**Q1.**

The algorithm below shows a bubble sort algorithm represented using pseudo-code. The algorithm sorts the data in a list L.

PROCEDURE BubbleSort(L)

  N ← LEN(L) – 2

  Count1 ← 0

  WHILE Count 1 < LEN(L) – 1

     FOR Count2 ← 0 TO N

       IF L[Count2] > L[Count2 + 1] THEN

         Temp ← L[Count2]

         L[Count2] ← L[Count2 + 1]

         L[Count2 + 1] ← Temp

       ENDIF

     ENDFOR

     Count1 ← Count1 + 1

  ENDWHILE

ENDPROCEDURE

‘Sorting a list becomes an intractable problem when the size of the list is very large; it is a tractable problem when the size of the list is small.’

Explain why this statement is wrong.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q2.**

The algorithm below shows a bubble sort algorithm represented using pseudo-code. The algorithm sorts the data in a list L.

PROCEDURE BubbleSort(L)

  N ← LEN(L) – 2

  Count1 ← 0

  WHILE Count 1 < LEN(L) – 1

     FOR Count2 ← 0 TO N

       IF L[Count2] > L[Count2 + 1] THEN

         Temp ← L[Count2]

         L[Count2] ← L[Count2 + 1]

         L[Count2 + 1] ← Temp

       ENDIF

     ENDFOR

     Count1 ← Count1 + 1

  ENDWHILE

ENDPROCEDURE

Explain what approach(es) a programmer might take if asked to ‘solve’ an intractable problem.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

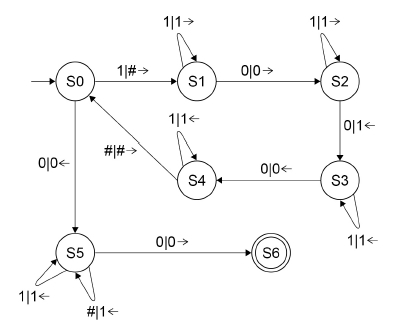
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q3.**

The diagram below shows the transition function (represented as a state transition diagram) and part of the tape for a Turing machine designed to complete a task. The current position of the read / write head is indicated by the \* symbol.

**Figure 1**

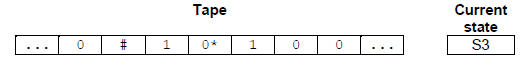
****



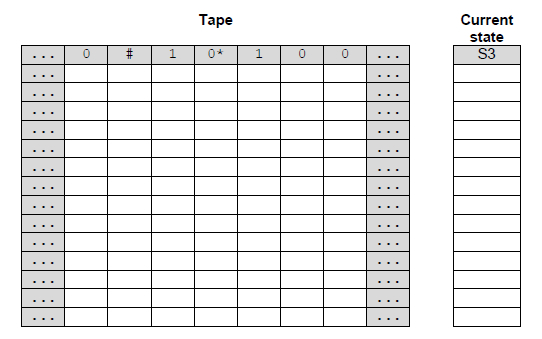
The label 1|#→ on the transition from S0 to S1 means if the machine is in state S0 and a 1 is read from the tape then a # should be written to the tape and then the read / write head moved one cell to the right.

(a)     After four steps of the computation have been completed, the current state, the tape contents and the position of the read / write head are:

**Figure 2**

****

Complete the unshaded cells of the table below to show the result of tracing the computation of this Turing machine, from step five onwards. Show the contents of the tape, the current position of the read / write head and the current state as the input symbols are processed. Step four has been repeated at the start of the trace.



**(5)**

(b)     What is the purpose of the Turing machine shown in **Figure 2**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     What is the purpose of the transition from S5 to S6 in **Figure 2**?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 7 marks)**

**Q4.**

Explain what a Universal Turing machine is.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

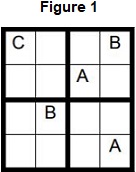
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

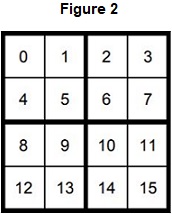
**Q5.**

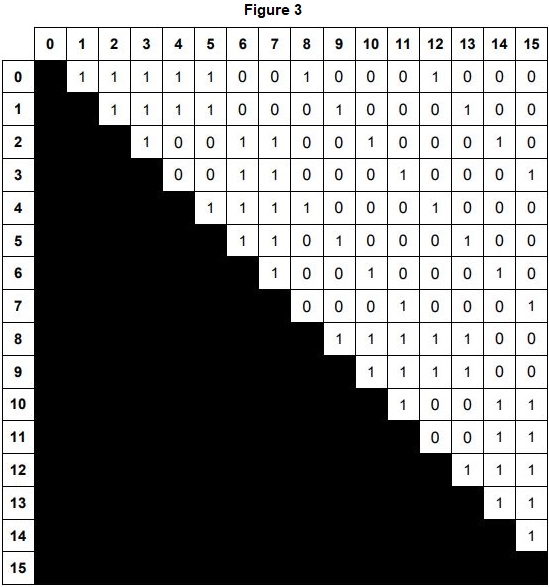
**Figure 1** shows a simpler example of a type of logic puzzle with fewer cells. In this simpler puzzle only the letters A-D are used.



It is possible to represent this type of puzzle as a graph. To do this a unique number is given to each cell and a node containing this unique number is added to the graph. An edge between two nodes denotes a link between those two cells, meaning they cannot contain the same letter as each other.

**Figure 2** shows how unique numbers have been allocated to each cell in the puzzle in **Figure 1** and **Figure 3** shows an adjacency matrix that represents this puzzle.





The graph in **Figure 3** can be considered to be both a representational abstraction and an abstraction by generalisation of the puzzle from **Figure 1**.

What is abstraction by generalisation?

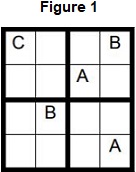
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

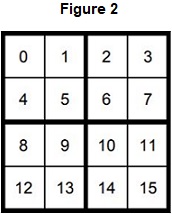
**Q6.**

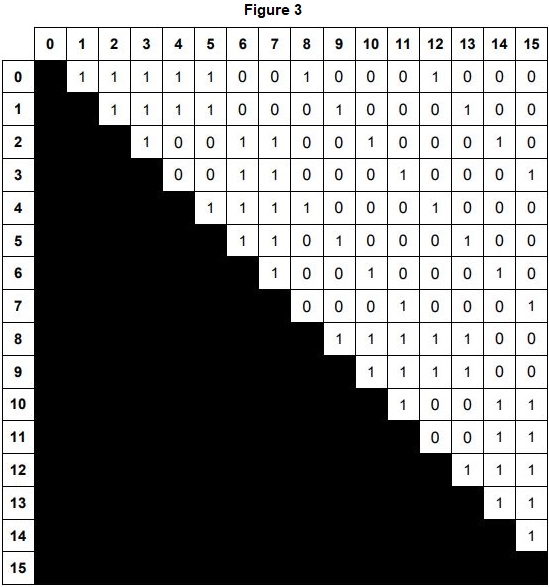
**Figure 1** shows a simpler example of a type of logic puzzle with fewer cells. In this simpler puzzle only the letters A-D are used.



It is possible to represent this type of puzzle as a graph. To do this a unique number is given to each cell and a node containing this unique number is added to the graph. An edge between two nodes denotes a link between those two cells, meaning they cannot contain the same letter as each other.

