OCR GCSE (J277)

# Module L: Standard algorithms

Workbooklet mark scheme · 7 parts · 146 marks

## Contents


| Part | Topic | Marks |
|---|---|---|
| L1 | Linear search: mark scheme | 21 |
| L2 | Binary search: mark scheme | 21 |
| L3 | Bubble sort: mark scheme | 21 |
| L4 | Merge sort: mark scheme | 20 |
| L5 | Insertion sort: mark scheme | 20 |
| L6 | Comparing the search and sort algorithms: mark scheme | 22 |
| L7 | Totalling, counting, maximum, minimum and average: mark scheme | 21 |
| | **Total** | **146** |

MODULE L STANDARD ALGORITHMS · L1

# Linear search: mark scheme

21 marks · OCR J277 spec 2.1.3 (searching algorithms: linear search), with 2.1.2 (trace tables)

**How to use.** 1 mark per bullet up to the Max shown. **/** separates alternative answers. Words in (brackets) are not needed for the mark. Students are not expected to remember the code for a linear search (J277 2.1.3 "Not required"); Q4 gives the code.

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **1(a)** J277 2.1.3 | Starts by checking the first item: True<br>Data must be sorted first: False<br>Can stop as soon as the item is found: True<br>Checks the middle item first: False | **4** | 1 mark per correct row. Two ticks in a row: 0 for that row. Row 2: linear search has no pre-requisite; it works on unsorted data. Row 4 describes a binary search. |
| **2(a)** J277 2.1.3 | Start at the first item (in the list)<br>Compare the (current) item with the item being searched for / target<br>If they match, the item is found / stop (and give its position)<br>If not, move to the next item (in order)<br>Repeat until the item is found / until the end of the list is reached<br>If the end of the list is reached with no match, the item is not in the list / output not found | **3** | Max 3. Steps must be described, not just the result: "it looks through the list" alone = 0. Do not accept steps of a binary search (middle item, discard half). Credit any other valid step of a linear search. |
| **3(a)** J277 2.1.3 | P41, C08, R17 (compared first, in this order) ...<br>... then D55, and no further codes | **2** | Order must be correct. Listing any code after D55 (e.g. the second C08): max 1. Accept indexes 0, 1, 2, 3 if clearly linked to the codes. |
| **3(b)** J277 2.1.3 | 3 | **1** | Do not accept 4 (that is the position counting from 1, not the index). |
| **3(c)** J277 2.1.3 | 1 | **1** | The search stops at the first match, so index 4 is never reached. Do not accept 4, or "1 and 4". |
| **4(a)** J277 2.1.3 | Linear search | **1** | Accept "linear" by itself. Do not accept "linear sort" or "binary search". |
| **4(b)** J277 2.1.3 | (Line) 06 / 6 | **1** | Do not accept line 05 (it compares `i` and `found`, not an item with the target). |
| **4(c)** J277 2.1.3 | To record / show whether the target has been found (yet)<br>To stop the loop / search once the target is found | **1** | Max 1. Accept "it is a flag (that shows if the sensor ID was found)". Do not accept "to store the sensor ID" / "to store the index". |

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **4(d)** J277 2.1.3, 2.1.2 | `i`: 0, 1, 2 (in order)<br>`found`: False, True (in order)<br>Output: Found at index 2 | **3** | 1 mark per correct column. Values may be repeated on later rows. Line numbers are not marked. A blank cell means no change. Accept 0/1 or F/T for False/True. Output: ignore quotation marks; do not accept extra outputs. Example: line 03 found False; 04 i 0; 09 i 1; 09 i 2; 07 found True; 13 output Found at index 2. |
| **4(e)** J277 2.1.3 | Not found | **1** | Ignore quotation marks. Do not accept "Found at index 5" or any index. |
| **5(a)** J277 2.1.3 | 50 000 | **1** | Accept "all of them / every record" only with the number 50 000 or clear equivalent (all 50 000). |
| **5(b)** J277 2.1.3 | The record being searched for is the last one in the list / at the end<br>The record being searched for is not in the list (so every record is checked before reporting not found) | **2** | 1 mark each. Do not accept "the list is not sorted" (linear search checks items in order whether sorted or not) or "the list is very big" alone. |

