# 3.2 Tree traversal Vectors Mark Scheme

## Q1.

(a) **Mark is for AO1 (knowledge)**

A subroutine that calls itself;

**1**

(b) **Mark is for AO1 (understanding)**
When target equals node // (When target does not equal node and) node is a leaf // `node = target;`

**1**

(c) **Marks are for AO2 (apply)**

| Function Call | Output |
|---|---|
| `TreeSearch(Olivia, Norbert)` | (Visited) Norbert; |
| `TreeSearch(Olivia, Phil);` | (Visited) Phil; |
| | |

**MAX 2** if any errors eg additional outputs / function calls after output of Phil

**I**. minor spelling and punctuation errors

**3**

**[5]**

## Q2.

(a) John, Rachel, Paul;
**R.** If not in correct order
**I.** Incorrect spellings of names, as long as the name is comprehensible
**I.** Quotation marks

**1**

(b)

| Time Complexity | Tick one box |
|---|---|
| O(n) | |
| O(log n) | ✔ |
| $O(n^2)$ | |

**A.** Alternative symbol which clearly indicates just one box
e.g. cross, Y, Yes

**R.** Answers in which more than one row is ticked

**1**

(c) **1 mark** for Start Index containing the index of the array item storing the root node (John)
**1 mark** for John, Hannah and Rachel being stored, each with left and right pointer values storing the indices of the correct child nodes

**1 mark** for Bradley, Jo, Paul and Tina being stored, each with left and right pointers storing appropriate rogue values e.g. -1, 0, NIL, NULL **R.** A dash or blank as the rogue value

**Three example solutions are shown below, but the data items can be stored at any positions within the array, as long as the pointers correctly reflect this.**

**Start Index = 1**

| Index | Left Pointer | Data | Right Pointer |
|---|---|---|---|
| [1] | 2 | John | 3 |
| [2] | 4 | Hannah | 5 |
| [3] | 6 | Rachel | 7 |
| [4] | –1 | Bradley | –1 |
| [5] | –1 | Jo | –1 |
| [6] | –1 | Paul | –1 |
| [7] | –1 | Tina | –1 |

**Start Index = 4**

| Index | Left Pointer | Data | Right Pointer |
|---|---|---|---|
| [1] | –1 | Bradley | –1 |
| [2] | –1 | Jo | –1 |
| [3] | 1 | Hannah | 2 |
| [4] | 3 | John | 6 |
| [5] | –1 | Paul | –1 |
| [6] | 5 | Rachel | 7 |
| [7] | –1 | Tina | –1 |

**Start Index = 1**

| Index | Left Pointer | Data | Right Pointer |
|---|---|---|---|
| [1] | 2 | John | 5 |
| [2] | 3 | Hannah | 4 |
| [3] | –1 | Bradley | –1 |
| [4] | –1 | Jo | –1 |
| [5] | 6 | Rachel | 7 |
| [6] | –1 | Paul | –1 |
| [7] | –1 | Tina | –1 |



**3**

(d) **Difference between Static and Dynamic (2 marks):**

Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time whereas size of dynamic structure can change at run-time;
Static structures can waste storage space/memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;
Static structures (typically) store data in consecutive memory

locations, which dynamic data structures (typically) do not // Dynamic data structures (can) (require memory to) store pointer(s) to the (next items which static structures typically do not need); **MAX 2**
**A.** Just one side of points, other side is by implication

**Heap (1 mark):**
Memory allocated/deallocated at run-time/for new items (to dynamic data structure);
(Provides a) pool of free/unused/available memory;
**N.E.** To store new items

**3**

(e) (i) Bradley, Hannah, Jo, John, Paul, Rachel, Tina;
**R.** If not in correct order
**I.** Incorrect spelling of names, so long as name is comprehensible
**I.** Quotation marks

**1**

(ii) (Ascending) Alphabetic order;
**A.** Alphabetic, it is sorted

**1**

(f) Graph may contain cycles / loops / circuits (so must keep track of which nodes already visited);
Graph may not be connected (so some nodes may be unreachable);
Graph may be weighted (so a more complex algorithm that accounts for the weights may be required);
**N.E.** Graphs can be directed

**MAX 1**

**1**

**[11]**



## Q3.

(a)

| Algorithm Name | Requires Sorted List? (Tick one box) |
|---|---|
| Binary search | ✓ |
| Linear search | |

**1 mark** for having a tick in the "Binary search" row.
**A** alternative indicators for tick eg "Yes"
**A** a tick for "Binary search" and a cross for "Linear search"
**R** answers where two ticks have been used.

**1**

(b)

| | | | | List | | | |
|---|---|---|---|---|---|---|---|
| List Length | Outer Pointer | Current Value | Inner Pointer | [1] | [2] | [3] | [4] |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | 9 | 8 | 5 | 6 |
| 4 | 2 | 8 | 1 | | 9 | | |
| | | | 0 | 8 | | | |
| | 3 | 5 | 2 | | | 9 | |
| | | | 1 | | 8 | | |
| | | | 0 | 5 | | | |
| | 4 | 6 | 3 | | | | 9 |
| | | | 2 | | | 8 | |
| | | | 1 | | 6 | | |

Award **1 mark** for each of the highlighted rectangles which has the correct values written in it in the unshaded cells.
Accept responses in which correct values are unnecessarily written out again.
Do not award a mark for any rectangle which has an incorrect value written in it.

**3**

(c) The value being moved / CurrentValue / 6 does not need to be put at the start of the list / / should be inserted at position 2 not position 1;
Because the second condition (in the While statement) is not satisfied;
**MAX 1**

**1**

(d)

| Order of Time Complexity | Tick one box |
|---|---|
| O(n) | |
| $O(n^2)$ | ✓ |
| $O(2^n)$ | |

**A** alternative indicators instead of a tick eg a cross, Y, Yes
**R** responses in which more than one box is ticked

**1**

(e) Insertion sort;
**A** Insert sort

**1**

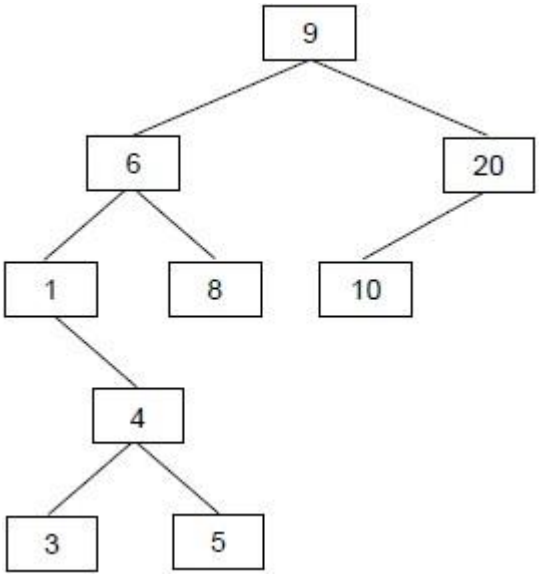
(f) (i) 9, 6, 8;
Must be in the order above. Can be separated by any character or a space

**1**

(ii) 9, 20,10;
Must be in the order above. Can be separated by any character or a space.

