Asymptotic Notations Overview

Asymptotic notations provide a simple, communicable way to represent the **time complexity** of algorithms using mathematical function classes.

They belong to mathematics but are used in algorithms for comparing functions and showing growth rates.

Three main notations:

- **Big O**: Upper bound
- **Big Omega (Ω)**: Lower bound
- **Theta (Θ)**: Tight/average bound

Key principle: Any algorithm's time complexity belongs to one (or a multiple) of these; prefer the tightest bound for usefulness.

Big O Notation (Upper Bound)

Definition: $f(n) = O(g(n))$ if there exist positive constants C and n_0 such that $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$.

Step-by-Step Understanding

1. **Identify dominant term** in $f(n)$ (highest growth rate).
2. **Choose single-term $g(n)$** that bounds $f(n)$ from above (can include coefficient C).
3. **Verify inequality** holds for all $n \geq n_0$.

Example: $f(n) = 2n + 3$

- Convert lower terms to dominant: $2n + 3 \leq 5n$ (for $n \geq 1$).
- Here, $C=5$, $g(n)=n$, $n_0=1$.
- Thus, $f(n) = O(n)$.

Alternative (looser) bounds:

- $2n + 3 \leq 5n^2$ → $f(n) = O(n^2)$ (true but less tight).

- $$f(n) = O(2^n)$$ also true (any faster-growing function works).
- **Wrong**: $$O(\log n)$$ (grows slower, violates upper bound).

Best practice: Choose **tightest (closest) upper bound** for usefulness.

Big Omega Notation (Lower Bound)

Definition: $$f(n) = \Omega(g(n))$$ if there exist positive constants $$C$$ and $$n_0$$ such that $$f(n) \geq C \cdot g(n)$$ for all $$n \geq n_0$$.

Difference from Big O: Inequality reverses ($$\geq$$ instead of $$\leq$$).

Step-by-Step Understanding

1. **Choose $$g(n)$** that $$f(n)$$ grows at least as fast as.
2. **Use dominant term** with minimal $$C$$ (often 1).

Example: $$f(n) = 2n + 3$$

- $$2n + 3 \geq 1 \cdot n$$ (for $$n \geq 0$$ or $$1$$).
- Thus, $$f(n) = \Omega(n)$$.

Valid looser bounds:

- $$f(n) = \Omega(\log n)$$ (true, since $$n$$ grows faster).
- $$\Omega(\sqrt{n})$$ also true.

Wrong: $$\Omega(n^2)$$ ($$f(n)$$ doesn't grow that fast).

Best practice: Tightest lower bound is most useful.

Theta Notation (Tight Bound)

Definition: $$f(n) = \Theta(g(n))$$ if there exist positive constants $$C_1$$, $$C_2$$, $$n_0$$ such that $$C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$$ for all $$n \geq n_0$$.

Combines both bounds using **same $$g(n)$$.

Step-by-Step Understanding

1. **Find lower bound**: $$f(n) \geq C_1 \cdot g(n)$$.
2. **Find upper bound**: $$f(n) \leq C_2 \cdot g(n)$$.
3. **Same $$g(n)$$ on both sides** → tight bound.

Example: $$f(n) = 2n + 3$$

- Lower: $$2n + 3 \geq 1 \cdot n$$
- Upper: $$2n + 3 \leq 5 \cdot n$$
- Thus, $$1n \leq 2n + 3 \leq 5n$$ → $$f(n) = \Theta(n)$$.

Restrictions:

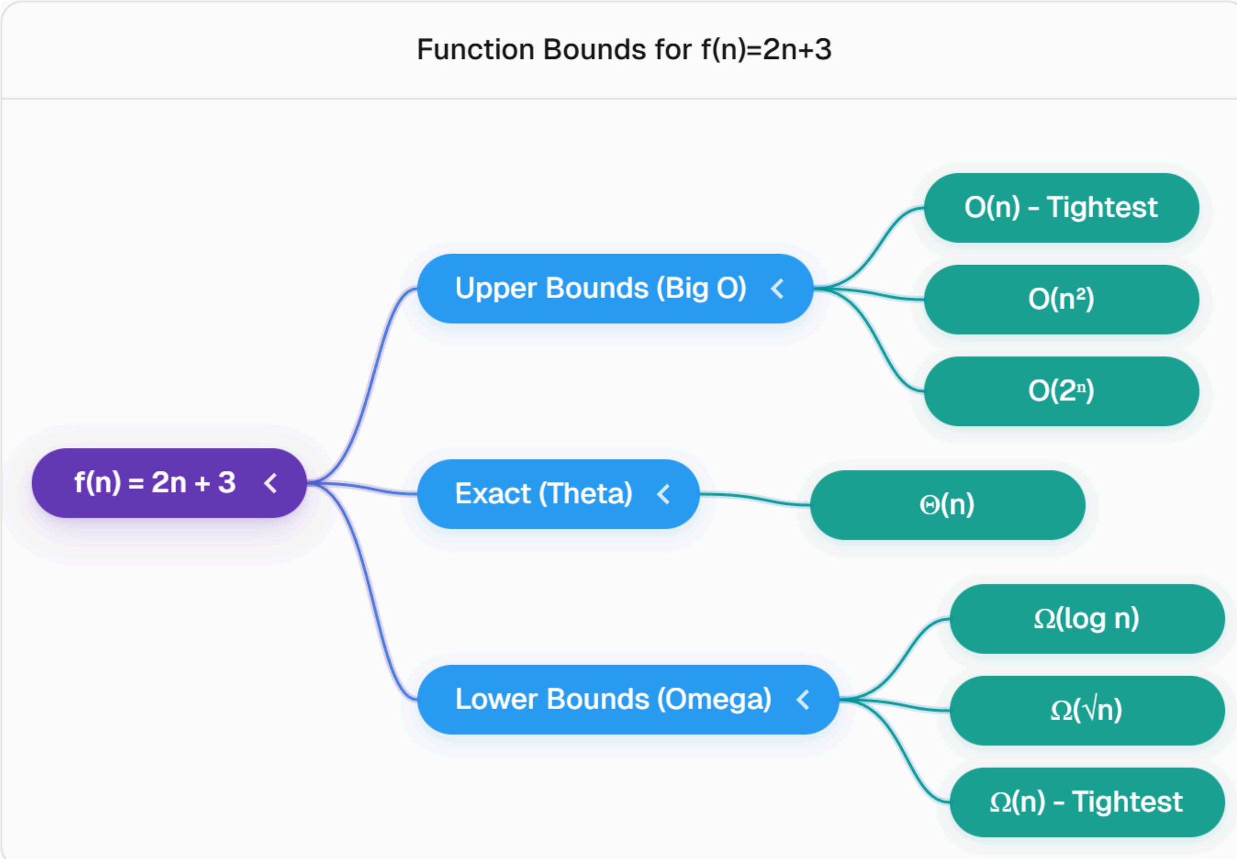
- Cannot use $$n^2$$ or $$\log n$$ (bounds don't match).
- **Most precise** and **preferred** when possible.

Comparison of Notations

Notation	Bound Type	Inequality	Example for $$f(n)=2n+3$$	Tightness
Big O	Upper	$$f(n) \leq C \cdot g(n)$$	$$O(n)$$, $$O(n^2)$$, $$O(2^n)$$	Loose to tight
Omega	Lower	$$f(n) \geq C \cdot g(n)$$	$$\Omega(n)$$, $$\Omega(\log n)$$	Loose to tight
Theta	Tight	$$C_1 g(n) \leq f(n) \leq C_2 g(n)$$	$$\Theta(n)$$ only	Exact match

Visual function relationships (for $$f(n)=2n+3$$):

Rule: Prefer **Theta** → **Big O** → **Omega** (in that order of precision).



Common Misconceptions

- **Not related to best/worst case**: These are function bounds, not algorithm cases (e.g., Big O can describe best or worst case).
- **Multiple valid representations**: Always true for looser bounds, but choose closest for analysis.
- **Single-term focus**: Simplify to one dominant term with coefficient.