Bubble sort and merge sort

1.2.6b · Computational thinking · revision notes

Sorting algorithms put a list into order. You need to know how a bubble sort and a merge sort work, show the list at each stage, and compare the two. Expect 'show the list after each pass', 'describe how a merge sort works' and 'discuss which sort is better for this task' questions.

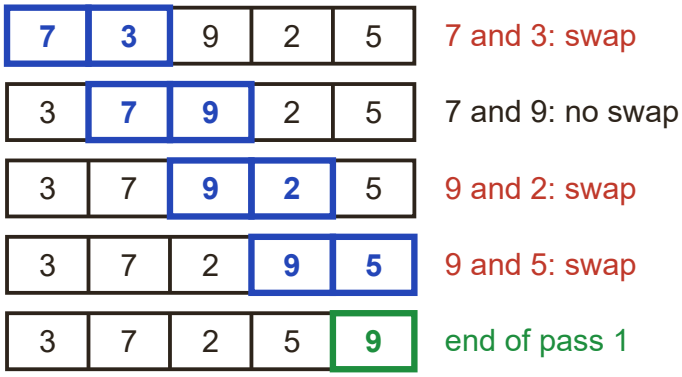
Key facts

- Bubble sort: compare adjacent items and swap them if they are in the wrong order
- After pass 1 (ascending order), the largest item is at the end
- Basic version: comparisons per pass = n − 1 (n = number of items)
- Merge sort: faster on large lists, but needs more memory
- Pass: one run through the list from the start to the end
- Bubble sort stops after a pass with no swaps
- Merge sort: split into single items, then merge pairs back together in order
- Bubble sort: simple, sorts in place, slow on large lists

Notes

Bubble sort

- Compare the first two items. If they are in the wrong order, swap them.
- Move along one place and compare the next pair of **adjacent** items. Carry on to the end of the list. This is one **pass**.
- After the first pass, the largest item (for ascending order) has 'bubbled' to the end, into its final place.
- Repeat passes until a pass makes no swaps. The list is then sorted.
- In the basic version, each pass through n items makes n − 1 comparisons. At most n − 1 passes are needed to put n items in order.
- Example: first pass of [7, 3, 9, 2, 5] into ascending order.
  7 > 3, swap: [3, 7, 9, 2, 5]. 7 and 9, no swap. 9 > 2, swap: [3, 7, 2, 9, 5]. 9 > 5, swap: [3, 7, 2, 5, 9].
  4 comparisons and 3 swaps, and 9 is now in its final place.



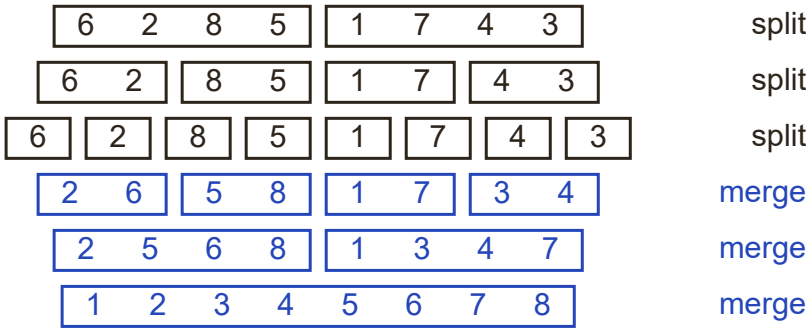
One pass: compare each adjacent pair (blue) and swap if they are in the wrong order. The largest item ends up at the end

- Improvements: stop after a pass with no swaps (a flag records whether a swap happened), and make one fewer comparison on each pass, because the items at the end are already in place.
- Bubble sort is simple to write and needs very little extra memory (one temporary variable for a swap), but it is slow on large lists.