1

(g)




**1 mark** for inserting number 4 in the correct place
**1 mark** for inserting both numbers 3 and 5 in the correct place relative to 4
**MAX 1** if any numbers added in the wrong place / any extra numbers added

2

**[11]**

## Q4.

(a) +;
4, 9, 6; (in any order)

2

(b) **A** Store the data / value (in the vertices / nodes);
A: holds the expression
B: Left pointer // points to the left child / left sub tree;
C: Right pointer // points to the right child / right sub tree;
**A** "indicates", "index" or other synonym for "points" / "pointer"
**R** Stores left / right subtree

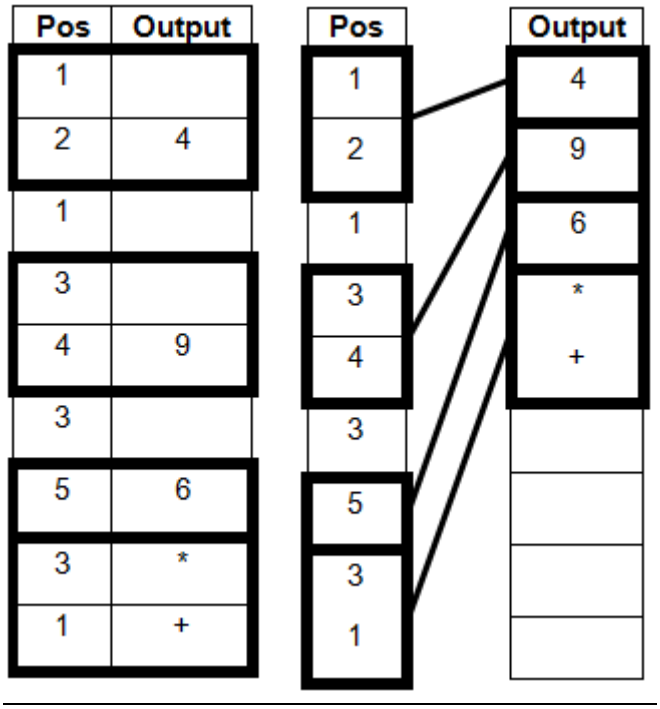
3

(c) The node has no left child / sub tree;
**A** there is nothing to the left
**A** this is a null pointer

1

(d) One mark for each area outlined with a dark rectangle. Lines that are not outlined can be missed out.

Alternative 1Alternative 2




Mark against whichever alternative gives the highest mark.

**Stop marking as soon as incorrect output is given.**

**4**

(e) Post-order;
**A** Depth-first
**A** Depth-first search as BOD
**TO** Depth-first pre / in-order

**1**

(f) ( 4 + 9 * 6 in) Reverse Polish (Notation) // Postfix (Notation) // RPN;

**1**

**[12]**



## Q5.

(a) **Connected** // There is a path between each pair of vertices;
**Undirected** // No direction is associated with each edge;
**Has no cycles** // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;
**A** no loops
**Alternative definitions:**
A simple cycle is formed if any edge is added to graph;
Any two vertices can be connected by a unique simple path;

**Max 1**

(b) No route from entrance to exit / through maze;
Maze contains a loop/circuit ;
**A** more than one route through maze;
Part of the maze is inaccessible / enclosed;
**R** Responses that clearly relate to a graph rather than the maze

**Max 1**

(c)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

(allow some symbol in the central diagonal to indicate unused)

**or**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | | 0 | 1 | 1 | 0 | 0 | 0 |
| 3 | | | 0 | 0 | 0 | 0 | 0 |
| 4 | | | | 0 | 1 | 0 | 0 |
| 5 | | | | | 0 | 1 | 1 |
| 6 | | | | | | 0 | 0 |
| 7 | | | | | | | 0 |

(with the shaded portion in either half – some indication must be made that half of the matrix is not being used. This could just be leaving it blank, unless the candidate has also represented absence of an edge by leaving cells blank)

*1 mark for drawing a 7x7 matrix, labelled with indices on both axis and filled only with 0s and 1s, or some other symbol to indicate presence/absence of edge. e.g. T/F. Absence can be represented by an empty cell.*
*1 mark for correct values entered into matrix, as shown above;*

**2**

(d) (i) Routine defined in terms of itself // Routine that calls itself;
**A** alternative names for routine e.g. procedure, algorithm
**NE** repeats itself

**1**

(ii) Stores return addresses;
Stores parameters;
Stores local variables; NE temporary variables
Stores contents of registers;
**A** To keep track of calls to subroutines/methods etc.

*Max 1*

Procedures / invocations / calls must be returned to in reverse order (of being called);
As it is a LIFO structure;
**A** FILO
As more than one / many return addresses / sets of values may need to be stored (at same time) // As the routine calls itself and for each call/invocation a new return address / new values must be stored;

*Max 1*

**2**

(e)

| | | | | Discovered | | | | | | | Completely Explored | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Call | V | U | EndV | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | F |
| | - | - | 7 | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| DFS(1,7) | 1 | 2 | 7 | **T** | F | F | F | F | F | F | F | F | F | F | F | F | F | F |
| DFS(2,7) | 2 | 1 | 7 | T | **T** | F | F | F | F | F | F | F | F | F | F | F | F | F |
| | | 3 | 7 | T | T | F | F | F | F | F | F | F | F | F | F | F | F | F |
| DFS(3,7) | 3 | 2 | 7 | T | T | **T** | F | F | F | F | F | F | **T** | F | F | F | F | F |
| DFS(2,7) | 2 | 4 | 7 | T | T | T | F | F | F | F | F | F | T | F | F | F | F | F |
| DFS(4,7) | 4 | 2 | 7 | T | T | T | **T** | F | F | F | F | F | T | F | F | F | F | F |
| | | 5 | 7 | T | T | T | T | F | F | F | F | F | T | F | F | F | F | F |
| DFS(5,7) | 5 | 4 | 7 | T | T | T | T | **T** | F | F | F | F | T | F | F | F | F | F |
| | | 6 | 7 | T | T | T | T | T | F | F | F | F | T | F | F | F | F | F |
| DFS(6,7) | 6 | 5 | 7 | T | T | T | T | T | **T** | F | F | F | T | F | F | **T** | F | F |
| DFS(5,7) | 5 | 7 | 7 | T | T | T | T | T | T | F | F | F | T | F | F | T | F | F |
| DFS(7,7) | 7 | 5 | 7 | T | T | T | T | T | T | **T** | F | F | T | F | F | T | **T** | **T** |
| DFS(5,7) | 5 | - | 7 | T | T | T | T | T | T | T | F | F | T | F | **T** | T | T | T |
| DFS(4,7) | 4 | - | 7 | T | T | T | T | T | T | T | F | F | T | **T** | T | T | T | T |
| DFS(2,7) | 2 | - | 7 | T | T | T | T | T | T | T | F | **T** | T | T | T | T | T | T |
| DFS(1,7) | 1 | - | 7 | T | T | T | T | T | T | T | **T** | T | T | T | T | T | T | T |