**Total: 21 marks**

MODULE L STANDARD ALGORITHMS · L2

# Binary search: mark scheme

21 marks · OCR J277 spec 2.1.3 (searching algorithms: main steps, segments of code, pre-requisites, apply to a data set, identify from code) and 2.1.2 (trace tables)

**How to use.** 1 mark per bullet up to the Max shown. **/** separates alternative words; **//** separates alternative answers for the same mark. Words in (brackets) are not needed for the mark. **...** at the start of a bullet means it depends on the bullet before it. Candidates are not required to remember the code for a binary search (J277 2.1.3).

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **1(a)** J277 2.1.3 | 4, 9, 15, 22, 31, 48: Yes<br>"Ash", "Elm", "Fir", "Oak", "Pine", "Yew": Yes<br>12, 5, 18, 3, 27, 30: No | **3** | 1 mark per correct row. Two ticks in a row: 0 for that row. Row 2 is in alphabetical order, so it is sorted. Row 3 is not sorted. |
| **1(b)** J277 2.1.3 | The list / data must be sorted / in order | **1** | Accept: in ascending / descending / alphabetical / numerical order. Do not accept "the list must be organised" or "the item must be in the list". |
| **2(a)** J277 2.1.3 | 167 is the (first) middle item checked / compared with 205<br>205 is greater than 167, so 167 and the items to its left / 103 to 167 are discarded // search continues with 171, 186, 194, 205, 219<br>194 is the next middle item; 205 is greater, so 171, 186 (and) 194 are discarded // search continues with 205, 219<br>205 is the next middle item ... it matches / equals the target, so it is found / the search stops | **4** | Max 4. Accept the stages shown as a series of lists with the middle item marked. Do not award a discard mark if the wrong half is discarded (FT not allowed for the half kept). Allow the item to the right of centre if used consistently: 167, then 194, then 219 (205 < 219, so 219 discarded), then 205 found. Award the same four points in this order.<br>Do not credit a description of a linear search (checking each item in turn). |
| **2(b)(i)** J277 2.1.3 | 167, 125 (in this order)<br>... 140 (last, and nothing else) | **2** | Correct answer: 167, 125, 140. 2 marks for all three in this order and no others; 1 mark for 167, 125 in order. Allow 167, 125, 152, 140 (taking the item right of centre in a part-list of 2) for 2 marks. |

