue,
(ii) reading an address from the queue. [5]

Oct/NOV 2007

6. Names are to be stored in a binary tree according to the algorithm

```
1. Repeat
2. If Name > node then take right pointer
3. Else take left pointer
```


4.1.2 Algorithms

```
4. Until empty node
5. Insert Name
```

(a) Given that the root node is DAMON, create the binary tree resulting from inserting:
CANDU RISH AAMON TENAR GLAN
in the given order. [3]

(b) Describe an algorithm for using the tree to read the names in alphabetic order. [3]

May/June 2008

7. List A is 2,4,7,9

List B is 15,3,8,10,1

These two lists are to be merged into one list in numerical order, smallest first.

(a) List B must first be sorted into order.
Describe how an insertion sort can be used to do this. [4]

(b) After both lists have been sorted they are to be combined into a single list in numerical order.
Describe how a merge sort can be used to do this. [4]

Oct/NOV 2008

9. The names of 20 students in a computing class are stored in an array called NAME(X) where X stands for a number between 0 and 19.

(a) Describe an algorithm to find the position of a particular student in the array, using a serial search. **[5]**

(b) (i) Explain why the search in part (a) would not be suitable if the array was large enough to store the names of all 1000 students in the school. **[2]**

(ii) Suggest a better method of searching for a particular name, justifying your answer. **[3]**

May/June 2009

2. (c) A set of data items is stored in a sorted binary tree.

Describe a procedure which will insert a new data item into the correct position in the sorted binary tree. **[4]**

Oct/NOV 2009. P31

4. A school is designing a computer system to satisfy its administration requirements.
The head teacher, the system administrator and five office staff will all require regular access.
It is decided to use a single, large, computer with a number of terminals to satisfy demand.
(d) The student records are stored using direct access with a full index based on student name.
The index is arranged in alphabetical order of student name.
The index can be stored in a static or dynamic data structure.

4.1.2 Algorithms

(i) State an advantage and a disadvantage of using a static data structure for storing this index. **[2]**

(ii) It is decided to store the index as a linked list. Describe an algorithm for the insertion of a new student. **[6]**

Oct/NOV 2009. P32

3. This question is about sorting data in files into order.

```
File A: CHO, SYG, DAN, ROG, BRI
File B: ADA, COU, LAC, LOV
File C: HEL, SAR, PAU, GRA, CHR, STE
```

(c) (i) Draw a binary tree using the data in file C and HEL as the root node.

(ii) Explain the rules which you used to draw the tree.

(iii) Describe how your tree can be used to output the file in alphabetical order. **[7]**

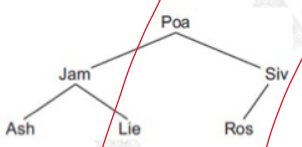
Oct/NOV 2010. P31/P32

6. A set of data is stored in a sorted binary tree.
Describe how to insert a new data item into the correct position in the sorted binary tree. **[4]**

Oct/NOV 2010. P33

6. The following data are placed in a binary tree:

```
Poa, Siv, Jam, Ash, Ros, Lie
```



(a) The tree is read using in-order traversal which follows the algorithm
Traverse the left-hand subtree
Read the root node
Traverse the right-hand subtree
Write down the nodes of the tree in the order that they will be read. **[2]**

(b) The tree is read using pre-order traversal which follows the algorithm
Read the root node
Traverse the left-hand subtree
Traverse the right-hand subtree
Write down the nodes of the tree in the order that they will be read. **[2]**

Computer Science 9608
Topical Past Papers

4.1.2 Algorithms

May/June 2011. P31

8 (a) (i) Explain the difference between static and dynamic implementation of data structures. **[2]**
(ii) Give one advantage and one disadvantage of storing a queue in an array rather than in a linked list. **[2]**

(b) (i) Draw a diagram to show how the following members of a Computing class can be stored in a linked list in alphabetic order:
FRO, TSI, DON, ROS, BEV **[5]**

(ii) Describe an algorithm to insert a new member of the class into the correct position in the list. **[5]**

May/June 2011. P33

8. (a) (i) Explain the difference between static and dynamic implementation of data structures. **[2]**
(ii) Give two advantages of storing a stack in a linked list rather than in an array. **[2]**

(b) (i) Draw a diagram to show how the following members of a Computing class can be stored in a linked list in alphabetic order:
FRO, TSI, DON, ROS, BEV **[5]**

(ii) Describe an algorithm to insert a new member of the class into the correct position in the list. **[5]**

May/June 2012. P33/P32

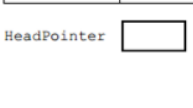
3 A linked list is to be implemented with the data structures described in the variable table.
The countries are to be organised in alphabetical order.

Computer Science 9608
Topical Past Papers

Zak ZAFAR ALI KHAN

4.1.2 Algorithms

Identifier	Data Type	Description
Country	ARRAY[250] OF STRING	Stores the country names
Pointer	ARRAY[250] OF INTEGER	Array index which points to the next country in the linked list
HeadPointer	INTEGER	Array index pointing to the first country in the linked list



1	SWEDEN	1	
2	DENMARK	2	
3	INDIA	3	
4	COLUMBIA	4	
5	BANGLADESH	5	4
6	NEPAL	6	
7	MAURITIUS	7	
250		250	

(a) Complete the above diagram showing all the pointer values for this linked list. **[4]**

(b) The following pseudocode uses the linked list to output all country names which are alphabetically before a requested country.

For example, the user inputs NEPAL – the pseudocode outputs all the values which are alphabetically before NEPAL.

Fill in the gaps in the pseudocode.

```
INPUT RequestedValue
IF ……
   THEN
     //special case - the list is empty ...
     OUTPUT "Linked list is empty"
   ELSE
     ……
     Current ← HeadPointer
     REPEAT
       IF Country[Current] < RequestedValue
         THEN
           OUTPUT Country[Current]
           Current ← ……
         ELSE
           NoMoreValues ← TRUE
       ENDIF
     UNTIL NoMoreValues = TRUE
```

[3]

(c) An algorithm is to be designed which inputs a requested country and outputs all the values in the linked list after this country.

