LECTURE 3

Algorithm Analysis and Complexity

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

AGENDA

Learning Objectives

1. Why we measure algorithms by counting operations, not with a stopwatch
2. Time complexity vs. space complexity
3. Best-case, average-case, and worst-case analysis
4. Asymptotic analysis: Big-O, Big-Theta, Big-Omega
5. The common complexity classes you'll see again and again this semester
6. Deriving Big-O by hand for a nested loop whose bound isn't obvious

MEASURING ALGORITHMS

Count Operations, Not Time

A stopwatch depends on the CPU, the language, how busy the machine is. What we want is a measure that's true on ANY computer, forever: count how the number of basic operations grows as input size n grows.

growth_demo.cpp (excerpt)

```
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        comparisons++;

// n x n nested loop
```

n	comparisons
10	100
20	400
40	1,600
80	6,400

Every time n doubles, the count QUADRUPLES — a mathematical property of n × n, true on any machine, any run.

MEASURING ALGORITHMS

Time Complexity vs. Space Complexity

Time Complexity

- How the number of basic operations grows as a function of n
- What this course focuses on almost exclusively

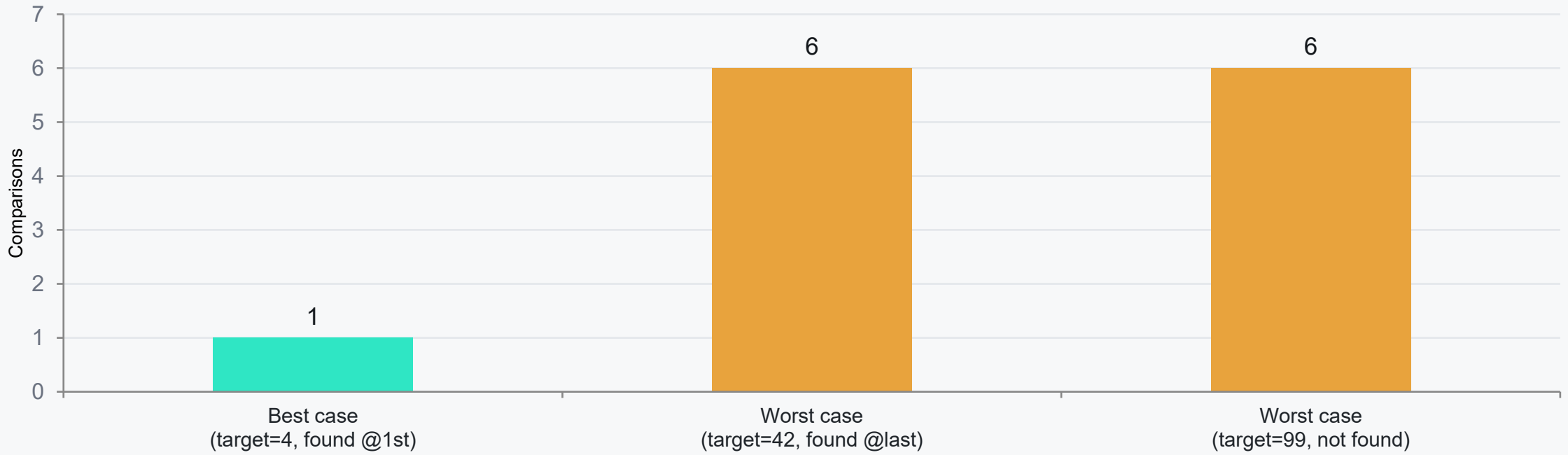
Space Complexity

- How much EXTRA memory an algorithm needs, beyond the input itself, as a function of n
- Often in tension with time — caching trades memory for speed

An algorithm can be sped up by using more memory (caching results instead of recomputing), or made more memory-frugal at the cost of extra time.

Best, Average, and Worst Case

Linear search on {4, 8, 15, 16, 23, 42} — the SAME algorithm, different inputs:



This course (and the industry) defaults to WORST-CASE analysis when we say "the complexity of an algorithm."

NOTATION

Asymptotic Analysis

Asymptotic analysis studies how an algorithm's resource use grows as n approaches infinity, deliberately ignoring constant factors and lower-order terms — because for large enough n, they stop mattering.

=

Same growth category

3n + 20 and n both belong to the same class — O(n). Constants don't change the class.

