4 Study the algorithm:

```
DECLARE Chars : ARRAY[1:4] OF CHAR
DECLARE I, J : INTEGER
DECLARE Key : CHAR
I ← 2
Chars[1] ← 'D'
Chars[2] ← 'T'
Chars[3] ← 'H'
Chars[4] ← 'R'
WHILE I <= 4                              //Outer loop
   Key ← Chars[I]
   J ← I - 1
   WHILE J >= 0 AND Chars[J] > Key        //Inner loop
      Chars[J + 1] ← Chars[J]
      J ← J - 1
   ENDWHILE
   Chars[J + 1] ← Key
   I ← I + 1
ENDWHILE
```

(a) The outer loop structure used in the algorithm is **not** the most appropriate one to use.

State the type of loop structure that would be the most appropriate to use and justify why it is the most appropriate.

Loop structure .....

Justification .....

.....

[2]

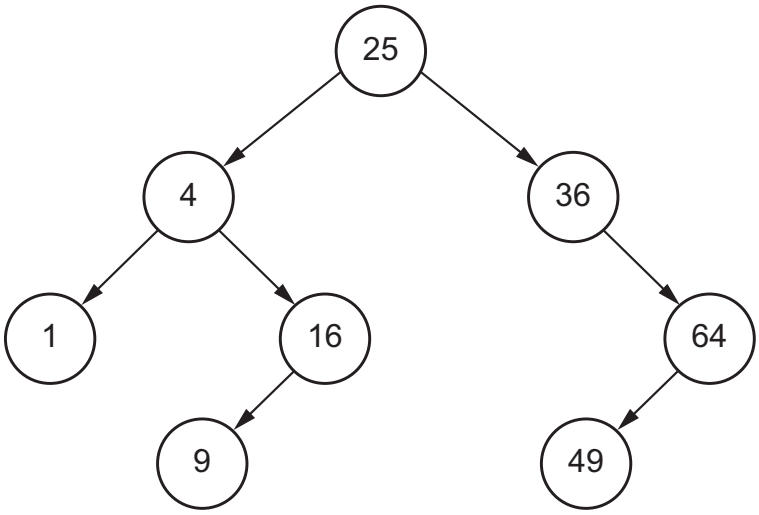
(b) Complete the trace table by dry running the algorithm.

The first row has been completed.

I	Key	J	Chars[J]	Chars			
				[1]	[2]	[3]	[4]
2				'D'	'T'	'H'	'R'
				(2)			

[6]

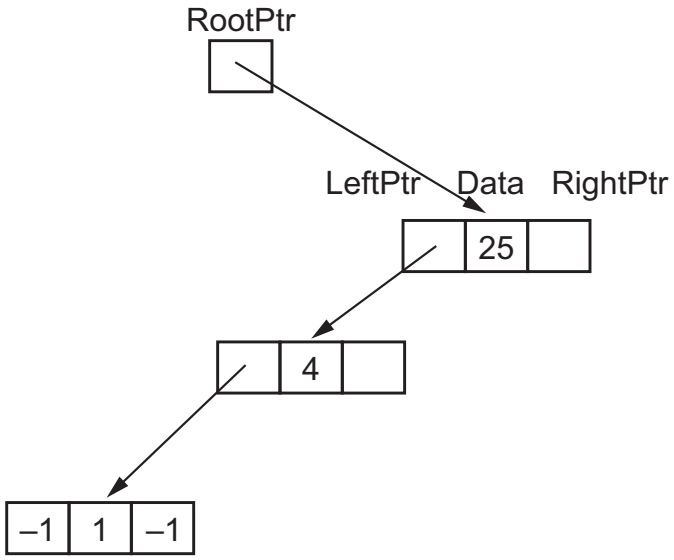
11 This binary tree shows an ordered list of integers.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

−1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree organisation.



[4]

(b) A 2D array is used to store the nodes of the linked list in part **(a)**.

Complete the diagram using your answer for part **(a)**.

RootPtr

0

FreePtr

Index	LeftPtr	Data	RightPtr
0		25	
1		4	
2		36	
3		1	
4		16	
5		64	
6		9	
7		49	
8			

