Bubble sort: compare 1

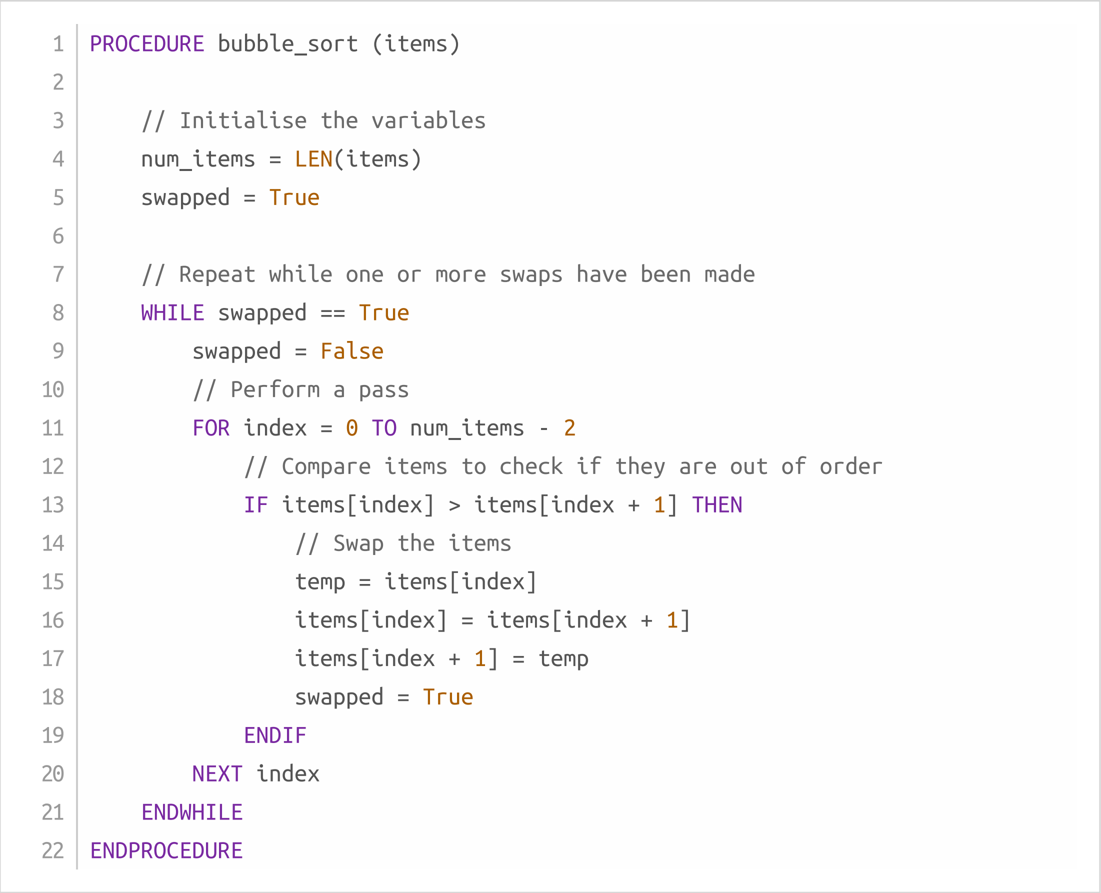
GCSE

C C

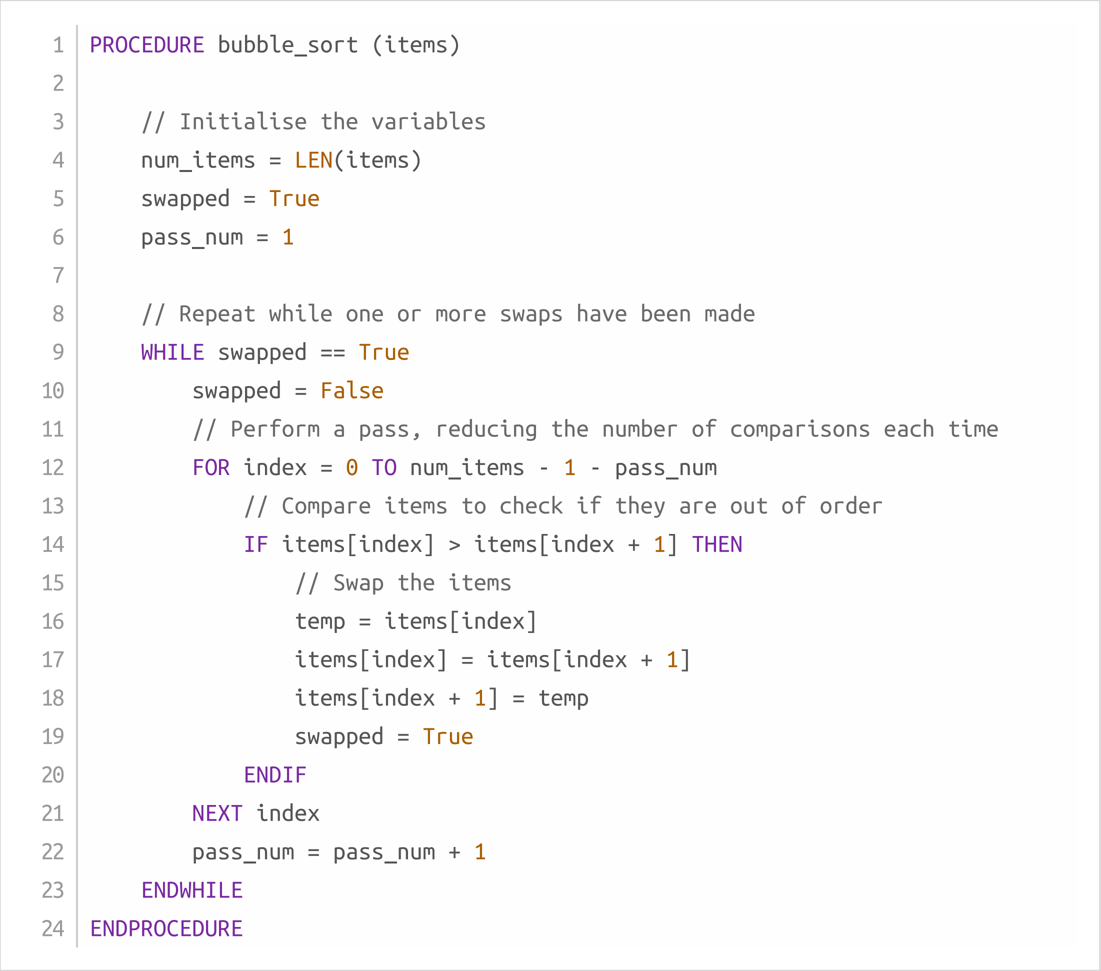
A bubble sort algorithm can be programmed in multiple ways. Look at the three versions of the bubble sort algorithm (use the tabs to see each version) and put them in order of efficiency with the **most** efficient first and the **least** efficient last in the list.

