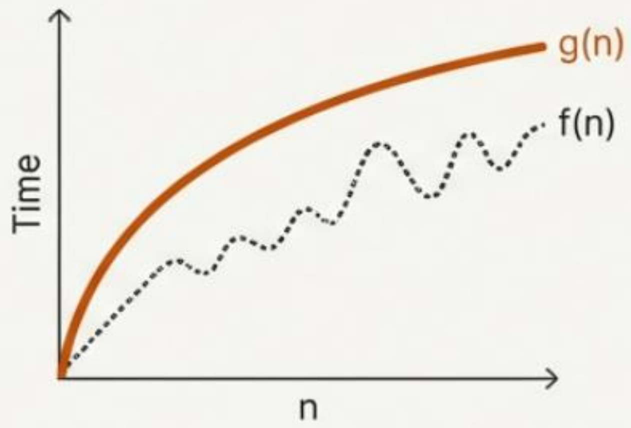
Lecture 2

Beyond T(n): Defining Asymptotic Boundaries

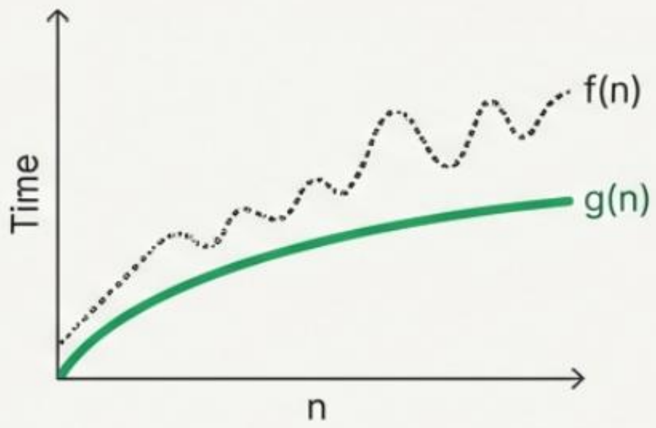
Calculating the time equation T(n) is only the first step. To truly understand performance, we must define the algorithm's limits using **Asymptotic Notation**.



O The Upper Bound

Worst Case Scenario

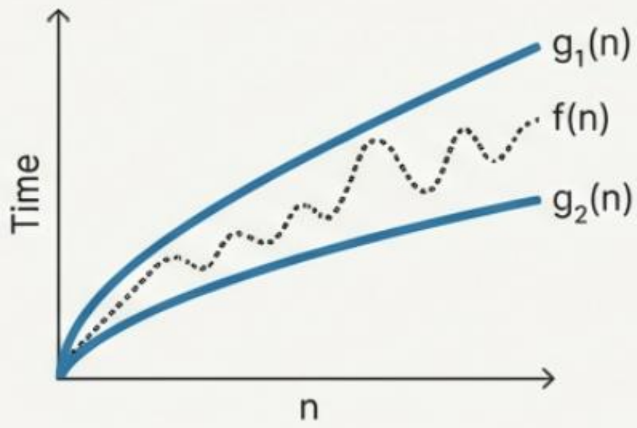
The algorithm will never perform worse than this limit. It encompasses the maximum number of steps required if the input configuration is maximally unhelpful.



Ω The Lower Bound

Best Case Scenario

The algorithm will never perform better than this limit. It represents the minimum effort required if the input configuration is ideal.



Θ The Tight Bound

Average Case / Exact Order

The precise order of growth that sits "tightly" between the upper and lower bounds. It represents the median or expected scenario.

KEY INSIGHT: These noutations allow us to predict behavior across three dimensions: the pessimistic view (Worst), the optimistic view (Best), and the realistic view (Average).

Method A: Mathematical Approximation

When only the equation T(n) is available—without access to the algorithm's logic—we use "Dominant Term Analysis" to determine growth.

- [1] **Ignore Constants:** Coefficients like '3' are irrelevant.
- [2] **Compare Exponents:** Identify highest power.
- [3] **Identify Dominant Term:** The largest exponent dictates complexity.

RAW EQUATION

$$T(n) = 3n^3 + n^2 + \sqrt{n}$$

Coefficient (Constant)

Lower Order Term

FILTERING NOISE

$$T(n) = \cancel{3}n^3 \underbrace{\cancel{+\ n^2}}_{\text{Filtering Noise}} \cancel{+\ \sqrt{n}}$$

FINAL RESULT

$$O(\boxed{n^3}) = \boxed{\Omega(n^3)} = \boxed{\Theta(n^3)}$$

The Dominant Term

CRITICAL NOTE: This is a 'quick solution.' It determines the polynomial order of growth but ignores specific algorithmic behaviors like 'Best Case' logic.

e.g., early exit in search.