Describe how, using the pointers, this algorithm works. **[4]**

(d) A linked list is maintained for capital cities using arrays `Capital` and `Pointer`.

An algorithm is required to delete a value from the linked list.

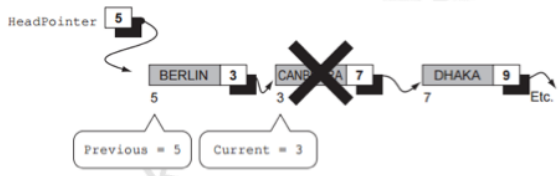
The algorithm will use the following variables:

Identifier	Data Type	Description
`Current`	`INTEGER`	Array index for the current capital
`Previous`	`INTEGER`	Array index for the previous capital

The following diagram shows the first three capitals in the linked list.

We are about to delete CANBERRA.

The list has been searched from the `HeadPointer` position until the capital to be deleted, CANBERRA, is found.



Describe the steps in the algorithm to delete CANBERRA from the linked list.

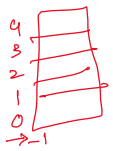
(Do not attempt to write the complete algorithm.) **[4]**

Oct/NOV 2012. P32

5 (a) Describe the operation of a stack data structure. LIFO [1]

(b) A stack is to be implemented to store string data using the following variables.

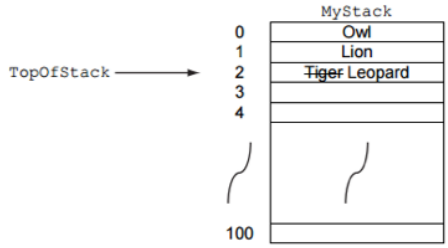
Variable	Data Type	Description
MyStack	ARRAY[100]: STRING	Stores the string data values
TopOfStack	INTEGER	Stores the index position of the MyStack array for the current 'top of stack' position. TopOfStack has value -1 when the stack is empty.



The diagram below shows the state of the array and TopOfStack after the following:

4.1.2 Algorithms

- three values have been pushed on to the stack (Owl, Lion and Tiger)
- a value is popped from the stack
- then the value Leopard is pushed on to the stack.



Popping a single value is to be implemented with the procedure PopFromStack.

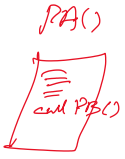
```
PROCEDURE PopFromStack
   IF ..... TopOfStack = -1 .....
     THEN
       OUTPUT "Stack is already EMPTY"
     ELSE
       OUTPUT MyStack [ TopOfStack ] "is popped"
       TopOfStack ← ..... TopOfStack - 1 .....
   ENDIF
ENDPROCEDURE
```

Complete the pseudocode by filling in the three answer spaces. [3]

(c) (i) State when a stack would be required in the operation of a computer system. [1]

(ii) Explain how the stack is used. [2]

→ When a call is made, the return address to resume will be pushed on stack. To control subroutines calling.



Oct/NOV 2012. P33

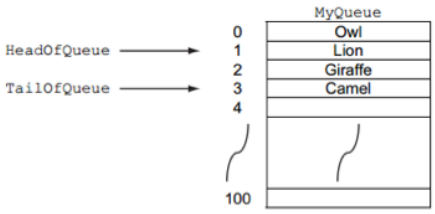
5 (a) Describe the operation of a linear queue data structure. [1]

(b) A linear queue is to be implemented to store data using the following variables.

Identifier	Data Type	Description
MyQueue	ARRAY[100]: STRING	Stores the data values
HeadOfQueue	INTEGER	Stores the index position of the item currently at the head of MyQueue
TailOfQueue	INTEGER	Stores the index position of the item currently at the tail of MyQueue
NewItem	STRING	Stores a data value to be added to MyQueue

The diagram shows the state of MyQueue, HeadOfQueue and TailOfQueue after four values (Owl, Lion, Giraffe and Camel) have been inserted and one value (Owl) has been deleted.

Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN



Tail = Tail + 1
Insert at Tail

Inserting and deleting a single item to/from the queue are to be implemented with two procedures `AddToQueue` and `RemoveFromQueue` respectively.

(i) Shown below is the incomplete pseudocode for the `AddToQueue` procedure.
Using the variables given, fill in the missing code.

```
PROCEDURE AddToQueue
   IF ..... TailOfQueue = 100 .....
     THEN
       OUTPUT "Refused - Queue is already FULL"
     ELSE
       INPUT NewItem
       TailOfQueue ← ..... TailOfQueue + 1 .....
   ENDIF
ENDPROCEDURE
```

[4]

(ii) Write the algorithm for the `RemoveFromQueue` procedure, using the variables given.

```
PROCEDURE RemoveFromQueue
```

Self

[2]

(c) Describe an application in the operation of a computer system where a queue data structure would be required.

[2]

May/June 2013. P31/32

4 A binary tree is implemented with three 1-dimensional arrays.

Identifier	Data Type	Description
Data	ARRAY[100] OF STRING	Stores the data values
LeftP	ARRAY[100] OF INTEGER	Stores the left index pointer
RightP	ARRAY[100] OF INTEGER	Stores the right index pointer
Root	INTEGER	Stores the index position of the root value

(a) An array is a static data structure.

(i) Explain the difference between a static and a dynamic data structure. **[2]**

(ii) What benefit would be gained from using a dynamic data structure to implement a binary tree? **[1]**

The initially empty tree has the following items added in this order:

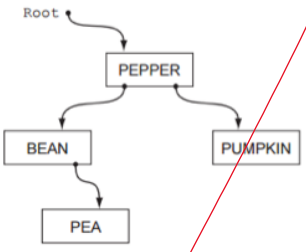
4.1.2 Algorithms

MELON, PEAR, BANANA, ORANGE

(b) Draw the binary tree after these four items have been added. **[3]**

(c) The following algorithm traverses the tree shown and outputs the nodes in order i.e. an 'in-order traversal'.

```
PROCEDURE InOrder(Root)
  IF LeftP[Root] <> 0
    THEN
      // move left
      InOrder(LeftP[Root])
  ENDIF

  OUTPUT Data[Root]

  IF RightP[Root] <> 0
    THEN
      // move right
      InOrder(RightP[Root])
  ENDIF
ENDPROCEDURE
```



(i) Copy a line from procedure `InOrder` that makes the procedure recursive. **[1]**

(ii) The diagram shows a trace of the execution of this algorithm for the given tree data. Fill in the missing lines of code.