Version 1 Version 2 Version 3

```
PROCEDURE bubble_sort (items)

    // Initialise the variables
    num_items = LEN(items)

    // Pass through the array of items n-1 times
    FOR pass_num = 1 TO num_items - 1
        // Perform a pass
        FOR index = 0 TO num_items - 2
            // Compare items to check if they are out of order
            IF (items[index] > items[index + 1]) THEN
                // Swap the items
                temp = items[index]
                items[index] = items[index + 1]
                items[index + 1] = temp
            ENDIF
        NEXT index
    NEXT pass_num
ENDPROCEDURE
```



```
PROCEDURE bubble_sort (items)

    // Initialise the variables
    num_items = LEN(items)
    swapped = True

    // Repeat while one or more swaps have been made
    WHILE swapped == True
        swapped = False
        // Perform a pass
        FOR index = 0 TO num_items - 2
            // Compare items to check if they are out of order
            IF items[index] > items[index + 1] THEN
                // Swap the items
                temp = items[index]
                items[index] = items[index + 1]
                items[index + 1] = temp
                swapped = True
            ENDIF
        NEXT index
    ENDWHILE
ENDPROCEDURE
```



```
PROCEDURE bubble_sort (items)

    // Initialise the variables
    num_items = LEN(items)
    swapped = True
    pass_num = 1

    // Repeat while one or more swaps have been made
    WHILE swapped == True
        swapped = False
        // Perform a pass, reducing the number of comparisons each time
        FOR index = 0 TO num_items - 1 - pass_num
            // Compare items to check if they are out of order
            IF items[index] > items[index + 1] THEN
                // Swap the items
                temp = items[index]
                items[index] = items[index + 1]
                items[index + 1] = temp
                swapped = True
            ENDIF
        NEXT index
        pass_num = pass_num + 1
    ENDWHILE
ENDPROCEDURE
```

○ 3, 2, 1

○ 1, 2, 3

○ 2, 1, 3

Bubble sort: purpose of temp

A Level

P P

Bubble sort moves each item in a list up until it reaches a value that is higher than that item. It then looks at the next item and does the same. Once all of the items have been moved, the collection is sorted.

Consider the pseudocode below.

Pseudocode

```
PROCEDURE bubblesort_ascending(data_set)
    num_items = LEN(data_set)
    swaps = True
    WHILE i < (num_items - 2) AND swaps == True
        swaps = False
        FOR j = 0 TO (num_items - i - 2)
            IF (data_set[j] > data_set[j + 1]) THEN
                temp = data_set[j]
                data_set[j] = data_set[j + 1]
                data_set[j + 1] = temp
                swaps = True
            ENDIF
        NEXT j
        i = i + 1
    ENDWHILE
ENDPROCEDURE
```

Explain the purpose of the `temp` variable in the algorithm.

○ `temp` is used to store a copy of one of the values that is being swapped.

○ `temp` is used to count the number of comparisons between items.

○ `temp` is used to stop the algorithm when the list is sorted.

○ `temp` is used to store the total number of items in the list.

Home > Bubble sort: maximum swaps

Bubble sort: maximum swaps

A Level

P P

Michael is going to sort an array `items` that holds **5 random numbers** generated using a subroutine that produces random integers. As the numbers are generated at runtime, it is not possible to guess how the unsorted array will look.

The array is to be sorted into ascending order using the version of the bubble sort algorithm shown in pseudocode below:

Pseudocode

```
PROCEDURE bubble_sort(items)
    // Initialise variables
    num_items = LEN(items)
    temp = 0
    pass_number = 1
    swapped = True

    // Continue while swaps have been made and there are more passes to evaluate
    WHILE swapped == True AND pass_number <= num_items - 1
        swapped = False
        FOR index = 0 TO num_items - 2
            // Check if items are out of order
            IF (items[index] > items[index + 1]) THEN
                // Swap items
                temp = items[index]
                items[index] = items[index + 1]
                items[index + 1] = temp
                swapped = True
            ENDIF
            pass_number = pass_number + 1
        NEXT index
    ENDWHILE
ENDPROCEDURE
```

What is the **maximum** number of swaps that might be carried out?

What is the **maximum** number of passes that might be performed?

Bubble sort: trace 3

GCSE

P P

Geraint is using the **bubble sort** algorithm to put their list of top five countries to visit into alphabetical order, from A to Z.

Using the `countries` list below, fill in the blanks to show the order of the countries after **each pass of the bubble sort algorithm**, using a new row for each pass. The algorithm will pass over the entire list four times.

`countries`				
Laos	Egypt	Peru	Fiji	Cuba

Items:

Cuba Egypt Fiji Laos Peru

Insertion: complete algorithm

A Level

P P

The **insertion sort** algorithm works by building up a sorted sublist, one item at a time. To do that, one by one the items in the list are examined and inserted into the correct position in the sorted sublist. The sorted sublist grows until the whole list is sorted.

