

THEME B: COMPUTATIONAL THINKING · B4.1 · HL ONLY

Fundamentals of ADTs — Mark scheme

34 marks · covers B4.1.1–B4.1.6

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
HL 1(a) B4.1.1	Award [3] max Award [1] per point up to [3] max. The answer must address why the implementation is irrelevant to the classification. An ADT is defined by the operations it offers and what each of them does, not by how it is built; A stack is anything supporting push, pop, peek and isEmpty with last in, first out behaviour; Both versions provide exactly those operations and behave identically as far as the user is concerned; ... (so) the array and the linked list are two <i>implementations</i> of the one ADT; The choice between them affects performance and memory, but not what the structure is; Code written against the ADT works with either, and one could be swapped for the other without changes;	3
HL 1(b) B4.1.6	Award [2] max Award [1] for the point and [1] for the expansion. Any one benefit. The implementation can be replaced with a more efficient one later; ... (and) no code that uses the ADT has to change, provided the operations behave the same way; The programmer using it only has to understand the operations; ... (so) the complexity of the internal mechanics is hidden, which reduces the chance of misuse; The internal data cannot be corrupted from outside; ... (because) every change must go through a defined operation that can enforce the rules of the structure;	2
HL 2(a) B4.1.2	Award [3] max Award [1] for three nodes each showing a data part and a pointer part; [1] for pointers linking 7 to 3 to 9 in that order; [1] for the head labelled and the final pointer shown as null. <pre> +-----+---+ +-----+---+ +-----+-----+ 7 -----> 3 -----> 9 null +-----+---+ +-----+---+ +-----+-----+ head</pre>	3

Q	ANSWER	MARKS
	Accept any clear notation, including arrows drawn between boxes. Accept a diagonal stroke or the word NULL or a grounding symbol to mark the end. Do not award the third mark if the last node's pointer is left blank with no indication that the list ends.	
HL 2(b) B4.1.2	Award [4] max This is an evaluate question (AO3), so both sides are required. Award [1] per point up to [4] max, but award a maximum of [3] if only one side is given. Arguments for a linked list It grows and shrinks one node at a time, so its size does not have to be known in advance; Memory is allocated per node as needed, so none is wasted on unused capacity; Inserting or deleting only changes a pointer or two, which is $O(1)$ once the position is known, whereas an array must shift the remaining elements; It never has to be resized and copied, which an array does when it becomes full; Arguments against a linked list Elements cannot be accessed by index: reaching the nth item means following the pointers from the head, which is $O(n)$; An array gives direct access to any element in $O(1)$; Each node needs extra memory for its pointer as well as its data; The nodes are scattered in memory rather than contiguous, so access is less cache-friendly and slower in practice; Judgement (credit a supported conclusion, for example) A linked list suits this collection because it changes size constantly and is processed in order rather than accessed by index;	4
HL 3(a) B4.1.3	Award [3] max Award [1] for each type correctly distinguished. In a singly linked list each node points only to the next node, so it can be traversed in one direction only; In a doubly linked list each node points to both the next and the previous node, so it can be traversed in either direction, at the cost of an extra pointer per node; In a circular linked list the last node points back to the first instead of to null, so traversal can continue round the list indefinitely with no end;	3
HL 3(b) B4.1.3	Award [3] max Award [1] for each correct step, [3] max. The order matters. Create the new node and store the value in it; Set the new node's next pointer to the current head of the list; Update head so that it points to the new node; The order of the second and third steps is essential: if head is reassigned first, the rest of the list is lost. Award [2] max for the steps in the wrong order. Accept a note that this also works when the list is empty, since head would be null.	3
HL 4(a) B4.1.4	Award [3] max Award [1] for 50 as the root with 30 on the left and 70 on the right; [1] for 20 and 40 as the children of 30, on the correct sides; [1] for 60 as the left child of 70. <pre> 50 / \ 30 70 </pre>	3

Q	ANSWER	MARKS
	$\begin{array}{ccccc} & / & & \backslash & / \\ 20 & & 40 & & 60 \end{array}$ Each value is compared with the nodes already present and goes left when smaller and right when larger. 60 is greater than 50 so goes right, then less than 70 so becomes its left child. 70 has no right child.	
HL 4(b) B4.1.4	Award [2] max Award [2] for the fully correct sequence. Award [1] for a sequence that is correct apart from one value, or that follows correctly (FT) from an incorrect tree in (a). 20, 30, 40, 50, 60, 70; An in-order traversal of a binary search tree always produces the values in ascending order, which is a useful check.	2
HL 4(c) B4.1.4	Award [2] max Award [1] for 3 and [1] for the explanation. 3 comparisons; 40 is compared with 50 (smaller, so go left), then with 30 (larger, so go right), then with 40 itself, which matches;	2
HL 5(a) B4.1.5	Award [2] max Award [1] per characteristic, [2] max. The elements are unordered, so there is no index or position; Every element is unique: adding a value that is already present does not change the set; There is no concept of a duplicate count, only whether an element is a member or not;	2
HL 5(b) B4.1.5	Award [3] max Award [1] for each correct result. $A \cup B = \{2, 4, 6, 8, 12\}$; $A \cap B = \{4, 8\}$; $A - B = \{2, 6\}$; Order does not matter, since a set is unordered. Do not credit $A - B = \{12\}$, which is $B - A$.	3
HL 6(a) B4.1.6	Award [2] max Award [1] per point up to [2] max. A hash function converts the value being stored into an index within the table; The same value always produces the same index, so the value can be found again directly; ...(so) lookup, insertion and deletion take roughly constant time, $O(1)$, rather than requiring a search;	2
HL 6(b) B4.1.6	Award [2] max Award [1] for the definition and [1] for a resolution strategy. A collision occurs when two different values are hashed to the same index in the table; Resolution (any one) Chaining: each slot holds a list, and colliding values are appended to the list at that slot; Open addressing (linear probing): the table is searched forward from the collision point for the next free slot; Rehashing: a second hash function is applied to find an alternative index;	2

Total for this paper: 34 marks