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

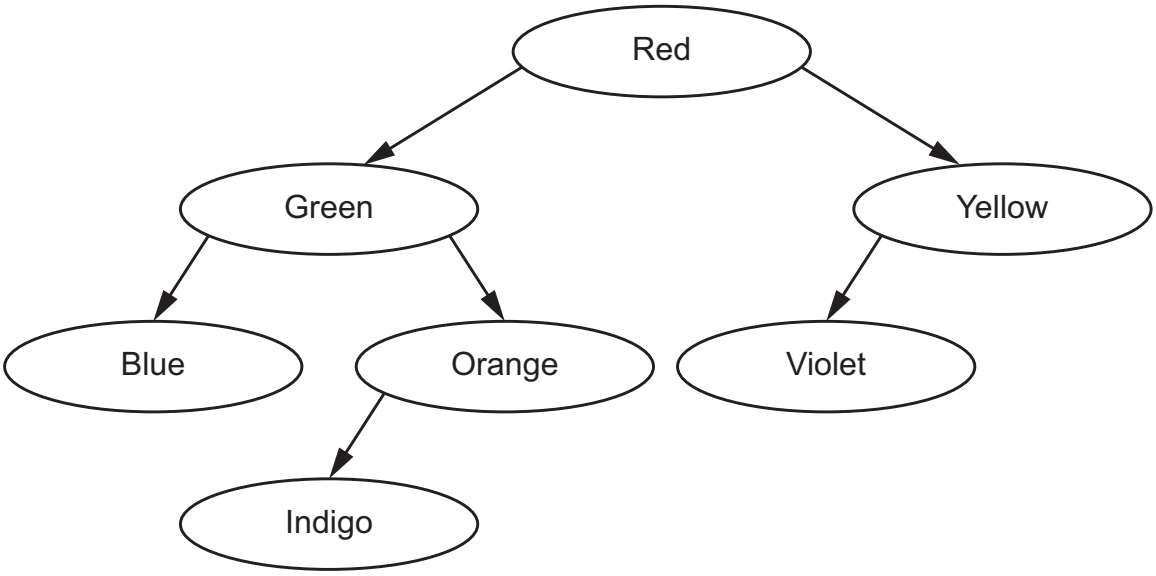
The following pseudocode represents a function that searches for an element in the array of records `LinkList`. It returns the index of the record if the element is found, or it returns a null pointer if the element is not found.

Complete the pseudocode for the function.

```
FUNCTION SearchList(Item : INTEGER).....
   NullPtr ← -1

   ..... ← RootPtr
   WHILE NowPtr <> NullPtr
      IF LinkList[NowPtr].Data < Item THEN
         NowPtr ← LinkList[NowPtr].RightPtr
      ELSE

         IF ..... THEN

            NowPtr ← .....
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NullPtr
ENDFUNCTION
```

[4]

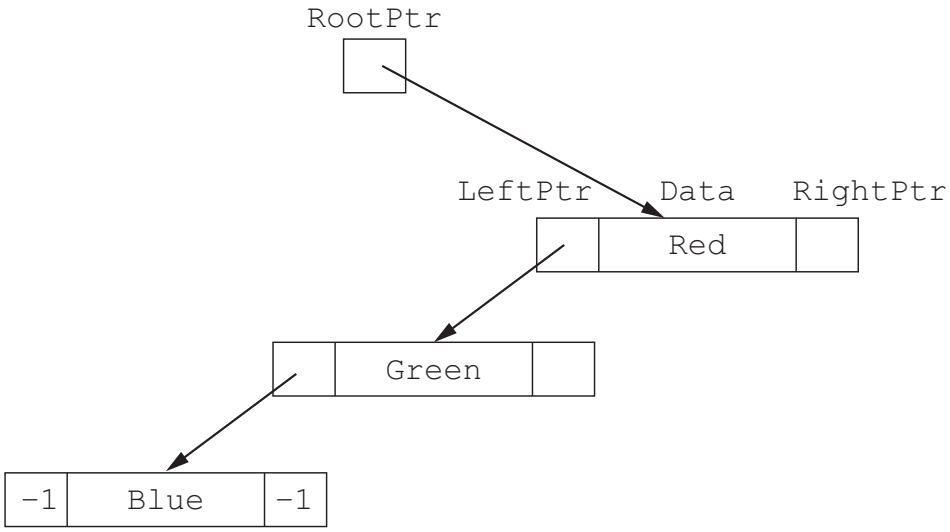
11 The following diagram shows an ordered binary tree.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

–1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree.



[4]

(b) A user-defined record structure is used to store the nodes of the linked list in part **(a)**.

Complete the diagram, using your answer for part **(a)**.

RootPtr

0

Index	LeftPtr	Data	RightPtr
0		Red	
1		Green	
2		Yellow	
3		Blue	
4		Orange	
5		Indigo	
6		Violet	
7			

FreePtr

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

The following pseudocode represents a function that searches for an element in the array of records `BinTree`. It returns the index of the record if the element is found, or it returns a null pointer if the element is **not** found.