The following pseudocode subroutine is an implementation of the insertion sort algorithm. There is a block of code missing.

Drag and drop the statements into the correct order to complete the missing part of the code. Make sure that you **indent** the statements as appropriate.

Pseudocode

```
PROCEDURE insertion_sort (items)
    num_items = LEN(items)
    FOR index = 1 TO num_items - 1
        item_to_insert = items[index]
        previous = index - 1
        >>>>  [missing block of code]  <<<<
    NEXT index
ENDPROCEDURE
```

Available items

```
WHILE previous >= 0 AND items[previous] > item_to_insert
```

```
previous = previous - 1
```

```
items[previous + 1] = item_to_insert
```

```
items[previous + 1] = items[previous]
```

```
ENDWHILE
```

Insertion: trace 2

A Level

P P

Pam is a teacher. She wants to sort her students test scores so that the highest score appears at the start of the list.

She uses an **insertion sort** to sort the data.

Drag and drop the lines into the correct order to show the state of the list of scores as the data is progressively sorted (i.e. as each item is correctly positioned).

The initial order of the list is: $43, 74, 64, 68, 49, 70$

Available items

- 74, 64, 43, 68, 49, 70
- 74, 68, 64, 43, 49, 70
- 74, 70, 68, 64, 49, 43
- 74, 68, 64, 49, 43, 70
- 74, 43, 64, 68, 49, 70

Insertion: trace 4

GCSE

P P

Geraint is using the **insertion sort** algorithm to put their list of top five countries to visit into alphabetical order, from A to Z.

countries				
Peru	Laos	Fiji	Egypt	Cuba

The insertion sort algorithm has been started on the list below. In the first row, the first item is highlighted to show it is in the sorted sublist and the unsorted sublist contains the remaining items.

In the second row, the first pass has completed and the sorted sublist now contains two items, which are highlighted.

Fill in the blanks to show the order of the countries at the end of each pass when performing an **insertion sort**. Use a new row for each pass.

countries				
Peru	Laos	Fiji	Egypt	Cuba
Laos	Peru	Fiji	Egypt	Cuba

Items:

Cuba Egypt Fiji Laos Peru

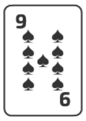
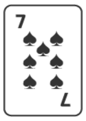
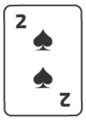
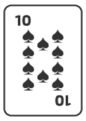
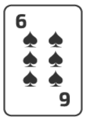
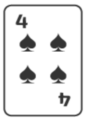
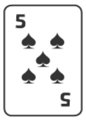
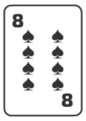
Home > Merge sort: trace 6

Merge sort: trace 6

GCSE

P P

You have been given some playing cards that have not been sorted:

Original order							
							

Drag and drop the cards provided to show the order they would be in at the **end** of each **step** of a **merge sort**.

For this part of the question, you just need to show the **splitting** part of the algorithm.

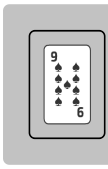
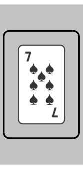
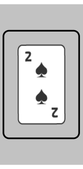
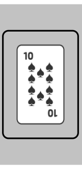
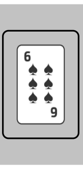
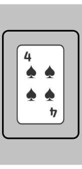
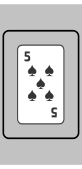
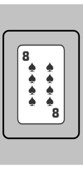
Splitting

	group 1				group 2			
Step 1								

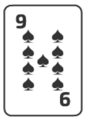
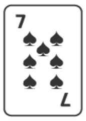
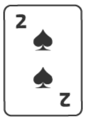
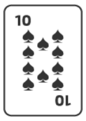
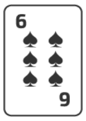
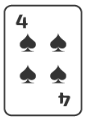
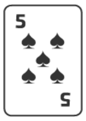
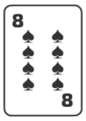
	group 1		group 2		group 3		group 4	
Step 2								

	group 1	group 2	group 3	group 4	group 5	group 6	group 7	group 8
Step 3								

Items:

You have been given some playing cards that have not been sorted:

Original order							
							

Drag and drop the cards provided to show the order they would be in at the **end** of each **step** of a **merge sort**.