4.1.2 Algorithms

```
InOrder(PEPPER)
LeftP[PEPPER] <> 0 is TRUE
    InOrder(BEAN)
    LeftP[BEAN] <> 0 is FALSE
    OUTPUT BEAN
    RightP[BEAN] <> 0 is TRUE
        InOrder(PEA)
        LeftP[PEA] <> 0 is FALSE
        OUTPUT PEA
        ..........
        ENDPROCEDURE
    ENDPROCEDURE
..........
RightP[PEPPER] <> 0 is TRUE
    InOrder(PUMPKIN)
    LeftP[PUMPKIN] <> 0 is FALSE
    ..........
    RightP[PUMPKIN] <> 0 is FALSE
    ..........
ENDPROCEDURE
```

[4]

(iii) What do the arrows in the diagram represent? **[1]**

May/June 2013. P33

4 A linked list is implemented with an array of records of data type `Node`.
The `Node` record has two fields as defined below:

```
RECORD Node
  Data : STRING
  Pointer : INTEGER
ENDRECORD
```

A program is to create a linked list using the array and variable shown below.

Identifier	Data Type	Description
MyList	ARRAY[100] OF Node	An array to store the data and pointer values
HeadPointer	INTEGER	Stores the index position of the node at the head of the linked list

(a) An array is a static data structure.

(i) Explain the difference between a static and a dynamic data structure. **[2]**

(ii) What benefit would be gained from using a dynamic data structure to implement a linked list? **[1]**

The linked list has the following items: BEAN, COURGETTE, APPLE, PEPPER

The data is stored as shown below:

HeadPointer: 3

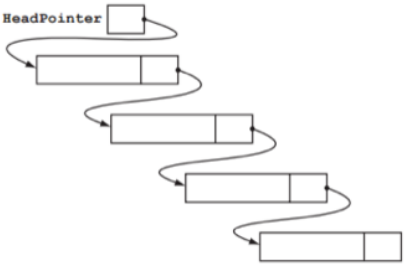
	MyList Data	MyList Pointer
1	BEAN	2
2	COURGETTE	4
3	APPLE	1
4	PEPPER	0
...		
99		
100		

(b) What is the value of:

MyList[HeadPointer].Data? **[1]**

MyList[3].Pointer? **[1]**

(c) Complete the linked list diagram by filling in the data and pointer values for each node.



[4]

Computer Science 9608
Topical Past Papers

Zak ZAFAR ALI KHAN

4.1.2 Algorithms

(d) The following algorithm traverses the linked list and outputs the data values.

```
PROCEDURE ListTraversal(Index)
  IF MyList[Index].Pointer <> 0
    THEN
      // follow the pointer to the next node
      ListTraversal(MyList[Index].Pointer)
  ENDIF
  OUTPUT MyList[Index].Data
ENDPROCEDURE
```

(i) Copy the line from procedure `ListTraversal` that makes the procedure recursive. **[1]**

(ii) The diagram shows a trace of the execution of this algorithm for the given linked list data.

`HeadPointer: 3`

	MyList	
	Data	Pointer
1	BEAN	2
2	COURGETTE	4
3	APPLE	1
4	PEPPER	0
...		
99		
100		

[4]

(iii) What do the arrows in the diagram represent? **[1]**

Oct/Nov 2013.P31

5 (a) Describe the operation of a stack data structure. LIFO **[1]**

A stack is to be implemented to manage the spooled print jobs sent to a network printer. A job reference and the user ID of the network account are recorded for each print job.

The stack is implemented using the following user-defined data type and variables.

```
TYPE Stack
  JobReference : STRING
  UserID : STRING
ENDTYPE
```


4.1.2 Algorithms

Identifier	Data Type	Description
`SpoolJob`	`ARRAY[1000] OF Stack`	Stores the job reference and user ID for each print job
`TopOfStack`	`INTEGER`	Stores the index position of the print job currently at the top of the stack
`NewReferenceNo`	`STRING`	Stores the job reference of the new print job added to `SpoolJob`
`NewUserID`	`STRING`	Stores the user ID of the new print job added to `SpoolJob`

(b) The diagram shows the state of `SpoolJob` and `TopOfStack` after three print jobs were received from users LEWIS, CHI and SINGH (in that order), a print job was sent to the printer, then a new print job received from user HARRIS.

	SpoolJob .UserID	SpoolJob .JobReference
1000		
...		
3		
TopOfStack → 2	SINGH HARRIS	~~17231~~ 17856
1	CHI	~~17843~~
0	LEWIS	~~17805~~

(i) What is the value of:

`SpoolJob[2].UserID?` HARRIS

`SpoolJob[TopOfStack - 1].JobReference?` 17843 **[2]**

(ii) Spooling a new print job is to be implemented with a procedure `PushJob`.

Shown below is the incomplete pseudocode for the `PushJob` procedure.