**Figure 2** shows how unique numbers have been allocated to each cell in the puzzle in **Figure 1** and **Figure 3** shows an adjacency matrix that represents this puzzle.





The graph in **Figure 3** can be considered to be both a representational abstraction and an abstraction by generalisation of the puzzle from **Figure 1**.

Other than the contents of the cells, what information has been removed from the puzzle in **Figure 1** when it has been represented as a graph?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q7.**

Explain the functionality of the \* metacharacter when it is used in a regular expression.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q8.**

Explain the functionality of the ? metacharacter when it is used in a regular expression.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q9.**

Complete the below to show which of the strings belong to the language defined by the regular expression 1|01+.

|  |  |
| --- | --- |
| **String** | **Belongs to language (Y/N)?** |
| 1 |  |
| 11 |  |
| 01 |  |
| 0111 |  |
| 0101 |  |
| 111 |  |
| 0011 |  |

**(Total 3 marks)**

**Q10.**

There are three boxes containing vegetables. One contains onions, one contains carrots and one contains onions and carrots. The three boxes have been labelled. One is labelled “onions”, one is labelled “carrots” and the other is labelled “onions and carrots”. You know that all three have been labelled incorrectly.

Describe how you can work out what each box actually contains by taking just **one** vegetable out of **one** box, without looking inside any of the boxes.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

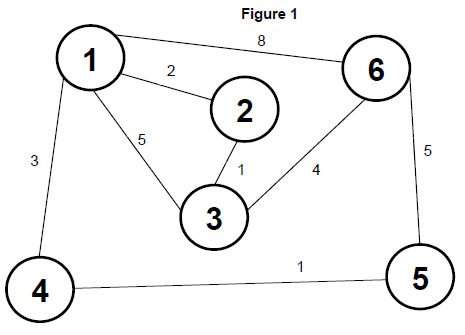
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q11.**

**Figure 1** is a graph that shows the time it takes to travel between six locations in a warehouse. The six locations have been labelled with the numbers 1 - 6. When there is no edge between two nodes in the graph this means that it is not possible to travel directly between those two locations. When there is an edge between two nodes in the graph the edge is labelled with the time (in minutes) it takes to travel between the two locations represented by the nodes.



(a)  The graph is represented using an adjacency matrix, with the value 0 being used to indicate that there is no edge between two nodes in the graph.

A value should be written in every cell.

Complete the unshaded cells in **Table 1** so that it shows the adjacency matrix for **Figure 1**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table 1** | | | | | | |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |

**(2)**

(b)  Instead of using an adjacency matrix, an adjacency list could be used to represent the graph. Explain the circumstances in which it would be more appropriate to use an adjacency list instead of an adjacency matrix.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

(c)  State **one** reason why the graph shown in **Figure 1** is **not** a tree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(d)  The graph in **Figure 1** is a weighted graph. Explain what is meant by a **weighted graph**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**Figure 2** contains pseudo-code for a version of Djikstra’s algorithm used with the graph in **Figure 1**.

Q is a priority queue which stores nodes from the graph, maintained in an order based on the values in array D. The reordering of Q is performed automatically when a value in D is changed.

AM is the name given to the adjacency matrix for the graph represented in **Figure 1**.

**Figure 2**

Q ← empty queue

FOR C1 ← 1 TO 6

  D[C1] ← 20

  P[C1] ← −1

  ADD C1 TO Q

ENDFOR

D[1] ← 0

WHILE Q NOT EMPTY

  U ← get next node from Q

  remove U from Q

  FOR EACH V IN Q WHERE AM[U, V] > 0

    A ← D[U] + AM[U, V]

    IF A < D[V] THEN

      D[V] ← A

      P[V] ← U

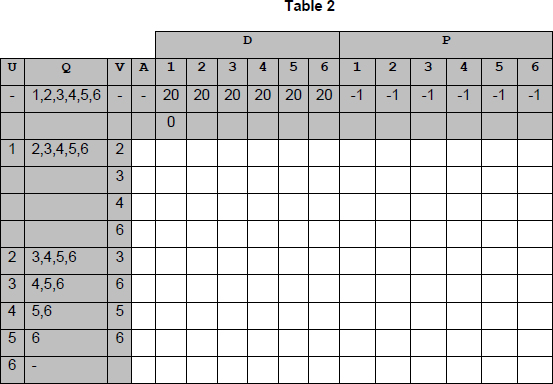
    ENDIF

  ENDFOR

ENDWHILE

OUTPUT D[6]

(e)  Complete the unshaded cells of **Table 2** to show the result of tracing the algorithm shown in **Figure 2**. Some of the trace, including the maintenance of Q, has already been completed for you.



**(7)**

(f)   What does the output from the algorithm in **Figure 2** represent?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(g)  The contents of the array P were changed by the algorithm. What is the purpose of the array P?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 16 marks)**

**Q12.**

Describe the Halting problem.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q13.**

Why is it not possible to create a Turing machine that solves the Halting problem?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q14.**

To define a Turing machine the finite alphabet of symbols that it can use needs to be specified and there needs to be a tape.

State **two** other components of a Turing machine.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q15.**

Explain what a Universal Turing Machine is.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q16.**

Why can a Universal Turing Machine be considered to be more powerful than any computer that you can purchase?

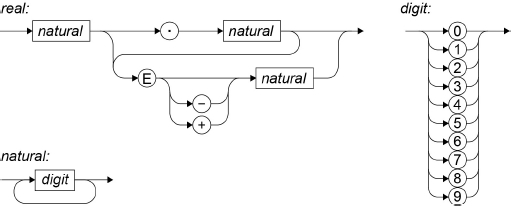
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q17.**

In a particular programming language, the correct syntax for a real number, natural number and digit is defined by the syntax diagrams in the diagram below.



(a)     Write **Yes** or **No** in the unshaded cells in the table to identify whether or not the numbers listed in the table are valid real numbers which conform to the correct syntax for this language.

|  |  |
| --- | --- |
| **Real number** | **Valid? (Yes/No)** |
| 87.000 |  |
| 97+12 |  |
| 12.31E+12 |  |

**(3)**

(b)     In Backus-Naur Form (BNF) the following production rule has been written to define a digit:

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Write a BNF production rule to define a natural number that is equivalent to the definition in the syntax diagram in the diagram.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 5 marks)**

**Q18.**

The table below lists some well-known algorithms.

|  |
| --- |
| **Algorithm** |
| Linear search |
| Merge sort |
| Binary search |
| Post-order tree-traversal |

(a)     Which of the algorithms listed in the table has *0*(*n* log *n*) time complexity?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(b)     How many of the algorithms listed in the table are algorithms used to solve tractable problems?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 2 marks)**

**Q19.**

(a)     State the time complexity for the bubble sort algorithm in terms of 𝑛, where 𝑛 is the number of items in the list to be sorted.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(b)     Explain why the bubble sort algorithm has the time complexity stated in your answer to part **(a)**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

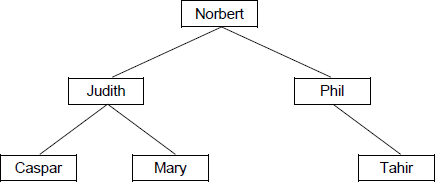
**(Total 3 marks)**

**Q20.**

**Figure 1** shows the data Norbert, Phil, Judith, Mary, Caspar and Tahir entered into a binary search tree.

**Figure 2** contains pseudo-code for a recursive binary tree search algorithm.

**Figure 1**

****

**Figure 2**

FUNCTION TreeSearch(target, node)

  OUTPUT ‘Visited ’, node

  IF target = node THEN

    RETURN True

  ELSE IF target > node AND Exists(node, right) THEN

    RETURN TreeSearch(target, node.right)

  ELSE IF target < node AND Exists(node, left) THEN

    RETURN TreeSearch(target, node.left)

  ENDIF

  RETURN False

ENDFUNCTION

The subroutine Exists takes two parameters – a node in the binary tree and a direction (left or right). It returns a Boolean value indicating if the node given as a parameter has a child node in the direction specified by the second parameter. For instance, Exists(Mary, left) will return a value of False as there is no node to the left of Mary in the binary tree.

node.right evaluates to the child node to the right of node, eg Judith.right is Mary.

node.left evaluates to the child node to the left of node, eg Judith.left is Caspar.