The Context: Anatomy of a Sequential Search

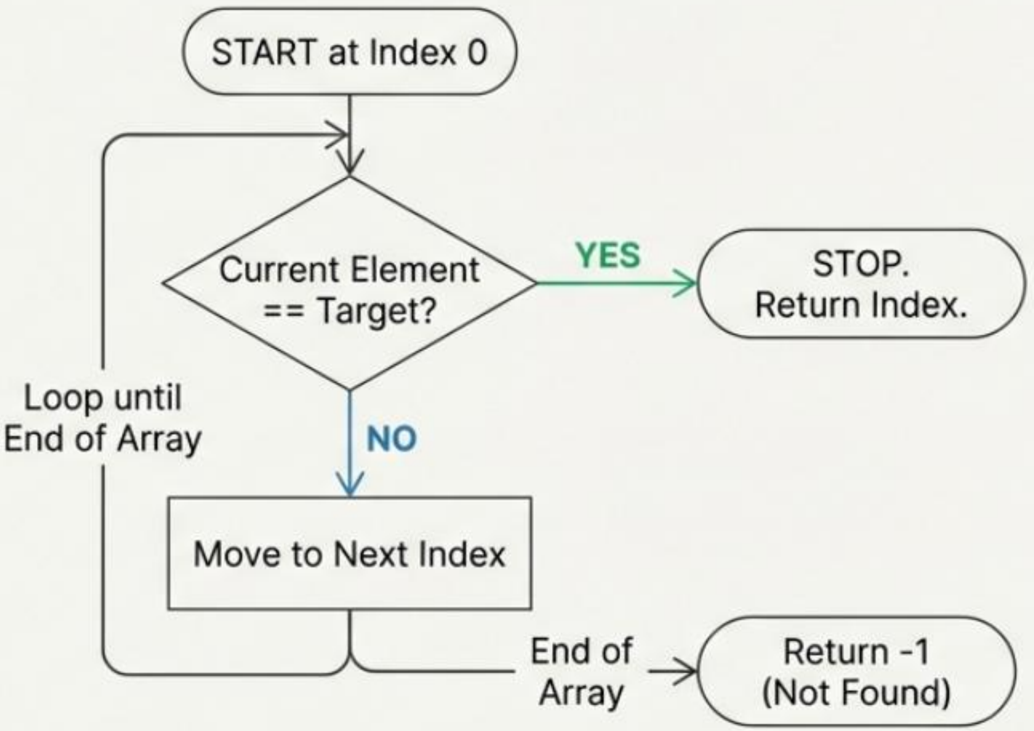
The Objective

`Goal: Find a specific "Target" value within an array of size n.`

Index 0	Index 1	Index 2	Index 3	Index 4		Target:
12	7	5	9	2		5

`The element layout is unknown. We must inspect elements one by one.`

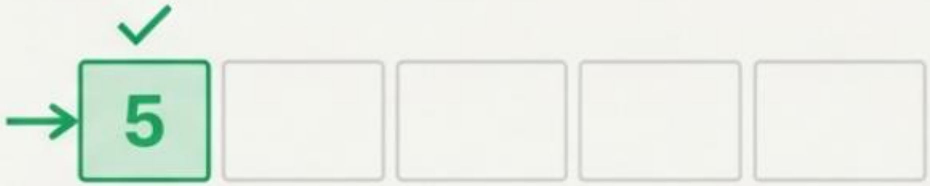
Algorithm Flow



KEY MECHANIC: The algorithm's runtime is strictly dependent on WHERE the target is located. It stops the moment success is achieved.

Method B: Logical Analysis of Performance Scenarios

SCENARIO: BEST CASE



`Effort: 1 step`
`Result: Ω(1) [Constant Time]`

SCENARIO: AVERAGE CASE

`Effort: n/2 steps`
`Result: Θ(n) [Linear Order]`

SCENARIO: WORST CASE

`Effort: n steps`
`Result: O(n) [Linear Time]`

OBSERVATION:

Unlike the mathematical polynomial, logical analysis reveals that the Best Case (Ω(1)) is fundamentally faster than the Worst Case (O(n)).

Synthesis: Mathematical Approximation vs. Logical Precision

The Equation Strategy

- **When to use:** Use when you only have $T(n)$ and no code context.
- **Technique:** Dominant Term Analysis (Highest Exponent).
- **Limitation:** Provides a "Quick Solution." Treats Best, Worst, and Average as the same polynomial order. Misses behavioral nuance.

The Algorithmic Strategy

- **When to use:** Use when you understand the code flow and mechanics.
- **Technique:** Scenario Analysis (Best / Worst / Average).
- **Advantage:** High Precision. Captures variance in performance based on input data (e.g., distinguishing $\Omega(1)$ from $O(n)$).

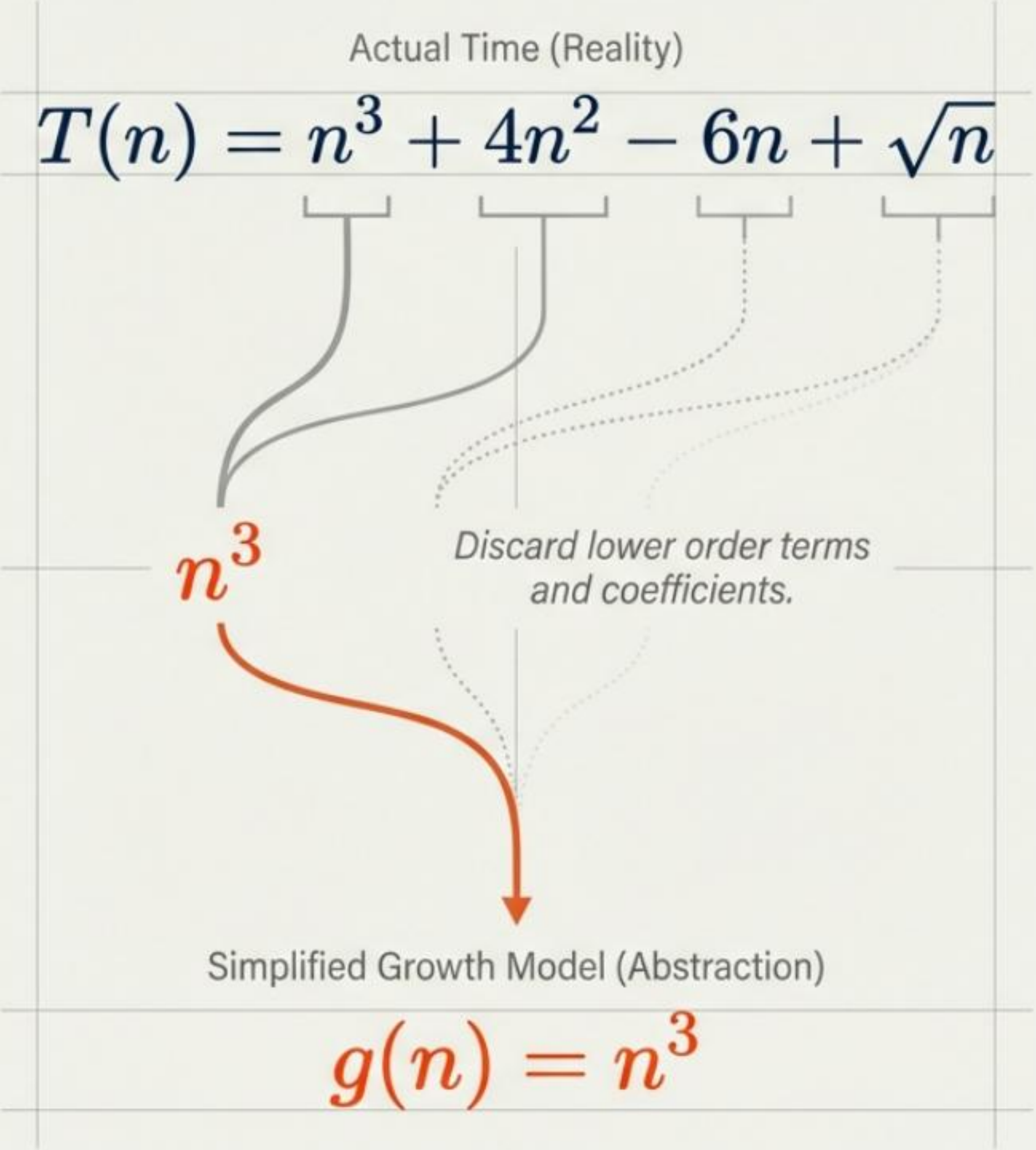
FINAL TAKEAWAY: Math gives us the order of growth for a function. Logic gives us the behavioral limits of a solution. True Asymptotic Analysis requires understanding HOW the code executes, not just the equation it produces.

