Characterizing Running Times

Definitions, Analysis, and Asymptotic Efficiency

Tarek Hany

January 27, 2026

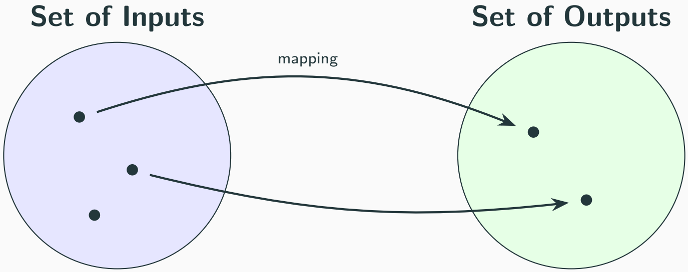
Recap: Core Definitions

What is an Algorithm?
An **algorithm** is a sequence of finite procedures designed to solve a computational problem.

What is a Computational Problem?
A **problem** is formally defined as a relation between:

- A set of **inputs**
- A set of **outputs**

Visualizing a Problem: The Relation



The **Problem** defines this relation.

The **Algorithm** computes it.

Example: The Sorting Problem

Problem Statement:

- **Input:** A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Instance Example:

$$\text{Input: } [31, 41, 59, 26, 41, 58]$$

$$\text{Solution: } [26, 31, 41, 41, 58, 59]$$

Measures of a Good Algorithm

When designing algorithms, we evaluate them based on several criteria:

1. **Clarity:** Is the process easy to understand?
2. **Simplicity:** Is the implementation straightforward?
3. **Correctness:** Does it produce the correct output for *every* input?
4. **Efficiency:** How much time and memory does it require?

Focus for Today

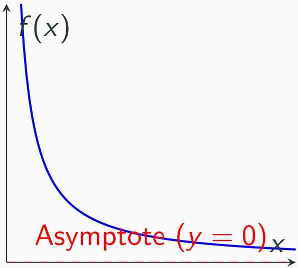
We established **Correctness** in the last lecture. Today, we will focus specifically on **Efficiency**.

Etymology: What is an Asymptote?

Origin: From the Greek *asymptōtos*, meaning "not falling together."

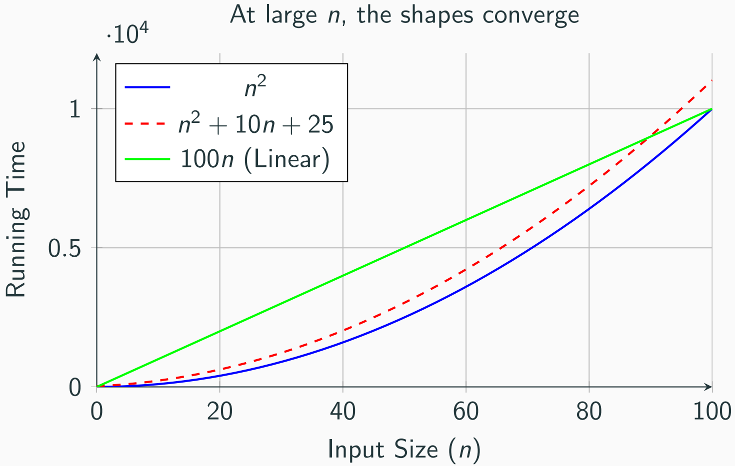
Definition: A line that a curve approaches arbitrarily closely as it heads towards infinity, but never quite touches.

Example: $f(x) = \frac{1}{x}$
As $x \to \infty$, $f(x) \to 0$.



Visualizing Dominance at Infinity

Why do we ignore coefficients and lower-order terms?
Consider a quadratic function $f(n) = n^2$ versus a "complex" one $g(n) = n^2 + 10n + 25$.



From Exact Time to Order of Growth

- **Exact Running Time:** computing every step and constant is difficult and rarely worth the effort.
- **The Observation:** For large enough inputs, multiplicative constants and lower-order terms are *dominated* by the input size itself.

Simplifying the Analysis
We focus on the **Order of Growth**. This characterizes efficiency simply and allows us to compare algorithms easily.

Asymptotic Efficiency

Definition: We study how the running time increases as the size of the input (n) increases without bound ($n \to \infty$).