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>2(b)(ii)<br>J277 2.1.3</td><td>There are no items left to check // the part of the list left to search is empty // the lower pointer / position passes the upper pointer / position</td><td>1</td><td>Accept: after 140 is discarded nothing is left between 125 and 140. Do not accept "it reaches the end of the list" (that is a linear search) or "130 is not found" alone.</td></tr>
<tr><td>3(a)<br>J277 2.1.3</td><td>Binary search<br>... it finds the middle position / item (line 05) // it compares the middle item with the target (lines 06 / 08) // it moves <code>low</code> or <code>high</code> to discard half of the array (lines 09 / 11) // it repeats while <code>low <= high</code></td><td>2</td><td>Reason mark only if binary search is identified. A line number alone is not enough; the reason must say what the code does. Do not accept "the array is sorted" (that is a pre-requisite, not something in the code) or "it searches the array".</td></tr>
<tr><td>3(b)<br>J277 2.1.3<br>J277 2.1.2</td><td><table><tr><th>LOW</th><th>HIGH</th><th>MID</th><th>VALUE RETURNED</th></tr><tr><td>0</td><td>7</td><td>3</td><td></td></tr><tr><td></td><td>2</td><td>1</td><td></td></tr><tr><td>2</td><td></td><td>2</td><td>2</td></tr></table><code>low</code>: 0, 2<br><code>high</code>: 7, 2<br><code>mid</code>: 3, 1, 2<br>Value returned: 2 (only)</td><td>4</td><td>1 mark per column: each variable's values in the correct order. Ignore the row a value is written on and ignore repeated values on later rows. Value returned: 2 (the index), not 23. Do not award the last mark if -1 is also returned.</td></tr>
<tr><td>3(c)<br>J277 2.1.3</td><td>It discards the lower half / the middle item and everything before it (because the target is larger than the middle item) // the next search starts just after the middle position</td><td>1</td><td>Accept "moves the start of the search to after mid". Do not accept "adds 1 to low" alone (repeats the code without saying why).</td></tr>
<tr><td>3(d)<br>J277 2.1.3</td><td>The target / serial number is not in the array / was not found</td><td>1</td><td>Accept: "-1 is not a valid index, so it shows the number is not stored". Do not accept "an error has happened".</td></tr>
<tr><td>4(a)<br>J277 2.1.3</td><td>Binary search halves the number of records left / discards half of the remaining records with each comparison<br>Linear search checks the records one at a time / in turn (from the start)<br>... so a linear search may have to check all / most of the 100 000 records<br>Binary search needs at most about 17 comparisons for 100 000 records</td><td>2</td><td>Max 2. Do not accept "binary search is faster / quicker" alone (it repeats the question). Do not accept "binary search starts in the middle" alone without the halving. Credit any other valid point. The number 17 is not needed; accept "about 16 or 17".</td></tr>
</table>

**Total: 21 marks**

MODULE L STANDARD ALGORITHMS · L3

# Bubble sort: mark scheme

21 marks · OCR J277 spec 2.1.3 (searching and sorting algorithms: bubble sort) and 2.1.2 (trace tables)

**How to use.** 1 mark per bullet up to the Max shown. **/** separates alternative answers. Words in (brackets) are not needed for the mark. **...** means the mark depends on the bullet before it. J277 does not require students to remember bubble sort code: credit understanding of the steps and of the given code. Code answers may be in OCR Exam Reference Language or any high-level language; ignore minor syntax slips.

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **1(a)** J277 2.1.3 | Compares items next to each other: True<br>Only used if data already partly in order: False<br>Largest value in final position after first pass: True<br>Splits the list into sub-lists of one item: False | **4** | 1 mark per correct row. Both boxes ticked in a row: 0 for that row. Row 2 tests the pre-requisite: a bubble sort has none (it works on unsorted data). Row 4 describes merge sort. |
| **2(a)** J277 2.1.3 | After pass 1: 7, 18, 12, 9, 25, 30<br>After pass 2: 7, 12, 9, 18, 25, 30<br>After pass 3: 7, 9, 12, 18, 25, 30 | **3** | 1 mark per correct row. Follow through: credit pass 2 or pass 3 if it is one correct bubble sort pass of the student's own previous row. Do not credit a row that is simply the fully sorted list unless it follows from the previous row (e.g. no marks for writing 7, 9, 12, 18, 25, 30 in every row). |
| **2(b)** J277 2.1.3 | The algorithm only stops when a (whole) pass makes no swaps<br>Pass 3 made a swap (9 and 12) / swaps were made in pass 3 ...<br>... so it cannot tell / does not know the list is sorted until it checks again<br>Pass 4 makes no swaps, so the sort then stops / ends | **2** | Max 2. Do not accept "to check it is sorted" alone for 2 marks (1 mark only if no other point is made). Do not accept "it always does one pass per item" or "it does a fixed number of passes": the stem states it stops when a pass makes no swaps. |
| **3(a)** J277 2.1.3 | Compares adjacent items / each item with the next one / `prices[i]` with `prices[i + 1]` (line 04)<br>Swaps the two items if they are in the wrong order (lines 05 to 07)<br>Goes through the whole list in a pass / inner (for) loop through the list (line 03)<br>Repeats passes until a pass makes no swaps / uses a flag (swapped) to stop (lines 01, 02, 08, 11) | **2** | Max 2. Line numbers are not needed. Do not accept "it sorts the prices" or "it puts them in order" (outcome, not a feature). Do not accept "it uses a loop" alone. |