Distilling Complexity into **Shape**

In the real world, an algorithm's running time—denoted as **T(n)**—is a complex equation. It accounts for every operation, loop, and system overhead, often resulting in a messy polynomial.

To categorize efficiency, we ignore the minor details. We look for the **Order** (O). We identify the term with the highest exponent and strip away all other terms and coefficients.

This simplified term is **g(n)**. It represents the fundamental 'shape' of the growth. Our goal is to prove that the complex reality fits within this simple model.



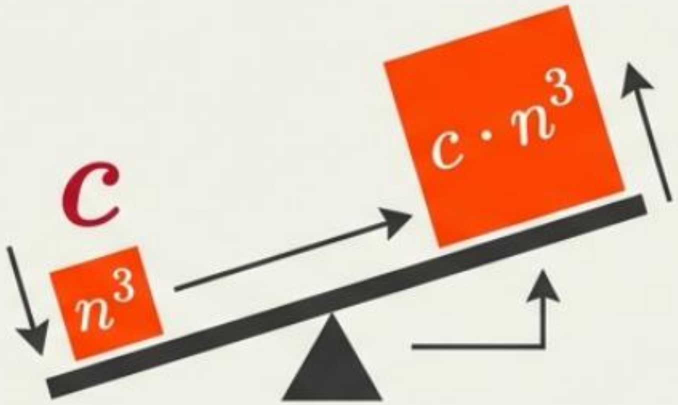
Constructing the Ceiling

Here lies the problem: simple simplification fails the math. The model (n^3) is numerically smaller than the reality ($n^3 + 4n^2 \ldots$) because it lacks the extra positive terms.

Big O defines an **Upper Bound**. It must be a ceiling that the algorithm never breaks through.

The Solution: The **Constant (c)**. We do not use $g(n)$ alone; we use $c \cdot g(n)$. By multiplying our model by a factor (e.g., 9), we force the simplified curve to rise above the complex equation, creating a valid mathematical ceiling.

$$n^3 < n^3 + 4n^2 - 6n + \sqrt{n} \quad \text{✗ False}$$



$$c \cdot n^3 \geq n^3 + 4n^2 - 6n + \sqrt{n}$$

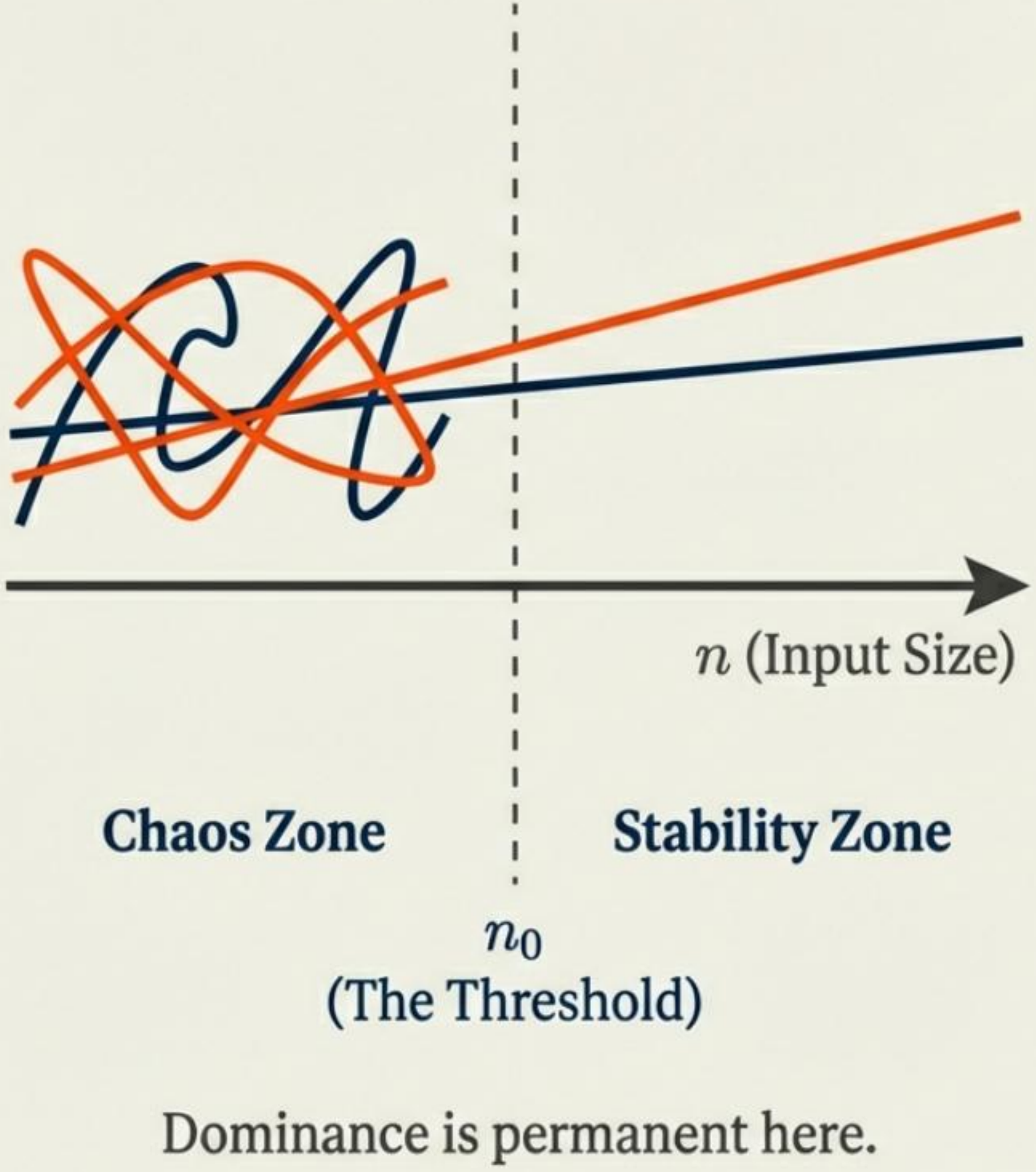
Example: If $c = 9$, then $9n^3$ dominates the equation.