(a)     What is meant by a recursive subroutine?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(b)     There are two base cases for the subroutine TreeSearch. State **one** of the base cases.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     Complete the unshaded cells of the table below to show the result of tracing the TreeSearch algorithm shown in **Figure 2** with the function call TreeSearch(Olivia, Norbert). You may not need to use all of the rows.

|  |  |
| --- | --- |
| **Function call** | **Output** |
| TreeSearch(Olivia, Norbert) |  |
|  |  |
|  |  |
|  |  |
|  |  |

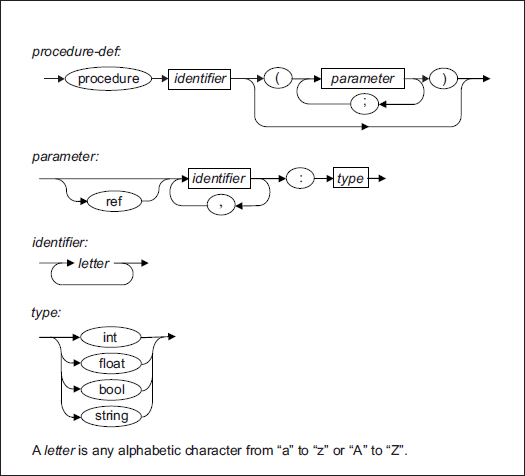
**(3)**

**(Total 5 marks)**

**Q21.**

In a particular programming language, the correct syntax for four different constructs is defined by the syntax diagrams in **Figure 1**.

**Figure 1**

****

In this language an example of a valid *identifier* is loopCount and an example of a valid *type* is int.

(a)     For each row in the table below, write **Yes** or **No** in the **Valid?** column to identify whether or not the **Example** is a valid example of the listed **Construct**.

|  |  |  |
| --- | --- | --- |
| **Construct** | **Example** | **Valid? (Yes/No)** |
| *identifier* | Game\_Over |  |
| *parameter* | ref x,y:bool |  |
| *procedure-def* | procedure square(s:float) |  |
| *procedure-def* | procedure rect(w:int,h:int) |  |

**(4)**

A student has written Backus-Naur Form (BNF) production rules that are supposed to define the same constructs as the syntax diagrams in **Figure 1**. Their BNF rules are shown in **Figure 2**.

**Figure 2**

<procedure-def> ::= procedure <identifier> ( <paramlist> )

<paramlist>     ::= <parameter> | <parameter> ; <paramlist>

<parameter>     ::= <identlist> : <type> |

                    ref <identlist> : <type>

<identlist>     ::= <identifier> | <identifier> , <identlist>

<identifier>    ::= <letter> | <letter> <identifier>

<type>          ::= int | float | bool

A <letter> is any alphabetic character from “a” to “z” or “A” to “Z”.

(b)     The BNF production rules in **Figure 2** contain two errors. These errors mean that the production rules do not represent the same statement types as the syntax diagrams in **Figure 1**.

Describe the **two** errors.

Error 1: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Error 2: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

(c)     The production rule for a <paramlist> is recursive.

Explain why recursion has been used in this production rule.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 7 marks)**

**Q22.**

An algorithm is a sequence of unambiguous instructions for solving a problem.

Three different algorithms, A, B and C, have the following orders of time complexity:

Algorithm A: O(2n)

Algorithm B: O(n)

Algorithm C: O(n3)

List the algorithms A, B and C in order with the most efficient at the top of the list.

Most efficient: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Least efficient: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 1 mark)**

**Q23.**

Some problems are tractable.

What does it mean for a problem to be described as tractable?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(Total 2 marks)**

**Q24.**

One of the problems listed in the table below is tractable.

Place **one** tick next to the tractable problem.

|  |  |
| --- | --- |
| **Problem** | **Tractable? (Tick one)** |
| The travelling salesman problem. |  |
| The problem of searching a list. |  |
| The Halting problem. |  |

**(Total 1 mark)**

**Q25.**

A Turing machine has been designed to recognise palindromic binary numbers, ie numbers such as 101 and 0110 that read the same from left to right as from right to left.

The machine has states SB, S0, S1, SC0, SC1, SL, SY and SN.

SB is the start state and SY and SN are the stop states.

The machine stores data on a single tape which is infinitely long in one direction. The machine’s alphabet is 0, 1 and □, where □ is the symbol used to indicate a blank cell on the tape. The machine will enter state SY if the value represented on the tape is a palindromic binary number, otherwise it will enter state SN.

The transition rules for this Turing machine can be expressed as a transition function δ.

Rules are written in the form:

δ(Current State, Input Symbol) = (Next State, Output Symbol, Movement)

So, for example, the rule:

δ(SB, 0) = (S0 , □, →)

means:

|  |  |
| --- | --- |
| IF | the machine is currently in state SB AND the input symbol read from the tape is 0 |
| THEN | the machine should change to state S0, write a blank symbol (□) to the tape and move the read/write head one cell to the right |

The machine’s transition function, δ, is defined by:

δ (SB, 0) = (S0, □, →)    δ (SC0, 0) = (SL, □, ←)

δ (SB, 1) = (S1, □, →)    δ (SC0, 1) = (SN, 1, ←)

δ (SB, □) = (SY, □ →)      δ (SC0, □) = (SY, □, →)

δ (S0, 0) = (S0, 0, →)      δ (SC1, 0) = (SN, 0, ←)

δ (S0, 1) = (S0, 1, →)      δ (SC1, 1) = (SL, □, ←)

δ (S0, □) = (SC0, □, ←)    δ (SC1, □) = (SY, □, →)

δ (S1, 0) = (S1, 0, →)      δ (SL, 0) = (SL, 0, ←)

δ (S1, 1) = (S1, 1, →)      δ (SL, 1) = (SL, 1, ←)

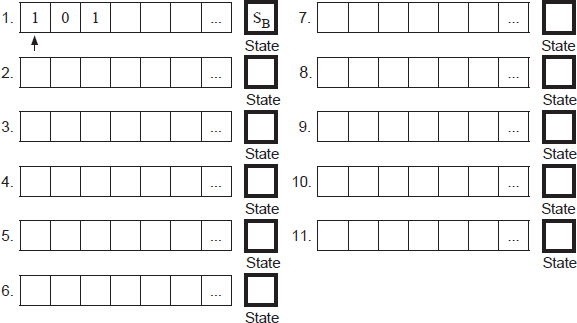
δ (S1, □) = (SC1, □, ←)    δ (SL, □) = (SB, □, →)

(a)     This Turing machine is carrying out a computation. The machine starts in state SB with the string 101 on the tape. All other cells contain the blank symbol, □ (not shown). The read/write head is located at the left hand symbol of the string and is indicated with an upward arrow.