*1 mark for having the correct values changes in each region highlighted by a rectangle and no incorrect changes in the region. Ignore the contents of any cells that are not changed.*

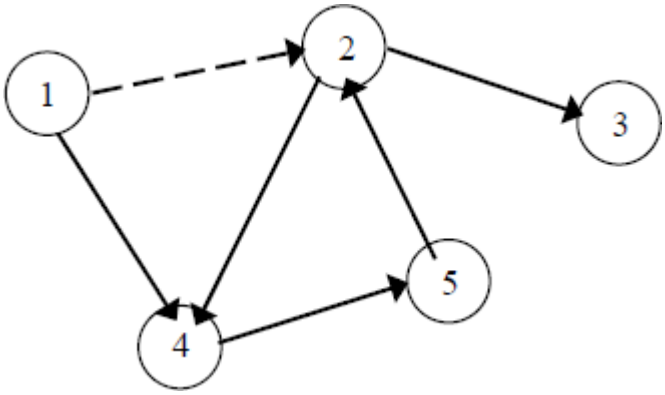
**A** alternative indicators that clearly mean True and False.
**A** it is not necessary to repeat values that are already set (shown lighter in table)

**5**

**[12]**

## Q6.

(a)




*1 mark for all 5 lines correctly drawn*
*1 mark for all 5 arrowheads pointing in correct directions*
*Max 1 if more than 5 lines drawn by candidate (note that dotted arrow is given in question)*

**A** arrowheads at any position on line

**2**

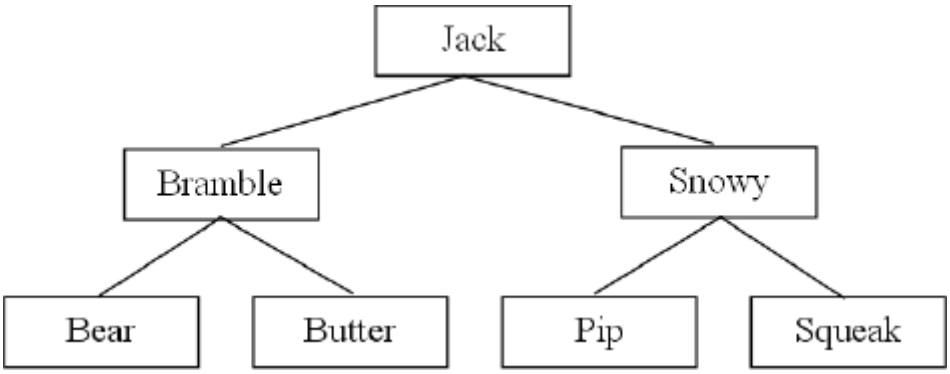
(b) Adjacency matrix appropriate when there are many edges between vertices // when edges may be frequently changed // when presence/absence of specific edges needs to be tested (frequently)
Adjacency list appropriate when there are few edges between vertices // when graph is sparse // when edges rarely changed //when presence/absence of specific edges does not need to be tested (frequently)
**A** alternative words which describe edge e.g. connection, line

**2**

(c) Connected // There is a path between each pair of vertices;
Undirected // No direction is associated with each edge;
Has no cycles // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;
**Alternative definitions:**
Graph with no cycles, and a simple cycle is formed if any edge is added to it;;
Graph which is connected, and it is not connected anymore if any edge is removed from it;;
Graph in which any two vertices can be connected by a unique simple path;;
(Note: not just adjacent vertices)
Graph which is connected and has n - 1 edges where n is no of vertices;;
Graph which has no simple cycles and has n - 1 edges where n is no of vertices;;

**Max 2**

(d)




*1 mark for Jack as root*
*1 mark for Bramble and Snowy as children of Jack*
*1 mark for four correct children of Bramble and Snowy*

**DPT** if arrows drawn instead of lines
**DPT** if any node has more than 2 child nodes
**A** "mirror image" answers which are consistent.

**3**

(e) **For solution with 3 arrays:**
One array stores data items;
One array for left child pointers;
One array for right child pointers;
Pointers stored at same location in arrays as corresponding data item;
**For solution with 1 array of records:**
Record created to store data item and pointers;
One pointer to left child;
One pointer to right child;
**For either of the above solutions**:
Rogue value (allow example) used to indicate no child;
Variable indicates position in array(s) of root node // Root node stored at first location/start of array(s);
**If answered as diagram:**
Column for data with at least three correct data items in it;
Use of rogue value for a node that does not have child;
Correct value for a start pointer variable indicating position of root node in the array (not drawn as an arrow, array indices must be labelled);
Column for left child pointers*;
Column for right child pointers*;
* = To get these marks, there must be a sufficient number of pointers to demonstrate that the data structure is a representation of a binary tree, but it is not necessary for every item to be shown. Also the array indices must be shown.

**Max 3**

**[12]**



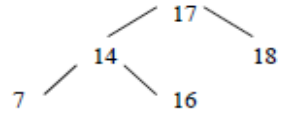
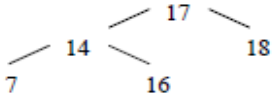
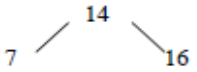
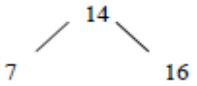
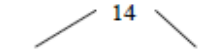
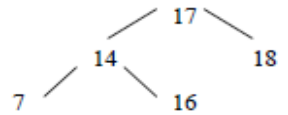
## Q7.

