5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

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

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

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

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in **part (a)(i)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

.....

.....

.....

.....

.....

..... [3]

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

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

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

.....

.....

.....

..... [2]

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

3 The arrays `PollData[1:10]` and `CardData[1:10]` store data.

PollData	12	85	52	57	25	11	33	59	56	91

CardData	11	12	25	33	52	56	57	59	91	85

An **insertion sort** sorts these data.

(a) State why it will take less time to complete an insertion sort on `CardData` than on `PollData`.

.....

.....[1]

(b) The following pseudocode algorithm performs an insertion sort on the `CardData` array.

Complete the following **pseudocode** algorithm.

```
01  ArraySize ← 10
02  FOR Pointer ← 2 TO .....
03    ValueToInsert ← CardData[Pointer]
04    HolePosition ← .....
05    WHILE (HolePosition > 1 AND (..... > ..... ))
06      CardData[HolePosition] ← CardData[.....]
07      HolePosition ← .....
08    ENDWHILE
09    CardData[HolePosition] ← .....
10  ENDFOR
```

[7]

(c) (i) A binary search algorithm is used to find a specific value in an array.

Explain why an array needs to be sorted before a binary search algorithm can be used.

.....

.....

.....

.....

.....

.....

.....

.....[2]

(ii) The current contents of `CardData` are shown.

11	12	25	33	52	56	57	59	85	91

Explain how a binary search will find the value `25` in `CardData`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[4]

(d) Complete this procedure to carry out a binary search on the array shown in **part (c)(ii)**.

```
PROCEDURE BinarySearch(CardData, SearchValue)
   DECLARE Midpoint : INTEGER
   First ← 1
   Last ← ARRAYLENGTH(.....)
   Found ← FALSE
   WHILE (First <= Last) AND NOT(Found)
      Midpoint ← .....
      IF CardData[Midpoint] = SearchValue
         THEN
            Found ← TRUE
         ELSE
            IF SearchValue < CardData[Midpoint]
               THEN
                  Last ← .....
               ELSE
                  First ← .....
            ENDIF
         ENDIF
   ENDWHILE
ENDPROCEDURE
```

[4]

5 The following procedure performs an insertion sort on the global array `TheArray` that has 10 elements.

Complete the pseudocode for the procedure `InsertionSort()`.

```
PROCEDURE InsertionSort()

  DECLARE Count : INTEGER

  DECLARE Counter : INTEGER

  DECLARE Temp : INTEGER

  Count ← .....

  WHILE Count < 10

    Temp ← TheArray[Count]

    Counter ← Count .....

    WHILE ..... >= 0 AND TheArray[Counter] > .....

    TheArray[Counter + 1] ← TheArray[Counter]

    Counter ← Counter - 1

    ENDWHILE

    TheArray[.....] ← Temp

    Count ← Count + 1

  ENDWHILE

ENDPROCEDURE
```

[5]

4 **(a)** The array `Numbers[0 : Max]` stores numbers. An insertion sort can be used to sort these numbers into ascending order.

Complete the following **pseudocode** for the insertion sort algorithm.

```
FOR Pointer ← 1 TO (Max - 1)
   ItemToInsert ← .....
   CurrentItem ← .....
   WHILE (CurrentItem > 0) AND (Numbers[CurrentItem - 1] > ItemToInsert)
      Numbers[.....] ← Numbers[CurrentItem - 1]
      CurrentItem ← CurrentItem - 1
   ENDWHILE
   Numbers[CurrentItem] ← .....
ENDFOR
```

[4]

(b) Identify **two** features of the array `Numbers` that would have an impact on the performance of this insertion sort algorithm.

1 .....

2 .....

[2]

3 A bubble sort algorithm is used to sort an integer array, `List`. This algorithm can process arrays of different lengths.