Comparison: Large Inputs
Even if an algorithm has large constants, a better growth rate wins eventually:

- **Merge Sort:** $\Theta(n \lg n)$
- **Insertion Sort:** $\Theta(n^2)$

Once n is large enough, Merge Sort **beats** Insertion Sort.

[0] Usually, the asymptotically more efficient algorithm is the best choice for all but very small inputs.

Understanding Bounds: Ceiling and Floor

Before defining formal notation, let's visualize "Bounds" with a simple analogy.

Upper Bound (The Ceiling):

- No matter how tall you grow, you will never pass the ceiling.
- In algorithms: The runtime will **never exceed** this rate.

Lower Bound (The Floor):

- You will always stand at least at this level.
- In algorithms: The runtime will **at least** take this long.

Asymptotic notations (O, Ω, Θ) are just mathematical ways to describe these ceilings and floors for functions.

O-notation: The Upper Bound

Definition: O-notation characterizes an **upper bound** on the asymptotic behavior of a function.

- It says a function grows **no faster than** a certain rate.

Example: $f(n) = 7n^3 + 100n^2 - 20n + 6$

- Highest-order term is $7n^3$, so the growth rate is n^3.
- Since it grows no faster than n^3, it is $O(n^3)$.

Note: It is also $O(n^4)$, $O(n^5)$, etc., because if it doesn't exceed n^3, it certainly doesn't exceed n^4.

Ω-notation: The Lower Bound

Definition: Ω-notation characterizes a **lower bound** on the asymptotic behavior of a function.

- It says a function grows **at least as fast as** a certain rate.

Example: $f(n) = 7n^3 + 100n^2 - 20n + 6$

- The function grows at least as fast as n^3.
- Therefore, it is $\Omega(n^3)$.

Note: It is also $\Omega(n^2)$ and $\Omega(n)$, because if it grows as fast as n^3, it definitely outpaces n^2.

Θ-notation: The Tight Bound

Definition: Θ-notation characterizes a **tight bound**.

- It describes the rate of growth precisely (within constant factors).
- It sandiwches the function between an upper and lower bound.

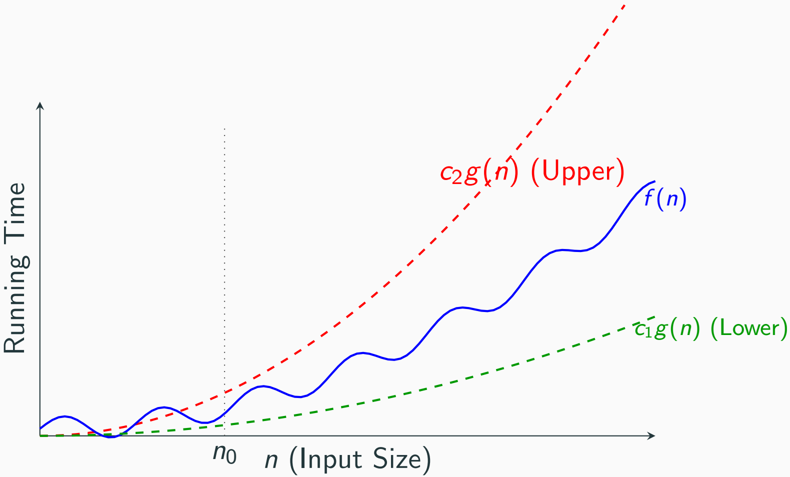
The Theorem

If a function $f(n)$ is both $O(g(n))$ AND $\Omega(g(n))$, then:

$$f(n) = \Theta(g(n))$$

Back to our Example: Since $f(n)$ is $O(n^3)$ AND $\Omega(n^3)$, it is $\Theta(n^3)$.

Visualizing Tight Bounds (Θ-notation)



Definition:

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{for all } n \geq n_0$$

Example: Analyzing Insertion Sort

Let's apply asymptotic notation to **Insertion Sort** without complex summations.

The Procedure:

```
for i = 2 to n
    key = A[i]
    j = i - 1
    while j > 0 and A[j] > key
        A[j + 1] = A[j]  // Shift elements
        j = j - 1
    A[j + 1] = key
```

Key Observation: The runtime depends entirely on how many times the inner `while` loop executes (the "Shift" operation).