(a) A procedure/routine that calls itself/ is defined in terms of itself;

**A** Function instead of procedure
**R** re-entrant
**R** program iteration (TO)

**1**

(b) (i)

| Procedure Call | T |
|---|---|
| $P_1$ |   |
| $P_2$ | 18 ; |
| $P_1$ |   |
| $P_3$ |   ; |
| $P_4$ | 16 ; |
| $P_3$ |   |
| $P_5$ | 7 ; |
| $P_3$ |   |
| $P_1$ |   |



Output

| | | | | |
|---|---|---|---|---|
| 18; | 17; | 16 | 14 | 7; |

**7**

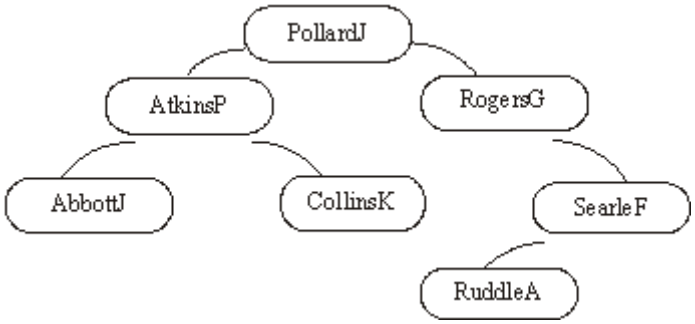
(ii) Reversed Inorder; Tree traversal;
**I** Sort/ Re-arrange

**2**

**[10]**

## Q8.

(a)




Correct root + left subtree;
Correct root + right subtree;

**I.** identification of PollardJ as the root
**A** a complete 'left-right' mirrored image.

**2**

(b) (i) PollardJ, AtkinsP, CollinsK
from a correctly drawn left sub-tree;

**1**

(ii) 4 from a correctly drawn right sub-tree;

**1**

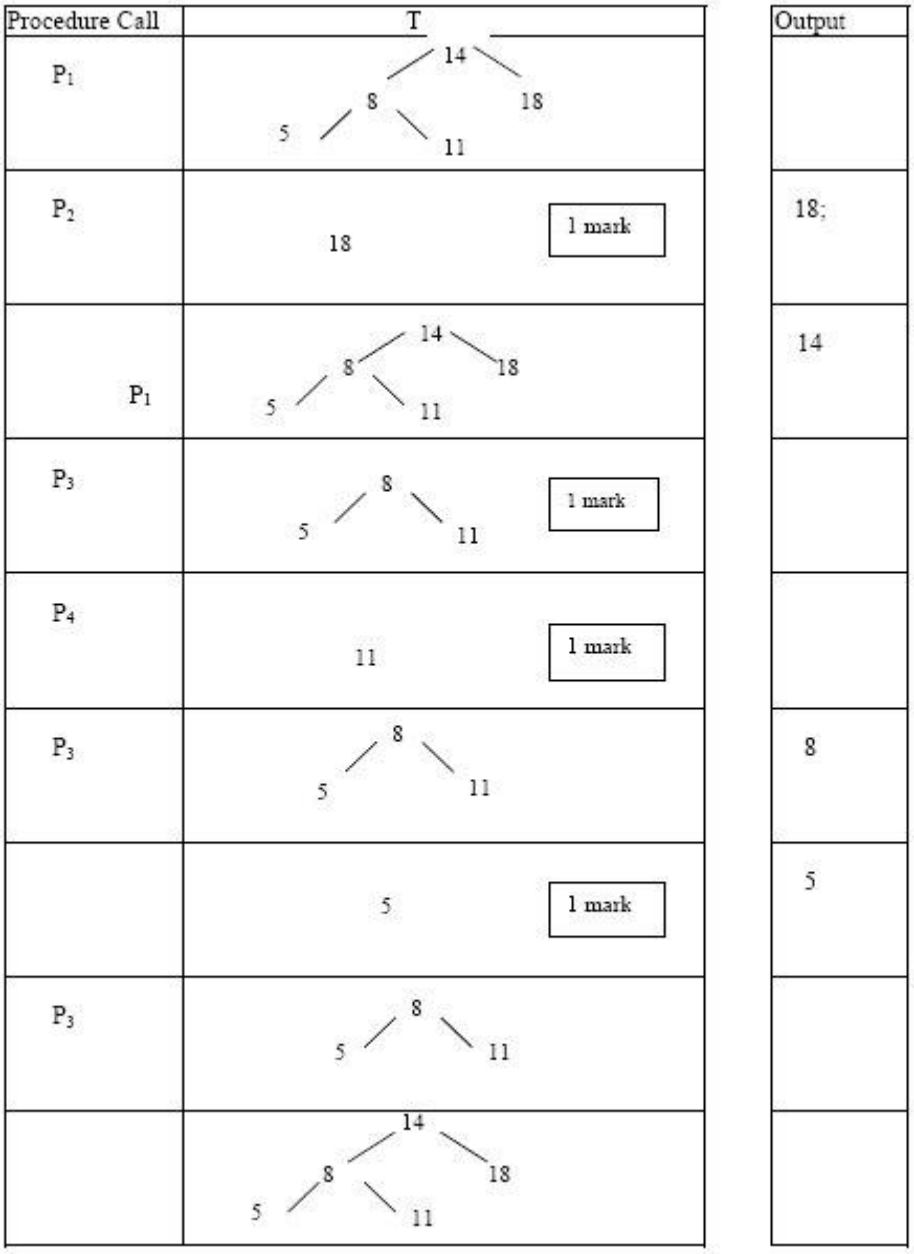
**[4]**

## Q9.

(a) A procedure/routine that calls itself/ is defined in terms of itself;
**A** Function instead of procedure
**R** re-entrant **R** program **R** iteration

**1**

(b) (i)




**6**

(ii) Reverse Inorder// Reverse order; (tree) traversal;