Using the variables and user-defined type given, fill in the missing pseudocode.

```
PROCEDURE PushJob
   IF TopOfStack = 1000
     THEN
```

```
    OUTPUT "Stack is already FULL"
  ELSE
    INPUT NewUserID
    SpoolJob[TopOfStack].UserID ← NewUserID
    TopOfStack ← TopOfStack - 1
    SpoolJob[TopOfStack].JobReference ← NewReferenceNo
    TopOfStack ← TopOfStack + 1
  ENDIF
ENDPROCEDURE
```

[4]

4.1.2 Algorithms

(c) Processing a print job is to be implemented with a `PopJob` procedure.
Complete the pseudocode for this `PopJob` procedure.

```
PROCEDURE PopJob
  IF TopOfStack = -1
    THEN
      OUTPUT "Underflow"

    ELSE
      PROCESS SpoolJob[TopOfStack]
      TopOfStack = TopOfStack - 1
  ENDIF
ENDPROCEDURE
```

[3]

(d) Explain why the choice of a stack data structure for this application is a poor choice.
Suggest an alternative data structure. [3]

Oct/Nov 2013.P32

5 Customer names are stored in the array `Customer`.

An algorithm is to be designed to perform a serial search of the array for a requested customer name.

The algorithm will use the variables shown in the table.

(a) Study the table and the algorithm and fill in the gaps.

Identifier	Data Type	Description
Customer	ARRAY[100] OF STRING	Array of customer names
Index	INTEGER	Used to index the array elements
IsFound		
SearchName	STRING	The requested customer name

```
//Serial search algorithm
INPUT SearchName
IsFound ← FALSE
Index ← 1
REPEAT
  IF Customer[Index] = SearchName
    THEN
      IsFound ← TRUE
      OUTPUT "Found at position ", Index
    ELSE
```


Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

```
      Index ← Index + 1
  ENDIF
UNTIL (IsFound = TRUE) OR Index > 100
IF IsFound = False
  THEN
     OUTPUT "Customer name was NOT FOUND"
ENDIF
```

[7]

(b) How many comparisons on average will be needed to find a requested customer from the Customer array? [1]

(c) A binary search may be an alternative algorithm to a serial search.

(i) What condition is put on the Customer array for a binary search to be used? [1]

The following recursive function is for the binary search algorithm.

```
FUNCTION BinarySearch(ThisArray, FindValue, Low, High) : INTEGER
  IF High < Low
    THEN
      RETURN -1 // not found
    ELSE
      Middle ← INT((High + Low) / 2)
      IF ThisArray[Middle] > FindValue
         THEN
           BinarySearch(ThisArray, FindValue, Low, Middle - 1)
         ELSE
           IF (ThisArray[Middle] < FindValue)
               THEN
                  BinarySearch(ThisArray, FindValue, Middle + 1, High)
               ELSE
                  RETURN Middle // found
           ENDIF
      ENDIF
  ENDIF
ENDFUNCTION
```

(ii) How can you recognise that the function is recursive? [1]

(iii) A binary search is carried out on the data in the Surname array shown.

	Surname
1	Ban
2	Chae
3	Dang
4	Hwang
5	Jeong
6	Jin
7	Jo
8	Ju
9	Ma
10	So
11	Song

4.1.2 Algorithms

Complete the trace table below for the following function call:

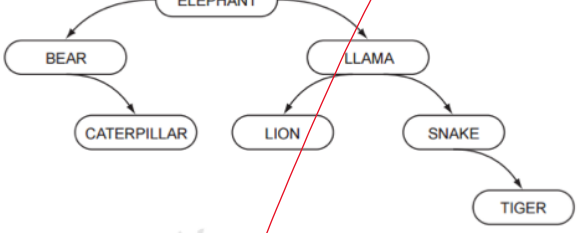
```
BinarySearch(Surname, "Hwang", 1, 11)
```

Low	High	Middle	RETURN
1	11		

[4]

May/June 2014.P31/P32

4 A binary tree is maintained for a list of animals.
The tree currently has seven animals as shown below:



(a) (i) Draw a line around the left subtree. **[1]**

(ii) State the number of leaf nodes in the current tree. **[1]**

The binary tree is implemented in a high-level language with the following data structures:

Variable	Data type	Description
RootPtr		The array subscript of the root of the tree
Data		An array of animal names
RightPtr	ARRAY[1 : 2000] OF INTEGER	Array of right pointer values
LeftPtr	ARRAY[1 : 2000] OF INTEGER	Array of left pointer values

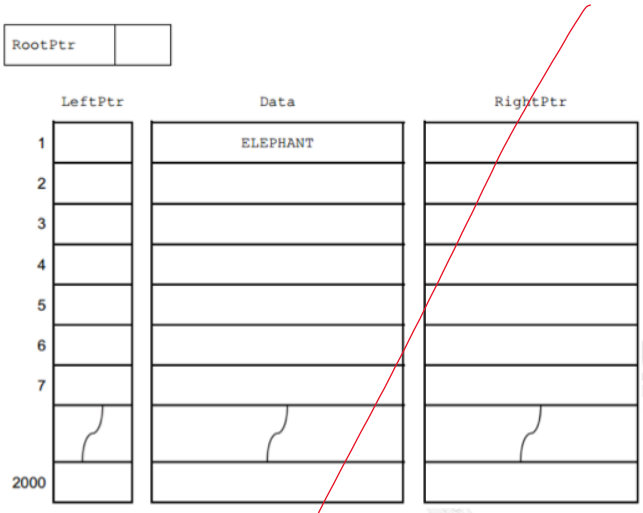
(b) Complete the two table entries above. **[3]**

(c) The animals joined the tree in the order:

4.1.2 Algorithms

ELEPHANT, LLAMA, SNAKE, BEAR, LION, CATERPILLAR, TIGER.

The diagram below is to show the contents of the arrays and the root pointer variable.
Complete the diagram.



[4]

(d) (i) How many comparisons will be made to locate LION in the current tree? **[1]**

(ii) An algorithm is designed in pseudocode. It will search the binary tree for a particular animal.

