2.3 ALGORITHMS · 2.3.1(C)(D)

Complexity & Big-O — Mark scheme

30 marks · spec 2.3.1(c)(d)

AO key: AO1 = knowledge · AO2 = application. Accept O(...) in any clear form. Do not award the same point twice.

Q	ANSWER	AO	MARKS
1(a)	The worst-case rate at which time/space grows (1); as the input size n increases (keeping the dominant term) (1). (2)	AO1	2
1(b)	Time = how the number of steps grows with n (1); space = how the memory used grows with n (1). (2)	AO1	2
1(c)	O(1) (constant). (1)	AO1	1
1(d)	Seconds depend on the hardware; complexity measures how work scales with n. (1)	AO1	1

Q	ANSWER	AO	MARKS
2(a)	O(n). (1)	AO2	1
2(b)	$O(n^2)$. (1)	AO2	1
2(c)	O(log n). (1)	AO1	1
2(d)	O(n) (drop the constant and the +7). (1)	AO2	1
2(e)	The inner loop runs n times (1); for each of the outer loop's n iterations → $n \times n = n^2$ operations (1). (2)	AO2	2

Q	ANSWER	AO	MARKS
3(a)	O(1), O(log n), O(n), $O(n^2)$. All correct (2); one slip (1). (2)	AO1	2
3(b)	O(n) ≈ 1,000 operations (1); $O(n^2)$ ≈ 1,000,000 operations (1). (2)	AO2	2
3(c)	O(log n) grows far more slowly than O(n) (1); so for very large n it needs vastly fewer operations (e.g. ~20 vs a million for a million items) (1). (2)	AO2	2

Q	ANSWER	AO	MARKS
4(a)	It uses nested loops / repeatedly passes over the list (1); comparing roughly n items n times → n^2 comparisons (1). (2)	AO2	2
4(b)	Merge sort (1); O(n log n) grows much more slowly than $O(n^2)$, so it is far faster on large lists (1). (2)	AO2	2
4(c)	For small n the number of operations is small anyway (1); and the algorithm may be simpler to write / has lower constant overhead (1). (2)	AO2	2

Q	ANSWER	AO	MARKS
5(a)	O(n) (linear). (2) (1 if "linear" without notation, or close). (2)	AO2	2
5(b)	It is a single loop over the list (1); the number of iterations grows in direct proportion to the list length n (1). (2)	AO2	2
5(c)	How the memory/storage used grows with the input size. (1)	AO1	1
5(d)	Any one: O(n log n) or $O(n^2)$ (both worse than O(n), better than $O(2^n)$). (1)	AO1	1

Total for paper: 30 marks