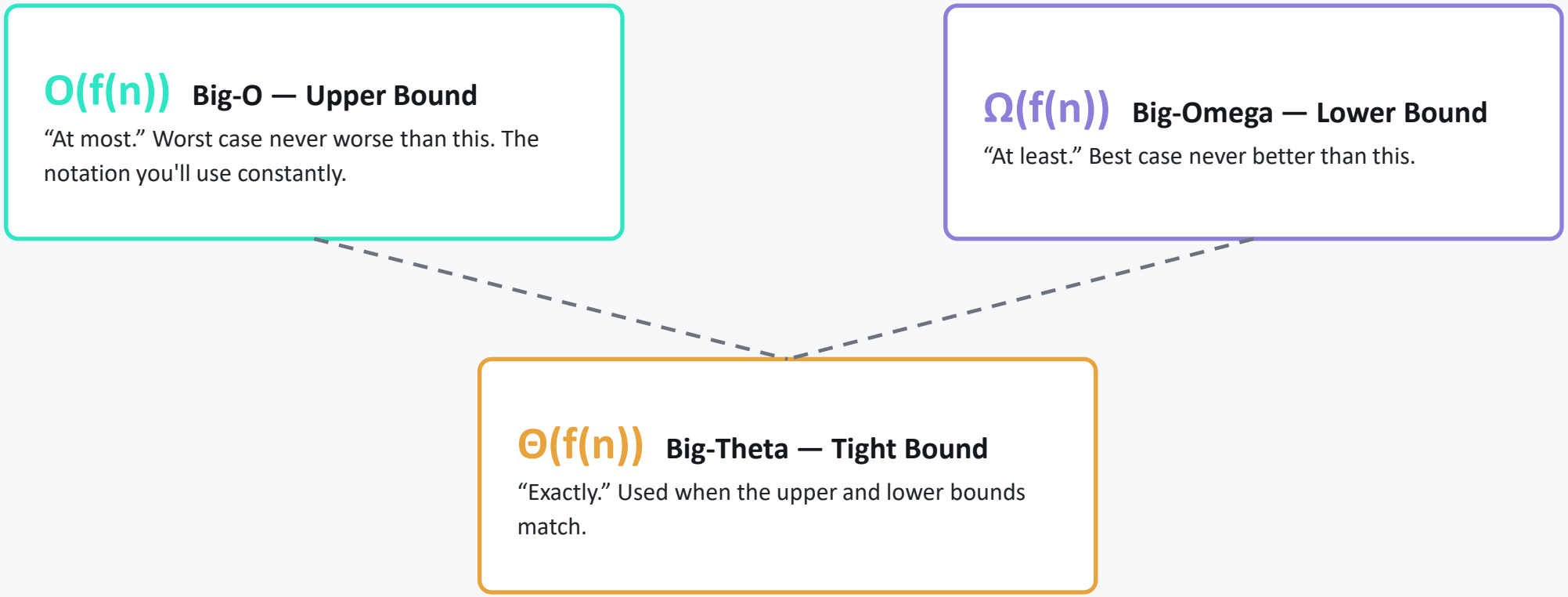
≠

Different growth category

n^2 belongs to a fundamentally different, worse class than n, no matter the constants.

NOTATION

Big-O, Big-Omega, Big-Theta



Linear search is Ω(1) (might get lucky on the first element) but also O(n) (might check every element). Summing an array is Θ(n) — it always visits every element, no more, no less.

