Coding a Bubble Sort Trace Table

You have studied a flowchart for the *bubble sort* algorithm, and examined both Pearson pseudocode and Java code that implements the algorithm. Below a Python implementation is given.

```
 1  def bubble_sort(arr):
 2      """
 3      Basic bubble sort algorithm. Time Complexity: O(n²) worst
 4      case, O(n) best case
 5      Space Complexity: O(1)
 6      """
 7      n = len(arr)
 8      # Traverse through all elements
 9      for i in range(n-1):
10          for j in range(0, n - i - 1):
11              if arr[j] > arr[j + 1]:
12                  arr[j], arr[j + 1] = arr[j + 1], arr[j]
13      return arr
14  # Example usage
15  numbers = [ 25, 11, 22, 12, 64, 90, 34 ]
16  sorted_numbers = bubble_sort(numbers.copy())
17  print("Sorted:",sorted_numbers)
```

```
1  Sorted: [11, 12, 22, 25, 34, 64, 90]
```

You have also worked through trace tables to show the flow of the algorithm. Now let's confirm the trace table by modifying the above code so that it prints out a trace table. Add the following functionality:

- A counter to keep track of the number of passes through the loop. A pass is one time through the entire list, swapping adjacent elements.
- A counter to keep track of the number of times a comparison is made of two adjacent elements.
- A printed line of text after each swap that contains the loop counters (`i` and `j`), and the array that is being sorted. (BONUS: add the string `"<- swap"` after any line that contains a swap.
- A constant named `DIVIDER` that is a string of 35 dashes (`-`)
- A printed dividing line after each pass of the algorithm (using the constant `DIVIDER`)
- A printed line of text that reports the number of passes made and the number of comparisons made by the completion of the algorithm.

1. Examine the output of the completed program, given on the next page, and answer the questions.

a) How many passes were made by the algorithm, total.	
b) On which pass was the last swap performed	
c) How many comparisons were made of two adjacent list elements by the algorithm?	
d) How many swaps of two adjacent list elements were made by the algorithm	
e) How many comparisons were made after the array was sorted?	

Coding a Bubble Sort Trace Table

The output of the updated program is given below:

```
i j - Array
-----------------------------------
0 0 - [11, 25, 22, 12, 64, 90, 34] <- swap
0 1 - [11, 22, 25, 12, 64, 90, 34] <- swap
0 2 - [11, 22, 12, 25, 64, 90, 34] <- swap
0 3 - [11, 22, 12, 25, 64, 90, 34]
0 4 - [11, 22, 12, 25, 64, 90, 34]
0 5 - [11, 22, 12, 25, 64, 34, 90] <- swap
-----------------------------------
1 0 - [11, 22, 12, 25, 64, 34, 90]
1 1 - [11, 12, 22, 25, 64, 34, 90] <- swap
1 2 - [11, 12, 22, 25, 64, 34, 90]
1 3 - [11, 12, 22, 25, 64, 34, 90]
1 4 - [11, 12, 22, 25, 34, 64, 90] <- swap
-----------------------------------
2 0 - [11, 12, 22, 25, 34, 64, 90]
2 1 - [11, 12, 22, 25, 34, 64, 90]
2 2 - [11, 12, 22, 25, 34, 64, 90]
2 3 - [11, 12, 22, 25, 34, 64, 90]
-----------------------------------
3 0 - [11, 12, 22, 25, 34, 64, 90]
3 1 - [11, 12, 22, 25, 34, 64, 90]
3 2 - [11, 12, 22, 25, 34, 64, 90]
-----------------------------------
4 0 - [11, 12, 22, 25, 34, 64, 90]
4 1 - [11, 12, 22, 25, 34, 64, 90]
-----------------------------------
5 0 - [11, 12, 22, 25, 34, 64, 90]
-----------------------------------
Passes: 6 Comparisons: 21 Swaps: 6
Sorted: [11, 12, 22, 25, 34, 64, 90]
```

So you should realize an inefficiency in the algorithm that we could easily improve on. This can be corrected by keeping track of whether any swap was made during a pass through the list: if no swap was made during a pass through the list, then the array must be sorted. Update the your bubble sort code further to:

- initialize a boolean variable (representing whether or not swapping occurred) to `False` at the start of each pass through the list – no element has been swapped (yet).
- if any adjacent elements in the list are swapped during the pass, update the variable to `True`.
- after each pass is complete of the list, if no swaps were made, use a `break` statement to break out of the outer loop (i.e.: don't go through any further passes of the list).

The expected output of the program is given on the following page. Examine it, then answer the questions below it.

Coding a Bubble Sort Trace Table


```
i j - Array
----------------------------------
0 0 - [11, 25, 22, 12, 64, 90, 34] <- swap
0 1 - [11, 22, 25, 12, 64, 90, 34] <- swap
0 2 - [11, 22, 12, 25, 64, 90, 34] <- swap
0 3 - [11, 22, 12, 25, 64, 90, 34]
0 4 - [11, 22, 12, 25, 64, 90, 34]
0 5 - [11, 22, 12, 25, 64, 34, 90] <- swap
----------------------------------
1 0 - [11, 22, 12, 25, 64, 34, 90]
1 1 - [11, 12, 22, 25, 64, 34, 90] <- swap
1 2 - [11, 12, 22, 25, 64, 34, 90]
1 3 - [11, 12, 22, 25, 64, 34, 90]
1 4 - [11, 12, 22, 25, 34, 64, 90] <- swap
----------------------------------
2 0 - [11, 12, 22, 25, 34, 64, 90]
2 1 - [11, 12, 22, 25, 34, 64, 90]
2 2 - [11, 12, 22, 25, 34, 64, 90]
2 3 - [11, 12, 22, 25, 34, 64, 90]
----------------------------------
Passes: 3 Comparisons: 15 Swaps: 6
Sorted: [11, 12, 22, 25, 34, 64, 90]
```

2. Answer the questions by giving a value.

a) Recall that 6 passes were made by the original algorithm (without early termination). How many passes does the updated algorithm make?

b) Recall that 21 comparisons of adjacent list elements were made by the original algorithm. How many comparisons were made by the updated algorithm?

c) Recall that 6 swaps of two adjacent elements were made by the original algorithm. How many swaps does the updated algorithm make?

d) Recall that 10 comparisons were made after the list was sorted by the original algorithm. How many comparisons were made after the array was sorted for the updated algorithm?

3. Notice that the list is sorted after the second pass through, but the algorithm makes three passes. Why does the algorithm perform an additional pass through the list after it is sorted?