

Computing

Lesson 9: Coding Sorting Algorithms

Algorithms

Kashif Ahmed

Materials from the Teach Computing Curriculum created by the National Centre for Computing Education



Task 1 - Code for bubble sort

An implementation of a bubble sort in Python is shown in **Figure 1**. Read through the code to familiarise yourself with it; don't worry if you don't understand all of it yet.



```
1 def bubble_sort(items):
2     num_items = len(items) # Initialise the variables
3     passes = 1
4     # Repeat while the maximum numbers of passes has not been made
5     while passes < num_items:
6         # Repeat for each pair of items
7         for current in range(num_items - 1):
8             # Compare the item at the current position with the next item
9             if items[current] > items[current+1]:
10                # Swap the out-of-order items
11                temp = items[current]
12                items[current] = items[current+1]
13                items[current+1] = temp
14            # Increase the number of passes by 1
15        passes = passes + 1
```



Task 1 - Code for bubble sort

The following questions will be based on executing the algorithm in **Figure 1** when **items** is the list: **['Maya', 'Dan', 'Vivian', 'Tobi', 'Areeji']**.

Examine Line 5 and state how many times the inner loop is performed on the list above, i.e. how many pairs of items every single pass examines.

Examine Line 4 and state how many times the outer loop is performed on the list above, i.e. how many passes the algorithm makes.



Task 1 - Code for bubble sort

Complete the trace table below for lines 7 to 9 of the algorithm. The first line in the trace table contains the values for the **current** variable and the **items** list.

| | | | items | | | | |
|------|---------|------|-------|-----|--------|------|-------|
| Line | current | temp | [0] | [1] | [2] | [3] | [4] |
| | 0 | - | Maya | Dan | Vivian | Tobi | Areej |
| 7 | | | | | | | |
| 8 | | | | | | | |
| 9 | | | | | | | |
| | | | | | | | |



Task 2 - Improving bubble sort

Explain the purpose of Lines 7 to 9 in the bubble sort algorithm in **Figure 1**.

What happens when Line 12 is omitted from the algorithm in **Figure 1**?



Task 2 - Improving Bubble Sort - part 1

Reducing the number of comparisons

One improvement that could be made to the bubble sort algorithm is to change the range of the inner loop on Line 5 from **num_items - 1** to **num_items - passes**.



```
1 def bubble_sort(items):
2     num_items = len(items) # Initialise the variables
3     passes = 1
4     # Repeat while the maximum numbers of passes has not been made
5     while passes < num_items:
6         # Repeat for the range num_items - passes
7         for current in range(num_items - passes):
8             # Compare the item at the current position with the next item
9             if items[current] > items[current+1]:
10                # Swap the out-of-order items
11                temp = items[current]
12                items[current] = items[current+1]
13                items[current+1] = temp
14            # Increase the number of passes by 1
15        passes = passes + 1
```

Figure 2



Task 2 - Improving Bubble Sort - part 1

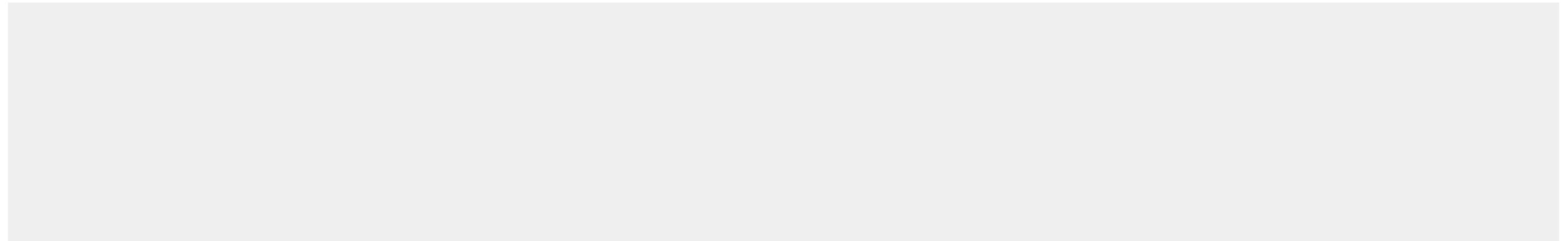
Complete the table below for tracing the two expressions **num_items - 1** and **num_items - passes** when **items** is a list of **eight items**.

| passes | num_items - 1 | num_items - passes |
|--------|---------------|--------------------|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |



Task 2 - Improving Bubble Sort - part 1

Explain how changing the range of the inner loop to **num_items - passes** increases the efficiency of the bubble sort algorithm compared to **num_items - 1**.



Task 2 - Improving Bubble Sort - part 2

Stopping when no swaps were made.

Now you are going to make a second improvement to the bubble sort algorithm in Figure 2 by following the instructions below:

- Insert the statements `swapped = False` and `swapped = True` in the algorithm so that `swapped` is reset to **False** at the beginning of each pass and set to **True** only when a swap occurs.



Task 2 - Improving Bubble Sort - part 2

- Modify the **while** condition so that the iteration continues only as long as `swapped` has been set to **True** in the previous pass, i.e. if at least one pair of elements was swapped.
- Add comments to the code to explain the changes you made.