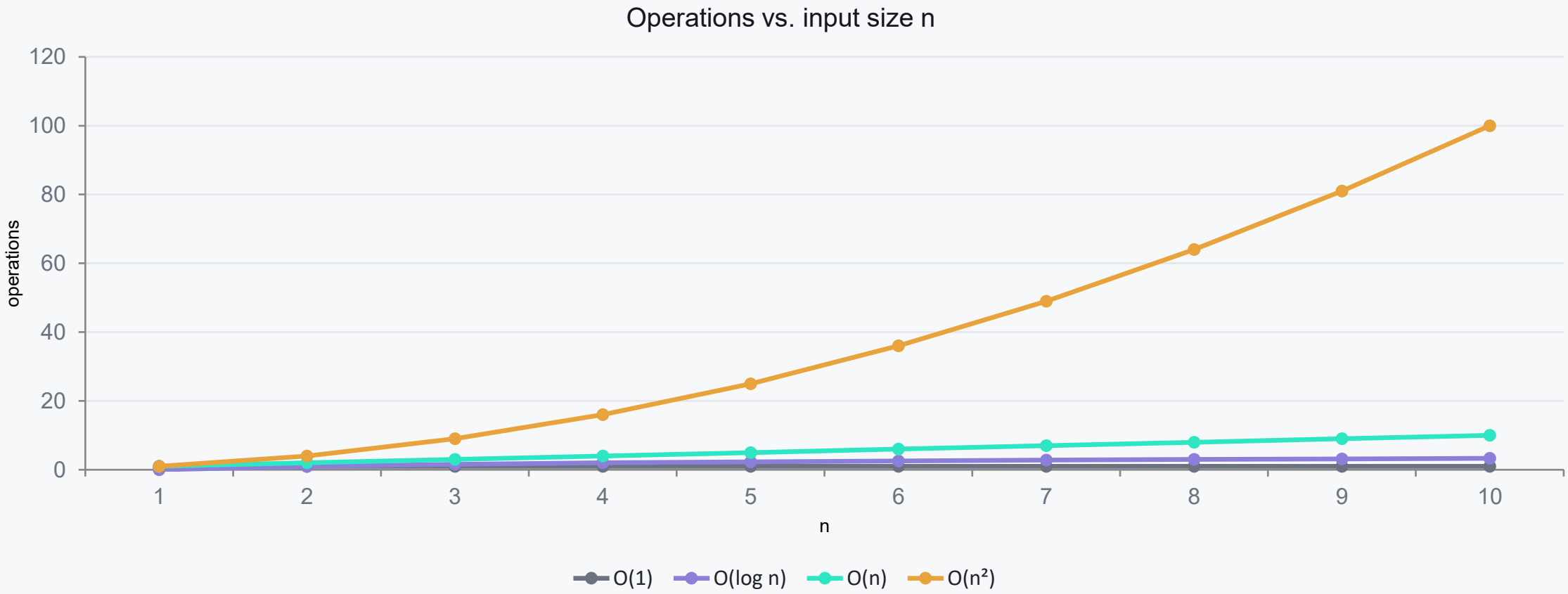
THE VOCABULARY

Common Complexity Classes

Notation	Name	Example
$O(1)$	Constant	Accessing array[5] — one step, regardless of size
$O(\log n)$	Logarithmic	Binary search — halves the search space each step
$O(n)$	Linear	Linear search, printing every element once
$O(n \log n)$	Linearithmic	Merge sort, quick sort
$O(n^2)$	Quadratic	Bubble sort, nested loops over the same input
$O(n^3)$	Cubic	Naive matrix multiplication
$O(2^n)$	Exponential	Naive recursive Fibonacci — grows explosively

THE VOCABULARY

Growth Rates, Plotted



O(n²) pulls away fast — at n = 10 it's already at 100 while O(n) is only at 10. That gap only widens as n grows: a faster machine buys a constant-factor speedup, a better algorithm changes the entire curve.

THE VOCABULARY

Why Each Class Looks the Way It Does

1

O(1) — array access

One multiplication, one addition — regardless of array size.

L

O(log n) — binary search

Each comparison eliminates HALF of what's left: $\log_2 n$ halvings to reach 1 element.

N

O(n) — linear search

Worst case: every one of the n elements gets exactly one comparison.

M

O(n log n) — merge sort

log n levels of splitting in half, each level doing a full O(n) merge pass.

Q

O(n^2) — nested loops

For every one of the n elements, the inner loop does up to n more comparisons.