Trace the computation of the Turing machine, using the transition function δ.

Show the contents of the tape, the current position of the read/write head and the current state as the input symbols are processed.

The initial configuration of the machine has been completed for you in step 1.



**(5)**

(b)     The three rules shown below are part of the machine’s transition function.

δ (S0, 0) = (S0, 0, →)

δ (S0, 1) = (S0, 1, →)

δ (S0, □) = (SC0, □, ←)

Explain what effect these three rules, taken together, have on the tape, the read/write head and the state of the Turing machine.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 7 marks)**

**Q26.**

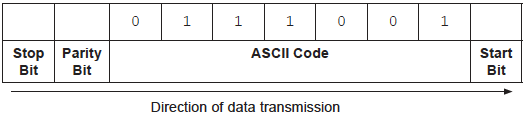
A bar code scanner is connected to a computerised point of sale system (till). When a product is sold, the bar code that is printed on the product is scanned by the scanner and transmitted to the point of sale system.

This transmission uses asynchronous serial communication and odd parity.

**Figure 1** shows the ASCII code for the character "9", which has been read from the bar code, being transmitted to the point of sale system.

(a)     Write the missing values of the stop bit, parity bit and start bit on **Figure 1**.

**Figure 1**

****

**(2)**

(b)     Explain what asynchronous data transmission is and why stop and start bits are required when asynchronous data transmission is used.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

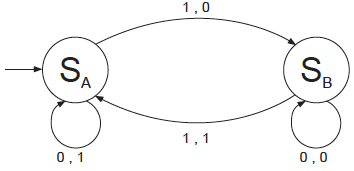
**(3)**

As part of the process of preparing the data for transmission, the 7-bit ASCII code (0111001) is processed by a Mealy machine (a type of Finite State Machine with output).

The ASCII code is processed from left to right, i.e. the leftmost 0 is the first digit to be processed.

**Figure 2** shows a diagram of the Mealy machine. Each transition is labelled with the input symbol that will trigger the transition, followed by a comma, followed by the output that will be produced.

**Figure 2**

****

(c)     What output is generated by the Mealy machine in **Figure 2** for the input 0111001?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(d)     The last digit output by the Mealy machine is used in the transmission.

Explain what this last digit represents.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(e)     Serial communication has been chosen instead of parallel communication even though the scanner and point of sale system are located next to each other.

State **two** reasons why this choice is appropriate.

Reason 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Reason 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 9 marks)**

**Q27.**

(a)     Two important components of a Turing machine are the transition function and the tape.

Five components of a typical modern computer system are listed below:

1. DVD-ROM

2. Compiler

3. Main Memory

4. Processor

5. Program

Complete the table by writing into it the numbers of the modern computer system components from the list above that would fulfil the role most similar to the transition function and the tape in a Turing machine.

|  |  |
| --- | --- |
| **Turing machine component** | **Number (1-5) of modern computer system component with most similar role** |
| Transition function |  |
| Tape |  |

**(2)**

(b)     Explain the relationship between the Turing machine, as a model of computation, and an algorithm.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     A Turing machine has been designed to complete a task.

The machine has states SB, SF, S0, S1, SR, SE, and SH. SB is the start state and SH is the stop state.

The machine stores data on a single tape which is infinitely long in one direction. The machine’s alphabet is 0, 1, #, □, where □ is the symbol used to indicate a blank cell on the tape.

The transition rules for this Turing machine can be expressed as a transition function *δ*. Rules are written in the form:

*δ*(Current State, Input Symbol) = (Next State, Output Symbol, Movement)

So, for the example, the rule:

*δ*(SB , □) = (SF , #, ←)

means:

|  |  |
| --- | --- |
| IF | the machine is currently in state SB AND the input symbol read from the tape is a blank symbol (□) |
| THEN | the machine should change to state SF , write a # to the tape and move the read/write head one cell to the left |

The machine's transition function, *δ*, is defined by:

|  |  |  |
| --- | --- | --- |
| *δ*(SB, 0)   =   (SB, 0, →)  *δ*(SB, 1)   =   (SB, 1, →)  *δ*(SB, □)   =   (SF, #, ←) |  | *δ*(S1, #)   =   (S1, #, →)  *δ*(S1, 0)   =   (S1, 0, →)  *δ*(S1, 1)   =   (S1, 1, →)  *δ*(S1, □)   =   (SR, 1, ←) |
| *δ*(SF, #)   =   (SF, #, ←)  *δ*(SF, 0)   =   (S0, #, →)  *δ*(SF, 1)   =   (S1, #, →)  *δ*(SF, □)   =   (SE, □, →) |  | *δ*(SR, #)   =   (SF, #, ←)  *δ*(SR, 0)   =   (SR, 0, ←)  *δ*(SR, 1)   =   (SR, 1, ←) |
| *δ*(S0, #)   =   (S0, #, →)  *δ*(S0, 0)   =   (S0, 0, →)  *δ*(S0, 1)   =   (S0, 1, →)  *δ*(S0, □)   =   (SR, 0, ←) |  | *δ*(SE, #)   =   (SE, □, →)  *δ*(SE, 0)   =   (SH, 0, →)  *δ*(SE, 1)   =   (SH, 1, →)  *δ*(SE, □)   =   (SH, □, →) |

This Turing machine is carrying out a computation. The machine starts in state SB with the string 10 on the tape. All other cells contain the blank symbol, □ (not shown). The read/write head is positioned at the left hand end of the string on the tape, as indicated by the arrow:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 0 |  |  |  | ... |  | SB |  |
|  |  | ↑ |  |  |  |  |  | State | | |

After eight steps of the computation have been completed, the tape contents, state and position of the read/write head are:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | # | # | 0 |  | ... |  | SF |  |
|  |  | ↑ |  |  |  |  |  | State | | |

Trace the computation of the Turing machine, from step nine onwards, using the transition function *δ*. Show the contents of the tape, the current position of the read/write head and the current state as the input symbols are processed. Step eight has been repeated at the start of the trace.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8. |  | 1 | # | # | 0 |  | ... |  | SF |  |  | 16. |  |  |  |  |  |  | ... |  |  |  |
|  |  | ↑ |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 9. |  |  |  |  |  |  | ... |  |  |  |  | 17. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 10. |  |  |  |  |  |  | ... |  |  |  |  | 18. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 11. |  |  |  |  |  |  | ... |  |  |  |  | 19. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 12. |  |  |  |  |  |  | ... |  |  |  |  | 20. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 13. |  |  |  |  |  |  | ... |  |  |  |  | 21. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 14. |  |  |  |  |  |  | ... |  |  |  |  | 22. |  |  |  |  |  |  | ... |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  | State | | |
| 15. |  |  |  |  |  |  | ... |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | State | | |  |  |  |  |  |  |  |  |  |  | | |

**(6)**

(d)     Explain the purpose of the Turing machine that you have traced the execution of.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 10 marks)**

**Q28.**

Problems can be classified into different categories based upon how efficiently they can be solved, or if they can be solved at all.

Three such classifications are:

•        tractable

•        intractable

•        unsolvable.

