2.3 ALGORITHMS · 2.3.1(A)(B)

Analysis & design — Mark scheme

30 marks · spec 2.3.1(a)(b)

AO key: AO1 = knowledge · AO2 = application · AO3 = programming/design. Credit any valid alternative; do not award the same point twice.

Q	ANSWER	AO	MARKS
1(a)	Analysis = defining what the solution must do (requirements/I-O/criteria) (1); design = planning how it will work (structures/algorithm/pseudocode) (1). (2)	AO1	2
1(b)	Measurable statements (1); of what a correct, finished solution must achieve (1). (2)	AO1	2
1(c)	Any two: pseudocode · flowchart · structure diagram. (2)	AO1	2

Q	ANSWER	AO	MARKS
2(a)	1 mark each, max 2: inputs · outputs · processing required · requirements/success criteria. (2)	AO1	2
2(b)	Normal/typical data (within range) (1); boundary/extreme data (at the limits) (1); erroneous/invalid data (should be rejected) (1). (3)	AO1	3
2(c)	So the finished solution can be checked against the success criteria / proven to work. (1)	AO2	1

Q	ANSWER	AO	MARKS
3(a)	Any two: execution time (speed) · memory/space used. (2)	AO1	2
3(b)	Binary search halves the search space each step (≈ $\log_2 n$) (1); far fewer checks than linear search (up to n) on a large list (1). (2)	AO2	2
3(c)	The data must already be sorted. (1)	AO1	1
3(d)	Any one: the list is small · the list is unsorted and rarely searched (sorting wouldn't be worth it). (1)	AO2	1

Q	ANSWER	AO	MARKS
4(a)	A measure of how the running time / number of steps (1); grows as the size of the input increases (1). (2)	AO1	2
4(b)	How much memory/storage the algorithm needs (as input grows). (1)	AO1	1
4(c)	You can often reduce time by using more memory, or save memory at the cost of time (1); valid example (1) + explanation (1), e.g. caching results uses extra memory but makes repeated work much faster. (3)	AO1	3

Q	ANSWER	AO	MARKS
5(a)	Input: the list of scores (1); output: the highest score (1). (2)	AO2	2
5(b)	Any one: returns the correct maximum for any non-empty list / handles an empty list appropriately. (1)	AO2	1

5(c)	Indicative design (award for logic): set `highest` to the first score (1); loop through the rest of the list (1); if an item > `highest`, update `highest`; output `highest` at the end (1). (3)
	`highest = scores[0]`
	`for i = 1 to length(scores) - 1`
	`    if scores[i] > highest then`
	`        highest = scores[i]`
	`    endif`
	`next i`
	`print(highest)`
		AO3	3

Total for paper: 30 marks