**2**

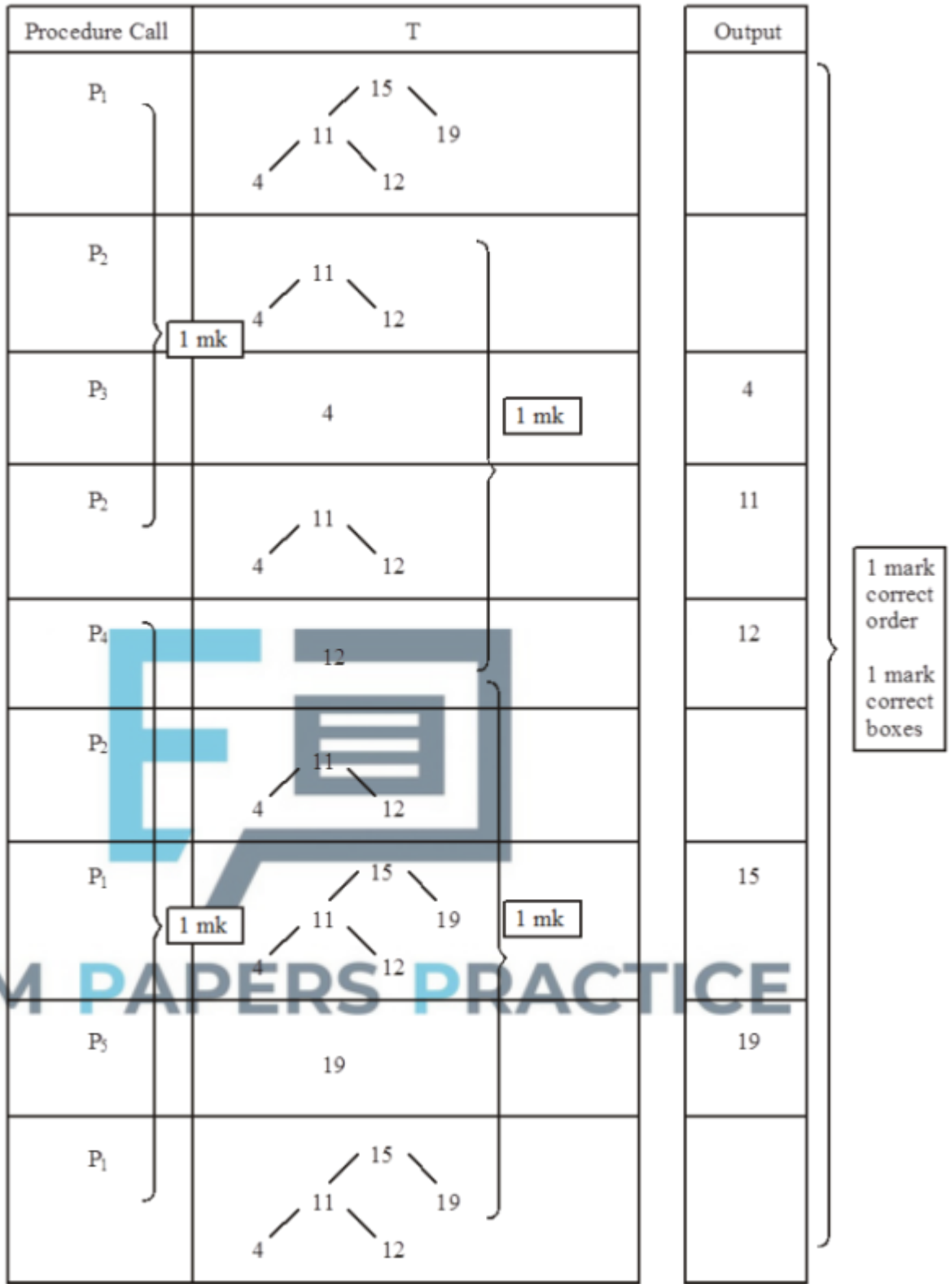
**[9]**

## Q10.

(a) A procedure/routine that calls itself/ is defined in terms of itself;
**A** Function instead of procedure
**R** re-entrant
**R** program
**R** iteration

**1**

(b) (i)




**6**

(ii) In order; (tree) traversal

**2**

**[9]**

## Q11.

(a) It calls itself / is defined in terms of itself / contains within its body a reference

to itself;

*Ensure 'it' refers to procedure, if meaning program or object no mark*

1

(b) The current state of the machine is saved/preserved;
So can return correctly (to previous invocation/call of **Process)**;
**OR**
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved;
So can return correctly to "*correctly' can be implied* (previous invocation of Process);

2

(c) Printed Output:
1; 3; 5, Bird; Bremner; 4, Fortune, Jones; 2, Smith;

*Mark from left and stop marking when error encountered*
*Ignore punctuation.*

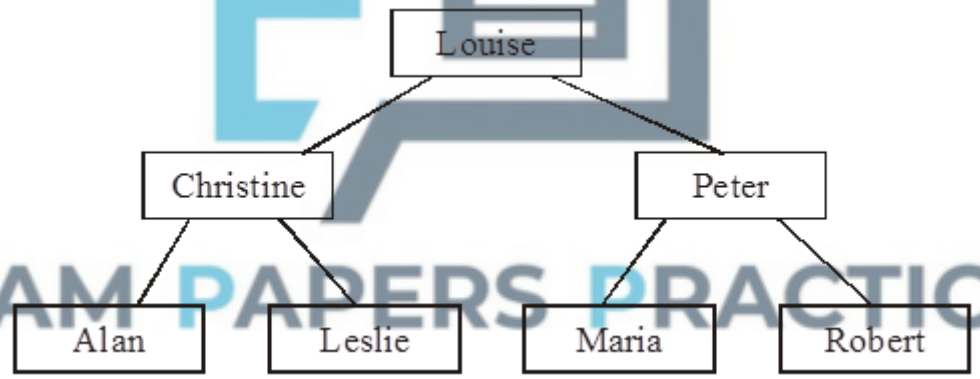
6

(d) (in-order) traversal of a tree; **A** printing of tree (elements in order)
**I** wrong order

1

**[10]**

## Q12.






(a) Correct position of Louise;
Correct position of Christine and Peter;
Correct position of Alan and Leslie;
Correct position of Maria and Robert;

*(If consistent mirror image give full marks)*

4

(b) Root node marked correctly;

1

(c) Louise, Peter, Maria in correct order (allow follow through from (a));

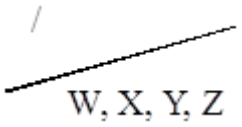
1

**[6]**

## Q13.

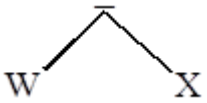
(a) Root, (1)

Branch (1)
Leaf node(1)
Must circle!

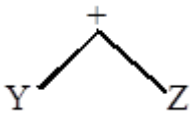



3

(b) Left sub-tree (1)




Right sub-tree(1)




2

(c) W-X / Y+Z
1 1 1

**A** column vector
Spurious punctuation (1)

3

**[8]**

## Q14.

(a) +3x;

1

(b) 3+x;

1

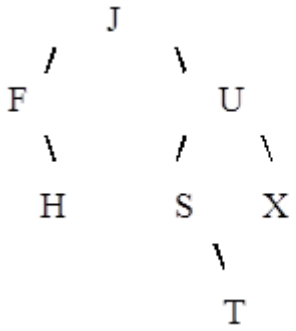
(c) 3x+;

1

**[3]**

## Q15.

(a)




*J 1 mark*

*F and U 1 mark*

*H 1 mark*

*S, X, T 1 mark*

**A** mirror image (this time)

4

(b) ‘T’ 4; ‘U’;

‘T’ 5; ‘S’;

‘T’ 7; ‘T’;

