Lesson Plan: Complexity Olympics

Year Group: 9 | **Duration:** 50 minutes | **Topic:** Algorithmic Complexity / Big-O Notation

---

1. Overview

Core Concept: Algorithmic complexity — how the number of steps grows as the input size grows. O(1), O(log n), O(n), $O(n^2)$ demonstrated through physical activities.

Learning Objectives:

- Experience O(1), O(log n), O(n), and $O(n^2)$ through physical activities
- Plot growth curves and identify their shapes (flat, logarithmic, linear, quadratic)
- Connect complexity classes to real algorithm choices
- Understand why $O(n^2)$ algorithms become unusable at large scale

Key Vocabulary:

Term	Definition
Complexity	How the number of steps grows as input size grows
Big-O notation	A way of expressing the growth rate of an algorithm
O(1)	Constant time — always the same number of steps
O(log n)	Logarithmic time — steps grow very slowly
O(n)	Linear time — steps grow proportionally with input
$O(n^2)$	Quadratic time — steps grow as the square of input size
Input size (n)	The amount of data being processed

---

2. Before the Lesson

Print:

- [ ] `resource-station-cards.md` — 1 per station (laminate if possible)
- [ ] `worksheet-results-recording.md` — 1 per student

Room Setup:

- 4 stations around the room, each with its instruction card
- Each station needs a small stack of numbered index cards
- Groups of 5–6 students, rotating every ~7 minutes

---

3. Timed Lesson Flow

0–5 min — Introduce the Olympics

1. Today: four events. Each measures how the WORKLOAD grows as the INPUT gets bigger.
2. At each station: read the card, do the activity for small n, medium n, large n. Record on worksheet.

5–35 min — Rotation Through Stations

Groups rotate, ~7 minutes per station. Teacher circulates.

Station 1 — O(1): Direct lookup. Given a position number, retrieve that card instantly. Always 1 step, regardless of stack size.

Station 2 — O(log n): Binary search on a sorted stack. Count guesses for different stack sizes.

Station 3 — O(n): Find the maximum in an unsorted pile. Must look at every card.

Station 4 — $O(n^2)$: Check every pair of cards for duplicates. Count total comparisons.

35–45 min — Graph

Students complete the worksheet graph — plotting all 4 curves on the same axes.

45–50 min — Debrief

- Which algorithm would you want for 1 million items?
- Why does $O(n^2)$ become unusable at scale?
- *"If sorting 10 items takes 100 steps with $O(n^2)$ sort, how many steps for 100 items? For 1000?"*

---

4. Teacher Facilitation Notes

What to look for:

- Station 4: students undercount pairs — remind them: card 1 pairs with ALL others, then card 2 pairs with remaining others. Formula: $n \times (n-1) \div 2$
- Station 1: students feel like they're "cheating" — that IS the point. O(1) means the data structure gives instant access (like an array index or dictionary key)
- Help students predict before doing: "If n=5 takes 10 comparisons, how many will n=10 take?" The doubling to 40 (not 20) surprises people

Common misconceptions:

- O(1) means fast — no, it means constant. An O(1) operation could still take 1 second. O(n) on 3 items might be faster.
- Big-O is the exact number of steps — no, it's the growth RATE (proportionality). Constants and lower-order terms are ignored.

---

5. Extension Tasks

1. For n=100, station 4 takes 4,950 comparisons. For n=1000: ~500,000. Plot these. Why is $O(n^2)$ impractical at scale?

2. If a computer does 1 billion operations per second, how long does $O(n^2)$ take for n=1,000,000?
3. Research: what is $O(n \log n)$? Which sorting algorithms achieve this?

---

6. Key Takeaway

> **The growth rate of an algorithm matters as much as its correctness. $O(n^2)$ algorithms become impractical at scale. $O(\log n)$ algorithms scale magnificently. Choosing the right complexity class is a core engineering decision.**