Insertion sort

Algorithms and programming (Paper 2) · revision notes

An insertion sort builds up a sorted part of the list one item at a time, taking each item in turn and inserting it into the correct place in the sorted part. You need to show the list after each insertion, describe the steps, recognise the code and compare it with bubble sort and merge sort.

Key facts

- First item = sorted part; start by inserting the second item
- n items need n − 1 insertions
- Pro: simple, in place (little memory), fast for small or nearly sorted lists
- Differs from bubble sort: builds a sorted part, rather than swapping adjacent pairs in passes
- Take the next item; shift larger sorted items one place right; insert it into the gap
- After each insertion, the sorted part at the start grows by one
- Con: slow for large, unsorted lists

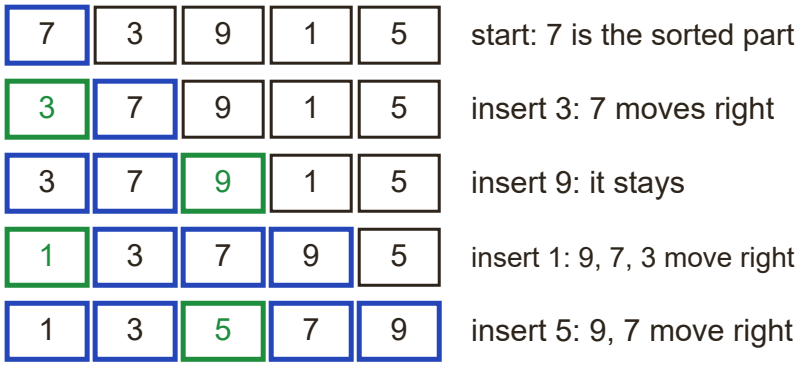
Notes

How an insertion sort works (ascending order)

- The first item on its own is treated as a sorted part of the list, so start with the **second** item.
- Take the next item from the unsorted part. Compare it with the items in the sorted part, working backwards from the right.
- Move each sorted item that is larger one place to the right, then insert the item into the gap.
- Repeat until the last item has been inserted. A list of n items needs n − 1 insertions.

Worked example: 7, 3, 9, 1, 5

- Insert 3: it is smaller than 7, so 7 moves right. List: 3, 7, 9, 1, 5.
- Insert 9: it is larger than 7, so it stays. List: 3, 7, 9, 1, 5.
- Insert 1: 9, 7 and 3 all move right. List: 1, 3, 7, 9, 5.
- Insert 5: 9 and 7 move right. List: 1, 3, 5, 7, 9.



Blue: the sorted part. Green: the item just inserted

The code

- The outer loop starts at position 1 (the second item). The inner while loop shifts larger items right.

```
for i = 1 to items.length - 1
   current = items[i]
   j = i - 1
   while j >= 0 AND items[j] > current
      items[j + 1] = items[j]
      j = j - 1
   endwhile
   items[j + 1] = current
next i
```

Advantages and disadvantages

- Advantages: simple to code; sorts in place, so it uses very little extra memory; quick on small lists and on lists that are already nearly sorted, because items already in place need only one comparison.
- Disadvantage: slow on large lists that are in a random order, because of the number of comparisons and shifts.

How to answer each type of question

Show the list after each insertion

3 to 4 marks | Grade 5

1. Leave the first item alone; insert the second item first.
2. For each item, find where it belongs in the sorted part at the start.
3. Write the whole list after each insertion, even if nothing moved.

Example. An insertion sort is used to put 12, 5, 18, 2, 9 into ascending order.
Show the list after each item is inserted. [4]

Model answer.

After inserting 5: 5, 12, 18, 2, 9 (1)
After inserting 18: 5, 12, 18, 2, 9 (1)
After inserting 2: 2, 5, 12, 18, 9 (1)
After inserting 9: 2, 5, 9, 12, 18 (1)

Describe how an insertion sort works

3 marks | Grade 5

1. Say that the first item is the sorted part and the sort starts with the second item.
2. Say that each item is compared with the sorted part and inserted in the correct place (larger items move right).
3. Say that this repeats until every item has been inserted.

Example. Describe how an insertion sort puts a list into ascending order. [3]

Model answer. The first item is treated as a sorted list, and the sort starts with the second item (1). Each item in turn is compared with the items in the sorted part and inserted into the correct position, with larger items moved one place to the right (1). This repeats until the last item has been inserted (1).

Explain when an insertion sort is a good choice

2 to 3 marks | Grade 7

1. Look at the data: is it small, or nearly sorted already?
2. Explain why: items already in place need only one comparison, so there are few moves.
3. Memory: it sorts in place.

Example. A list of 1000 scores is in ascending order. One new score is added to the end, and the list must be sorted again.
Explain why an insertion sort is a good choice. [2]

Model answer. Every score that is already in place needs only one comparison (1), so the only real work is inserting the new score into its correct position, which needs far fewer comparisons and shifts than sorting a random list (1).

Identify a sorting algorithm from code

1 to 2 marks | Grade 6

1. Outer loop from the second item and an inner loop that shifts items right: insertion sort.
2. A swapped flag and comparisons of items[i] with items[i + 1]: bubble sort.
3. Splitting into halves and merging: merge sort.

Example. Name the sorting algorithm in the code below and give **one** reason for your answer.

```
for i = 1 to items.length - 1
   current = items[i]
   j = i - 1
   while j >= 0 AND items[j] > current
      items[j + 1] = items[j]
      j = j - 1
   endwhile
   items[j + 1] = current
next i
```

[2]

Model answer. Insertion sort (1). Each item from the second onwards is stored in current, larger items before it are shifted one place right, and current is placed in the gap (1).

Don't lose marks

- Starting with the first item: it is already the sorted part, so the first insertion is the second item.
- Changing the order of the unsorted items that have not been inserted yet.
- Leaving out a step where the item stays where it is: the question may still want that list written out.
- Describing a bubble sort (swapping pairs in passes) when asked about insertion sort.

More tips

Memory tricks

- Think of sorting playing cards in your hand: pick up the next card and slide it into the right place among the cards you already hold.
- Check after the k-th insertion: the first k + 1 items should be in order (the rest are untouched).
- Spot the algorithm: 'shift and insert' = insertion; 'compare and swap neighbours in passes' = bubble; 'split and merge' = merge.

Exam technique

- Write one line per insertion, and check that the items after the sorted part are unchanged.
- Use the words 'sorted part', 'insert' and 'correct position' in descriptions.
- For a nearly sorted list, insertion sort is usually the best answer; for a very large random list, merge sort.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3

State what an insertion sort does

It takes each item in turn and inserts it into the correct place in a sorted part of the list.

Grade 5

Show the list after each insertion

Insert the second item, then the third, and so on, writing the list after each one.

Grade 5

Describe the steps of an insertion sort

Start with the second item, shift larger sorted items right, and insert the item in the gap.

Grade 6

Recognise insertion sort code

Spot an outer loop from the second item and an inner loop that shifts items right.

Grade 7

Explain when insertion sort is a good choice

It is quick for small lists and lists that are nearly sorted, and uses little memory.

Grade 8

Compare insertion, bubble and merge sort

Weigh up speed on large lists, memory use and how sorted the data already is.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.