The algorithm uses the variables below:

Variable	Data type	Description
SearchAnimal	STRING	Animal to search for
Current	INTEGER	The array subscript for the item currently considered
IsFound	BOOLEAN	Flag set to TRUE when SearchAnimal is found

Complete the algorithm below:

```
//binary tree search
INPUT .....
IsFound ← FALSE
Current ← RootPtr
REPEAT
   IF .....
```

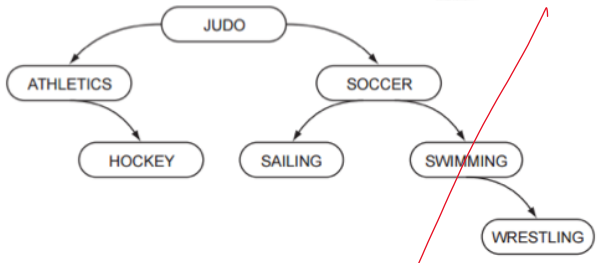
4.1.2 Algorithms

```
        THEN
           //found
           OUTPUT 'Found'
           IsFound ← TRUE
        ELSE
           IF SearchAnimal > Data[Current]
              THEN
                 // move right
                 .....

              ELSE
                 Current ← LeftPtr[Current]
           ENDIF
     ENDIF
UNTIL ..... OR Current = 0
IF .....
     THEN
       OUTPUT SearchAnimal ' not found'
ENDIF
```

[5]

May/June 2014.P33

4 A binary tree is used to store a list of sports.
The tree currently has seven sports as shown below:



The binary tree is implemented in a high-level language with the following data structures:

Variable	Data type	Description
RootPtr		Array subscript for the root of the tree
Data		Array of sport names
RightPtr	ARRAY[1 : 2000] OF INTEGER	Array of right pointer values
LeftPtr	ARRAY[1 : 2000] OF INTEGER	Array of left pointer values

Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

4.1.2 Algorithms

(a) Complete the two table entries above. **[2]**

(b) Two new sports, CYCLING and TAI-CHI are to be added.
Add these to the diagram above. **[2]**

(c) The first seven sports joined the tree in the order:
JUDO, SOCCER, SAILING, SWIMMING, ATHLETICS, WRESTLING, HOCKEY.
Complete the diagram below showing the contents of the arrays and the `RootPtr` variable.

RootPtr	

	LeftPtr	Data	RightPtr
1		JUDO	
2			
3			
4			
5			
6			
7			
2000			

[4]

(d) An algorithm is designed in pseudocode to search the binary tree for a particular sport.
The algorithm uses the variables below:

Variable	Data type	Description
SearchSport	STRING	Sport to search for
Current	INTEGER	Array subscript for the item currently considered
IsFound	BOOLEAN	Flag set to TRUE when SearchSport is found

```
//binary tree search
INPUT SearchSport
IsFound ← FALSE
Current ← RootPtr
REPEAT
  IF SearchSport = Data[Current]
     THEN
```



```
        OUTPUT 'found'
        IsFound ← TRUE
      ELSE
        IF SearchSport > Data[Current]
           THEN
             OUTPUT 'Moving right'
             Current ← RightPtr[Current]
           ELSE
             OUTPUT 'Moving left'
             Current ← LeftPtr[Current]
        ENDIF
  ENDIF
UNTIL IsFound = TRUE OR Current = 0
IF Current = 0
  THEN
    OUTPUT SearchSport ' not found'
ENDIF
```

The algorithm is tested using this new dataset.

RootPtr	1

	LeftPtr	Data	RightPtr
1	2	KARATE	3
2	0	ARCHERY	4
3	6	ROWING	5
4	0	HANDBALL	0
5	0	RUGBY	7

6	0	LACROSSE	0
7	0	SHOOTING	0
2000			

4.1.2 Algorithms

Complete the trace table below to search for LACROSSE.

SearchSport	IsFound	Current	OUTPUT
LACROSSE	FALSE	1	

[5]

Oct/Nov 2014.P32

1 (c) (i) Describe the operation of a stack. LIFO [1]

An expression in reverse Polish notation can be evaluated on a computer system using a stack.

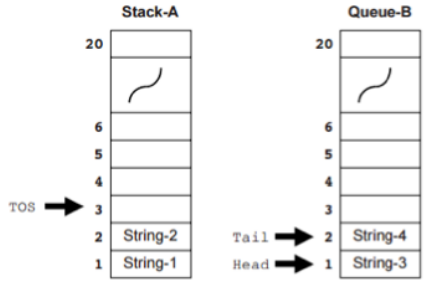
(ii) Give one further use of a stack data structure by a computer system. To control Subroutines calling. [1]

(d) (i) Describe the operation of a queue. FIFO [1]

(ii) Give one use of a queue data structure by a computer system. printing over network. [1]

(e) A stack (Stack-A) and a queue (Queue-B) are to be implemented as follows:
Stack-A has a single pointer TOS which shows the index position available for the next new item to be added ('pushed') to the stack.

Queue-B is controlled by two pointers Head and Tail.
The diagram shows Stack-A and Queue-B after two items have been added to each.



Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

Stack-A and Queue-B are both initially empty

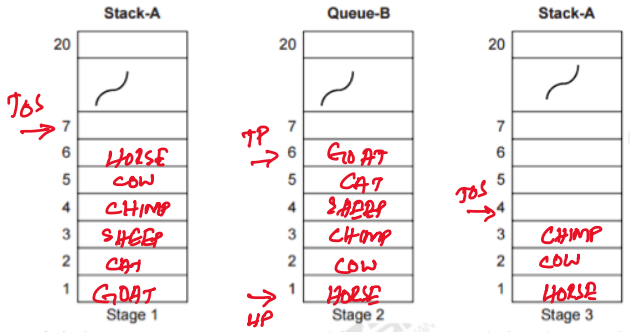
The sequence of six items of string data: GOAT CAT SHEEP CHIMP COW HORSE

is stored on Stack-A.

(i) Show on the diagram below, the contents of Stack-A and its pointer at this stage (Stage 1). **[2]**

The contents of Stack-A are then removed and placed on Queue-B.
(ii) Show on the diagram below, the contents of Queue-B and its pointers at this stage (Stage 2). **[3]**

Three items are removed from Queue-B and input to Stack-A.
(iii) Show on the diagram below, the contents of Stack-A and its pointer at this stage (Stage 3). **[2]**



(iv) Three more items are removed from Queue-B and placed on Stack-A. Comment on the final contents of Stack-A.