Defining Asymptotic Stability

Even with a multiplier, the functions might intersect multiple times when the input size (n) is small. The "Upper Bound" might temporarily fail.

We call this the instability zone. But Big O analyzes asymptotic behavior—the long run. We don't care about the start; we care about the end game.

We must identify n_0. This is the **last point of intersection.** For all inputs $n \geq n_0$, the Upper Bound must stay above the running time permanently.

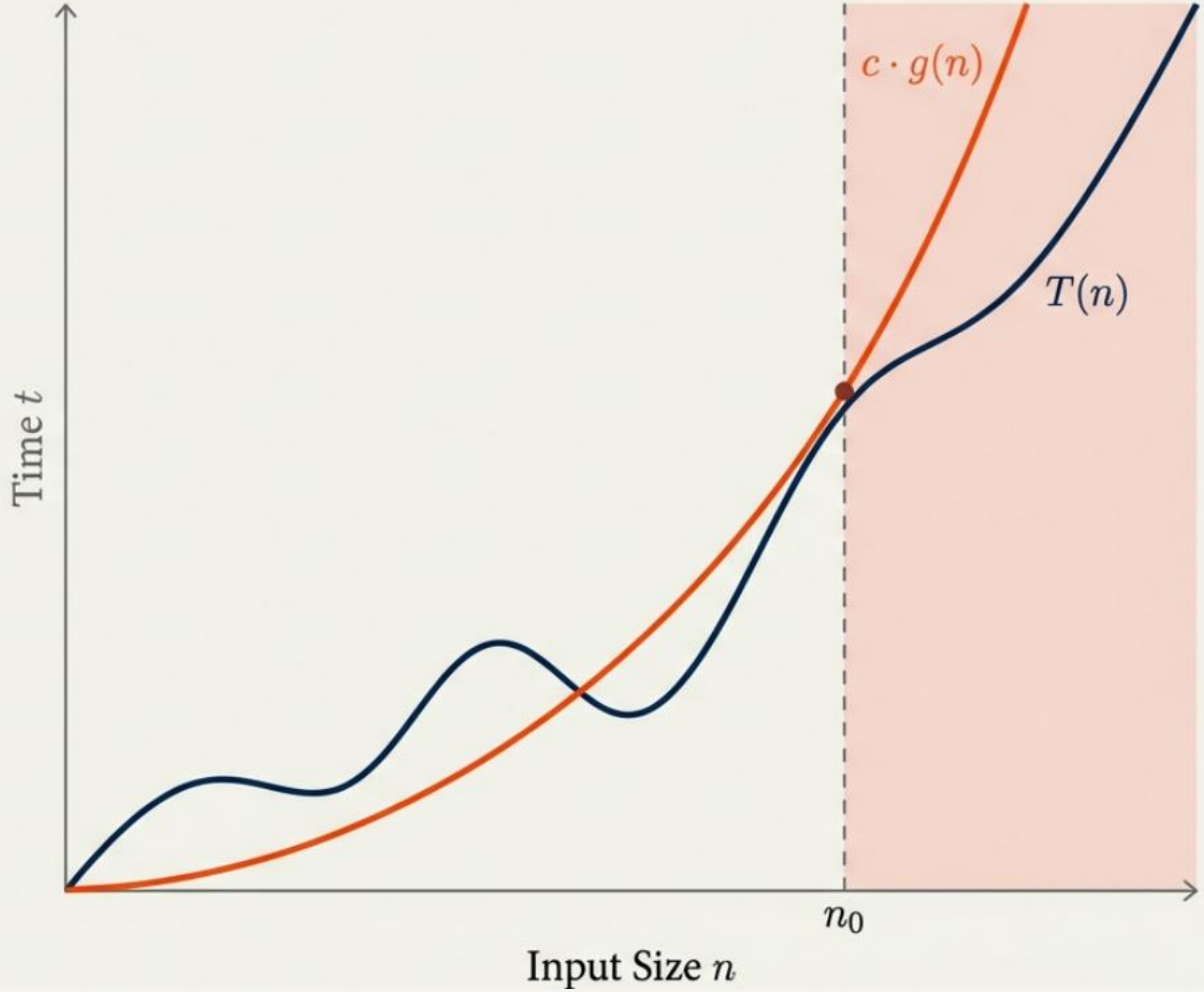


The Geometry of Big O

Visualizing the Asymptotic dominance.

1. The Reality (Blue) fluctuates.
2. The Model (Red) is smooth.
3. At n_0, the Red line overtakes the Blue line for the final time.

In the shaded region, $c \cdot g(n)$ is always higher than $T(n)$.



The Mathematical Contract

$$T(n) = O(g(n))$$

if and only if there exist positive constants c and n_0 such that:

$$0 \le T(n) \le c \cdot g(n) \quad \text{for all } n \ge n_0$$

$T(n)$	$g(n)$	c	n_0
in Reality Blue (#002147)	in Model Red (#FF4F00)	in Model Red (#FF4F00)	in Graphite (approx. #444444)
The Algorithm (Reality)	The Shape (Model)	The Scaler (Multiplier)	The Threshold (Start Point)

Translation: For sufficiently large inputs ($n \ge n_0$), the algorithm's running time will never grow faster than our scaled model.