Complete the pseudocode for the function.

```
FUNCTION SearchTree(Item : STRING) .....

   NowPtr ← .....
   WHILE NowPtr <> -1

      IF ..... THEN
         NowPtr ← BinTree[NowPtr].LeftPtr
      ELSE
         IF BinTree[NowPtr].Data < Item THEN

            .....
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NowPtr
ENDFUNCTION
```


[4]

1 A program sorts string data using different sorting methods.

(a) The text file `Data.txt` stores string data items. Each data item is on a new line in the text file.

The function `ReadData()`:

- has a local array of strings that can store 45 items
- reads each line of data and stores it in the array
- returns the array.

Write program code for the function `ReadData()`.

Save your program as **Question1_N24**.

Copy and paste the program code into part **1(a)** in the evidence document.

[6]

(b) The function `FormatArray()` takes an array of strings as a parameter. It concatenates the contents of the array into one string with a space between each array element. The function returns the concatenated string.

(i) Write program code for `FormatArray()`.

Save your program.

Copy and paste the program code into part **1(b)(i)** in the evidence document.

[2]

(ii) The main program:

- calls `ReadData()` and stores the returned array
- calls `FormatArray()` with the returned array and outputs the returned string.

Write program code for the main program.

Save your program.

Copy and paste the program code into part **1(b)(ii)** in the evidence document.

[3]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(b)(iii)** in the evidence document.

[1]

(c) The function `CompareStrings()`:

- takes two strings as parameters
- compares each string, one character at a time, to identify which string comes first alphabetically. If the first two characters are the same, the second character of each string is compared. This continues until the two characters are different.

The function:

- returns `1` if the first parameter comes before the second alphabetically
- returns `2` if the second parameter comes before the first alphabetically.

Write program code for `CompareStrings()`.

Assume that all strings are in lower case.

Assume that a difference between two strings will always be identified before the end of one string is reached.

Do **not** use an in-built string comparison function.

The strings must be compared one character at a time.

Save your program.

Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The function `Bubble()` takes an array of strings as a parameter and sorts the data into ascending alphabetical order, using a bubble sort. The bubble sort uses `CompareStrings()` to compare each string.

The function returns the sorted list.

(i) Write program code for `Bubble()`.

Save your program.

Copy and paste the program code into part **1(d)(i)** in the evidence document.

[3]

(ii) Write program code to amend the main program to:

- call `Bubble()` with the unsorted array as a parameter
- call `FormatArray()` with the sorted array and output the returned string.

Save your program.

Copy and paste the program code into part **1(d)(ii)** in the evidence document.

[2]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(d)(iii)** in the evidence document.

[1]

1 A program sorts string data using different sorting methods.

(a) The text file `Data.txt` stores string data items. Each data item is on a new line in the text file.

The function `ReadData()`:

- has a local array of strings that can store 45 items
- reads each line of data and stores it in the array
- returns the array.

Write program code for the function `ReadData()`.

Save your program as **Question1_N24**.

Copy and paste the program code into part **1(a)** in the evidence document.

[6]

(b) The function `FormatArray()` takes an array of strings as a parameter. It concatenates the contents of the array into one string with a space between each array element. The function returns the concatenated string.

(i) Write program code for `FormatArray()`.

Save your program.

Copy and paste the program code into part **1(b)(i)** in the evidence document.

[2]

(ii) The main program:

- calls `ReadData()` and stores the returned array
- calls `FormatArray()` with the returned array and outputs the returned string.

Write program code for the main program.

Save your program.

Copy and paste the program code into part **1(b)(ii)** in the evidence document.

[3]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(b)(iii)** in the evidence document.

[1]

(c) The function `CompareStrings()`:

- takes two strings as parameters
- compares each string, one character at a time, to identify which string comes first alphabetically. If the first two characters are the same, the second character of each string is compared. This continues until the two characters are different.

The function:

- returns `1` if the first parameter comes before the second alphabetically
- returns `2` if the second parameter comes before the first alphabetically.

Write program code for `CompareStrings()`.

Assume that all strings are in lower case.

Assume that a difference between two strings will always be identified before the end of one string is reached.

Do **not** use an in-built string comparison function.

The strings must be compared one character at a time.

Save your program.

Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The function `Bubble()` takes an array of strings as a parameter and sorts the data into ascending alphabetical order, using a bubble sort. The bubble sort uses `CompareStrings()` to compare each string.

The function returns the sorted list.

(i) Write program code for `Bubble()`.

Save your program.

Copy and paste the program code into part **1(d)(i)** in the evidence document.

[3]

(ii) Write program code to amend the main program to:

- call `Bubble()` with the unsorted array as a parameter
- call `FormatArray()` with the sorted array and output the returned string.

Save your program.

Copy and paste the program code into part **1(d)(ii)** in the evidence document.

[2]

(iii) Test your program.

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **1(d)(iii)** in the evidence document.

[1]