Step 1: The Upper Bound (O)

To find the Upper Bound, we ask: *"What is the absolute maximum number of operations possible?"*

- The **Outer Loop** runs n times.
- The **Inner Loop** runs at most i times (where $i \approx n$ in the end).

Rough Calculation

$$\text{Total Steps} \approx n \times n = n^2$$

Since the algorithm **never** exceeds this quadratic growth, we say:

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

Step 2: The Lower Bound (Ω)

To find the Lower Bound for the *Worst Case*, we ask: *"Is there a specific input that forces the algorithm to work hard?"*

The Worst Case Input: Reverse Sorted Array

$$[n, n-1, \ldots, 3, 2, 1]$$

- To insert the next number, it must move **all the way** to the start.
- Element 2 moves past 1 item.
- Element 3 moves past 2 items...
- Element n moves past $n-1$ items.

Since we **must** perform proportional to $n^2/2$ shifts:

$$T(n) = \Omega(n^2) \quad \text{(in the worst case)}$$

Step 3: The Tight Bound (Θ)

Conclusion

We have established two facts for the **Worst Case**:

1. Upper Bound: It is $O(n^2)$ (Never takes longer).
2. Lower Bound: It is $\Omega(n^2)$ (There is a case where it takes this long).

Therefore, the Worst-Case Running Time is:

$$\Theta(n^2)$$

[0] Note: This does not mean Insertion Sort is $\Theta(n^2)$ in *all* cases. In the Best Case (already sorted), it is $\Theta(n)$.

But... What about the Best Case?

We proved the Worst Case is $\Theta(n^2)$. Does Insertion Sort *always* take this long?

The Best Case Input: Already Sorted Array

$$[1, 2, 3, \ldots, n]$$

- **Work Done:** We just loop through the array once.

Best Case Complexity

$$T(n) = \Theta(n) \quad \text{(Linear Time)}$$

Key Takeaway: Because the Best Case ($\Theta(n)$) and Worst Case ($\Theta(n^2)$) are different, we cannot give a single Θ bound for Insertion Sort in general! We can only give a general $O(n^2)$ upper bound.

Formalizing the Notation

The Mathematical Reality: Asymptotic notations are actually **Sets of Functions**.

- Technically, we should write: $f(n) \in O(g(n))$
- Conventionally, we write: $f(n) = O(g(n))$

Standard "Abuse" of Notation

When we write $f(n) = O(g(n))$, we are not saying "f(n) is equal to O(g(n))."

We mean: **"f(n) is a member of the set O(g(n))."**

Formal Definition: O-notation

Definition: $O(g(n))$ is the set of functions:

$$O(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

In English: There exist positive constants c and n_0 such that $f(n)$ is always **on or below** $c \cdot g(n)$ for large enough n.

Example from Text:

$$4n^2 + 100n + 500 = O(n^2)$$

Proof on next slide...

Example: Proving the Bound

Goal: Find c and n_0 such that:

$$4n^2 + 100n + 500 \leq cn^2$$

Step 1: Divide by n^2

$$4 + \frac{100}{n} + \frac{500}{n^2} \leq c$$

Step 2: Choose n_0 and calculate c

- If we pick $n_0 = 1$, the left side is $4 + 100 + 500 = 604$.
- So, $c = 604$ works.

Conclusion

Since we found a pair $(c = 604, n_0 = 1)$ that makes the inequality true, the function is indeed $O(n^2)$.

Formal Definition: Ω-notation

Definition: $\Omega(g(n))$ is the set of functions:

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$

In English: There exist positive constants c and n_0 such that $f(n)$ is always **on or above** $c \cdot g(n)$.

Example: $4n^2 + 100n + 500 = \Omega(n^2)$
(Just pick $c = 4$ and $n_0 = 1$, and clearly $4n^2 ... \geq 4n^2$)

Formal Definition: Θ-notation

Definition: $\Theta(g(n))$ is the set of functions:

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ such that}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$$

Theorem 3.1
For any two functions $f(n)$ and $g(n)$:

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ **AND** } f(n) = \Omega(g(n))$$

This confirms that a Tight Bound (Θ) requires both an Upper Bound (O) and a Lower Bound (Ω).