E

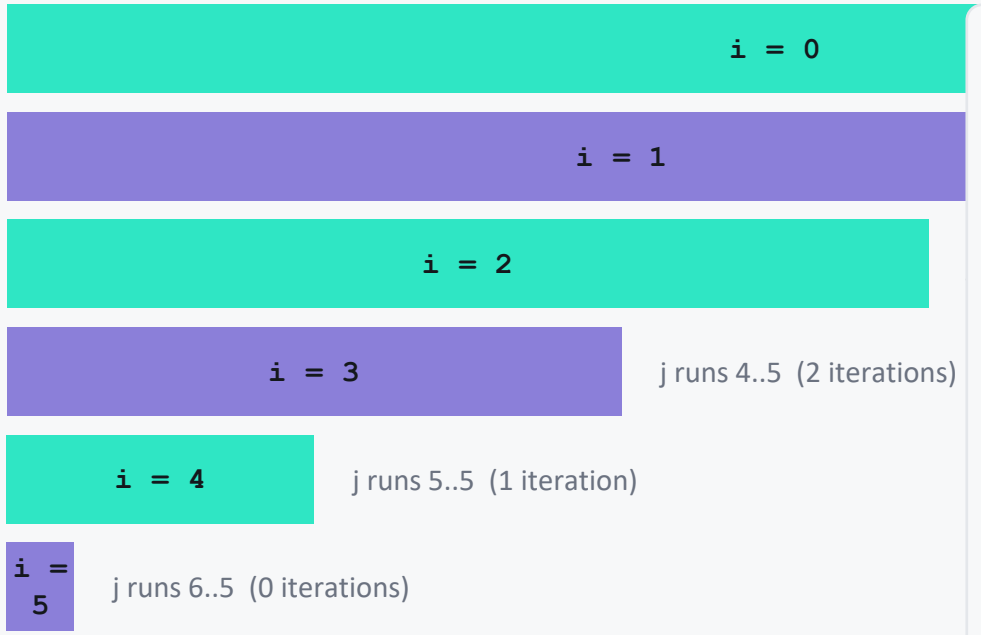
O(2^n) — recursive Fibonacci

Every call spawns two more, doubling the work at every level of recursion.

DERIVING BIG-O BY HAND

A Nested Loop With a Non-Obvious Bound

`for (i = 0; i < n; i++)  for (j = i+1; j < n; j++)  count++;` — the inner bound depends on i.



It's a TRIANGLE, not a rectangle.

The inner loop's work shrinks by one every time the outer loop advances — which is exactly why the total works out to roughly HALF of the full n × n square.

for n = 8, real output: 28 pairs checked — matches the shape above exactly.

DERIVING BIG-O BY HAND

From a Sum to a Big-O Class

```
summing across every outer iteration
total = (n-1) + (n-2) + (n-3) + ... + 1 + 0
      = sum from k=0 to n-1 of k
      = n(n-1) / 2            // standard sum 0..n-1
      = (n² - n) / 2
```

Drop the lower-order term and the constant factor:

$O((n^2 - n) / 2) = O(n^2)$

Even though this loop does roughly HALF the work of a naive n×n double loop, it is still, correctly, $O(n^2)$ — constant factors never change the complexity class.

PROOF, NOT JUST ASSERTION

Timed Comparisons Across Complexity Classes

n = 4,000, same run, same machine — sumAll (O(n)), countEqualPairs (O(n²)), binarySearch (O(log n)):

10 µs
sumAll — O(n)

190,000 µs
countEqualPairs — O(n²)

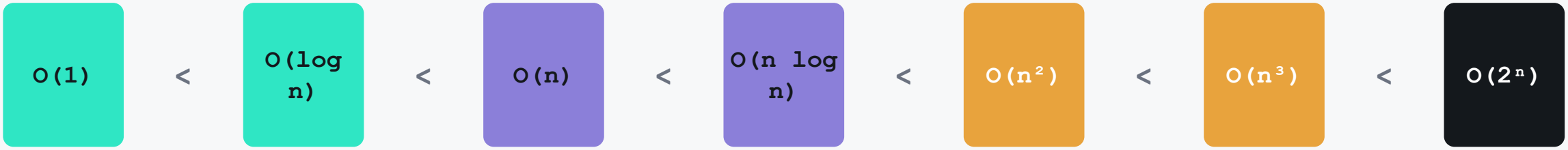
0 µs
binarySearch — O(log n), too fast to register

Roughly a 19,000-fold slowdown from a mere quadratic exponent, on the same input size — comfortably clearing the "at least 1000x slower" bar the program checks automatically.

```
long quadraticTime = duration_cast<microseconds>(e2 - s2).count();
// quadraticTime >= linearTime * 1000  ->  yes, every run
```

THE VOCABULARY

Memorize the Order



fastest-growing (best) → slowest-growing (worst)

Every remaining lecture in this course will describe its operations in these exact terms — this ordering is the single most-used vocabulary of the semester.

KEY TAKEAWAYS

Key Takeaways

- Algorithms are analyzed by counting how basic operations grow with n, not by timing them — this makes the analysis true on any machine, forever.
- Time complexity measures operation growth; space complexity measures extra memory growth.
- Best-, average-, and worst-case describe the same algorithm on different inputs; this course defaults to worst-case.
- Big-O is an upper bound, Big-Omega a lower bound, Big-Theta a tight bound where both match.
- Real timing confirms the theory: at n = 4,000, an $O(n^2)$ loop measured roughly 20,000× slower than an O(n) loop, while O(log n) finished too fast to even register.