Omega Notation (Ω)

The Asymptotic Lower Bound

Definition: Omega represents the "Best Case" scenario or the absolute performance floor. An algorithm cannot run faster than this bound.

> **Key Insight:** While Big O (O) defines the ceiling (Worst Case), Omega (Ω) defines the floor. Both are derived from the highest degree term.

$$T(n) = n^3 + 4n^2 + n$$

Asymptotic Reduction

$$1\boxed{n^3} + \cancel{4n^2} + \cancel{n}$$

Ignore coefficients & lower-order terms.

$$T(n) = \Omega(n^3)$$

Reduction Rule: Identify the highest power in its simplest form. The oscillation of lower-order terms is negligible as $n \to \infty$.

Mathematical Definition & Constraints

Establishing the Lower Bound Inequality

$$0 \le c_2 \cdot g(n) \le T(n)$$

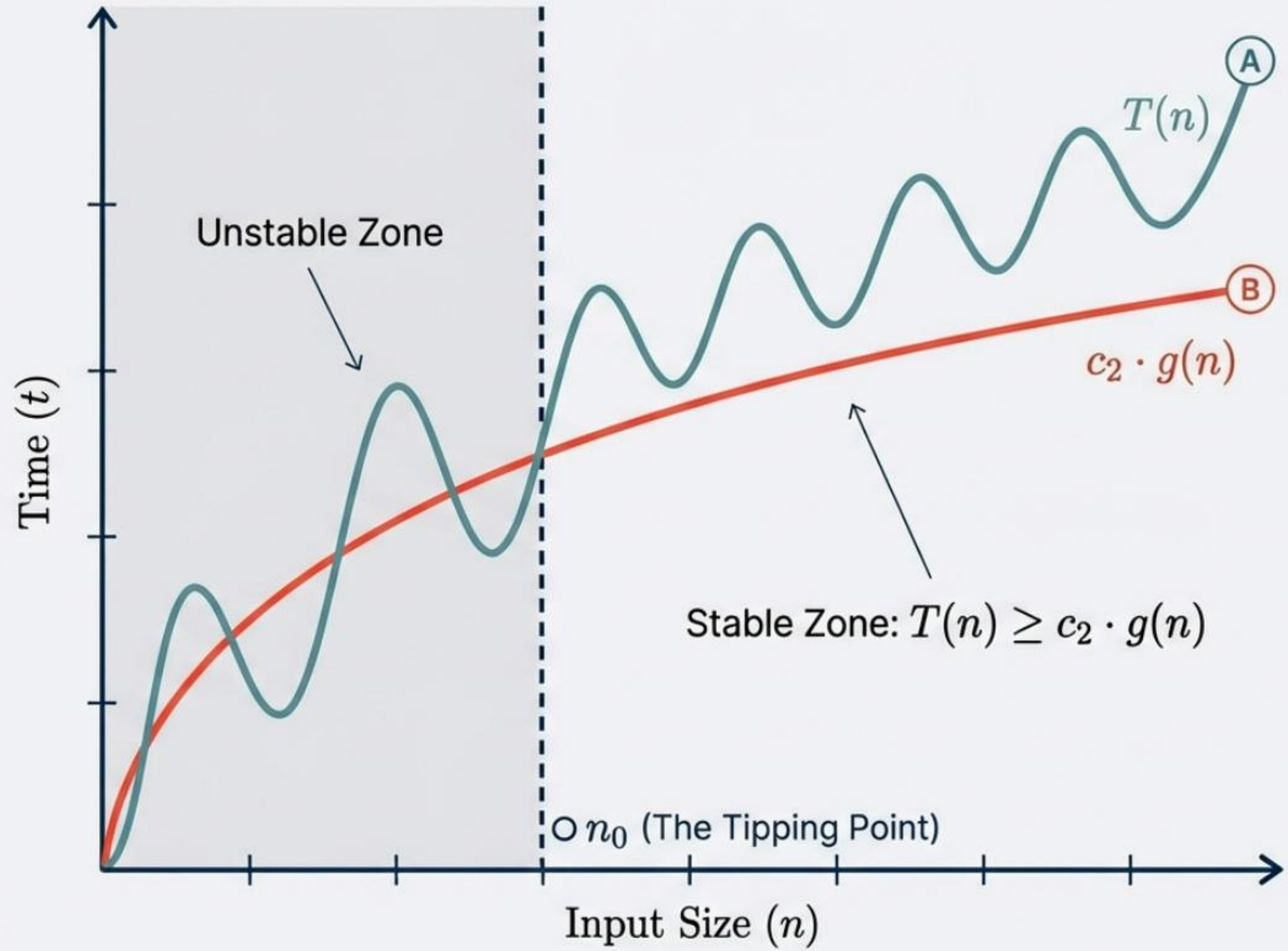
$$\text{for all } n \ge n_0$$

The Floor Condition
Unlike Big O, the bound here is *less than or equal to* the function. $T(n)$ sits above the curve.

The Scaling Constant (c_2)
A positive constant ($c_2 > 0$) specifically chosen to scale $g(n)$ so it fits *underneath* the execution time. Distinct from the Big O constant (c_1).

The Threshold
The point where stability begins. Pre-n_0 behavior is irrelevant.

Formal Logic: $T(n) = \Omega(g(n))$ iff there exist positive constants c_2, n_0 such that the function remains bounded from below. If we can find a c_2 that satisfies this inequality, the Lower Bound is proven.



Graphical Analysis

Lower Bound / Best Case

The function never dips below this line after n_0.

Upper Bound / Worst Case

Contrast: Big O acts as the ceiling.

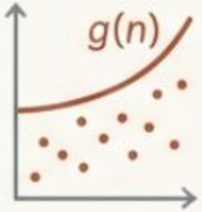
n_0 represents the specific input size required for the asymptotic behavior to become valid.