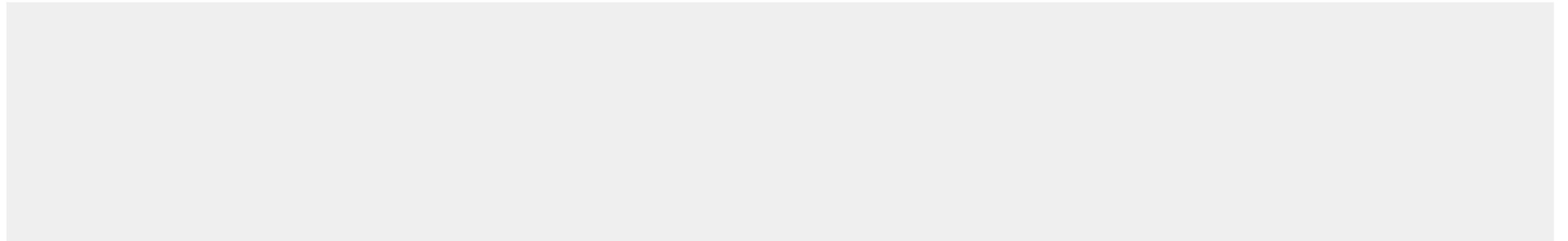
```
1 def bubble_sort(items):
2     num_items = len(items) # Initialise the variables
3     passes = 1
4     # Repeat while the maximum numbers of passes has not been made
5     while passes < num_items:
6         # Repeat for each pair of items, reducing the number of
7         # comparisons by the number of passes that have been completed
8         for current in range(num_items - passes):
9             # Compare the item at the current position with the next item
10            if items[current] > items[current+1]:
11                # Swap the out-of-order items
12                temp = items[current]
13                items[current] = items[current+1]
14                items[current+1] = temp
15            # Increase the number of passes by 1
16            passes = passes + 1
```



Task 3 - Code for Insertion Sort - part 1

Demonstrating insertion sort

Describe how an insertion sort is performed..



Task 3 - Code for Insertion Sort - part 1

Show the steps of an insertion sort on the list of data in **Figure 3** to put the elements into alphabetical order. Each pass should be on a new line and you should clearly highlight which part of the list is the sorted sublist. The first row has been filled in for you.

| | | | | | | |
|----------------|-------|--------|---------|--------|------|---------|
| Element | Chile | Guyana | Ecuador | Brazil | Peru | Bolivia |
| Index | 0 | 1 | 2 | 3 | 4 | 5 |

Figure 3



Task 3 - Code for Insertion Sort - part 1

| Chile | Guyana | Ecuador | Brazil | Peru | Bolivia |
|-------|--------|---------|--------|------|---------|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



Task 3 - Code for Insertion Sort - part 1

Demonstrate how an insertion sort would place the following numbers into ascending numerical order:

32, 8, 128, 16, 64, 256



Task 3 - Code for Insertion Sort - part 1

| | | | | | |
|--|--|--|--|--|--|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



Task 3 - Code for Insertion Sort - part 2

An insertion sort algorithm

An implementation of an insertion sort in Python is shown in **Figure 4**. Read through the code to familiarise yourself with it; don't worry if you don't understand all of it yet.



```
1 def insertion_sort(items):
2     num_items = len(items) # Initialise the variables
   # Repeat for each item in the unsorted part of the list
3     for first_unordered in range(1, num_items):
4         value = items[first_unordered] # Copy the first unordered item into value
5         current = first_unordered - 1 # set current to the position before
   # Repeat while the start of the list has not been reached
   # and the current item is greater than value
6         while current >= 0 and items[current] > value:
           # Copy the item from the current position to the next element
7             items[current+1] = items[current]
8             current = current - 1 # Proceed to the previous item in the list
   # Copy the value of the first unordered item into the correct position
9     items[current+1] = value
```

Figure 4



Task 3 - Code for Insertion Sort - part 2

State the number of times the outer **for** loop would repeat if **items** were a list of 10 items.

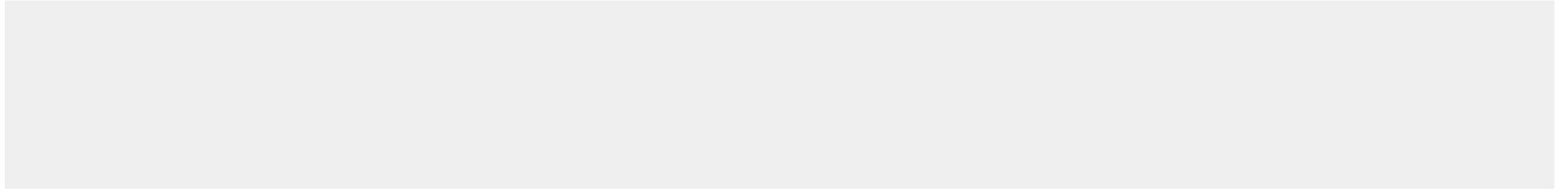
Hint: The first value of range is the start value and the second value is the stop value.

Describe what Line 3 does during each iteration of the outer for loop.



Task 3 - Code for Insertion Sort - part 2

Explain the purpose of the condition `items[current] > value` on Line 6.



Task 3 - Code for Insertion Sort - part 2

Complete the trace table below for Lines 6 to 9 of the algorithm. The first line in the trace table contains the **items** list after two passes of the algorithm (**first_unordered** is now 3). The variables **value** and **current** after executing Lines 4 and 5 have also been included in the table.



Task 3 - Code for Insertion Sort - part 2

| | | | items | | | | |
|------|--------|---------|-------|------|------|--------|------|
| Line | value | current | [0] | [1] | [2] | [3] | [4] |
| | | | Abeer | Lola | Yara | Carlos | Tami |
| 4 | Carlos | | | | | | |
| 5 | | 3 | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |



Task 3 - Code for Insertion Sort - part 2

Explain the purpose of Lines 7 to 8 in the insertion sort algorithm in **Figure 4**, using the table above as an example.

What happens when line 9 is omitted from the algorithm in **Figure 4**?