Precision in Reporting Running Times

We must be precise about *which* case we are describing.

Correct Statements:

- "Worst Case is $\Theta(n^2)$"
- "Best Case is $\Theta(n)$"
- "General Runtime is $O(n^2)$"

INCORRECT Statement:

- "Insertion Sort is $\Theta(n^2)$"

Why is it incorrect?
Saying "Insertion Sort is $\Theta(n^2)$" implies it is n^2 for **ALL** inputs.
Since the best case is $\Theta(n)$, this statement is false.

Asymptotic Notation in Equations (RHS)

We often use notation like $\Theta(n)$ inside equations to represent **Anonymous Functions**.

Example:

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

Interpretation: The term $\Theta(n)$ stands for *some specific function* $f(n)$ that we don't care to name, but we know belongs to the set $\Theta(n)$.

In this case, the anonymous function is $f(n) = 3n + 1$, which is indeed $\Theta(n)$.

Formal Rule: The Right-Hand Side

When asymptotic notation appears on the **Right-Hand Side (RHS)** of an equation:

$$\cdots = \cdots + \Theta(g(n))$$

Meaning (Existential): "There **exists** some function $f(n) \in \Theta(g(n))$ that makes this equation true."

Why do this? It allows us to hide lower-order details during intermediate steps of a proof.

Asymptotic Notation on the Left-Hand Side

What if the notation appears on the **Left-Hand Side (LHS)**?

Example:

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

Meaning (Universal): "No matter which function $f(n) \in \Theta(n)$ you choose for the left side, there is a way to equate it to the right side."

Rule of Thumb
The LHS represents **any** function in that set. The RHS represents **some** function that satisfies the equation.

Reading a Chain of Equations

We can chain these together to simplify expressions step-by-step.

Example:

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

How to read it:

1. **Step 1:** $2n^2 + 3n + 1$ can be written as $2n^2 + f(n)$ where $f(n) \in \Theta(n)$.
2. **Step 2:** $2n^2 + f(n)$ is a member of the set $\Theta(n^2)$.

Common "Abuses" of Notation

In mathematics, "abuse of notation" means using notation slightly loosely for clarity, as long as the meaning is clear.

Case 1: Small Inputs Statement: "$T(n) = O(1)$ for $n < 3$"
Meaning: For small inputs, the runtime is bounded by **some constant**. We don't care what the constant is.

Case 2: Defined Domains Statement: "Merge Sort is $\Theta(n \lg n)$" (even if n isn't a power of 2).
Meaning: The bound holds for the domain where the function is defined.

Little-o Notation (o): Strict Upper Bound

Concept: O-notation is like "$\leq$". It allows equality ($n^2 = O(n^2)$).
o-**notation** is like "$<$". It does **not** allow equality.

Formal Definition: $o(g(n))$
$f(n) = o(g(n))$ if for **ALL** constants $c > 0$, there exists an n_0 such that:

$$0 \leq f(n) < cg(n) \quad \text{for all } n \geq n_0$$

Intuitive Limit Definition:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

(The function $f(n)$ becomes insignificant relative to $g(n)$).

Example: $2n = o(n^2)$, but $2n^2 \neq o(n^2)$.

Little-omega Notation (ω): Strict Lower Bound

Concept: Ω-notation is like "$\geq$".

ω-**notation** is like "$>$". It denotes a lower bound that is *not tight*.

Formal Definition: $\omega(g(n))$
$f(n) = \omega(g(n))$ if for **ALL** constants $c > 0$, there exists an n_0 such that:

$$0 \leq cg(n) < f(n) \quad \text{for all } n \geq n_0$$

Intuitive Limit Definition:

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

(The function $f(n)$ becomes arbitrarily large relative to $g(n)$).

Example: $n^2/2 = \omega(n)$, but $n^2/2 \neq \omega(n^2)$.

Analogy to Real Numbers

The relational properties of asymptotic notations behave very similarly to comparing real numbers a and b.