The Anatomy of Theta (Θ): The Asymptotic Tight Bound

Defining the "Average" Case through Upper and Lower Limits

Boundary Conditions

Upper Bound (O)

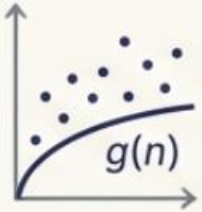


Represents the scenario where the equation grew larger.

$$c_2 \cdot g(n)$$

Highest power × Derived Constant

Lower Bound (Ω)



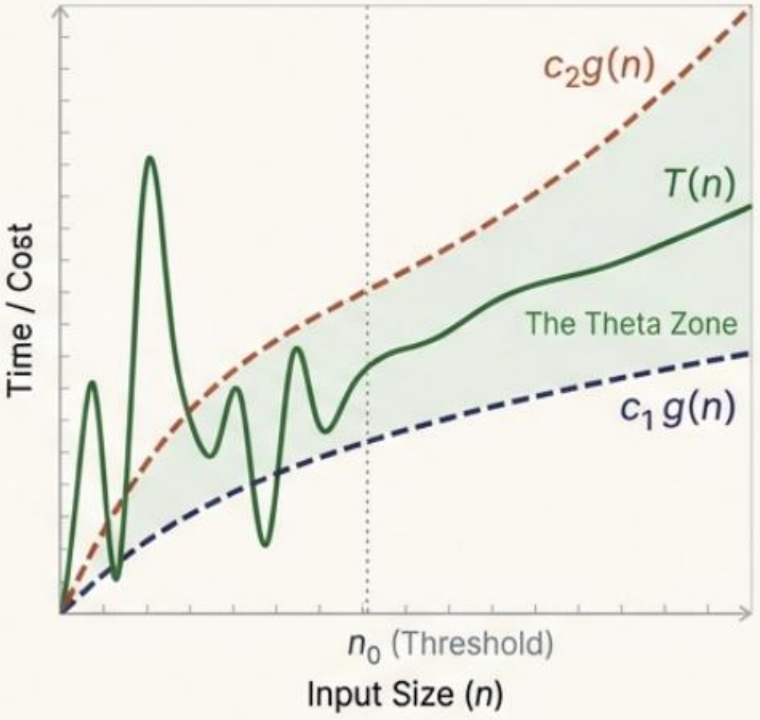
Represents the scenario where the equation grew smaller.

$$c_1 \cdot g(n)$$

Highest power × Calculated Constant

Defining the Range

Asymptotic Visualization



Theta is the "In-Between". It implies the function is bounded from above and below.

$$\Omega(g(n)) \leq \Theta(g(n)) \leq O(g(n))$$

For all $n \geq n_0$

Determining Theta

Mathematically, Theta acts as the median behavior. It ignores specific constants (c_1, c_2) to focus purely on the growth rate.

It resides in the middle ground between the Best Case and Worst Case bounds.

The "Close Your Eyes" Rule

If the Upper and Lower bounds converge to the same order, Theta is that order.

$$O(n) = n^2 \quad + \quad \Omega(n) = n^2$$

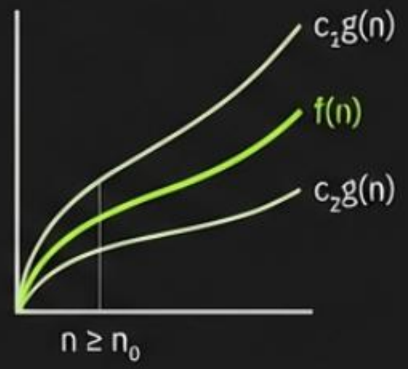
$$\Downarrow$$

$$\Theta(n) = n^2$$

If the ceiling and floor are the same shape, the room is that shape.

DIRECT INSPECTION: DOMINANT TERM ANALYSIS

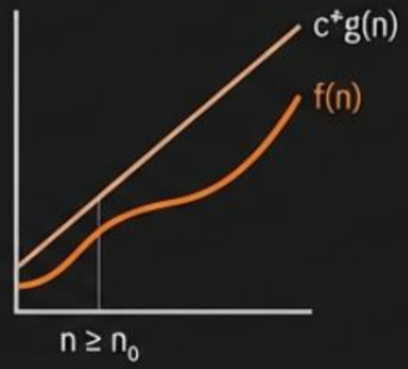
THETA (Exact Match)



Rule: Ignore constants and lower-order terms.
Time cannot be 0; upgrade 0 to 1.

```
1. T(n) = c_1 * n^2 + c_2 * n --> Theta(n^2)
2. T(n) = 10 --> Theta(1)
3. T(n) = sqrt(91) --> Theta(1)
```

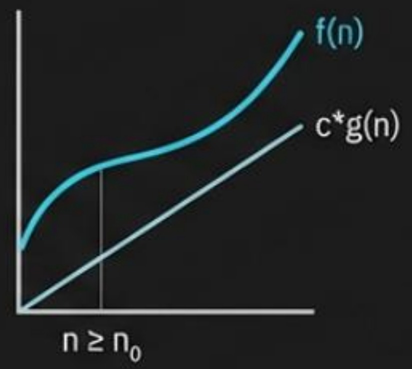
BIG-O (Upper Bound)



The Ceiling. Valid for the dominant term and anything faster.

```
T(n) = n^2 + n
Is O(n^2)? TRUE
Is O(n^3)? TRUE
```

BIG-OMEGA (Lower Bound)



The Floor. Valid for the dominant term and anything slower.

```
T(n) = 2^n
Is Omega(2^n)? TRUE
Is Omega(n^100)? TRUE (Exp > Poly)
```

ALGEBRAIC NUANCES: SIMPLIFY BEFORE CLASSIFYING

CASE 01: THE EXPONENT INSIDE THE LOG

$f(n) = \log(n^2)$

Not a quadratic function.

$\xrightarrow{\text{TRANSFORM}}$

$\dashrightarrow 2 * \log(n) \dashrightarrow$ Theta(log n)

Log Rule: log(a^b) = b * log(a). The square becomes a constant coefficient.

