Insertion sort

VB Code
Explanation on Page 2.

```
Dim outer As Integer
Dim inner As Integer
Dim temp As Integer

For outer = 1 To 9
    temp = MyList(outer)
    inner = outer

    Do While inner > 0 AndAlso MyList(inner - 1) > temp
        MyList(inner) = MyList(inner - 1)
        inner = inner - 1
    Loop

    MyList(inner) = temp
Next outer
'End of insertion sort
```

An insertion sort traverses an array from the second element to the last. Each element is compared to the elements before in turn, working backwards down the list. Values are swapped until the element being compared is placed in order.

The following is a worked example of an insertion sort.

Iteration 1

Start with element 1 of the list to be sorted. This value is temporarily stored.

0	1	2	3	4	5	6	7	8	temp
45	23	99	7	8	64	37	63	34	23

If the temporary value (23) is smaller than the value before it (45), then the value before it is copied to the right.

45	45	99	7	8	64	37	63	34	23

Each value, to the left of the element where the temporary value was originally stored, is compared in turn until:

- the value being compared is smaller than the stored temporary value

or

- the start of the list has been reached

When either of the previous bullets is true, the temporary value is copied back into the list.

23	45	99	7	8	64	37	63	34	23

Iteration 2

When the next element (99) is examined, the element before it (45) is smaller, so no further action is required.

0	1	2	3	4	5	6	7	8	temp
23	45	99	7	8	64	37	63	34	99

Iteration 3

When the value in element 3 is compared to every element before it, the result is that the values in indexes 0, 1 and 2 are all copied one element to the right (as 7 is smaller than 23, 45 and 99).

0	1	2	3	4	5	6	7	8	temp
23	45	99	7	8	64	37	63	34	7

The temporary value is copied into element index 0.

7	23	45	99	8	64	37	63	34	7

Iteration 4

When the value in element 4 is compared to the values in indexes 0 to 3, it is smaller than every value, except element 0 (7).

0	1	2	3	4	5	6	7	8
7	23	45	99	8	64	37	63	34

temp
8

The values 23, 45 and 99 all move right. The temporary value this time is copied into index 1.

7	8	23	45	99	64	37	63	34

8

By this stage, the algorithm of an insertion sort should be apparent, as follows:

- Each element from 1 to the length of the array is copied into temporary storage and dealt with in turn.
- Every larger value to the left is moved up one element.
- The temporary value is copied back into the list when the next value is smaller, or when the end of the array is reached.

Pseudocode

Design	Commentary
`fixed loop i = 1 to length(list)-1`	Loop from the second element to the last
`  store the value at array index i`	Store the current temporary value
`  store the starting position of the inner loop`	Store the current position in the array — this will be used as a starting point to count backwards during the comparisons
`  while index > 0 and value < list[index`	Continue comparing previous values in the list with the temporary value until the start of the array is reached or the two values are in the correct order
`    copy the value at index i into index i+1`	The compared value is copied into the element to the right
`    reduce the index by 1`	Decrement the element being compared next
`  end while`	
`  copy the stored value into index i`	The temporarily stored value is copied into the correct place
`end fixed loop`	

SQA reference language: insertion sort procedure

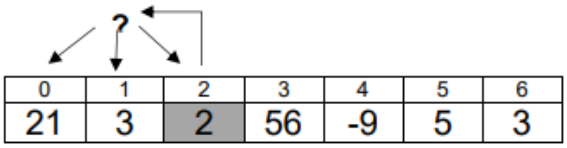
```
PROCEDURE insertion_sort(list)
      DECLARE value INITIALLY 0
      DECLARE index INITIALLY 0
      FOR i = 1 to length(list)-1 DO
            SET value TO array[i]
            SET index TO i
            WHILE (index > 0) AND (value < list[index-1]) DO
                  SET list[index] TO list[index-1]
                  SET index TO index - 1
            END WHILE
            SET list[index] TO value
      END FOR
END PROCEDURE
```

INSERTION SORT

Insertion sort is a comparative sort, which builds the sorted array one entry at a time.

It works by ordering the first two items, then inserting the 3rd item in the correct place, relative to the first two, the 4th item in the correct place, relative to the first 3, and so on.

This means that there is a continual reapplication of comparisons and swaps for each element in the array.



By the end of each traversal, each item will be in the correct position in the list of sorted items so far.

Another way of understanding the steps taken by this algorithm:

Pass 1

- Store value in list position 2 in temporary variable
- Compare temporary value with value in list position 1
- If list position 1 value is larger than temporary value
  - copy list position 1 to list position 2
- If list position 1 value is not larger
  - copy temporary value to list position 2

Pass 2

- Store value in list position 3 in temporary variable
- Compare temporary value with values in list positions 2 and then 1
- Each time list value is larger than temporary value
  - Copy list item to next list position
- If compared list item is not larger than temporary item
  - Copy temporary value to current list position (2 or 1)

Pass 3

- Store value in list position 4 in temporary variable
- Compare temporary value with values in list positions 3 and then 2 and then 1 … and so on.