(a)     Explain what it means for a problem to be described as tractable.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

(b)     What approach(es) might a programmer take if asked to ’solve’ an intractable problem?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

(c)     Tick **one** row in **Table 1** to indicate which of the problems listed in the table is **unsolvable**.

**Table 1**

|  |  |
| --- | --- |
| **Problem** | **Unsolvable?  ( one row)** |
| The problem of sorting a list into order |  |
| The Halting problem |  |
| The travelling salesman problem |  |

**(1)**

(d)     Sometimes more than one algorithm exists to solve the same problem.

In such cases, a programmer may select the algorithm to use based upon the time and space complexity of the algorithm.

**Table 2** below shows the order of time complexity of three different algorithms to solve a problem.

Tick **one** row in **Table 2** to indicate which of the algorithms is the **least time efficient**.

**Table 2**

|  |  |
| --- | --- |
| **Order of Time Complexity** | **Least Time Efficient?  ( one row)** |
| O(2n) |  |
| O(n) |  |
| O(n2) |  |

**(1)**

**(Total 6 marks)**

**Q29.**

The Backus-Naur Form (BNF) production rules below define the syntax of a number of programming language constructs.

<forloop>    ::=  FOR <variable>  = <integer>  TO  <integer>  
<variable>   ::=  <letter> | <letter> <string>  
<string>     ::=  <character> | <character> <string>  
<integer>    ::=  <digit> | <digit> <integer>  
<character>  ::=  <digit> | <letter>  
<digit>      ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

A <letter> is any alphabetic character from a to z or A to Z.

(a)     The table below contains a list of variable names. Place a tick in each row if the stated variable name is a valid <variable> for the production rules above.

You may tick **more than one** box.

|  |  |
| --- | --- |
| **<variable>** | **Valid? ( any number of rows)** |
| a |  |
| money\_paid |  |
| taxrate2 |  |
| 2ndPlayerName |  |

**(1)**

(b)     The production rule for an <integer> is recursive.

Explain why recursion has been used in this production rule.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(c)     Here is an example of a valid <forloop> :

FOR count = 1 TO 10

The BNF production rules above can be used to check whether or not a <forloop> is syntactically correct.

However, it is possible that a programming language statement that is a syntactically correct <forloop> may still produce an error when the program that it is part of is compiled.

Describe **one** example of why a syntactically correct <forloop> may still produce an error when a program is compiled.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

**(Total 3 marks)**

**Q30.**

Regular expressions can be used to search for strings.

(a)     For each of the following regular expressions, describe the set of strings that they would match.

(i)      b\*c

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(ii)     b?c

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(b)     Write a regular expression that matches the letter b, followed by zero or more occurrences of the string cd followed by either a single letter e or the string fg.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(2)**

**(Total 4 marks)**

**Q31.**

A computer program stores a list of integers in an array named List. The numbers in the array are to be sorted into ascending order so that a particular efficient search algorithm can be used to search for a number.

(a)     One of the search algorithms in **Table 1** can only be used successfully on a sorted list.

Place **one** tick next to the name of the algorithm that requires a list to be sorted.

**Table 1**

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

**(1)**

(b)     The pseudo-code for a standard algorithm that can be used to sort the data in the array List into order is shown in **Figure 1**. The variable ListLength stores a count of the number of items in the array List.

Array indexing starts at 1.

**Figure 1**

    For OuterPointer ← 2 To ListLength  
      CurrentValue ← List[OuterPointer]  
      InnerPointer ← OuterPointer - 1  
      While InnerPointer > 0 And  
            List[InnerPointer] > CurrentValue Do  
        List[InnerPointer + 1] ← List[InnerPointer]  
        InnerPointer ← InnerPointer - 1  
      EndWhile  
      List[InnerPointer + 1] ← CurrentValue  
    EndFor

Complete the empty (unshaded) cells in the trace table (**Table 2**) for an execution of the algorithm in **Figure 1** when the array List contains the values 9, 8, 5 and 6 in that order.

**Table 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | **List** | | | |
| **List Length** | **Outer Pointer** | **Current Value** | **Inner Pointer** | [1] | [2] | [3] | [4] |
| 9 | 8 | 5 | 6 |
| 4 | 2 |  | 1 |  |  |  |  |
|  |  |  | 0 |  |  |  |  |
|  | 3 |  | 2 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  | 0 |  |  |  |  |
|  | 4 |  | 3 |  |  |  |  |
|  |  |  | 2 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |

**(3)**

(c)     In the trace table (**Table 2**), when the variable OuterPointer contains the value 2 and then 3, the value of the variable InnerPointer decreases to 0. When OuterPointer contains 4, InnerPointer stops decreasing when it reaches the number 1.

Explain why InnerPointer does not decrease to 0 when OuterPointer contains 4.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(d)     Tick **one** box in **Table 3** to indicate the correct Order of **Time** Complexity of the standard algorithm in **Figure 1**.

**Table 3**

|  |  |
| --- | --- |
| **Order of Time Complexity** | **Tick one box** |
| O(n) |  |
| O(n2) |  |
| O(2n) |  |

**(1)**

(e)     State the name of the standard algorithm that is represented by the pseudo-code in **Figure 1**.

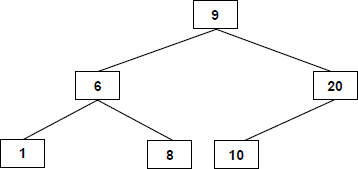
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(f)      Instead of storing a list of numbers in an array as in (b), the numbers could be stored in a binary search tree. This would also enable efficient searching.

The numbers 9, 6, 1, 8, 20 and 10 are put into a binary search tree in that order.  
**Figure 2** shows this binary search tree.

**Figure 2**

****

(i)      A search of the binary tree is performed for the number 8.

List the numbers, in the order that they would be checked, for the search to determine that the number 8 **is present** in the tree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(1)**

(ii)     A search of the binary tree is performed for the number 11.

List the numbers, in the order that they would be checked, for the search to determine that the number 11 **is not present** in the tree.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

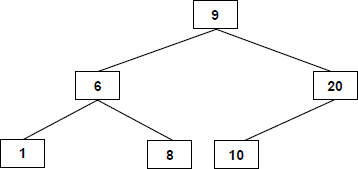
**(1)**

(g)     The numbers 4, 5 and 3 are to be added into the binary search tree, in that order.

**Figure 3** below is an identical copy of **Figure 2**.

Complete **Figure 3** below to show the binary search tree from **Figure 2** after the extra numbers have been added into it.