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>3(b)<br>J277 2.1.3</td><td>Records / shows whether a swap has been made (during a pass)<br>Set to false at the start of each pass and to true when a swap happens<br>Used to decide when to stop / the loop repeats until a pass makes no swaps / it stays false when the list is sorted</td><td>2</td><td>Max 2. Do not accept "to swap the items" (that is <code>temp</code>'s job) or "it counts the swaps".</td></tr>
<tr><td>3(c)<br>J277 2.1.3 / 2.1.2</td><td><code>if prices[i] < prices[i + 1] then</code></td><td>1</td><td>Accept <code>if prices[i + 1] > prices[i] then</code>. Accept the comparison written in any high-level language. Allow a missing <code>then</code>. Do not accept <= (equal prices would be swapped every pass, so the loop would never end if two prices are the same). Do not accept "change > to <" with no line written: the question asks for the line.</td></tr>
<tr><td>4(a)<br>J277 2.1.2 / 2.1.3</td><td>1 mark: <code>i</code> takes 0, 1, 2 (in order)<br>1 mark: <code>temp</code> takes 14 then 21<br>1 mark: <code>swaps</code> takes 0, 1, 2 (in order)<br>1 mark: Output 2 (once, at the end)<br><table><tr><th>i</th><th>temp</th><th>swaps</th><th>Output</th></tr><tr><td></td><td></td><td>0</td><td></td></tr><tr><td>0</td><td>14</td><td>1</td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td>21</td><td>2</td><td></td></tr><tr><td></td><td></td><td></td><td>2</td></tr></table></td><td>4</td><td>Mark each column as a sequence of values; rows may be laid out differently. Values may be repeated on later rows (e.g. swaps 1 repeated when i = 1). Do not credit <code>i</code> = 3. Do not credit <code>temp</code> = 9 or 6. Output must appear once only; "x" or "-" in the Output column counts as an output.</td></tr>
<tr><td>4(b)<br>J277 2.1.3</td><td>9, 14, 6, 21</td><td>1</td><td>All four in this order. Accept with or without brackets / commas. Do not accept the fully sorted list 6, 9, 14, 21 (only one pass is made).</td></tr>
<tr><td>4(c)<br>J277 2.1.3</td><td>Line 06 overwrites <code>speeds[i]</code> / the first item is overwritten (by the second)<br>... so its original value would be lost / both positions would end up holding the same value<br><code>temp</code> keeps a copy of the first item ...<br>... so it can be put into <code>speeds[i + 1]</code> / the second position (line 07)</td><td>2</td><td>Max 2. Credit an example using the data, e.g. "without temp, 14 would be lost and both would be 9". Do not accept "to store a temporary value" alone, or "to make the swap faster".</td></tr>
</table>

**Total: 21 marks**

MODULE L STANDARD ALGORITHMS · L4

# Merge sort: mark scheme

20 marks · OCR J277 spec 2.1.3

**How to use.** 1 mark per bullet up to the maximum shown. **/** separates alternative answers; **//** separates alternative statements for the same mark. Words in (brackets) are not needed for the mark. **...** means the mark depends on the one before. Credit any other valid answer that shows the same understanding.
**Spec points (J277 2.1.3).** Standard sorting algorithms: merge sort. Understand the main steps of the algorithm and the segments of code in it; apply the algorithm to a data set; identify an algorithm if given the code or Exam Reference Language for it. Candidates do **not** need to remember the code for merge sort: Q5 gives the code and asks only what it does.

