WORKSHEET

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

BugBotLab

Name _____ Class _____ Date _____

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]

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)

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)
```

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)
```

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)

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

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
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a5-4-bubble-and-insertion-sort/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. How many swaps does bubble sort make on the reversed list of 8? Is that also n(n - 1) / 2?
2. Rewrite bubble sort without the early stop. What is its best case now?
3. Change `insertion_sort` to use `>=` instead of `>`. Is it still stable? Test it on records with equal keys.