**Figure 3**

****

**(2)**

**(Total 11 marks)**

Mark schemes

**Q1.**

**All marks AO1 (understanding)**

Sorting a list is (always) a tractable problem // sorting a list is always polynomial time (or better);

A problem does not change from being tractable to intractable / polynomial to exponential as the problem size grows (an intractable problem is one that is not solvable in a reasonable amount of time as the size of the problem grows);

**[2]**

**Q2.**

**All marks AO1 (understanding)**

Use of heuristic;

An algorithm that makes a guess/estimate based on experience;

**N.E.** algorithm that uses previous knowledge/experience

That provide a close-to-optimal solution/approximation // that only works in some cases; **A.** non-optimal

Relax some of the constraints on the solution; **A.** solve simpler version of problem

**A.** Reduce the size of the search space

**Max 2 marks**

**[2]**

**Q3.**

**All marks AO2 (apply)**

(a)  **Mark as follows:**

**1 mark:** first row of tape is correct

**1 mark:** current state and read/write head position correct for first row of tape

**1 mark:** second and third rows of tape and current state are correct

**1 mark:** last row of tape is correct

**1 mark:** all other rows of current state are correct and read/write head in correct position for row two onwards

**A.** alternative, unambiguous, ways of representing read/write head position

**I.** inclusion of shaded rows/columns

**5**

(b)  **Mark is for AO2 (analyse)**

Make a copy of a string of 1s;

**A.** double the number of 1s on the tape

**1**

(c)  **Mark is for AO2 (analyse)**

Moves the read/write head to the start of the (original) string of 1s // moves the read/write head back to where it started from;

**1**

**[7]**

**Q4.**

**All marks AO1 (knowledge)**

Turing machine that can execute/simulate the behaviour of any other Turing machine // can compute any computable sequence;

Faithfully executes operations on the data precisely as the simulated TM does; (Note: must have idea of same process)

Description of/Instructions for TM (and the TM's input) are stored on the (Universal Turing machine's) tape // The UTM acts as an interpreter; **A.** take any other TM and data as input

*Alternative definition:*

A UTM, U, is an interpreter that reads the description <M> of any arbitrary Turing machine M;

and faithfully executes operations on data D precisely as M does.;

The description <M> is written at the beginning of the tape, followed by D.;

**Max 2 marks**

**[2]**

**Q5.**

**Mark is for AO1 (knowledge)**

Grouping by common characteristics // a hierarchical / ‘kind-of’ relationship;

**[1]**

**Q6.**

**Mark is for AO2 (analyse)**

(If there is a relationship between two cells is still represented but) if the relationship is because two cells are in the same row/column/two-by-two block is no longer represented // the nature of the link between the two cells is not represented; **A.** the location of a cell is not represented

**[1]**

**Q7.**

**Mark for AO1 (knowledge)**

Zero or more (of the preceding element/character/value);

**A.** any number of the preceding element/character/value

**[1]**

**Q8.**

**Mark for AO1 (knowledge)**

Zero or one (of the preceding element/character/value) // (the preceding element/character/value is) optional;

**[1]**

**Q9.**

**All marks AO2 (apply)**

|  |  |
| --- | --- |
| **String** | **Belongs to language (Y/N)?** |
| 1 | Y |
| 11 | N |
| 01 | Y |
| 0111 | Y |
| 0101 | N |
| 111 | N |
| 0011 | N |

**Mark as follows:**

**1 mark:** four rows correct

**2 marks:** five rows correct

**3 marks:** all seven rows correct

**[3]**

**Q10.**

**All marks AO2 (analyse)**

Take a vegetable from the box labelled "onions and carrots";

If it is an onion then the box labelled "onions" contains carrots and the box labelled "carrots" contains onions and carrots. If it is a carrot then the box labelled "carrots" contains onions and the box labelled "onions" contains carrots and onions;

**[2]**

**Q11.**

(a)  **All marks AO2 (analyse)**

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

**Alternative answer**

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

**Alternative answer**

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

**Mark as follows:**

**1 mark** 0s in correct places

**1 mark** all other values correct

**I.** non-zero symbols used to denote no edge but only for showing no edge going from a node to itself

**2**

(b)  **All marks for AO1 (understanding)**

Adjacency list appropriate when there are few edges between vertices // when graph/matrix is sparse; **NE**. few edges

Adjacency list appropriate when edges rarely changed;

Adjacency list appropriate when presence/absence of specific edges does not need to be tested (frequently);

**A.** Alternative words which describe edge, eg connection, line, arc

**Max 2**

**2**

(c)  **Mark is for AO2 (apply)**

It contains a cycle / cycles;

**1**

(d)  **Mark for AO1 (knowledge)**

A graph where each edge has a weight/value associated with it;

**1**

(e)  **All marks AO2 (apply)**

**Mark as follows:**

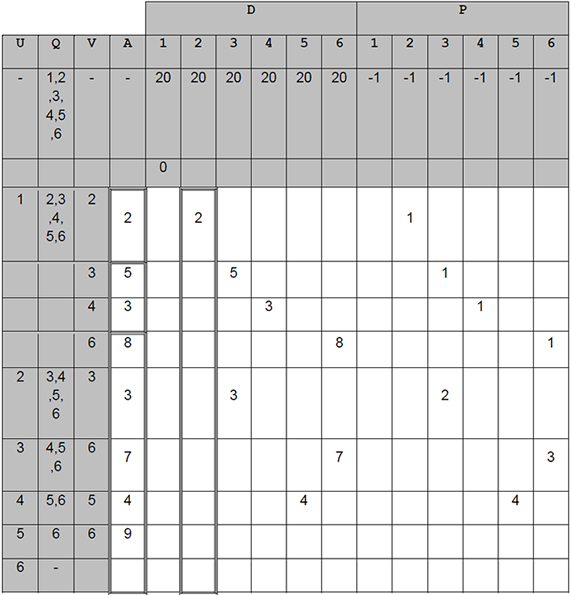
**I.** output column

**1 mark** first value of A is 2

**1 mark** second value of A is 5 and third value is 3

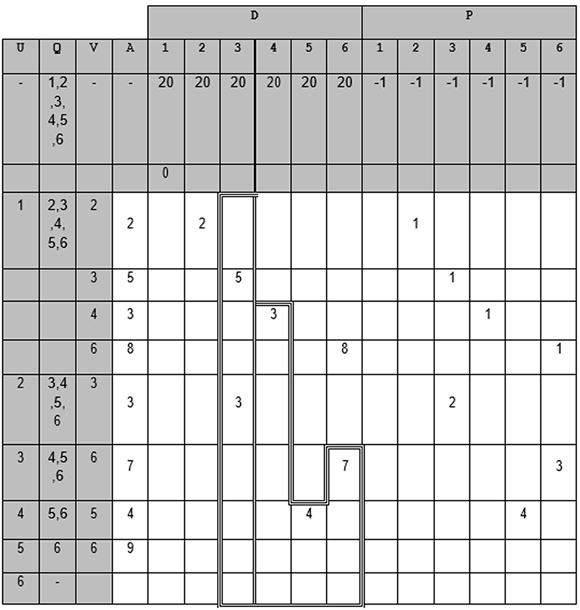
**1 mark** fourth and subsequent values of A are 8, 3, 7, 4, 9 with no more values after this

**1 mark** D[2] is set to 2 and then does not change

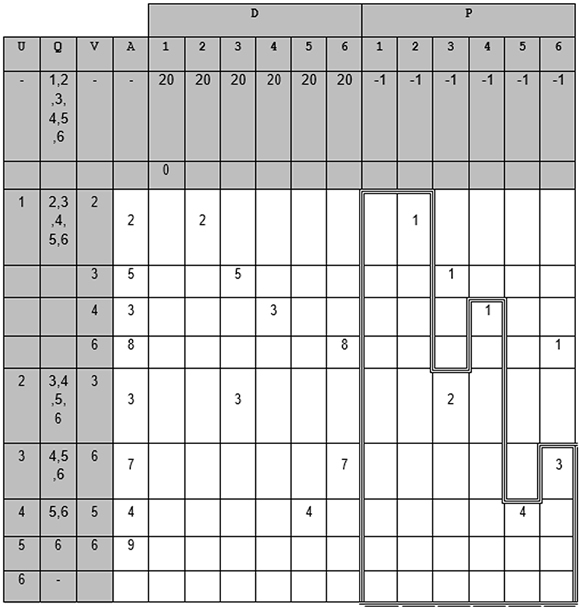


**1 mark** D[3] is set to 5 and then changes to 3 and does not change again

**1 mark** correct final values for each position of array P



**1 mark** correct final values for D[1], D[4], D[5], D[6]



**Max 6 marks if any errors**

**7**

(f)  **Mark is for AO2 (analyse)**

The shortest distance / time between locations/nodes 1 and 6;