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **1(a)** J277 2.1.3 | Only one item, already sorted: True<br>Data must already be in order: False<br>Joined end to end, then sorted: False | **3** | 1 mark per correct row. Two ticks in a row = 0 for that row. Row 2: being in order first is a pre-requisite of binary search, not merge sort. Row 3: the merging itself keeps the items in order; nothing is sorted afterwards. |
| **2(a)** J277 2.1.3 | (Repeatedly) split / divide the list (in half) ... until each list has one item<br>(Repeatedly) merge pairs of lists (back together) in order // compare the first items of two lists and move the smaller into a new list ... until there is one (sorted) list | **2** | 1 mark per stage. Each stage needs its stopping point (one item each // one list) **or** the idea of repeating. Accept "divide" and "conquer" as names only if the description is given. "Split the list" alone = NE. "Merge the lists" alone = NE (must be in order / sorted). Do not accept merging and then sorting the merged list. |

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **3(a)**<br>J277 2.1.3 | Split into individual lists of one item each: 9 \| 27 \| 3 \| 14 \| 21 \| 5 \| 11 \| 18<br>Merge into sorted lists of 2: [27, 9] [14, 3] [21, 5] [18, 11]<br>Merge into sorted lists of 4: [27, 14, 9, 3] [21, 18, 11, 5]<br>... Merge into one list: [27, 21, 18, 14, 11, 9, 5, 3] | **4** | 1 mark per bullet. Splitting may be shown in stages (4 + 4, then 2 + 2 + 2 + 2, then 1s) or all at once. If no splitting is shown, assume the list starts as single items and mark BP2 to BP4. Lists must clearly be the right size (boxes, brackets or gaps). Final mark only after a reasonable attempt at the merges; the sorted list on its own = 0. Do not award a merge mark where lists are merged out of order and then sorted. A different algorithm (e.g. bubble sort) = 0. If every merge is correct but in ascending order, penalise once: max 3. |
| **4(a)**<br>J277 2.1.3 | 2: Heron and Crane ... Crane<br>3: Heron and Fox ... Fox<br>4: Heron and Stoat ... Heron | **3** | 1 mark per row: both items compared **and** the item moved must be correct. The two items may be written in either order. Rule: compare the front item of each list and move the one that comes first alphabetically. |
| **4(b)**<br>J277 2.1.3 | Badger, Crane, Fox, Heron, Otter, Stoat, Vole, Wren | **1** | All eight in this order. Ignore capitals and the type of separator. |
| **5(a)**<br>J277 2.1.3 | Merge sort | **1** | Do not accept bubble sort, insertion sort, or "merge" alone without "sort" being clear (BOD "a merge" if the candidate names the sort elsewhere). |
| **5(b)**<br>J277 2.1.3 | They run when one of the lists has run out / all its items have been moved // the first loop has ended<br>... they copy / move the remaining items of the other list onto the end of `result`<br>The remaining items are already in order, so no comparisons are needed // they keep their order | **2** | 1 mark per bullet, max 2. "Copies the rest of left, then the rest of right" = 1 mark (BP2). "Adds the leftover items" = BP2. "Loops through the lists" alone = 0. Only one of the two loops actually runs; do not penalise a candidate who does not say so. |
| **5(c)(i)**<br>J277 2.1.3 | [2, 4, 9, 10, 15] | **1** | All five values in this order. Ignore brackets, quotes and separators. |
| **5(c)(ii)**<br>J277 2.1.3 | 4 | **1** | Correct answer only. (4 v 2, 4 v 10, 9 v 10, 15 v 10; then `right` is used up and 15 is copied by lines 16 to 20 without a comparison.) |

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **6(a)**<br>J277 2.1.3 | Quicker / more efficient (than bubble sort) for a large list // takes fewer comparisons / steps on a large list<br>Time taken grows much more slowly than bubble sort's as the list gets longer<br>Its speed does not depend much on the starting order of the list // always takes about the same time | **1** | Any one. "Faster" alone is BOD here because the question states the list is large. Do not accept "uses less memory" (wrong) or "it sorts the list" (true of both). |
| **6(b)**<br>J277 2.1.3 | Uses more memory // needs extra space for the split / merged lists // does not sort in place<br>More complex / harder to write (code) // harder to understand / debug<br>Can be slower on a small list or a list that is already (nearly) in order | **1** | Any one. Do not accept "slower" alone, or "slower for large lists" (wrong). Credit any other valid answer. |