(a) Write **pseudocode** to complete the bubble sort algorithm shown.

```
01 FOR Outer ← ..... TO 0 STEP - 1
02   FOR Inner ← 0 TO (.....)
03     IF .....  >  .....
04       THEN
05         Temp ← .....
06         List[Inner] ← .....
07         List[Inner + 1] ← .....
08     ENDIF
09   ENDFOR
10 ENDFOR
```

[7]

(b) (i) State the order of the sorted array.

..... [1]

(ii) State which line of the algorithm you would change to sort the array into the opposite order.

State the change you would make.

Line .....

Change .....

.....

[1]

(c) Use **pseudocode** to write an alternative version of this bubble sort algorithm that will exit the algorithm when the list is fully sorted.

.... [4]

1 There are several different searching and sorting algorithms.

(a) Identify **two** sorting algorithms.

1 .....

2 .....
[2]

(b) Consider the following pseudocode algorithm.

```
LowerBound ← 0
UpperBound ← LengthOfList - 1
ValueFound ← FALSE
OUTPUT "Value to find: "
INPUT ValueToFind
WHILE ValueFound = FALSE AND UpperBound <> LowerBound
    MidPoint ← (LowerBound + UpperBound)DIV 2
    IF List[MidPoint] = ValueToFind
        THEN
            ValueFound ← TRUE
        ELSE
            IF List[MidPoint] < ValueToFind
                THEN
                    LowerBound ← MidPoint + 1
                ELSE
                    UpperBound ← MidPoint - 1
            ENDIF
    ENDIF
ENDWHILE
IF ValueFound = FALSE
    THEN
        MidPoint ← (LowerBound + UpperBound) DIV 2
        IF List[MidPoint] = ValueToFind
            THEN
               OUTPUT "Item in position " & MidPoint & " in list"
            ELSE
               OUTPUT "Not in list"
        ENDIF
    ELSE
        OUTPUT "Item in position " & MidPoint & " in list"
ENDIF
```

Note: `DIV` is an operator that performs integer division.

The array `List` contains the following values:

2, 5, 21, 25, 36, 48, 51, 59, 65, 70

(i) Complete the trace table to show a dry run of the algorithm, when the value 21 is input.

LowerBound	UpperBound	ValueFound	ValueToFind	MidPoint

[3]

(ii) Identify this type of searching algorithm.

..... [1]

(iii) The value 59 is input.

State the number of times the while loop condition is executed.

..... [1]

(iv) State the minimum number of times the while loop condition will be executed to search for a value.

..... [1]

(v) `MidPoint` is calculated and checked again after the while loop is terminated.

Explain why this additional calculation and check is necessary.

.....

.....

.....

..... [2]

(vi) A new data set is used as follows:

5, 9, 10, 12, 15, 13, 17, 19, 20, 2

Explain why the algorithm will not find the value 2 in this data set.

.....

.....

.....

..... [2]

1 An array, `NumberArray`, stores 100 integer values. The array needs to be sorted into ascending numerical order.

(a) Describe how an insertion sort will sort the data in `NumberArray`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

(b) Another type of sorting algorithm is a bubble sort.

The procedure `Bubble()` takes an array as a parameter. It performs a bubble sort on the array. The sorting algorithm stops as soon as all the elements are in ascending order.

Complete the procedure `Bubble()`.

```
PROCEDURE Bubble(BYREF NumberArray : ARRAY[0 : 99] OF INTEGER)
    DECLARE Outer : INTEGER
    DECLARE Swap : BOOLEAN
    DECLARE Inner : INTEGER
    DECLARE Temp : INTEGER
    Outer ← LENGTH(NumberArray) - 1
    REPEAT
        Inner ← .....
        Swap ← FALSE
        REPEAT
            IF NumberArray[Inner] > NumberArray[Inner + 1]
                THEN
                    Temp ← NumberArray[Inner]
                    NumberArray[Inner] ← NumberArray[Inner + 1]
                    NumberArray[Inner + 1] ← Temp
                    Swap ← .....
            ENDIF
            Inner ← Inner + 1
        UNTIL Inner = .....
        Outer ← Outer - 1
    UNTIL Swap = ..... OR Outer = .....
ENDPROCEDURE
```

[5]