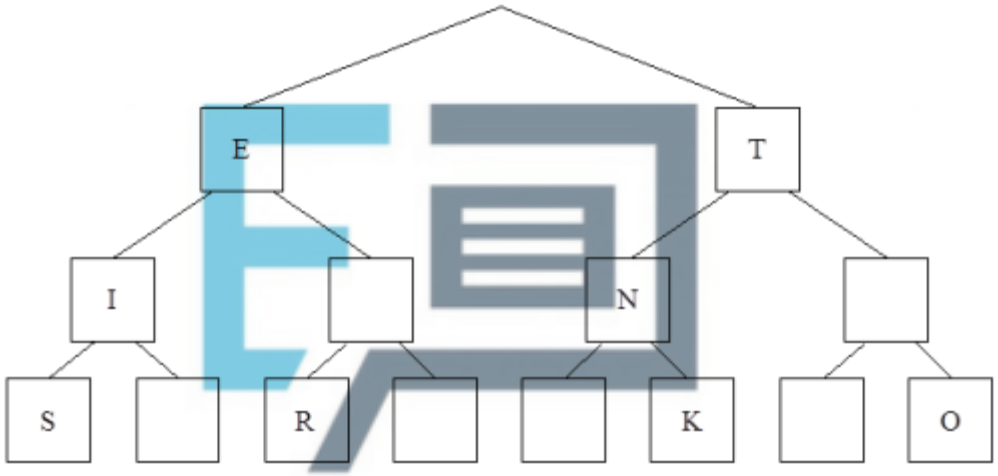
*No penalty if candidate gets ‘item’ wrong*
*Ignore ‘item’ column*

6

**[10]**

## Q16.

(a) *1 mark for each letter correctly placed:*






*1+1*

2

(b) I;T;

*1+1*

2

**[4]**

## Q17.

(a) (i) A

(ii) A B C G

(iii) D E F H I

*1 mark for part (i)*
*1 mark for any 2 correct and another of all correct in (ii) and (iii)*

5

(b) Left and right pointers

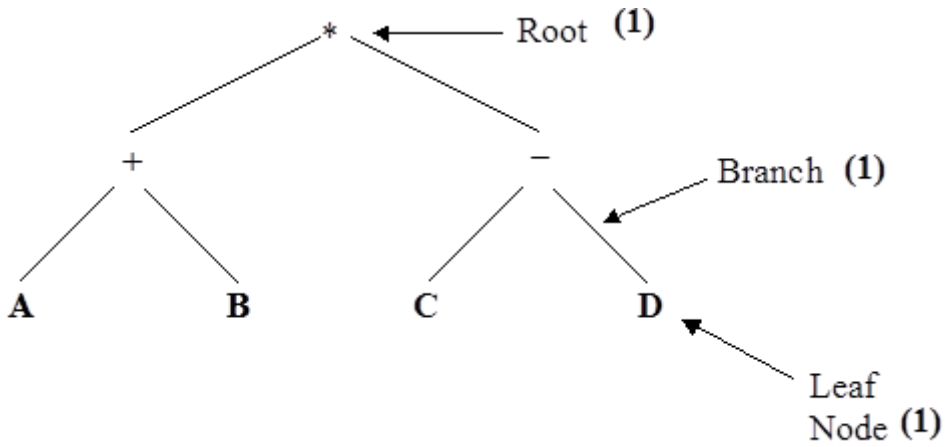
*1 mark for each pointer, 1 mark if just pointers allow addresses / locations*
***NOT** co–ordinates*

**2**

**[7]**

## Q18.

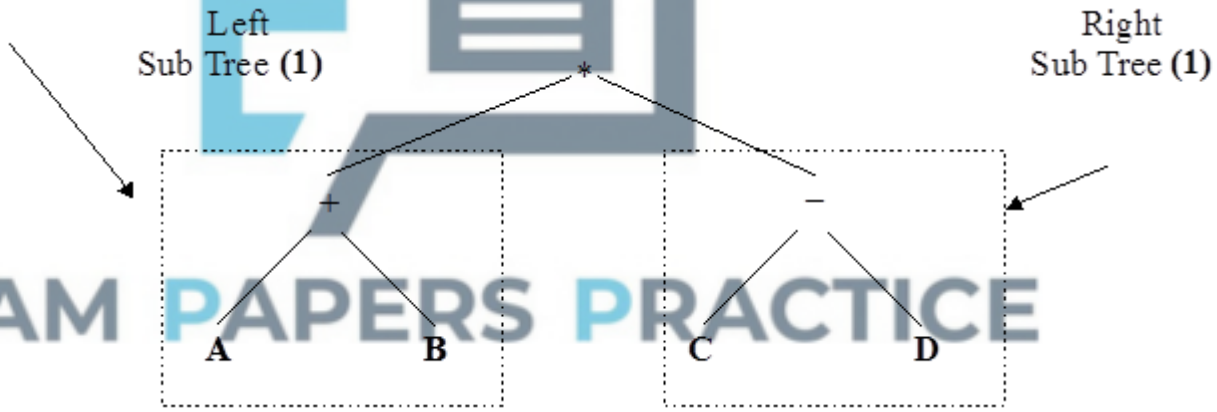
(a)




Labelling must clearly indicate term

**3**

(b)






*Must clearly indicate subsets*

**2**

(c) A procedure which is defined in terms of/ calls itself /re-entrant

**1**

(d) State of machine/return address/parameter (1) needs to be stored/held (1) to enable a previous execution of T to be resumed (1)

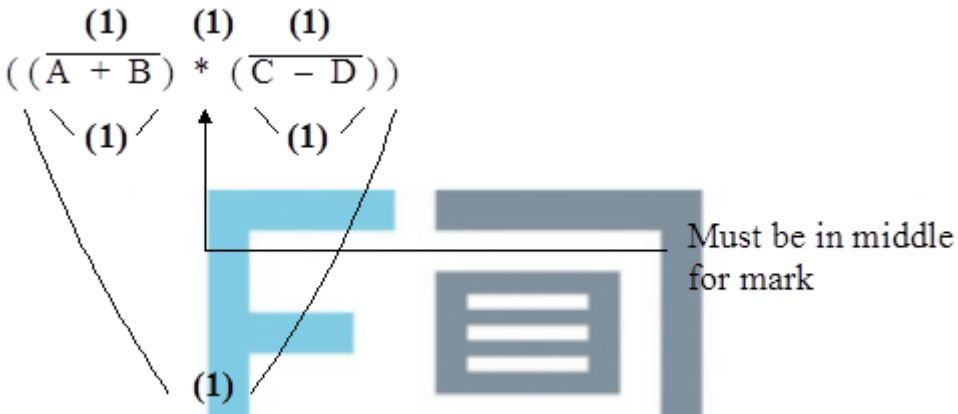
**Or**
So that each call to T(1) can pass(1) a new value of the parameter(1)