**Total: 20 marks**

MODULE L STANDARD ALGORITHMS · L5

# Insertion sort: mark scheme

20 marks · OCR J277 spec 2.1.3 (searching and sorting algorithms: insertion sort)

**How to use.** 1 mark per bullet up to the Max shown. **/** separates alternative answers. Words in (brackets) are not needed for the mark. Candidates do not need to remember the code for insertion sort (J277 2.1.3): credit understanding of the steps, not recall of code. Accept answers that describe swapping the item left one place at a time instead of shifting the larger items right: both are valid ways of inserting the item.

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **1(a)** J277 2.1.3 | A sorted part (at the start / left) **and** an unsorted part (the rest) | **1** | Both parts needed for the mark. Accept sub-list / section / partition for part. Do not accept "left half and right half" (the parts are not halves). |
| **1(b)** J277 2.1.3 | The first item is treated as sorted / starts as the sorted part<br>Take the next item (from the unsorted part) / start from the second item<br>Compare it with the items in the sorted part (working from right to left / from the end of the sorted part)<br>Move / shift each larger item one place to the right<br>Insert the item into the gap / its correct position (when a smaller or equal item, or the start of the list, is reached)<br>Repeat for every remaining item / until the unsorted part is empty / all items are inserted | **3** | Max 3. Accept "swap it with the item to its left while that item is larger" for the shift and insert points. Do not accept "it sorts the list" / "it puts the items in order" alone (outcome, not steps). Do not accept descriptions of comparing adjacent pairs through the whole list (bubble sort) or splitting the list (merge sort). |
| **2(a)** J277 2.1.3 | After insertion 1: 18, 31, 42, 7, 25<br>After insertion 2: 18, 31, 42, 7, 25<br>After insertion 3: 7, 18, 31, 42, 25<br>After insertion 4: 7, 18, 25, 31, 42 | **4** | 1 mark per correct row. Row 2 is unchanged because 42 is already larger than every item in the sorted part. Follow through: if one row is wrong, award a later row if it correctly inserts the next item into the candidate's own previous row. A row that is just the fully sorted list too early (e.g. row 1 = 7, 18, 25, 31, 42) scores 0 for that row. |
| **3(a)** J277 2.1.3 | Iris | **1** | The first item of the unsorted part. |

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **3(b)** J277 2.1.3 | Iris is compared with Mint, which comes after it (alphabetically) so Mint moves one place right / Iris and Mint swap<br>Iris is then compared with Fern, which comes before it, so the comparisons stop<br>Iris is inserted between Fern and Mint / list becomes Aster, Fern, Iris, Mint (Basil, Tulip) | **2** | Max 2. Must refer to the items in this list for full marks; a correct generic description of inserting an item: Max 1. Follow through a wrong answer to 3(a) if the description correctly inserts that item. |
| **3(c)** J277 2.1.3 | Aster, Basil, Fern, Iris, Mint, Tulip | **1** | All six items in this order. Separators (commas / spaces) not important. |
| **4(a)** J277 2.1.3 | Insertion sort | **1** | Do not accept bubble sort or merge sort. |
| **4(b)** J277 2.1.3 | (Line) 09 / 9 | **1** | Line number only. Accept the line copied out correctly. |
| **4(c)** J277 2.1.3 | Loops while the item to the left (`times[j]`) is greater than `current` / and the start of the list has not been passed (j is not below 0)<br>Moves / shifts / copies that item one place to the right<br>Moves left / decreases `j` by 1 to check the next item in the sorted part<br>(So that) a gap is made / found for `current` in its correct position | **2** | Max 2. Do not accept "sorts the list" alone. Do not accept "swaps" pairs of items through the whole list (bubble sort). Do not accept "moves the item to the left" for line 06 (the larger item moves right). |
| **4(d)** J277 2.1.3 (J277 2.1.2 trace tables) | `i`: 1, 2, 3<br>`current`: 35, 52, 29<br>`j`: 0, −1, 1 (while i is 1 and 2)<br>`j`: 2, 1, 0, −1 (while i is 3)<br>Example rows (i, current, j): (1, 35, 0); ( , , −1); (2, 52, 1); (3, 29, 2); ( , , 1); ( , , 0); ( , , −1) | **4** | 1 mark per bullet: each variable's values in the correct order. Values repeated on later rows are fine; row positions and line numbers are not marked. 52 is not moved, so j is only 1 while i is 2. Do not accept 0 in place of −1 for the final j values. Ignore a correct extra column showing the contents of times. |

