2.3 ALGORITHMS · 2.3.1(C)(D)

Complexity & Big-O notation

How time/space grow with n, the Big-O classes & comparing algorithms. Spec 2.3.1(c)(d).

01 What complexity means

Time	How steps grow with input **n**.
Space	How memory grows with **n**.
Big-O	**Worst-case** growth; drop constants, keep dominant term.
e.g.	`3n + 5 → O(n)`

02 Classes (best → worst)

O(1)	Constant — array index.
O(log n)	Logarithmic — binary search.
O(n)	Linear — linear search.
O(n^2)	Polynomial — bubble/insertion sort.
O(2^n)	Exponential — brute-force subsets.

03 Spotting Big-O

1 loop	≈ **O(n)**.
Nested loops	≈ **O(n^2)**.
Halve each step	≈ **O(log n)**.
No loop	≈ **O(1)**.

04 Comparing algorithms

Large n	Lower growth wins: log n < n < n^2 < 2^n.
Small n	Constants matter — Big-O is about **scaling**.
Also	Weigh **space** (e.g. merge sort needs extra memory).

FINAL PASS BEFORE THE EXAM

Rapid exam tips

Six slips on Big-O questions.

01

Drop constants: `2n+3 → O(n)`. Keep the dominant term.

02

$O(n^2)$ = **polynomial**; $O(2^n)$ = **exponential**. Don't swap.

03

Single loop → O(n); **nested loops** → $O(n^2)$.

04

Halving each step → **O(log n)** (binary search).

05

Big-O = **large-n scalability**; small n, constants can matter.

06

bubble $O(n^2)$ vs merge O(n log n): merge scales far better.