[1]

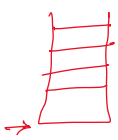
May/June 2015.P31/P32

4 (a) Briefly describe the operation of a stack data structure. **[1]**

High level language programs make extensive use of subroutines.
A stack is used to store data about each subroutine call.
The data are the return address, the register contents and the values of all local variables.
A student project is to simulate the saving and retrieval of the data for subroutine calls.
The student simplifies this for their project and will store on the stack only the return addresses.

4.1.2 Algorithms

Memory address (denary)	
100	// Main program
101	<statements>
...	
172	CALL SUB1
173	
...	// end main program
200	PROCEDURE SUB1
201	<statements>
...	
268	CALL SUB2
269	<statements>
...	
280	CALL SUB3
281	<statements>
...	
320	END PROCEDURE
321	PROCEDURE SUB2
322	<statements>
...	

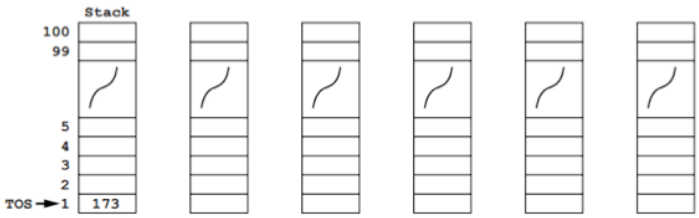


```
399  END PROCEDURE
400  PROCEDURE SUB3
401  <statements>
...
562  END PROCEDURE
```

The stack is implemented using the following data structure and variables:

Identifier	Data type	Description
Stack	ARRAY[1 : 100] OF INTEGER	Stores the return address for each procedure call
TOS	INTEGER	Stores the index position of the return address currently at the top of the stack
NewAddress	INTEGER	Stores the new return address to be added to Stack

(c)The diagram shows the stack after the first procedure call at line 172.



4.1.2 Algorithms

(i) Complete the diagram to show the contents of the stack and the value of TOS after each change. [5]

(ii) The student uses the procedure PushAddress to simulate adding a return address to the stack.
The incomplete pseudocode for the procedure PushAddress is shown below.
Using the given variables, fill in the missing pseudocode.

```
PROCEDURE PushAddress
 IF TOS = 100
   THEN
     OUTPUT "..."
   ELSE
     INPUT  ...
  TOS ← ...
        ... ← NewAddress
 ENDIF
ENDPROCEDURE
```

[4]

(c) The student uses the procedure PopAddress to simulate retrieving a return address from the stack.
Complete the pseudocode for this PopAddress procedure.

```
PROCEDURE PopAddress
  IF ...
    THEN
      OUTPUT "There are no current procedure calls"
    ELSE
      OUTPUT "Address" Stack[TOS]
      ...
  ENDIF
ENDPROCEDURE
```

[2]

May/June 2015.P33

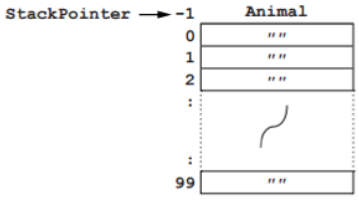
4 (a) Describe the operation of a stack data structure. **[1]**

A stack data structure is used to control the adding and removal of animal names.
The stack is implemented using the following data structure and variables.

Identifier	Data type	Description
Animal	ARRAY[0 : 99] OF STRING	Stores the animal names.
Index	INTEGER	Index pointer for the Animal array
StackPointer	INTEGER	Array index position of the item at the top of the stack. Value –1 indicates the stack is empty.
NewAnimal	STRING	Name of the new animal to be added to the Animal array.

4.1.2 Algorithms

(b) Complete the pseudocode procedure below to initialise the stack.

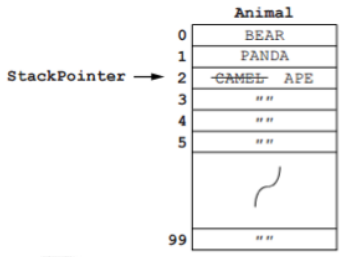



```
PROCEDURE InitialiseStack
  FOR Index ← 0 to 99
    Animal[.....] ← ""
  ENDFOR
  StackPointer ← .....
ENDPROCEDURE
```

[2]

(c) The diagram shows the state of the stack after the following operations:

- three values were added – BEAR, PANDA and CAMEL (in that order)
- a value was removed from the stack
- a new value APE was added



(i) State the current value of:

Animal[3] .....

StackPointer - 1 ..... [2]

(ii) Adding a value to the stack is done with a procedure Push.

Shown below is the incomplete pseudocode for procedure Push.

Using the variables given, fill in the missing pseudocode.

```
PROCEDURE Push
  IF .....
    THEN
      OUTPUT "REFUSED - stack is full"
    ELSE
```


Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

4.1.2 Algorithms

```
      INPUT .....
      StackPointer ← .....
      ..... ← NewAnimal
  ENDIF
ENDPROCEDURE
```

[4]

(d) Removal of a value is implemented with a procedure Pop. Write pseudocode for the procedure Pop.

(d) Removal of a value is implemented with a procedure `Pop`. Write pseudocode for the procedure `Pop`.

```
PROCEDURE Pop
ENDPROCEDURE
```
[4]

Oct/Nov 2015.P31/P33

5 (a) A dataset of city names is to be organised as an ordered binary tree.
The city names below join the binary tree in the order shown:
`PLYMOUTH, MUMBAI, DHAKA, SINGAPORE, NEW YORK, ROTTERDAM` and `TORONTO`.