CASE 02: CONSTANTS DISGUISED AS EXPONENTIALS

$$T(n) = 2^{\log_2 3} \xrightarrow[]{\text{Swap Rule: } a^{\log_b n} = n^{\log_b a}} 3^{\log_2 2} = 3^1 = 3 \longrightarrow$$

COMPLEXITY:
Theta(1)
(Constant Time)

FORMAL DEFINITIONS: FINDING CONSTANTS c AND n_0

O(g(n)) - THE ADDITIVE METHOD

$$T(n) = c_1 * n^2 + c_2$$

Find c such that $T(n) \le c * n^2$

$$T(n) \le |c_1|n^2 + |c_2|n^2$$

$$\downarrow$$

$$T(n) \le (|c_1| + |c_2|)n^2$$

$$\downarrow$$

$$\boxed{\begin{array}{l} c = c_1 + c_2 \\ n_0 = 1 \end{array}}$$

Inflate the terms to find the Ceiling.

Omega(g(n)) - THE SUBTRACTIVE METHOD

$$T(n) = c_1 * n^2 + c_2$$

Find c such that $T(n) \ge c * n^2$

Drop non-dominant term $\cancel{c_2}$

$$\downarrow$$

$$T(n) \ge c_1 * n^2$$

$$\downarrow$$

$$\boxed{\begin{array}{l} c = c_1 \\ n_0 = 1 \end{array}}$$

Reduce the terms to find the Floor.

THE LIMITS METHOD: CALCULUS FOR EXPONENTIALS

Using L'Hôpital's Rule concepts to break ties.

Limit (n->inf)
[T(n) / g(n)]

Result = 0	Result = Constant	Result = Infinity
Small-o (Strictly Smaller). T(n) is O(g(n)).	Theta (Exact Match). T(n) is Theta(g(n)).	Small-omega (Strictly Larger). T(n) is Omega(g(n)).

Example: 2^(n+1) vs 2^n	Example: 2^(2n) vs 2^n
Ratio: (2 * 2^n) / 2^n = 2	Ratio: 4^n / 2^n = 2^n --> Infinity
Verdict: Constant --> Theta(2^n)	Verdict: Infinity --> Omega(2^n)

CONCEPTUAL HIERARCHY: TRUE/FALSE LOGIC

Scenario 1: f(n) = n
Check vs g(n) = 1 (Constant)
Logic: 1 is smaller than n. It sits on the Floor.
Verdict: Omega(1) is TRUE. O(1) is FALSE.

Scenario 2: f(n) = n
Check vs g(n) = log n
Logic: log n is smaller than n. It sits on the Floor.
Verdict: Omega(log n) is TRUE.

Scenario 3: f(n) = n^2
Is O(n^3)? TRUE (n^3 is in the Ceiling)
Is Omega(n)? TRUE (n is on the Floor)
Is Theta(n^3)? FALSE (Not exact)

MASTER MATRIX: COMPREHENSIVE PROBLEM SET

FUNCTION T(n)	COMPARISON g(n)	VERDICT	REASONING / NOTE
c_1 n^2 + c_2	n^2	Theta	Dominant term rule; ignore constants.
10 (Constant)	1	Theta	Time cannot be 0; upgrade 0 to 1.
sqrt(91)	1	Theta	Constant value disguised with a root.
log(n^2)	log n	Theta	Log rule: 2 log n. Drop coefficient.
2^(log_2 3)	1	Theta	Evaluates to constant 3.
2^(n+1)	2^n	Theta	Limit ratio is 2 (constant).
2^(2n) (=4^n)	2^n	Omega Only	Limit ratio is infinity. Too big for O.
n	1	Omega Only	n > 1. True for lower bound.
n	log n	Omega Only	n > log n. True for lower bound.
n^2 + n	n^2	Theta	Exact match logic.

Moving Beyond Equality: The Strict Asymptotic Notations

Contrasting Inclusive Bounds (O, Ω) with Strict Bounds (o, ω).

UPPER BOUNDS (Growth ≤ or <)

Big O (O)

$$f(n) \le c \cdot g(n)$$

Inclusive Upper Bound. Can be Tight (equal) or Loose.

Little o (o)

$$f(n) < c \cdot g(n)$$

Strictly Loose Upper Bound. Equality is forbidden.

Example Box

Given $f(n) = 3n + 2$

- $O(n)$: VALID (Tight bound allowed) ✓
- $o(n)$: INVALID (Equality forbids this) ✗
- $o(n^2)$: VALID (Strictly loose) ✓

LOWER BOUNDS (Growth ≥ or >)

Big Omega (Ω)

$$f(n) \ge c \cdot g(n)$$

Inclusive Lower Bound. Can be Tight (equal) or Loose.

Little Omega (ω)

$$f(n) > c \cdot g(n)$$

Strictly Loose Lower Bound. Equality is forbidden.

Example Box

Given $f(n) = 3n + 2$

- $\Omega(n)$: VALID (Tight bound allowed) ✓
- $\omega(n)$: INVALID (Equality forbids this) ✗
- $\omega(\sqrt{n})$: VALID (Strictly loose) ✓

Key Takeaway: Big notations describe the class itself. Little notations describe the strict gaps between classes.

The "No Equality" Rule in Practice

$$f(n) = 2^n + n^2$$

(Dominant term is 2^n).

Notation Type	Big O / Ω (Inclusive)	Little o / ω (Strict)	Reasoning
Candidate $g(n) = 2^n$	$O(2^n)$ and $\Omega(2^n)$ are **TRUE**.	$o(2^n)$ and $\omega(2^n)$ are **FALSE**.	Equality is allowed in Big notation but forbidden in Little notation. 2^n is not strictly greater than 2^n.
Candidate $g(n) = 3^n$	$O(3^n)$ is **TRUE**.	$o(3^n)$ is **TRUE**.	3^n grows strictly faster than 2^n. Valid Strict Upper Bound.
Candidate $g(n) = n^2$	$\Omega(n^2)$ is **TRUE**.	$\omega(n^2)$ is **TRUE**.	2^n grows strictly faster than n^2. Valid Strict Lower Bound.

The Exclusionary Rule

If a bound is **Tight** (matches the highest exponent), it qualifies for **Big** notation but automatically fails **Little** notation.