Merge sort

- **Split** the list in half, then split each half again, until every list has one item. A list of one item is already sorted.
- **Merge** pairs of lists: compare the first items of the two lists and move the smaller one into a new list. Repeat until one list is empty, then move the rest of the other list across.
- Keep merging pairs of lists until there is one sorted list.

```
[6, 2, 8, 5, 1, 7, 4, 3]
[6, 2, 8, 5] [1, 7, 4, 3]
[6, 2] [8, 5] [1, 7] [4, 3]
[6] [2] [8] [5] [1] [7] [4] [3]
[2, 6] [5, 8] [1, 7] [3, 4]
[2, 5, 6, 8] [1, 3, 4, 7]
[1, 2, 3, 4, 5, 6, 7, 8]
```



Merge sort: split until every list has one item, then merge pairs back together in order (blue)

- If a list has an odd number of items, one half gets the extra item. Follow any rule the question gives.
- Merge sort is much faster than bubble sort on large lists, but it needs extra memory to hold the sublists and is more complex to write.

How to answer each type of question

Show a pass of a bubble sort

2 to 4 marks | **Grade 5**

1. Compare items 1 and 2, then items 2 and 3, and so on to the end.
2. Swap each pair that is in the wrong order, and use the new order for the next comparison.
3. Write the whole list after the pass, and count the swaps if asked.

Example. A bubble sort is used to put this list into ascending order.
[14, 6, 11, 3, 9]
(a) Give the list after the first pass.
(b) State the number of swaps made during the first pass.

Model answer.

14	**6**	11	3	9	14 and 6: swap
6	**14**	**11**	3	9	14 and 11: swap
6	11	**14**	**3**	9	14 and 3: swap
6	11	3	**14**	**9**	14 and 9: swap
6	11	3	9	**14**	end of pass 1

(a) [6, 11, 3, 9, 14]: 14 is in the last position (1); the other items are in the order 6, 11, 3, 9 (1).
(b) 4 (1): 14 is swapped with 6, 11, 3 and 9 in turn.

Describe how a merge sort works

3 to 4 marks | **Grade 6**

1. Splitting: divide the list in half repeatedly until each list has one item.
2. Merging: compare the first items of two lists and move the smaller into the new list.
3. Say that merging repeats until there is one sorted list.

Example. Describe how a merge sort puts a list of numbers into ascending order.

Model answer. The list is split in half repeatedly (1) until each sublist contains one item (1). Pairs of sublists are merged by comparing their first items and moving the smaller one into a new list (1). Merging continues until there is a single sorted list (1).

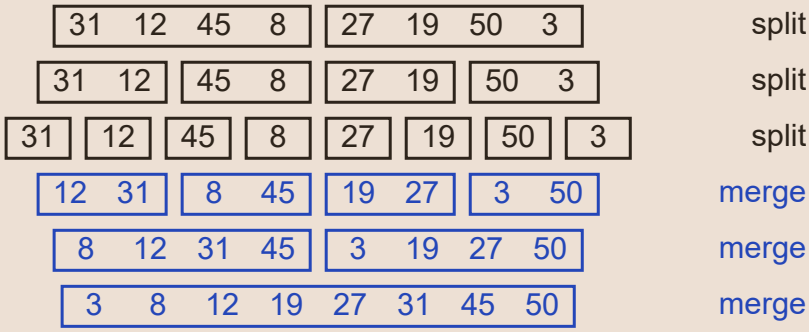
Show the stages of a merge sort

3 to 4 marks **Grade 6**

1. Show the splits until every list has one item.
2. Merge neighbouring pairs, putting each merged list in order.
3. Check that every row holds the same items as the original list.

Example. Show how a merge sort puts this list into ascending order.
[31, 12, 45, 8, 27, 19, 50, 3]

Model answer.




```
[31, 12, 45, 8] [27, 19, 50, 3]
[31, 12] [45, 8] [27, 19] [50, 3]
[31] [12] [45] [8] [27] [19] [50] [3]
[12, 31] [8, 45] [19, 27] [3, 50]
[8, 12, 31, 45] [3, 19, 27, 50]
[3, 8, 12, 19, 27, 31, 45, 50]
```

Splits down to single items (1); first merge into ordered pairs (1); second merge into two ordered lists of four (1); final sorted list (1).