(i) Draw the binary tree. [3]

(ii) On the tree drawn in **part (a)(i):**
- label the root
- draw a line around the left subtree [2]

(iii) State the number of leaf nodes for the tree drawn in **part (a)(i)**. [1]

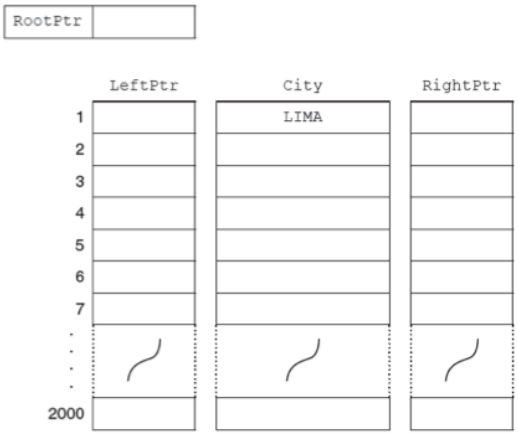
(b) The binary tree is implemented in a high-level language using a number of data structures and one variable:

Variable	Data type	Description
`RootPtr`	.....	The array subscript of the root of the tree
`City`	.....	Array of city names
`RightPtr`	`ARRAY[1 : 2000] OF INTEGER`	Array of right pointer values
`LeftPtr`	`ARRAY[1 : 2000] OF INTEGER`	Array of left pointer values

(i) Complete the entries in the table. [2]

A new dataset of cities is used as test data:
`LIMA, PARIS, KARACHI, MELBOURNE, WARSAW, CAPE TOWN, EDINBURGH`

(ii) Complete the diagram below showing the contents of the arrays and the root pointervariable.



[4]

(c) An algorithm is designed in pseudocode to search the binary tree for a particular city.
The algorithm uses the additional variables below:

Variable	Data type	Description
`SearchCity`	`STRING`	City to search for
`Current`	`INTEGER`	The array subscript for the item currently considered
`IsFound`	`BOOLEAN`	Flags to `TRUE` when `SearchCity` is found

Complete the algorithm below:

```
// binary tree search
INPUT .....
IsFound ← FALSE
Current ← RootPtr
```

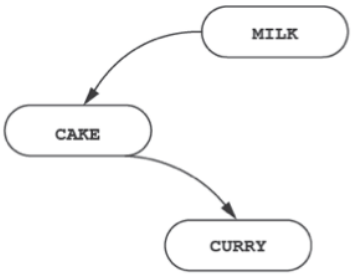
```
REPEAT
  IF City[Current] =.....
     THEN
       // found
       OUTPUT "Found"

       .....

     ELSE
       IF SearchCity > City[Current]
          THEN
            // move right
            .....
          ELSE
            Current ← LeftPtr[Current]
```



```
       ENDIF
  ENDIF
UNTIL Current = 0 OR .....
IF .....
   THEN
      OUTPUT SearchCity "Not Found"
ENDIF
```
[6]

Oct/Nov 2015.P32

4 (a) A dataset of food names are to be organised as an ordered binary tree.
The food names below join the binary tree in the order shown:
MILK, CAKE, CURRY



Four more food names join the binary tree in the order shown:
SCONES, RICE, BREAD, COUSCOUS

(i) Draw these values on the binary tree above. **[4]**

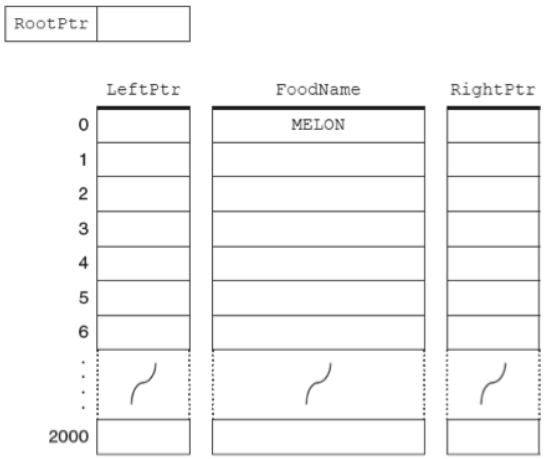
(ii) On the binary tree above, write on the tree as follows:
- label the root
- draw a line around the right subtree **[2]**

(iii) State the number of leaf nodes for the completed tree in **part (a)(i)**. **[1]**

(b) The binary tree is implemented in a high-level language using a number of data structures and one variable:

Variable	Data type	Description
RootPtr	INTEGER	The array subscript of the root of the tree
FoodName	ARRAY[0 : 2000] OF STRING	Array of food names
RightPtr	ARRAY[0 : 2000] OF INTEGER	Array of right pointer values
LeftPtr	ARRAY[0 : 2000] OF INTEGER	Array of left pointer values

A new dataset of food names is used as test data:
MELON, BEETROOT, TURNIP, APPLE, PARSNIP, SWEDE, QUINCE
Complete the diagram below showing the contents of the arrays and the root pointer variable.



[4]

(c) An algorithm is designed in pseudocode to search the binary tree for a particular food name.
The algorithm uses the additional variables below:

Variable	Data type	Description
SearchFood	STRING	Food name to search for
Current	INTEGER	The array subscript for the item currently considered
IsFound	BOOLEAN	Flags to TRUE when SearchFood is found

Complete the algorithm below:

```
//binary tree search
INPUT SearchFood
IsFound ← .....
Current ← .....
REPEAT
   IF Food[Current] = .....
      THEN
         //found
         OUTPUT "Found"
         .....
      ELSE
        IF SearchFood < Food[Current]
           THEN
             // move left
             Current ← LeftPtr[Current]
           ELSE
           .....
        ENDIF
   ENDIF
```



```
UNTIL IsFound = TRUE OR .....
IF IsFound = FALSE
  THEN
     OUTPUT SearchFood "Not Found"
ENDIF
```

[6]

(d) (i) The dataset used in **part (b)** was:
MELON, BEETROOT, TURNIP, APPLE, PARSNIP, SWEDE, QUINCE
Draw the binary tree. [2]
A well-balanced tree has approximately the same number of nodes in the left and right subtrees.
(ii) State whether or not this is a well-balanced tree. [1]

Computer Science (9608)

