ANSWERS

A5.5 Merge sort

Algorithms and complexity · A level · OCR H446 2.3.1, AQA 7517 4.3.5.2, Eduqas A500QS 1.3 · about 25 min

What this lesson is about

Divide and conquer, a recursive merge sort, why it is O(n log n) in every case, and its O(n) memory cost.

- Divide and conquer
- A trace
- In pseudocode
- Why it is O(n log n)
- The cost: memory

Questions

5 marks in all

1. What is the worst-case time complexity of merge sort? [1 mark]

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

> **Answer:** A. $\log_2 n$ levels of merging, each handling all n items.

2. Why is merge sort O(n log n) even on a list that is already sorted? [1 mark]

- ☐ **A** It still splits down to single items and merges every level back up
- ☐ **B** It checks whether the list is sorted first
- ☐ **C** Sorted lists need no comparisons when merging
- ☐ **D** It is not: on a sorted list it is O(n)

> **Answer:** A. The splitting does not look at the data, so every level of merging still happens.

3. What is the main disadvantage of merge sort compared with bubble sort? [1 mark]

- ☐ **A** It needs O(n) extra memory for the merged lists
- ☐ **B** It is slower on large lists
- ☐ **C** It is not stable
- ☐ **D** It only works on numbers

> **Answer:** A. Merge sort is not in place: merging builds new lists.

4. What does this print? It merges two sorted lists and counts the comparisons. [1 mark]

```
left = [3, 8, 20]
right = [5, 6, 30]
result, i, j, count = [], 0, 0, 0
while i < len(left) and j < len(right):
    count = count + 1
    if left[i] <= right[j]:
        result.append(left[i])
        i = i + 1
    else:
        result.append(right[j])
        j = j + 1
result = result + left[i:] + right[j:]
print(result, count)
```

> **Answer:**
>
> ```
> [3, 5, 6, 8, 20, 30] 5
> ```
>
> Five comparisons place 3, 5, 6, 8 and 20; then left is empty and 30 is copied across with no comparison.

5. Merge sort splits a list of 32 items in half repeatedly until every piece has one item. How many levels of splitting are there? [1 mark]

> **Answer:** 5. 32, 16, 8, 4, 2, 1: five halvings, $\log_2 32 = 5$.

The task: merge sort the readings

Write `merge_sort(items)` recursively (it must call itself; do not use `sort` or `sorted`). It takes a list of numbers and **returns a tuple** `(sorted_list, comparisons)`: a new list in ascending order, and the number of times two items were compared while merging. Compare with `<=` so the sort is stable; copying the leftovers at the end of a merge is not a comparison. Print exactly three lines: - `sorted:` followed by the readings in ascending order - `readings: <n> comparisons`, for sorting `readings` - `ordered: <n> comparisons`, for sorting `ordered`, which is already in order

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

readings = [57, 16, 34, 26, 51, 15, 31, 35, 26, 34, 16, 31, 48, 22, 40, 9]
ordered = list(range(1, 17))

def merge_sort(items):
    return items, 0
```

The hint students can ask for: Split the list at the middle, sort each half with a call to merge_sort, and add up the comparisons both calls report. Then merge: count one comparison each time you compare the two front items, and none for copying what is left once one list runs out.

A solution

```
from bugbot import *
connect()

readings = [57, 16, 34, 26, 51, 15, 31, 35, 26, 34, 16, 31, 48, 22, 40, 9]
ordered = list(range(1, 17))

def merge_sort(items):
    if len(items) <= 1:
        return items, 0
    mid = len(items) // 2
    left, a = merge_sort(items[:mid])
    right, b = merge_sort(items[mid:])
    result, i, j, count = [], 0, 0, 0
    while i < len(left) and j < len(right):
        count = count + 1
        if left[i] <= right[j]:
            result.append(left[i])
            i = i + 1
        else:
            result.append(right[j])
            j = j + 1
    return result + left[i:] + right[j:], a + b + count

result, count = merge_sort(readings)
print("sorted:", result)
print("readings:", count, "comparisons")
print("ordered:", merge_sort(ordered)[1], "comparisons")
```

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