4.3.4 Search & Sorting Algorithms

1) **Figure 1** shows a bubble sort algorithm represented using pseudo-code. The algorithm sorts the data in a list List.

Figure 1

```
PROCEDURE BubbleSort(List)
   N ← LEN(List) - 2
   I ← 0
   WHILE I < LEN(List) - 1
   FOR J ← 0 TO N
      IF L[J] > L[J + 1] THEN
         Temp ← L[J]
         L[J] ← L[J + 1]
         L[J + 1] ← Temp
      ENDIF
   ENDFOR
   I ← I + 1
   ENDWHILE
ENDPROCEDURE
```

a) Describe two changes that could be made to this bubble sort algorithm that would be likely to result in fewer comparisons being made when sorting the list List. The algorithm should still be a bubble sort algorithm if your suggested changes were made.

[4 marks]

Change 1

- **Have a flag variable that is set to True if a swap is made and reset to False at the start of each pass**
- **change the outer loop so that it would stop repeating if no swaps have been made;**

Change 2

- **After the inner loop;**
- **subtract 1 from N;**

b) A bubble sort algorithm could be used to sort the list of numbers into ascending order.

Complete the cells of **Table 1** to show the results of completing three passes through the list using a bubble sort algorithm.

You should state the values at the end of each pass.

Table 1

	[0]	[1]	[2]	[3]	[4]	[5]
	2	5	8	1	6	4
First pass	**2**	**5**	**1**	**6**	**4**	**8**
Second pass	**2**	**1**	**5**	**4**	**6**	**8**
Third pass	**1**	**2**	**4**	**5**	**6**	**8**

[3 marks]

2) **Figure 2** contains pseudo-code for a recursive merge sort algorithm. **Figure 3** contains pseudo-code for an algorithm called Merge that is called by the merge sort algorithm in **Figure 2**.

Figure 2

```
FUNCTION MergeSort(List, Start, End)
   IF Start < End THEN Mid ← (Start + End) DIV 2
      Left ← MergeSort(List, Start, Mid)
      Right ← MergeSort(List, Mid + 1, End)
      RETURN Merge(Left, Right)
   ELSE
      RETURN Append([], List[Start])
   ENDIF
ENDFUNCTION
```

Figure 3

```
   FUNCTION Merge(Left, Right)
      Result ← [ ]
      WHILE Len(Left) > 0 AND LEN(Right) > 0
         IF Left[0] < Right[0] THEN
            Result ← Append(Result, Right[0])
            Right ← RemoveFirstItem(Right)
         ELSE
            Result ← Append(Result, Left[0])
            Left ← RemoveFirstItem(Left)
         ENDIF
      ENDWHILE
      WHILE Len(Left) > 0
         Result ← Append(Result, Left[0])
         Left ← RemoveFirstItem(Left)
      ENDWHILE
      WHILE Len(Right) > 0
         Result ← Append(Result, Right[0])
         Right ← RemoveFirstItem(Right)
      ENDWHILE
      RETURN Result
   ENDFUNCTION
```

a) What is meant by a recursive subroutine?

[1 mark]

- **A subroutine that calls itself;**

b) What is the base case for the subroutine MergeSort?

[1 mark]

- **When Start is greater than or equal to End;**

c) Complete **Table 2** to show the result of tracing the MergeSort algorithm shown in **Figure 2** with the function call MergeSort(ListToSort, 0, 4). ListToSort is the list [7, 2, 3, 9, 4]. The first six rows and the Call number column have been completed for you.

Table 2

Call number	Start	End	Mid	List returned
1	0	4	2	
2	0	2	1	
3	0	1	0	
4	0	0		[7]
3	0	1	0	
5	1	1		[2]
3	0	1	0	[7, 2]
2	0	2	1	
6	2	3		[3]
2	0	3	1	[7, 3, 2]
1	0	4	2	
7	3	4	3	
8	3	3		[9]
7	3	4	3	
9	4	4		[4]
7	3	4	3	[9, 4]
1	0	4	2	[9, 7, 4, 3, 2]

[6 marks]

3) **Figure 4** shows an incomplete algorithm for a binary search.

Figure 4

```
PROCEDURE BinarySearch(List, First, Last, ItemToFind)
   Found ← False
   Failed ← (1).....................................
   WHILE NOT Failed AND NOT Found
      Mid ← (First + Last) DIV 2
      IF List[Mid] = ItemToFind THEN
         Found ←  True
      ELSE IF First >= Last
         (2).....................................
      ELSE IF List[Mid] > ItemToFind THEN
         (3).....................................
      ELSE
         First ← Mid + 1
      ENDIF
   ENDWHILE
   IF Found = True THEN
      OUTPUT "Item is in list"
   ELSE
      OUTPUT "Item is not in list"
   ENDIF
ENDPROCEDURE
```

The DIV operator calculates the whole number result of integer division. For example, 7 DIV 2 = 1.

a) What code should be added at position **(1)** in Figure **4**?

[1 mark]

- **False;**

b) What code should be added at position **(2)** in Figure **4**?

[1 mark]

- **THEN Failed ← True;**

c) What code should be added at position **(3)** in Figure **4**?

[2 marks]

- **Last ← Mid – 1;**