Asymptotic Notation	**Analogy**	**Meaning**
$f(n) = O(g(n))$	$a \leq b$	Grows no faster than
$f(n) = \Omega(g(n))$	$a \geq b$	Grows at least as fast as
$f(n) = \Theta(g(n))$	$a = b$	Grows at same rate
$f(n) = o(g(n))$	$a < b$	Grows strictly slower than
$f(n) = \omega(g(n))$	$a > b$	Grows strictly faster than

Mathematical Properties

Assuming $f(n)$ and $g(n)$ are asymptotically positive:

1. **Transitivity:** (Applies to all)

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \implies f(n) = \Theta(h(n))$$

2. **Reflexivity:**

$$f(n) = \Theta(f(n)), \quad f(n) = O(f(n)), \quad f(n) = \Omega(f(n))$$

3. **Transpose Symmetry:**

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

Trichotomy: Where the Analogy Fails

For Real Numbers (Trichotomy Property): For any numbers a, b, exactly one must be true:

$$a < b, \quad a = b, \quad \text{or} \quad a > b$$

Does this hold for functions?
NO. Not all functions are asymptotically comparable.

Counter-Example:

$$f(n) = \sin x, \quad g(n) = \cos x$$

Since the exponent oscillates between 0 and 2, $g(n)$ is sometimes larger and sometimes smaller than $f(n)$.
Result: Neither O, Ω, nor Θ applies.

Standard Functions: A Quick Refresher

1. **Monotonicity**
   - **Monotonically Increasing:** If $m \leq n \implies f(m) \leq f(n)$.
   - *Why we care:* Most algorithm runtimes are monotonic (sorting 100 items takes longer than sorting 10).

2. **Floors and Ceilings**
   - Floor $\lfloor x \rfloor$: Greatest integer $\leq x$.
   - Ceiling $\lceil x \rceil$: Least integer $\geq x$.

Note: For large n, we usually ignore floors/ceilings in asymptotic notation (e.g., $n/2$ vs $\lfloor n/2 \rfloor$ doesn't change the O-notation).

Exponentials vs. Polynomials

The most important race in Computer Science:

$$n^b \quad \text{vs} \quad a^n \quad (\text{where } a > 1)$$

The Golden Rule

Any exponential function with base > 1 grows **faster** than any polynomial.

Formally:

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 \implies n^b = o(a^n)$$

Example: $n^{100} = o(2^n)$. Even a huge polynomial loses to a small exponential eventually.

Logarithms: The CS Notation

In algorithms, we use specific shorthands:

- $\lg n = \log_2 n$ (Binary Log - most common in CS)
- $\ln n = \log_e n$ (Natural Log)
- $\lg^k n = (\lg n)^k$ (Exponentiation)
- $\lg \lg n = \lg(\lg n)$ (Composition)

Why Base Doesn't Matter in Big-O
The Change of Base formula:

$$\log_b n = \frac{\log_c n}{\log_c b}$$

Since $\frac{1}{\log_c b}$ is just a **constant factor**, we typically say $O(\log n)$ without specifying the base.

Factorials ($n!$)

Permutations lead to factorials: $n! = 1 \cdot 2 \cdot \cdots \cdot n$.

How fast does $n!$ grow? It grows faster than 2^n but slower than n^n.

Stirling's Approximation (Simplified):

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Key Takeaway for Analysis:

$$\lg(n!) = \Theta(n \lg n)$$

(This is crucial for proving the lower bound of sorting algorithms later).

The Iterated Logarithm ($\lg^* n$)

Definition: Let $\lg^{(i)} n$ be the log function applied i times.
$\lg^* n$ (read "log star of n") is the number of times you must apply lg to n before the result is ≤ 1.

Growth Rate: Extremely Slow.

- $\lg^* 2 = 1$
- $\lg^* 4 = 2$
- $\lg^* 16 = 3$
- $\lg^* 65536 = 4$
- $\lg^*(2^{65536}) = 5$

Fun Fact: Since 2^{65536} is greater than the number of atoms in the observable universe (10^{80}), for all practical purposes:

$$\lg^* n \leq 5$$

Fibonacci Numbers

Defined by the recurrence:

$$F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i-2}$$

Connection to Nature (Golden Ratio):

$$F_i \approx \frac{\phi^i}{\sqrt{5}} \quad \text{where } \phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

Asymptotic Growth

Fibonacci numbers grow **exponentially**.

$$F_i = \Theta(\phi^i)$$

(This is why naive recursive Fibonacci algorithms are very slow).

Thank You!

Questions?