**Total: 20 marks**

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<br>Middle item: Binary search only<br>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<br>Stops early after a pass with no swaps: Bubble sort<br>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)**<br>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<br>Compares 58 with 52 (middle of 36 to 70); 58 is larger so 36 to 52 are discarded<br>Compares 58 with 63 (middle of 58, 63, 70); 58 is smaller so 63 and 70 are discarded…<br>…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)**<br>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…<br>…so it needs only 2 comparisons (3, then 8)<br>A binary search starts at the middle item (31)…<br>…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)**<br>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<br>The list would have to be sorted (again) every time a badge number is added / removed (taking extra time)<br>The list is small / only 25 items, so a linear search is quick enough / there is little time difference<br>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)**<br>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<br>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)**<br>J277 2.1.3 Bubble, insertion and merge sort: main steps (recommend for a scenario) | **Recommendation (1):** insertion sort / bubble sort<br>**Justification (max 2):**<br>It needs little / no extra memory / sorts within the original list, which suits the sensor's small memory<br>Insertion sort: quick / efficient on small lists or lists that are nearly in order (few items need to move)<br>Bubble sort: if the readings are already in order, the first pass makes no swaps so it can stop after one pass<br>Only 20 items, so the speed advantage of merge sort on large lists is not needed<br>Simple to code<br>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**

MODULE L STANDARD ALGORITHMS · L7

# Totalling, counting, maximum, minimum and average: mark scheme

21 marks · OCR J277 spec 2.1.2 (creating algorithms, common errors, trace tables), 2.2.1 (variables, operators, inputs, outputs, assignment, iteration, selection), 2.2.3 (arrays)

**How to use.** 1 mark per bullet up to the Max shown. **/** separates alternative answers. Words in (brackets) are not needed for the mark. Code answers: mark the logic, not the language. Allow OCR Exam Reference Language or any high-level language, and allow any logically valid method. Do not penalise minor syntax slips (missing casting, capitals, a missing colon). A recurring slip: penalise once, then follow through. Do not accept maths symbols such as ×, ≥, ≤ in code.

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>1(a)<br>J277 2.2.1, 2.2.3</td><td>Line 01: <code>0</code><br>Line 03: +<br>Line 05: <code>sales.length / 12</code></td><td>3</td><td>Line 05: allow <code>len(sales)</code> or any expression that gives 12. Do not accept <code>i</code>, <code>11</code> or <code>sales.length - 1</code>.</td></tr>
<tr><td>2(a)<br>J277 2.1.2, 2.2.1</td><td><code>day</code>: 1, 2, 3, 4<br><code>wettest</code>: 4, 12, 15<br><code>dryDays</code>: 0, 1<br>Output: 15 then 1<br><table><tr><th>DAY</th><th>WETTEST</th><th>DRYDAYS</th><th>OUTPUT</th></tr><tr><td></td><td>4</td><td>0</td><td></td></tr><tr><td>1</td><td>12</td><td></td><td></td></tr><tr><td>2</td><td></td><td>1</td><td></td></tr><tr><td>3</td><td>15</td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>15</td></tr><tr><td></td><td></td><td></td><td>1</td></tr></table></td><td>4</td><td>1 mark for each column with the correct values in the correct order. Ignore the row a value is on, and allow values to be repeated on later rows. Do not award the <code>day</code> mark if 0 or 5 appears. Outputs written as <code>print(15)</code> or in quotation marks: do not award the Output mark.</td></tr>
<tr><td>2(b)<br>J277 2.1.2</td><td>The loop starts at index 1 / <code>rain[0]</code> is never checked (to see if it is 0) / the first day is missed (and it was a dry day)<br><code>for day = 0 to 4</code></td><td>2</td><td>Allow <code>for day = 0 to rain.length - 1</code>. Credit any other refinement that also checks <code>rain[0]</code> for the second mark. Do not accept "the loop is wrong" alone for the first mark.</td></tr>
</table>