Discuss which sort is more suitable

4 to 6 marks **Grade 7**

1. Give an advantage and a disadvantage of each algorithm.
2. Link each one to the scenario: how many items, how much memory, whether the list is nearly sorted.
3. Finish with a justified choice.

Example. A sports club needs to sort the names of its 40 000 members into alphabetical order. The club's computer has plenty of memory.
Discuss whether a bubble sort or a merge sort is more suitable.

Model answer. A merge sort makes far fewer comparisons than a bubble sort on a large list (1), so it will sort 40 000 names much more quickly (1). A merge sort needs extra memory for the sublists (1), but the computer has plenty of memory, so this is not a problem (1). A bubble sort is simpler to write and needs almost no extra memory (1), but it would be very slow with this many names, so the merge sort is more suitable (1).

Write code for a bubble sort 4 to 6 marks **Grade 8**

1. Use an outer loop that repeats passes until a pass makes no swaps.
2. Use an inner loop that compares each adjacent pair, stopping one before the end.
3. Swap using a temporary variable, and set the flag when a swap is made.

Example. The list `scores` holds integers. Write Python code that uses a bubble sort to put `scores` into ascending order.

Model answer.

```
swapped = True
while swapped:
    swapped = False
    for i in range(len(scores) - 1):
        if scores[i] > scores[i + 1]:
            temp = scores[i]
            scores[i] = scores[i + 1]
            scores[i + 1] = temp
            swapped = True
```

An outer loop that repeats until a pass makes no swaps (1); an inner loop over adjacent pairs that stops at `len(scores) - 1` (1); compares adjacent items with `>` (1); swaps the two items correctly (1); records that a swap was made (1).

Don’t lose marks

- Comparing items that are not next to each other in a bubble sort.
- Stopping a bubble sort as soon as the list looks sorted. The algorithm only stops after a pass with no swaps.
- Merging two lists by just joining them together, instead of comparing their first items to keep the order.
- Losing or duplicating an item while copying out the lists.
- Sorting into ascending order when the question asks for descending order.
- Saying that merge sort uses less memory. It uses more, because of the sublists.

More tips

Memory tricks

- Bubble: the biggest item bubbles to the end on every pass.
- Merge: split down to singles, then zip pairs back together in order.
- n items: n − 1 comparisons per pass in the basic version.
- Check each row of a merge sort: the same number of items as the original list, with nothing lost or repeated.

Exam technique

- Write out the whole list after every pass or merge; each row can earn a mark.
- Use the key terms: pass, adjacent, swap, split, merge, sublist.
- In 'discuss' or 'evaluate' questions, weigh up both algorithms and finish with a choice linked to the scenario.

What each grade needs

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

Grade 4 — **Carry out one pass of a bubble sort**
Compare adjacent pairs from left to right and swap any pair in the wrong order.

Grade 4 — **Describe how a bubble sort works**
Repeat passes of comparing and swapping adjacent items until a pass makes no swaps.

Grade 5 — **Complete a bubble sort and count swaps**
Show the list after every pass and count the comparisons and swaps.

Grade 6 — **Describe how a merge sort works**
Split the list into single items, then merge pairs of sorted lists until one list is left.

Grade 6 — **Show the stages of a merge sort**
Write the lists after each split and after each round of merging.

Grade 7 — **Compare bubble sort and merge sort**
Weigh speed on large lists against memory use and simplicity.

Grade 7 — **Explain improvements to a bubble sort**
Stop after a pass with no swaps, and skip the items already in place.

Grade 8 — **Write code for a bubble sort**
Nested loops that compare adjacent items, swap them with a temporary variable and use a flag.

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.