May/June 2015.P41/P42

5 Data is stored in the array NameList[1:10]. This data is to be sorted.
(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO ……
  // use a temporary variable to store item which is to
  // be inserted into its correct location
  Temp ← NameList[ThisPointer]
  Pointer ← ThisPointer - 1

  WHILE (NameList[Pointer] > Temp) AND ……
     // move list item to next location
     NameList[……] ← NameList[……]
     Pointer ← ……
  ENDWHILE

  // insert value of Temp in correct location
  NameList[……] ← ……
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in part (a)(i) is applied to this special case.
Explain how many iterations are carried out for each of the loops. **[3]**

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
   FOR Pointer ← 1 TO 9
     IF NameList[Pointer] > NameList[Pointer + 1]
        THEN
           Temp ← NameList[Pointer]
           NameList[Pointer] ← NameList[Pointer + 1]
           NameList[Pointer + 1] ← Temp
     ENDIF
```



```
   ENDFOR
ENDFOR
```

(i) As in part (a)(ii), a special case is when NameList is already in order. The algorithm in part (b) is applied to this special case.
Explain how many iterations are carried out for each of the loops. **[2]**

(ii) Rewrite the algorithm in part (b), using pseudocode, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate. **[5]**

May/June 2015.P43

5 A stack Abstract Data Type (ADT) has these associated operations:

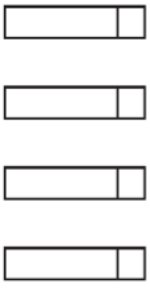
- create stack
- add item to stack (push)
- remove item from stack (pop)

The stack ADT is to be implemented as a linked list of nodes.
Each node consists of data and a pointer to the next node.

(a) There is one pointer: the top of stack pointer, which points to the last item added to the stack.
Draw a diagram to show the final state of the stack after the following operations are carried out.

```
CreateStack
Push("Ali")
Push("Jack")
Pop
Push("Ben")
Push("Ahmed")
Pop
Push("Jatinder")
```

Add appropriate labels to the diagram to show the final state of the stack. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

4.1.2 Algorithms

(b) Using pseudocode, a record type, Node, is declared as follows:

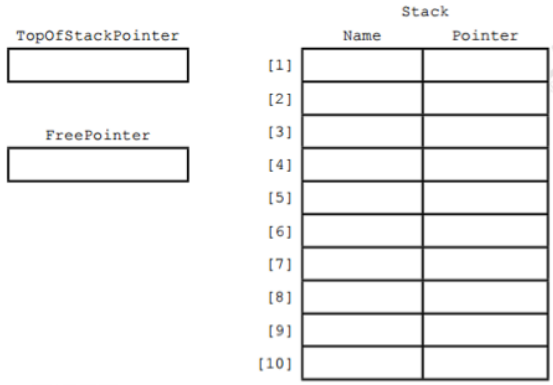
```
TYPE Node
  DECLARE Name : STRING
  DECLARE Pointer : INTEGER
ENDTYPE
```

The statement

```
DECLARE Stack : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Stack`.

(i) The `CreateStack` operation links all nodes and initialises the `TopOfStackPointer` and `FreePointer`.
Complete the diagram to show the value of all pointers after `CreateStack` has been executed.



[4]

(ii) The algorithm for adding a name to the stack is written, using pseudocode, as a procedure with the header
`PROCEDURE Push (NewName)`
Where `NewName` is the new name to be added to the stack. The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
`Stack`	`Array[1:10] OF Node`	
`NewName`	`STRING`	Name to be added
`FreePointer`	`INTEGER`	Pointer to next free node in array
`TopOfStackPointer`	`INTEGER`	Pointer to first node in stack
`TempPointer`	`INTEGER`	Temporary store for copy of `FreePointer`

Computer Science 9608
Topical Past Papers
Zak ZAFAR ALI KHAN

4.1.2 Algorithms

```
PROCEDURE Push(BYVALUE NewName : STRING)
// Report error if no free nodes remaining
IF FreePointer = 0
  THEN
     Report Error
  ELSE
```

```
  ELSE
    // new name placed in node at head of free list
    Stack[FreePointer].Name ← NewName
    // take a temporary copy and
    // then adjust free pointer
    TempPointer ← FreePointer
    FreePointer ← Stack[FreePointer].Pointer
    // link current node to previous top of stack
    Stack[TempPointer].Pointer ← TopOfStackPointer
    // adjust TopOfStackPointer to current node
    TopOfStackPointer ← TempPointer
ENDIF
ENDPROCEDURE
```

Complete the pseudocode for the procedure `Pop`. Use the variables listed in the identifier table.

```
PROCEDURE Pop()
  // Report error if Stack is empty
  ..........
  ..........
  ..........
  ..........
    OUTPUT Stack [..........].Name
    // take a copy of the current top of stack pointer
  ..........
    // update the top of stack pointer
  ..........
    // link released node to free list
  ..........
  ..........
  ..........
ENDPROCEDURE
```

[5]

May/June 2018.P41/43

2 The array `ItemList[1:20]` stores data. A **bubble sort** sorts these data.

(a) Complete the pseudocode algorithm for a bubble sort.

4.1.2 Algorithms

```
01   MaxIndex ← 20
02   NumberItems ← ..........
03   FOR Outer ← 1 TO ..........
04      FOR Inner ← 1 to NumberItems
05         IF ItemList[Inner] > ..........
06            THEN
07               Temp ← ItemList[..........]
08               ItemList[Inner] ← ItemList[..........]
09               ItemList[Inner + 1] ← ..........
10         ENDIF
11      ENDFOR
12      NumberItems ← ..........
13   ENDFOR
```

[7]

(b) The algorithm in **part (a)** is inefficient.

(i) Explain why the algorithm in **part (a)** is inefficient. **[2]**

(ii) Explain how you would improve the efficiency of this algorithm. **[3]**

(c) An insertion sort is another sorting algorithm.

State **two** situations when an insertion sort is more efficient than a bubble sort. Give a reason for each. **[4]**