| QUESTION | ANSWER | MARKS | GUIDANCE |
|---|---|---|---|
| **3(a)**<br>J277 2.1.2, 2.2.1, 2.2.3 | 1 mark each to max 4:<br>Sets a variable for the smallest to `heights[0]` / the first value / a value larger than any height (e.g. 1000) before the loop<br>Loop that visits every item (index 0 to 29 / `heights.length - 1`; starting at 1 is allowed only if `smallest` was set to `heights[0]`)<br>Compares each item with the smallest so far (`heights[i] < smallest`)<br>... and replaces the smallest with that item when it is smaller<br>Outputs the smallest after the loop<br>`smallest = heights[0]`<br>`for i = 1 to 29`<br>`  if heights[i] < smallest then`<br>`  smallest = heights[i]`<br>`  endif`<br>`next i`<br>`print(smallest)` | **4** | Do not award the first bullet for starting at 0 (no height is smaller than 0). A loop of 0 to 29 is fine. Start value of 1000 (or similar) with a loop starting at 1: do not award the loop bullet, max 3. Output inside the loop: do not award the last bullet. Use of `min()` or a built-in sort: max 1 (for the output). Allow <= for the comparison. |
| **4(a)**<br>J277 2.1.2, 2.2.1 | 0 is higher than every temperature (in the array) so `highest` is never replaced / the program outputs 0, which is not one of the temperatures | **1** | Do not accept "0 is the wrong number" or "it is a logic error" alone. |
| **4(b)**<br>J277 2.1.2, 2.2.1 | `highest = temps[0]` | **1** | Allow a starting value lower than any temperature the freezer could record, e.g. `highest = -1000`. Do not accept 0, or a value that could be higher than every reading (e.g. -20). |
| **5(a)**<br>J277 2.1.2, 2.2.1 | 1 mark each to max 6:<br>Inputs the number of repairs and stores it in a variable<br>Loop that repeats once for each repair (uses the number input)<br>Inputs a time inside the loop<br>Total set to 0 before the loop and the time added to it inside the loop<br>Counter set to 0 before the loop and increased by 1 inside the loop when time > 45<br>Mean calculated as total / number of repairs (after the loop)<br>Outputs the total, the count and the mean (after the loop)<br>`repairs = int(input("How many repairs today?"))`<br>`total = 0`<br>`over45 = 0`<br>`for r = 1 to repairs`<br>`  time = int(input("Time in minutes"))`<br>`  total = total + time`<br>`  if time > 45 then`<br>`  over45 = over45 + 1`<br>`  endif`<br>`next r`<br>`print("Total time: " + str(total))`<br>`print("Over 45 minutes: " + str(over45))`<br>`print("Mean time: " + str(total / repairs))` | **6** | Count bullet: do not accept `>= 45`. Allow a while loop with its own counter. Allow the mean worked out with a separate count of repairs. Allow uninitialised total/counter to be taken as 0 if the language allows it. Not required: checking for 0 repairs, rounding, messages with the outputs. Loop with a fixed number of repeats (e.g. 10): do not award the loop bullet. |

**Total: 21 marks**