For this part of the question, you just need to show the **merging** part of the algorithm. You will need to use your answer from part 1 of the question to help you.

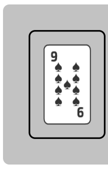
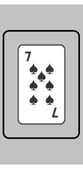
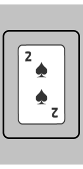
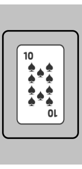
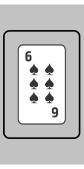
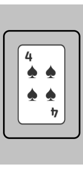
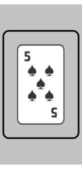
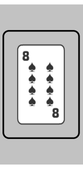
Merging

Step 4	group 1		group 2		group 3		group 4	

Step 5	group 1				group 2			

Step 6	Sorted cards							

Items:

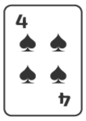
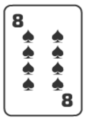
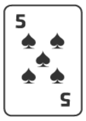
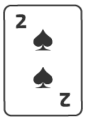
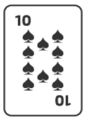
Home > Merge sort: trace 4

Merge sort: trace 4

GCSE

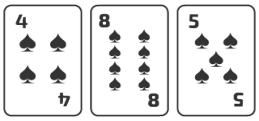
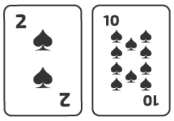
P P

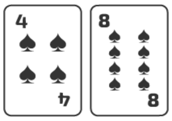
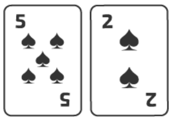
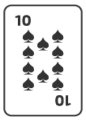
You have been given some playing cards that have not been sorted:

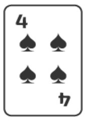
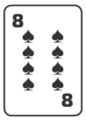
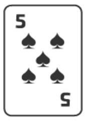
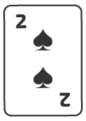
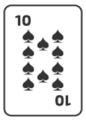
				

Apply a **merge** sort to sort the cards from the lowest to highest value.

The first three steps of splitting the data have been completed for you:

	Group 1	Group 2
Step 1		

	Group 1	Group 2	Group 3
Step 2			

	Group 1	Group 2	Group 3	Group 4	Group 5
Step 3					

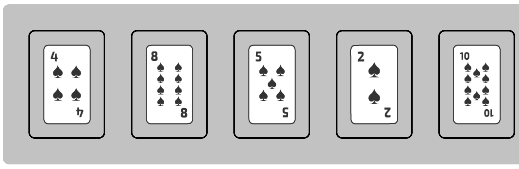
Drag and drop the cards provided at the bottom of the question into the correct positions to show the **merge** steps of the algorithm:

	Group 1	Group 2	Group 3
Step 4			

	Group 1	Group 2
Step 5		

	Sorted group
Step 6	

Items:

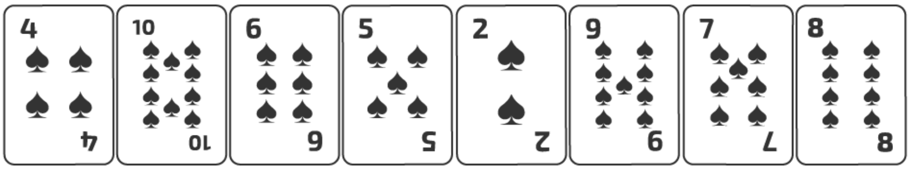


Merge sort: trace 5

GCSE

P P

You have been given some playing cards that have not been sorted:



Drag the lines of cards below into the correct order to show how **each** stage of a merge sort would be performed to sort the the data from lowest to highest.

Available items