**NE** distance / time between locations/nodes 1 and 6

**R.** shortest route / path

**1**

(g)  **All marks AO2 (analyse)**

Used to store the previous node/location in the path (to this node);

Allows the path (from node/location 1 to any other node/location) to be recreated // stores the path (from node/location 1 to any other node/location);

**Max 1** if not clear that the values represent the shortest path

**Alternative answer**

Used to store the nodes that should be traversed;

And the order that they should be traversed;

**Max 1** if not clear that the values represent the shortest path

**2**

**[16]**

**Q12.**

**All marks AO1 (knowledge)**

Determining if a program will halt;

**Max 1 for the following points, but only award mark if 1st mark was awarded:**

without running the program;

for a particular input;

**[2]**

**Q13.**

**Mark is for AO1 (understanding)**

The Halting problem is non-computable / undecideable // there is no algorithm that solves the Halting problem;

**A.** it is not computable

In general, inspection alone cannot always determine whether any given algorithm will halt for its given inputs // a program cannot be written that can determine whether any given algorithm will halt for its given inputs;

**Max 1 mark**

**[1]**

**Q14.**

**All marks AO1 (knowledge)**

Finite set of states (in a state transition diagram);

A set of transition rules;

A (sensing) read-write head (that can move along the tape one square at a time);

Start state;

(Set of) accepting / halting states;

State register // current state;

**[2]**

**Q15.**

**All marks AO1 (knowledge)**

A Turing machine that can execute/simulate the behaviour of any other Turing machine // can compute any computable sequence;

Faithfully executes operations on the data precisely as the simulated TM does; (Note: must have idea of same process)

Description of/Instructions for TM (and the TM's input) are stored on the (Universal Turing machine's) tape // The UTM acts as an interpreter; **A**. take any other TM and data as input

*Alternative definition:*

A UTM, U, is an interpreter that reads the description <M> of any arbitrary Turing machine M;

and faithfully executes operations on data D precisely as M does.;

The description <M> is written at the beginning of the tape, followed by D.;

**Max 2 marks**

**[2]**

**Q16.**

**Mark is for AO1 (understanding)**

Because it has an infinite amount of memory / tape;

**[1]**

**Q17.**

(a)  **Marks are for AO1 (understanding)**

|  |  |
| --- | --- |
| **Real number** | **Valid? (Yes/No)** |
| 87.000 | Yes |
| 97+12 | No |
| 12.31E+12 | Yes |

**A**. alternative indicators for Yes/No eg Y/N.

**Mark as follows:**

One mark per correct row

**3**

(b)  **Marks are for AO2 (apply)**

<natural> ::= <digit> | <digit> <natural>

**A**. alternative names for <natural>

**A**. recursive and non-recursive cases swapped around

**Mark as follows:**

**1 mark:** correct recursive case

**1 mark:** correct non-recursive case

**MAX 1** if any errors in answer eg missing |

**2**

**[5]**

**Q18.**

(a)  **Mark is for AO1 (knowledge)**

Merge sort;

**1**

(b)  **Mark is for AO1 (understanding)**

4;

**1**

**[2]**

**Q19.**

(a)  **Mark is for AO1 (knowledge)**

n2 // O(n2);

**A**. other ways of indicating n2 e.g. n^2

**A**. On2

**1**

(b)  **Marks are for AO1 (understanding)**

In each pass through the list n items will be examined;

There will be (at most) n passes through the list;

**2**

**[3]**

**Q20.**

(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]**

**Q21.**

(a)     One mark per correct response.

|  |  |  |
| --- | --- | --- |
| **Construct** | **Example** | **Valid? (Yes/No)** |
| *identifier* | Game\_Over | No; |
| *parameter* | ref x,y:bool | Yes; |
| *procedure-def* | procedure square(s:float) | Yes; |
| *procedure-def* | procedure rect(w:int,h:int) | No; |

**A**. Alternative clear indicators of Yes/No such as Y/N, True/False and Tick/Cross.

**4**

(b)     The <type> rule is missing type string;

The <procedure-def> rule does not allow a procedure without parameters // cannot be just an identifier;

Accept answers comparing the figures the other way around, ie

•        The type rule has an extra type string

•        The procedure does not have to have parameters / can be just an identifier

**2**

(c)     Required as there can be a list of parameters // required as there can be more than one parameter;

BNF does not support iteration // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise // recursion allows for more than one parameter;

**MAX 1**

**A.** Input for parameter

**NE.** Rule needs to loop

**1**

**[7]**

**Q22.**

Most efficient:  B // O(n)

   C // O(n3)

Least efficient: A // O(2n)

**[1]**

**Q23.**

The problem can be solved;

In polynomial time or better // in a reasonable amount of time;

**A**. "Algorithm exists" for can be solved

**A**. Answers relating to space rather than time

**[2]**

**Q24.**

|  |  |
| --- | --- |
| **Problem** | **Tractable? (Tick one)** |
| The travelling salesman problem. |  |
| The problem of searching a list. | ✔; |
| The Halting problem. |  |

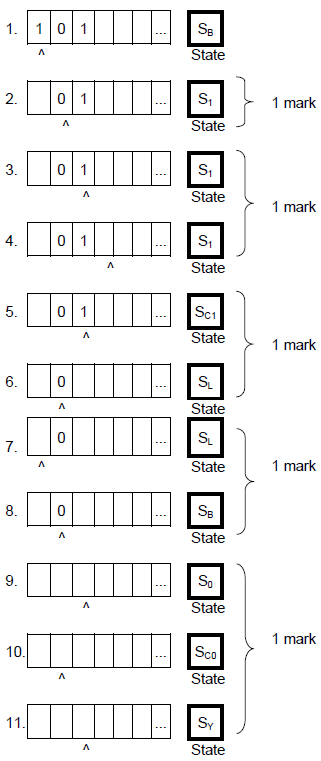
**A**. Alternative indicators for ticks

**Do not award mark if more than one box is ticked.**

**[1]**

**Q25.**

(a)  One mark per bracketed section.



**Must have correct tape contents and state for each mark**

**A**. Blank symbols instead of empty cells

**DPT** If the read/write head is not drawn on some rows, this should result in the loss of the mark on the first occasion that it is missing only. Marks should be awarded for subsequent rows, even if the read/write is not drawn.

**5**

(b)  (After a 0 has been read,) the rules keep moving the read/write head to the right (preserving the contents of the tape) // the read/write head skips right over 0s and 1s;

Until a blank symbol is encountered / the end of the number is reached, then the state is changed to SC0 (and the head is moved left/direction reversed);

**Note:** To achieve the first mark, it must be clear that the head moves right regardless of whether a 0 or 1 is read and also that this is a repeated process ie not just moving one place right.

**Note:** If it is stated that the process of moving continues until the end of the number is reached, then it can be inferred that the head was moving right for the first mark, if this was not explicitly stated.

