MODULE L STANDARD ALGORITHMS · L6

Comparing the search and sort algorithms: mark scheme

22 marks · OCR J277 spec 2.1.3 (binary and linear search; bubble, merge and insertion sort: main steps, pre-requisites, apply to a data set, identify from code)

How to use. 1 mark per bullet up to the maximum shown. **/** separates alternative answers. Words in (brackets) are not needed for the mark. **...** means the mark depends on the point above it. Credit any other valid answer that shows the same understanding. Throughout: "it's faster" / "it's quicker" alone is not enough (NE); the answer must say faster than what, or for what kind of list. Candidates do not need to remember code for these algorithms (spec 2.1.3).

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) J277 2.1.3 Binary and linear search: main steps; pre-requisites	Not sorted: Linear search only Middle item: Binary search only Ascending order: Both	**3**	1 mark per correct row. More than one tick in a row = 0 for that row.
2(a) J277 2.1.3 Bubble, insertion and merge sort: main steps	Most extra memory / creates new lists: Merge sort Stops early after a pass with no swaps: Bubble sort Inserts each item into the correct place among the items before it: Insertion sort	**3**	1 mark per correct row. More than one tick in a row = 0 for that row.
3(a) J277 2.1.3 Identify an algorithm from Exam Reference Language	Binary search	**1**	Accept "binary" alone. Do not accept linear search, or a sorting algorithm.
3(b) J277 2.1.3 Pre-requisites of an algorithm	The array / data must be sorted / in order (ascending)	**1**	Accept "in numerical order". Do not accept "must contain the target" or "must not be empty".
3(c) J277 2.1.3 Main steps and segments of code	(Line) 05	**1**	Accept 5. Accept the line written out (`mid = (low + high) DIV 2`) instead of its number.
4(a) J277 2.1.3 Linear search: apply to a data set	13	**1**	58 is at position 12 (the 13th item). Do not accept 12.

QUESTION	ANSWER	MARKS	GUIDANCE
4(b) J277 2.1.3 Binary search: apply to a data set	Compares 58 with 31 (the middle item); 58 is larger so 31 and the items before it (3 to 31) are discarded Compares 58 with 52 (middle of 36 to 70); 58 is larger so 36 to 52 are discarded Compares 58 with 63 (middle of 58, 63, 70); 58 is smaller so 63 and 70 are discarded... ...then compares with 58: found / stops	**3**	1 mark per bullet, max 3. The comparisons must be in this order: 31, 52, 63, 58. Discarding may be shown by listing the items that remain at each step (36 ... 70, then 58 63 70, then 58). Every sub-list here has an odd number of items, so there is no rounding choice. Do not award marks for steps that compare with items in a different order (e.g. a linear search).
4(c) J277 2.1.3 Linear and binary search: main steps; apply to a data set	A linear search starts at the first item / 8 is the second item / 8 is near the start of the list... ...so it needs only 2 comparisons (3, then 8) A binary search starts at the middle item (31)... ...so it needs 3 comparisons (31, 15, 8)	**2**	1 mark per bullet, max 2. Accept a correct comparison of the counts (2 against 3) for 1 mark. "Linear search is faster" alone = NE.
5(a) J277 2.1.3 Pre-requisites; linear and binary search	The list is not sorted / a binary search needs the list to be sorted / in order The list would have to be sorted (again) every time a badge number is added / removed (taking extra time) The list is small / only 25 items, so a linear search is quick enough / there is little time difference A linear search is simpler / easier to code	**2**	1 mark per bullet, max 2. Credit any other valid reason. Do not accept "a linear search is faster" (it is usually slower on large sorted lists).
6(a) J277 2.1.3 Merge sort: main steps (comparing sorting algorithms)	Merge sort is usually the quickest / most efficient of the three sorts on large lists / 2 million readings / needs fewer comparisons than bubble or insertion sort on large lists It needs extra memory (for the new lists), but the server has plenty of spare memory so this is not a problem	**2**	1 mark per bullet. "Merge sort is fast" alone = NE; accept "faster than bubble / insertion sort" or "fast on large lists". The memory point must link to the server's spare memory.
6(b) J277 2.1.3 Bubble, insertion and merge sort: main steps (recommend for a scenario)	**Recommendation (1):** insertion sort / bubble sort **Justification (max 2):** It needs little / no extra memory / sorts within the original list, which suits the sensor's small memory Insertion sort: quick / efficient on small lists or lists that are nearly in order (few items need to move) Bubble sort: if the readings are already in order, the first pass makes no swaps so it can stop after one pass Only 20 items, so the speed advantage of merge sort on large lists is not needed Simple to code Merge sort would need extra memory that the sensor does not have	**3**	Either insertion sort or bubble sort earns the recommendation mark. Justification must suit the algorithm chosen (e.g. the "no swaps" point only for bubble sort). If merge sort is recommended, 0 marks for the whole part. Justification with no recommendation: max 2.

Total: 22 marks