

THEME B: COMPUTATIONAL THINKING · B2.4

Programming algorithms — Mark scheme

35 marks · covers B2.4.1–B2.4.5

How to mark. Each creditworthy point is on its own line and ends with a semicolon (;). A line beginning with ... is an *expansion* of the line above it, so the pair is worth the marks shown. A slash (/) separates acceptable alternative wordings. Words in (brackets) are not needed for the mark. Mark positively: credit what the student has got right rather than penalising what is missing, and accept any valid alternative not listed here. Do not award more than the maximum for a part. Where a later part uses a wrong answer from an earlier part correctly, award follow-through marks (**FT**). **HL** marks parts that assess higher level only content.

Q	ANSWER	MARKS												
1(a) B2.4.1	Award [2] max Award [1] for each. Linear search: $O(n)$; Binary search: $O(\log n)$;	2												
1(b) B2.4.1	Award [3] max Award [1] per point up to [3] max. The time taken grows in proportion to the square of the number of items; ...(so) doubling the amount of data makes the algorithm take roughly four times as long, and ten times the data takes about a hundred times as long; It is usually acceptable on small data sets but becomes impractical as the data grows; An algorithm with lower complexity, such as $O(n \log n)$, should be preferred where the data set may become large; Big O describes the growth rate rather than the exact time, so constant factors and hardware speed are ignored;	3												
2(a) B2.4.2	Award [4] max Award [2] for each fully correct row. Award FT where a row follows correctly from an earlier error. <table><tr><th>LOW</th><th>HIGH</th><th>MID</th><th>VALUES[MID]</th></tr><tr><td>0</td><td>7</td><td>3</td><td>22</td></tr><tr><td>4</td><td>7</td><td>5</td><td>40</td></tr></table> First pass: $\text{mid} = (0 + 7) / 2 = 3$, and $\text{values}[3] = 22$. Since $40 > 22$, the lower half is discarded and low becomes $\text{mid} + 1 = 4$. Second pass: $\text{mid} = (4 + 7) / 2 = 5$, and $\text{values}[5] = 40$, so the value is found at index 5. Accept a third row only if the candidate shows the loop terminating.	LOW	HIGH	MID	VALUES[MID]	0	7	3	22	4	7	5	40	4
LOW	HIGH	MID	VALUES[MID]											
0	7	3	22											
4	7	5	40											
2(b) B2.4.2	Award [1] max 6; A linear search checks indices 0 to 5 inclusive before finding 40.	1												
2(c) B2.4.2	Award [2] max Award [1] for the point and [1] for the expansion. A binary search compares the target with the middle value and then discards one half of the data;	2												

Q	ANSWER	MARKS										
	...(and) it can only know which half to discard if the values are in order; in unsorted data the target could be on either side, so the discarded half might contain it;											
3(a) B2.4.3	Award [4] max Award [1] for each correct row. Award [1] for a row that follows correctly (FT) from an earlier error. <table><thead><tr><th>AFTER PASS</th><th>ARRAY</th></tr></thead><tbody><tr><td>start</td><td>5, 2, 9, 1</td></tr><tr><td>1</td><td>2, 5, 1, 9</td></tr><tr><td>2</td><td>2, 1, 5, 9</td></tr><tr><td>3</td><td>1, 2, 5, 9</td></tr></tbody></table> Pass 1: (5,2) swap, (5,9) no swap, (9,1) swap. Pass 2: (2,5) no swap, (5,1) swap; 9 is already in its final position. Pass 3: (2,1) swap. Accept an answer that stops after pass 3, and accept an extra pass with no swaps shown.	AFTER PASS	ARRAY	start	5, 2, 9, 1	1	2, 5, 1, 9	2	2, 1, 5, 9	3	1, 2, 5, 9	4
AFTER PASS	ARRAY											
start	5, 2, 9, 1											
1	2, 5, 1, 9											
2	2, 1, 5, 9											
3	1, 2, 5, 9											
3(b) B2.4.3	Award [1] max $O(n^2)$;	1										
3(c) B2.4.3	Award [3] max Award [1] per point up to [3] max. Both algorithms must be referred to. A bubble sort swaps every time it finds an adjacent pair out of order, so it can perform a very large number of swaps, up to $O(n^2)$; A selection sort finds the smallest remaining value first and makes at most one swap per pass, so it performs at most $n - 1$ swaps, which is $O(n)$; ...(so) selection sort moves data far less, which matters when the items being moved are large; Both make about the same number of <i>comparisons</i>, $O(n^2)$, so their overall time complexity is the same; A bubble sort can finish early on data that is already nearly sorted, which a standard selection sort cannot; On the array 5, 2, 9, 1 a bubble sort performs 4 swaps and a selection sort performs 2.	3										
4 B2.4.3	Award [6] max Award [1] for an outer loop over the positions; [1] for recording the position of the smallest value; [1] for an inner loop searching only the unsorted remainder; [1] for a correct comparison updating that position; [1] for a correct three-line swap (or equivalent); [1] for code that would run and sort the whole array. Accept either language. JAVA <pre>public static void selectionSort(int[] a) { for (int i = 0; i < a.length - 1; i++) { int smallest = i; for (int j = i + 1; j < a.length; j++) { if (a[j] < a[smallest]) smallest = j; } int temp = a[i]; a[i] = a[smallest]; a[smallest] = temp;</pre>	6										

Q	ANSWER	MARKS
	<pre> } } </pre> PYTHON <pre> def selection_sort(a): for i in range(len(a) - 1): smallest = i for j in range(i + 1, len(a)): if a[j] < a[smallest]: smallest = j a[i], a[smallest] = a[smallest], a[i] return a </pre> Accept an inner loop that starts at i rather than i + 1. Accept a guard that skips the swap when <code>smallest == i</code>. Award [4] max if the inner loop searches the whole array each time rather than only the unsorted part, since the sorted section would then be disturbed. Do not award the swap mark for a two-line assignment in Java that loses a value.	
HL 5(a) B2.4.4	Award [3] max Award [1] per point up to [3] max. The base case must be explained for full marks. Recursion is where a function calls itself to solve a smaller version of the same problem; Each call works on a smaller input, so the problem is broken down until it becomes trivial; The base case is the condition under which the function returns a value directly instead of calling itself again; ...(so) without it the calls would continue indefinitely, the call stack would fill and the program would crash with a stack overflow; The results of the smaller calls are then combined as the calls return, building up the final answer;	3
HL 5(b) B2.4.4	Award [2] max Award [1] for the point and [1] for the expansion. Any one limitation. Every call adds a frame to the call stack holding its own parameters and return address; ...(so) recursion uses more memory than a loop, and a deep recursion can exhaust the stack; There is an overhead in making and returning from each call; ...(so) a recursive version is usually slower than the equivalent loop; Recursive code can be harder to follow and to trace when debugging; ...(because) many calls are part-completed at the same time;	2
HL 6(a) B2.4.5	Award [2] max Award [2] for 10. Award [1] for correct working with an arithmetic slip. 10; $\text{total}(4) = 4 + \text{total}(3) = 4 + 3 + \text{total}(2) = 4 + 3 + 2 + \text{total}(1) = 4 + 3 + 2 + 1 + \text{total}(0) = 4 + 3 + 2 + 1 + 0 = 10.$	2
HL 6(b) B2.4.5	Award [2] max Award [1] per point up to [2] max. The base case would be gone, so the function would call itself with 0, then -1, then -2, and never stop; Each call adds a frame to the call stack, so the stack would fill and the program would crash with a stack overflow error;	2

Total for this paper: 35 marks