Proving Asymptotic Relationships via Limits

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)}$$

If result is 0

If result is Constant k
“ $(0 < k < \infty)$

If result is ∞

Scenario A: *L* = 0

Denominator $g(n)$ dominates numerator.

$$f(n) \in o(g(n))$$

Implies Big *O* is also true.

$$\lim_{n \to \infty} \frac{3n^2}{n^3} = 0$$

Scenario B: *L* = Constant

Same growth rate (Tight Bound).

$$\Theta(g(n))$$

Also implies *O* and Ω.

Little *o* and Little *ω* are **INVALID** here.

Scenario C: *L* = ∞

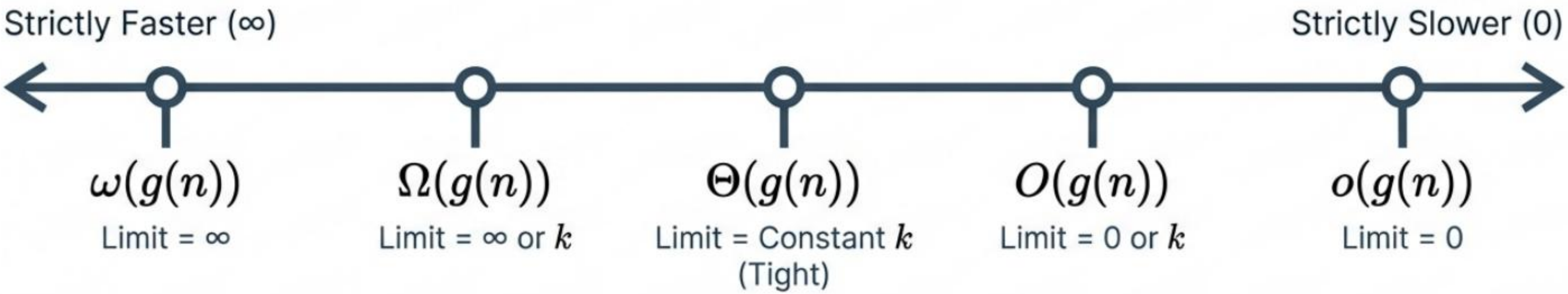
Numerator $f(n)$ dominates denominator.

$$f(n) \in \omega(g(n))$$

Implies Big Ω is also true.

Calculus Proof: Divide numerator and denominator by the highest power in the denominator to evaluate the limit.

Summary of Asymptotic Dominance



Limit Result	Meaning	o (Little)	O (Big)	Θ (Theta)	Ω (Big)	ω (Little)
0	$f < g$ (Strict)	✓	✓	✗	✗	✗
Constant (k)	$f = g$ (Tight)	✗	✓	✓	✓	✗
∞	$f > g$ (Strict)	✗	✗	✗	✓	✓

Logic Chain: $o \Rightarrow O$ but $O \neq o$. $\omega \Rightarrow \Omega$ but $\Omega \neq \omega$.

Questions

Identifying Asymptotic Dominance

Determine the complexity order by isolating the dominant term.

Function t(n)	Order
$t(n) = C_1 n^2 + C_2$	$O(\ldots\ldots)$
$t(n) = n^3 + n^4 + \sqrt{n}$	$O(\ldots\ldots)$
$t(n) = 10$	$O(\ldots\ldots)$
$t(n) = \sqrt{91}$	$O(\ldots\ldots)$
$t(n) = 2^n + n^2 + 3$	$O(\ldots\ldots)$
$t(n) = 2^{\log_3 n}$	$O(\ldots\ldots)$

Establishing Formal Bounds

$$t(n) = C_1 n^2 + C_2$$

Upper Bound (Big-O)

Find constants c, n_0 such that $t(n) \leq c \cdot n^2$

$$c = [\quad]$$

$$n_0 = [\quad]$$

Lower Bound (Big-Omega)

Find constants c, n_0 such that $t(n) \geq c \cdot n^2$

$$c = [\quad]$$

$$n_0 = [\quad]$$

Exponential Growth & Limit Theory

Is $2^{n+1} = O(2^n)$?

Evaluate the limit:

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n}$$

Proof Workspace

Is $2^{2n} = O(2^n)$?

Evaluate the limit:

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n}$$

Proof Workspace

Conceptual Verification

Verify the following relationships.

Functions	Proposed Relationship	Verdict
$F(n) = \log n^2, \quad g(n) = \log n + 5$	$F(n) = \Theta(g(n))$	☐ TRUE ☐ FALSE
$F(n) = n, \quad g(n) = \log n^2$	$F(n) = \Omega(g(n))$	☐ TRUE ☐ FALSE
$F(n) = n, \quad g(n) = 10$	$F(n) = O(g(n))$	☐ TRUE ☐ FALSE
$F(n) = 10, \quad g(n) = \log 10$	$F(n) = \Theta(g(n))$	☐ TRUE ☐ FALSE
$F(n) = 2^n, \quad g(n) = 3^n$	$F(n) = O(g(n))$	☐ TRUE ☐ FALSE
$F(n) = 2^n, \quad g(n) = 10n^2$	$F(n) = \Omega(g(n))$	☐ TRUE ☐ FALSE

Asymptotic Correctness Check

Determine whether the following statements are **True** or **False**.

$$\boxed{f(n) = 2^n + n^2}$$

Upper Bounds (O, o)

$$2^n + n^2 = O(2^n)$$

$$2^n + n^2 = O(3^n)$$

$$2^n + n^2 = o(2^n)$$

$$2^n + n^2 = o(3^n)$$

Lower Bounds (Ω, ω)

$$2^n + n^2 = \Omega(2^n)$$

$$2^n + n^2 = \Omega(n^2)$$

$$2^n + n^2 = \omega(2^n)$$

$$2^n + n^2 = \omega(n^2)$$

Proving Asymptotic Bounds

Establish the following relationships using the **Limit Definition**.

Method: $\lim_{n \to \infty} \frac{f(n)}{g(n)}$

Problem A: Little-o (Strict Upper Bound)

$$3n^2 + n = o(n^3)$$

Problem B: Little-omega (Strict Lower Bound)

$$n^3 + n^2 = \omega(n^2)$$