Solutions

Each answer shows the steps, then the **result**.

2.1

(a) $120 - 1 + 1 =$ **120**

(b) $8 - 0 + 1 =$ **9** – not 8

(c) $150 \times 2 =$ **300**

2.2

- lower bound **5**, upper bound **14**
- $14 - 5 + 1 =$ **10** elements

2.3

(a) **1-D** – the marks are a single sequence of values, one per student

(b) **2-D** – the board has two natural dimensions, three rows by three columns

2.4

- compare and swap each adjacent pair in turn: `7, 3` swap; `7, 9` in order; `9, 2` swap
- after one pass: **`3, 7, 2, 9`**
- the largest value, `9`, has reached the end

2.5

- a valid index can be `0` in an array whose lower bound is `0`, so `0` could be mistaken for a genuine result
- `-1` can never be a valid index, so after the loop `FoundAt = -1` unambiguously means "not found"

3.1

- an outer loop that repeats the passes
- an inner loop over the adjacent pairs
- the comparison `Data[Index] > Data[Index + 1]`
- the swap through a **temporary** variable, all three lines
- the loop ends correctly, leaving the array sorted

```
DECLARE Index, Pass, Temp : INTEGER
FOR Pass <- 1 TO n - 1
    FOR Index <- 1 TO n - 1
        IF Data[Index] > Data[Index + 1] THEN
            Temp <- Data[Index]
            Data[Index] <- Data[Index + 1]
            Data[Index + 1] <- Temp
        ENDIF
    NEXT Index
NEXT Pass
```

3.2

(a) a `Swapped` flag, set `FALSE` at the start of each pass and `TRUE` inside the `IF`

- the outer loop becomes `REPEAT ... UNTIL Swapped = FALSE`
- a `Limit` that decreases by one after each pass, with the inner loop running to `Limit - 1`

```
Limit <- n
REPEAT
    Swapped <- FALSE
    FOR Index <- 1 TO Limit - 1
        IF Data[Index] > Data[Index + 1] THEN
            Temp <- Data[Index]
            Data[Index] <- Data[Index + 1]
            Data[Index + 1] <- Temp
            Swapped <- TRUE
        ENDIF
    NEXT Index
    Limit <- Limit - 1
UNTIL Swapped = FALSE
```

(b) the **flag** stops the sort as soon as the array is in order, so an already-sorted array costs one pass instead of $n - 1$

- the **shrinking limit** avoids re-examining the values at the end, which each pass has already put in their final place

3.3 any four, in a sensible order:

- repeat the following until a pass makes no swaps
- compare each adjacent pair of elements in turn, starting at the beginning
- if a pair is out of order, swap the two values
- after each pass the largest unsorted value has reached the end, so the next pass can stop one place earlier
- no pseudocode keywords used (`FOR`, `IF`, `<-` are pseudocode, so they lose the style mark)

3.4

- header with the parameter, and the loop counter declared
- loop from `Position` to the second-from-last element
- `Stock[Index] <- Stock[Index + 1]` – copying **forwards**, so nothing is overwritten before it is moved
- set the last element to `""` after the loop
- `ENDPROCEDURE`, with everything closed

```
PROCEDURE Remove(Position : INTEGER)
    DECLARE Index : INTEGER
    FOR Index <- Position TO 99
        Stock[Index] <- Stock[Index + 1]
    NEXT Index
    Stock[100] <- ""
ENDPROCEDURE
```

- copying backwards here would fill the rest of the array with one repeated value