ANSWERS

A5.4 Bubble sort and insertion sort

Algorithms and complexity · A level · OCR H446 2.3.1, AQA 7517 4.3.5.1, Eduqas A500QS 1.3 · about 25 min

What this lesson is about

Tracing passes and insertions, counting comparisons in the best and worst case, in-place sorting and stability.

- Bubble sort
- Hear the passes
- Insertion sort
- Choosing between them

Questions

6 marks in all

1. Bubble sort with a shrinking pass sorts a reversed list of 10 items. How many comparisons does it make? [1 mark]

> **Answer:** 45. 9 + 8 + … + 1 = 10 × 9 / 2 = 45.

2. What is the best-case time complexity of bubble sort that stops after a pass with no swaps? [1 mark]

- ☐ **A** O(n)
- ☐ **B** $O(n^2)$
- ☐ **C** O(1)
- ☐ **D** O(log n)

> **Answer:** A. On a sorted list one pass of n - 1 comparisons makes no swaps, and it stops.

3. What does this print? It shows the list after the first pass of bubble sort. [1 mark]

```
items = [45, 12, 38, 7, 26, 19]
for i in range(len(items) - 1):
    if items[i] > items[i + 1]:
        items[i], items[i + 1] = items[i + 1], items[i]
print(items)
```

> **Answer:**
>
> ```
> [12, 38, 7, 26, 19, 45]
> ```
>
> The largest item, 45, is carried to the end; the others move up one place when passed.

4. What does this print? It shows the list after insertion sort has inserted the first three keys. [1 mark]

```
items = [30, 10, 50, 20, 40]
for i in range(1, 4):
    key = items[i]
    j = i - 1
    while j >= 0 and items[j] > key:
        items[j + 1] = items[j]
        j = j - 1
    items[j + 1] = key
print(items)
```

Answer:

```
[10, 20, 30, 50, 40]
```

10 goes before 30, 50 stays, then 20 is inserted between 10 and 30. The 40 has not been looked at yet.

5. Which are true of both bubble sort and insertion sort? [1 mark]

Tick every answer that is true.

- ☐ **A** They sort in place with O(1) extra space
- ☐ **B** They are stable
- ☐ **C** Their worst case is $O(n^2)$
- ☐ **D** Their best case is O(n log n)

Answer: A, B, C. Both are in place, stable and $O(n^2)$ at worst. With an already sorted list both are O(n) at best.

6. New distance readings arrive one at a time and must be kept in order. Which sort suits this best? [1 mark]

- ☐ **A** Insertion sort, because each new reading can be inserted into the sorted part
- ☐ **B** Bubble sort, because it has the fewest swaps
- ☐ **C** Bubble sort, because it is stable
- ☐ **D** Neither can be used

Answer: A. Insertion sort grows a sorted part one item at a time, so it can take items as they arrive.

The task: count the comparisons

Write `bubble_sort(items)` and `insertion_sort(items)`. Each sorts the list it is given **in place**, into ascending order, and **returns the number of comparisons between two items** it made. - `bubble_sort` must use both improvements from this lesson: pass p (counting from 0) compares positions 0 and 1 up to n - 2 - p and n - 1 - p, and the sort stops after a pass with no swaps (or after n - 1 passes). - `insertion_sort` counts one comparison each time it compares an item in the sorted part with the key. When `j` falls below 0 there is no comparison, so do not count one. Run each sort on a **copy** of each of these lists (use `list(...)` to copy) and print six lines, bubble first, in exactly this form: `bubble ordered: 7` then `bubble reversed: <n>`, `bubble tune: <n>`, `insertion ordered: <n>`, `insertion reversed: <n>`, `insertion tune: <n>` Finally sort `tune` with either sort and play it, each note for 0.2 seconds, lowest first.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

ordered = [262, 294, 330, 349, 392, 440, 494, 523]
reversed_notes = [523, 494, 440, 392, 349, 330, 294, 262]
tune = [392, 262, 494, 330, 523, 294, 440, 349]

def bubble_sort(items):
    return 0

def insertion_sort(items):
    return 0
```

The hint students can ask for: Put the counter next to each comparison of two items, not next to each swap. In insertion sort, check that j is still 0 or more before comparing, and stop the inner loop as soon as the item is not larger than the key. Sort a copy each time, or the second sort will be handed a list that is already sorted.

A solution

```
from bugbot import *
connect()

ordered = [262, 294, 330, 349, 392, 440, 494, 523]
reversed_notes = [523, 494, 440, 392, 349, 330, 294, 262]
tune = [392, 262, 494, 330, 523, 294, 440, 349]

def bubble_sort(items):
    n = len(items)
    count = 0
    for p in range(n - 1):
        swapped = False
        for i in range(n - 1 - p):
            count = count + 1
            if items[i] > items[i + 1]:
                items[i], items[i + 1] = items[i + 1], items[i]
                swapped = True
        if not swapped:
            break
    return count

def insertion_sort(items):
    count = 0
    for i in range(1, len(items)):
```

```
        key = items[i]
        j = i - 1
        while j >= 0:
            count = count + 1
            if items[j] > key:
                items[j + 1] = items[j]
                j = j - 1
            else:
                break
        items[j + 1] = key
    return count

for name, data in [("ordered", ordered), ("reversed", reversed_notes), ("tune", tune)]:
    print("bubble " + name + ":", bubble_sort(list(data)))
for name, data in [("ordered", ordered), ("reversed", reversed_notes), ("tune", tune)]:
    print("insertion " + name + ":", insertion_sort(list(data)))
insertion_sort(tune)
for note in tune:
    tone(note, 0.2)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.