**3**

(e)

| Call Number | Parameter |
|---|---|

| | | |
|---|---|---|
| 1 | **tree**('*', **tree**('+', **tree**('A', empty,empty), **tree**('B',empty,empty)),<br>**tree**('-', **tree**('C', empty,empty), **tree**('D',empty,empty)),<br>) | |
| 2 | **tree**('+', **tree**('A',empty,empty), **tree**('B',empty,empty)) | |
| 3 | **tree**('A',empty,empty), | |
| 4 | **tree**('B',empty,empty), | (1) |
| 5 | **tree**('-', **tree**('C',empty,empty),**tree**('D',empty,empty)) | (1) |
| 6 | **tree**('C',empty,empty), | (1) |
| 7 | **tree**('D',empty,empty), | (1) |




**10**

(f) In-order traversal

**1**

**[20]**

## Q1.

Most students could explain what was meant by a recursive subroutine though some answers showed that the difference between iteration and recursion was not always understood. The trace was reasonably well done with the most common error being to include additional function calls or outputs in the table.

## Q2.

This question was about data structures, with much of the emphasis placed on binary trees.

Parts (a) and (b), which related to searching a binary search tree, were both well answered with two thirds of candidates correctly identifying the items that would be examined during the search and over half correctly identifying the time complexity of the search operation for part (b).

For part (c) candidates had to represent a binary tree using an array of records. This was well tackled with candidates correctly using pointers to indicate the relationships between the data items. It was not enough to represent leaf node branches with a blank space or a dash for the left and right pointers. An appropriate value such as an unused index number eg 0 or a NULL value was required.

Most candidates achieved a reasonable number of marks for question part (d). The key difference between a static and dynamic data structure, that the former had a fixed size that was defined at compile time and that the latter had a variable size which could change at run time was well understood. However, not many candidates went on to explain any other differences, such as the fact that memory space might be wasted if a static structure was relatively empty or that a static structure would generally be allocated consecutive memory locations. The purpose of the heap was well understood, as being a pool of available unused memory that could be allocated to a dynamic structure at runtime. The most common misunderstanding was that the heap was where a dynamic structure stored its data. Some candidates made did not get the mark for explaining the purpose of the heap as whilst they made clear that the heap was used for dynamically allocating memory, their responses did not make clear that the heap was the unused memory rather than the memory that new data was stored in.

Part (e) (i) was well answered with the majority of candidates correctly identifying the order that the items would be output. Virtually everyone who correctly identified the order explained the significance of this for part (e) (ii).

For part (f) candidates had to explain why graph traversal was a more complex problem than tree traversal. Many responses recognised that features such as cycles and weighting were the key factors that might contribute to this. Some candidates went beyond what was required and provided excellent explanations of, for example, how a cycle in a graph might cause a problem for a traversal algorithm.

## Q3.

(a) The overwhelming majority of students were able to correctly identify that it was the binary search algorithm that required the list to be sorted for this part.

(b) The trace for this part was also well completed with about three quarters of students getting some marks and well over half getting full marks.

(c) For this part, around half of the candidates were correctly able to explain that the value of InnerPointer did not decrease to zero because either the second while loop condition was not

(d) For this part, about two thirds of students correctly identified that the algorithm that they had traced was of time complexity $O(n^2)$.

(e) This part was poorly answered, with only about one third of students correctly identifying that the algorithm they had traced was an insertion sort. Bubble sort was a far more common but incorrect response.

(f) Parts (i), (ii) were all well answered. The most common error in both parts of was to perform a traversal of the tree instead of using it as a binary search tree.

(g) This part was all well answered. The most common error in both parts of was to perform a traversal of the tree instead of using it as a binary search tree.

## Q4.

For part (a) almost all candidates were able to identify the content of the root node correctly. Leaf nodes were more problematic, with many candidates believing that all of the nodes that were not the root were leaves, and thus mistakenly included "+" in the list of contents of leaf nodes. Nevertheless just over half of the candidates achieved full marks for this question part.

Part (b) was poorly tackled, with only a third of candidates getting more than one mark. The most common error was to state that arrays B and C were used to store the left / right hand subtrees. Candidates needed to make clear that the arrays were storing pointers to these subtrees to be awarded the marks.

For (c), responses were disappointing with many candidates showing no understanding of the purpose of the entry 0 in the array. Others stated that it indicated that a node was a leaf node, or had no children, when in fact the 0 in array B only related to the left hand subtree so nothing could be told from it about whether or not a node had children without also consulting array C.



Part (d): candidates usually find recursive traces to be quite difficult. On this occasion, about half of them got at least one mark for the trace, and a quarter achieved all four marks.

For part (e), about a third of candidates made the expected response, but a disappointingly large number wrote answers that were not even types of tree traversal.

For part (f), almost all of the candidates who had correctly completed the trace table and obtained the output recognised that this was the Reverse Polish Notation equivalent of the infix expression in the expression tree.

## Q5.

Part (a): Two thirds of students were able to identify one property that a graph must have to be a tree. A small number confused a tree with a rooted tree and made assertions such as that a tree must have a root, which is incorrect.

Part (b): This question part tested students' understanding of the method being used to represent a maze as a graph. The majority of students correctly identified a feature of the maze that would stop its graph being a tree. The most commonly seen correct response identified that there could be a loop in the maze. Other possibilities included that part of

the maze could be inaccessible or that part of the maze might only be traversable in one direction. Some students failed to achieve the mark because they re-answered part (a), discussing a feature of a graph that would stop it being a tree, rather than a feature of a maze.

Part (c): Students were asked to represent the graph of the maze as an adjacency matrix. Three quarters of students scored both marks for this question part. Responses where symbols other than 0s and 1s were used in the matrix were accepted, as long as they could be viewed as an accurate representation of the graph.

Part (d)(i): The vast majority of students were able to identify that a recursive routine would call itself. A small number asserted that a recursive routine would repeat itself, which was not considered to be enough for a mark as this could equally have been a description of iteration.

Part (d)(ii): Most students scored some marks for this question part, but less than a fifth achieved both. The most widely understood point was that the data would need to be removed from the stack in the reverse of the order that it was put onto it so that the recursion could be unwound. Less well understood was the types of data that would be stored, such as return addresses and local variables.

Part (e): Most students achieved some marks on this question part and around a quarter achieved all five for a fully complete trace. The most commonly made mistake was to update, incorrectly, the Completely Explored array as the recursive calls were made, as opposed to when the recursion unwound.