**Note:** Marks should not be awarded for just explaining what the rules do individually.

**2**

**[7]**

**Q26.**

(a)     Parity Bit: 1;

**Start bit, Stop Bit**: Can be either 0 or 1, but must both be different to get mark;

**2**

(b)     **Definition (1 mark):**

Receiver and transmitter (clocks) do not need to be/are not (exactly) synchronised // transmission of data without use of external clock signal // receiver and transmitter clock only synchronised at start of/for length of transmission // data sent as soon as available rather than waiting for clock pulse/ synchronisation symbol;

**Explanation of start and stop bits (max 2 marks):**

Start bit synchronises receiver (clock) (to transmitter/data) // locks receiver and transmitter in phase // starts receiver's clock // wakes receiver;

Stop bit allows start bit to be recognised // allows receiver to process received bits;

**A.** Start and stop bits indicate when data is being transmitted/ begins – if neither of the other two marks for start and stop bits have been awarded

**3**

(c)     1010001;

**A.** Separator between digits e.g. comma

**1**

(d)     It is the parity bit;

**A.** Odd parity bit

**A.** If there are an even or odd number of 1s in the input

**1**

(e)     Only a small quantity of data to send // data transmission speed not important;

Widespread availability of USB/serial connections;

Serial communication avoids crosstalk // interference between signals on each wire;

Serial communication avoids data skew;

**A.** Serial communication is cheaper to implement with a suitable reason given

**A.** For future flexibility if devices were moved further apart

**N.E.** Serial is less error prone / fewer errors

**MAX 2**

**2**

**[9]**

**Q27.**

(a)

|  |  |
| --- | --- |
| **Turing machine component** | **Number (1-5) of modern computer system component with most similar role** |
| Transition function | 5; **A.** Program |
| Tape | 3; **A.** Main Memory / Memory |

**2**

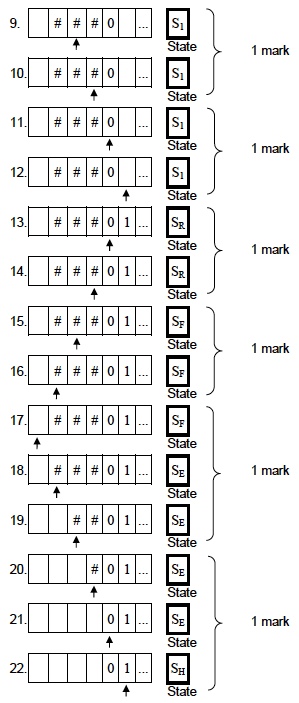
(b)     If (and only if) an algorithm exists to solve a problem then a Turing machine can be designed to solve the problem;

**A.** Statement made in reverse i.e. “if a Turing machine exists...”

**A.** Statement made as a negative i.e. “if no algorithm exists ...”

**A.** A Turing machine can compute any algorithm // any algorithm can be computed by a Turing machine

**1**

(c)     

**Must have correct tape contents and state for each mark**

**A.** Blank symbols instead of empty cells

**A.** Content written anywhere on the tape as long as the position is correct relative to the read/write head

**A.** Read/write head drawn off left hand end of tape at stage 17 if tape contents are written at left hand end of tape

**DPT** If the read/write head is not drawn on some rows, this should result in the loss of the mark on the first occasion that it is missing only. Marks should be awarded for subsequent rows, even if the read/write head is not drawn.

**6**

(d)     To reverse a (binary) string/number // to produce a copy of a (binary) string/number with the order of the characters/digits reversed;

**R.** Flips bits, but **A.** Flips order of bits

**A.** Mirror the input

**1**

**[10]**

**Q28.**

(a)     The problem can be solved // algorithm exists for problem;

in polynomial time (or less) // in a reasonable amount of time;

**2**

(b)     Use of heuristic;  
An algorithm that makes a guess / estimate based on experience;  
**NE** just algorithm that uses previous knowledge / experience  
That provides a close-to-optimal solution / approximation // that only works in some cases;  
**A** non-optimal  
Example of heuristic method e.g. hill-climbing / stochastic / local improvement / greedy algorithms / simulated annealing / trial and error / any reasonable example;  
Relax some of the constraints on the solution;  
**A** solve simpler version of problem  
**A** limit size of input  
**MAX 2**

**2**

(c)

|  |  |
| --- | --- |
| **Problem** | **Unsolvable? (Tick one row)** |
| The problem of sorting a list into order. |  |
| The Halting problem. |  |
| The travelling salesman problem. |  |

**R** Responses in which more than one row is ticked  
**A** Responses in which a symbol other than a tick is used, so long as it is only placed on one row.  
**A** Use of two symbols, with one indicating which problem is unsolvable and the other indicating which two are solvable, so long as the meaning of the symbols is clear.

**1**

(d)

|  |  |
| --- | --- |
| **Order of Time Complexity** | **Least Efficient? (Tick one row)** |
| O(2n) |  |
| O(n) |  |
| O(n2) |  |

**R** Responses in which more than one row is ticked  
**A** Responses in which a symbol other than a tick is used, so long as it is only placed on one row.  
**A** Use of two symbols, with one indicating which algorithm is least efficient and the other indicating which two are more efficient, so long as the meaning of the symbols is clear.

**1**

**[6]**

**Q29.**

(a)

|  |  |
| --- | --- |
| **<variable>** | **Valid? (Tick any number of rows)** |
| a |  |
| money-paid |  |
| taxrate2 |  |
| 2ndPlayerName |  |

**1 mark** for ticks in the correct two rows and other rows left blank.

**A** Use of alternative symbol for tick  
**A** Use of two symbols - one to indicate validity and one to indicate invalidity, so long as the meaning of the symbols is clear e.g. a tick and a cross or a Y and an N.

**1**

(b)     Required as an integer can contain any number of digits;  
**NE** More than one digit  
**A** “numbers” for “digits” as **BOD**BNF does not support iteration / looping // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise;  
**NE** Rule needs to loop  
**MAX 1**

**1**

(c)     Variable may not have been declared;  
Variable may be of inappropriate type;  
Position of statement within program may be invalid;  
Rightmost integer may be a lower value than the leftmost one;  
One of the numbers / limits may be outside of the range of valid integers;  
**A** Examples of any of the above  
**MAX 1**

**1**

**[3]**

**Q30.**

(a)     (i)      Zero or more bs followed by a / one c;

**A** answers by example but must be at least c, bc, bbc and indicate the sequence continues.

**1**

(ii)     Zero or one bs followed by (a / one) c / / the strings c or bc;

**1**

(b)     Correct expression: b(cd)\*(e|fg)

**A** use of incorrect bracket types

**A** accept brackets around fg

**A** (cd)+? for (cd)\*

**I** ^ at start, $ at end of expression

**2 marks** for the full correct expression.

**1 mark** for including either (cd)\* or (e|fg) in an incorrect expression.

**2**

**[4]**

**Q31.**

(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]  9 | [2]  8 | [3]  5 | [4]  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(n2) |  |
| O(2n) |  |

**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;

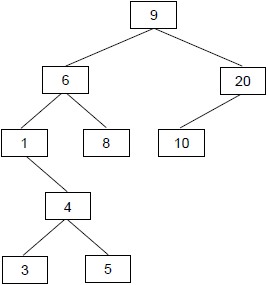
Must be in the order above. Can be separated by any character or a space

**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]**