## Q6.

Part (a): The use of the adjacency matrix was clearly well understood with all but a few candidates achieving full marks.

Part (b): There were some good responses to this question part, but also quite a lot of confused answers. An adjacency matrix is more appropriate when there are many edges in a graph, or if these edges need to be checked or updated frequently. An adjacency list is appropriate for graphs with few edges (sparse graphs) or where the edges are not checked / updated frequently.
Neither the number of vertices in a graph nor the available memory would influence the choice.
There was confusion over the use of terminology with some candidates apparently using the term vertex to mean edge.



Part (c): This question part was poorly answered, with only a third of candidates scoring any marks. A tree is a graph that is connected, undirected and has no cycles. Some candidates gave responses that referred only to specific types of tree, either rooted trees or binary trees.
These responses did not gain credit.

Part (d): The vast majority of candidates knew how to construct a binary search tree. The most common cause of error appeared to be candidates forgetting the order of the letters in the alphabet rather than forgetting the principles that should be used to construct the tree. A small number of candidates mistakenly drew arrows instead of lines between nodes.

Part (e): There were some good responses to this question part but many were disappointing and a surprising number of candidates did not write a response at all. Many candidates who did answer chose to use a diagram to illustrate their response which was quite acceptable, so long as the diagram included enough detail to make clear that it was

a representation of a binary tree.

## Q7.

It was pleasing to see the number of candidates that scored highly on this question. Most candidates were able to obtain the mark for part (a) and a large number did very well on part (b). It must be emphasised that candidates were asked to dry run the algorithm and complete the trace table. A small number of candidates were able to produce the correct output but did not produce a satisfactory trace. Marks were given for the trace and so it is essential that candidates fill this in correctly. Although most candidates obtained one mark for part (b)(ii), few obtained two. Candidates must realise that correct technical terminology should be used.

## Q8.

Despite what was considered to be a straightforward question, the range of the quality of answers seen was huge. Many failed even to appreciate what was meant by a binary tree (as was apparent by the suggested diagram!). Some answers correctly drew the structure for the tree but then failed to attempt (b)(i) or (b)(ii) – which is strange as both the 'list of comparisons' and the 'number of comparisons' had been asked for on previous papers. The most common error was not to include the final name in the list, and candidates who did not score for (b)(ii), then often incorrectly quoted 3 as their answer for (b)(ii).

## Q9.

Most candidates obtained the mark for part (a). It was also very pleasing to see the number of candidates who were able to correctly trace the algorithm. Many candidates obtained good marks on this question. Although many candidates did go wrong with the trace, very few candidates failed to attempt it.

## Q10.

This was another question which most candidates found difficult, if not impossible. However, some good candidates produced very good answers.



Most candidates were able to answer part (a).

The examiners only rarely awarded full marks for the trace table. A lot of candidates abandoned the trace once they realised that the numbers were being output in ascending order. This limited their reward to two or three marks at best since half of the marks depended on the trace being completed. Many candidates had difficulty logging the procedure calls even when they made a good attempt at showing the tree in the T column.

Some candidates got the two marks for part (b)(ii) without attempting the trace while others who showed the right output in (i) called the procedure a search or a bubble sort.

## Q11.

Part (a) and (b) have been asked many times before. However, many candidates were unable to explain how a stack is used in the execution of a recursive procedure. Correct responses included that the return address (held in the program counter) and other register values are saved so control can return correctly to the previous invocation of the procedure. Many candidates gained full marks for stating, correctly, the printed output after dry-running the procedure. However, many others could not even get the first few printed items in the correct order.

Many candidates correctly spotted that the procedure described an in-order traversal of a tree.

## Q12.

(a) It was pleasing to see the number of candidates that were able to draw a correct tree.

(b) Those candidates that were able to draw a tree were generally able to identify the root correctly.

(c) Those candidates that drew the tree were generally able to identify the data items that needed to be accessed. There were a number of candidates who failed to include Maria. It is not possible to know that you have found the correct node without accessing it.

## Q13.

This was well answered by many candidates, although here again, marks were lost through lack of care in carrying out the clearly written instructions. The instructions in part (a) were to circle and label the three parts of the tree. An un-circled, or (more rarely) unlabelled part was not credited. Very few candidates were unable to draw the left and right sub-trees correctly, and most could perform in-order traversal.

## Q14.

This question was on binary tree traversal. A large number of candidates gained full marks for this question.

## Q15.

(a) The tree was generally well created. A few candidates produced mirror images, which were accepted this time. However, candidates need to be aware that a binary search tree stores lower values in the left sub-tree and higher values in the right sub-tree. A few candidates produced a balanced binary tree with each node having two sub nodes, which was not correct.



(b) Only the better candidates seemed to score full marks here. Dry-running an algorithm does not seem to have received sufficient attention in the preparation of a number of candidates. The fact that 'Item' did not change its value seemed to be noticed by only a few candidates. Candidates did not appear to appreciate that the algorithm was using the binary search tree, which the candidates had drawn in part (a).

## Q16.

This was usually answered correctly.

## Q17.

Part (a) of this question was well done by nearly all candidates. The root node was almost always correctly identified but relatively few realised that node A is also a parent node. The terminal nodes were usually identified correctly, but some candidates missed out F while others included G.

Part (b) was often badly answered. The node must contain left and right pointers.

## Q18.

This question was answered successfully by the better candidates with many scoring full marks. Surprisingly, several candidates identified an internal node as a leaf node and others could not indicate a branch, clearly. In the latter case, candidates indicated their uncertainty by writing “branch” level with an internal node and omitting to link it to the tree diagram by arrow. The same identification problem arose with labelling the left and right sub-trees. Marks cannot be given when there is doubt in the mind of the examiner as to whether the candidate really knows the correct answer. The left and right sub-trees should be ringed and clearly labelled to eliminate doubt in the mind of the examiner.

Recursion has been examined frequently in recent years. It is therefore pleasing to note an increase in the number of candidates who can define the term accurately and who are able to explain why a stack is needed. Sadly, many candidates still have difficulty dry-running the execution of a recursively-defined procedure successfully. Many candidates completed the table correctly but a significant number made no attempt to show the printed output and many others produced output that was wrong. Many candidates looked at the first parameter of each call and incorrectly used it to generate the printed output. These candidates described the procedure as a *pre-order* tree traversal algorithm, whereas, in fact, it was an *in-order* traversal. Candidates who dry-ran the procedure successfully had little difficulty in stating in-order. Some candidates recognised in the layout of the procedure the structure of an *in-order* traversal. These candidates gained credit for their knowledge even though some did not complete the dry run successfully or at all.