Lecture 2 - Chpt 3: Characterizing Runing Times

Harley Caham Combest

Fa2025 CS4413 Lecture Notes – Mk1

. . . .

Chapter 3: Characterizing Running Times

. . . .

3.1 O-notation, Ω-notation, and Θ-notation

. Historical Context. The order of growth of an algorithm's running time provides a simple and powerful way to characterize efficiency. Constants and lower-order terms become irrelevant for sufficiently large n, leaving only the asymptotic behavior. This abstraction lets us compare algorithms meaningfully, independent of machine or implementation details.

Asymptotic Efficiency. When input sizes are large, the multiplicative constants and lower-order terms of exact running times are dominated by the input size itself.

- For example, insertion sort runs in $\Theta(n^2)$ time in the worst case, while merge sort runs in $\Theta(n \log n)$. For large enough n, merge sort will always outperform insertion sort.
- Thus, we focus on *asymptotic notation*, which captures growth rates and allows precise comparison.

Algorithm 1 Insertion-Sort(A, n)

```
for $i = 2$ to $n$ do
    $key \leftarrow A[i]$
    $j \leftarrow i - 1$
    while $j > 0$ and $A[j] > key$ do
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j - 1$
    end while
    $A[j+1] \leftarrow key$
end for
```

Analysis of Insertion Sort.

- The *outer loop* executes $n-1$ times.
- The *inner loop* executes at most $i-1$ times on iteration i.
- Total number of inner-loop iterations is bounded above by $(n-1)(n-1) < n^2$. Each iteration costs constant time, so the running time is $O(n^2)$.

Lower Bound Argument. To show a $\Omega(n^2)$ lower bound, suppose the input of size n has its largest $n/3$ elements in the first $n/3$ positions.

- Each of these values must move into the last $n/3$ positions.
- To do so, each must pass through the middle $n/3$ positions, one step at a time.
- Hence at least $(n/3)(n/3) = n^2/9$ moves are required.

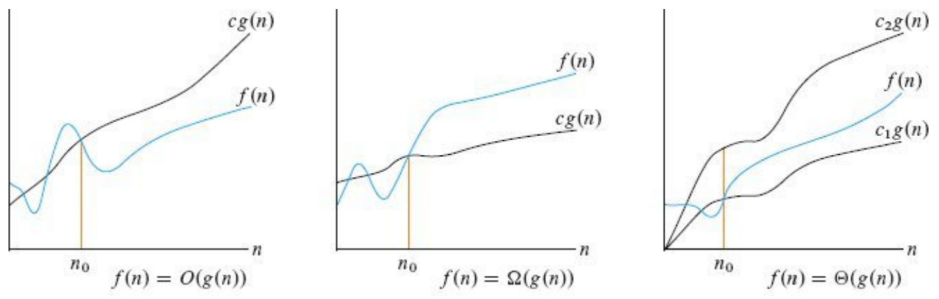
Thus, the worst-case running time of insertion sort is $\Omega(n^2)$.

Proposition 1. *Since insertion sort is $O(n^2)$ in all cases and $\Omega(n^2)$ in the worst case, its worst-case running time is $\Theta(n^2)$.*

Concluding Remark. Asymptotic notations abstract away machine-dependent constants, exposing the essential growth behavior. Insertion sort demonstrates $\Theta(n^2)$ worst-case time, while merge sort's $\Theta(n \log n)$ makes it asymptotically more efficient. These notations form the foundation for algorithm analysis.

3.2 Asymptotic notation: formal definitions

. **Motivation.** The informal use of asymptotic notation in §3.1 allows us to discard lower-order terms and constants. We now give precise mathematical definitions, expressed in terms of sets of functions.



Definition 1 (Big-O Notation)**.** For a given function $g(n)$, we define

$$O(g(n)) = \{f(n) : \exists\, c > 0,\ n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}.$$

We say $f(n) = O(g(n))$ if $f(n)$ grows no faster than $g(n)$, up to constant factors.

Definition 2 (Big-Ω Notation)**.**

$$\Omega(g(n)) = \{f(n) : \exists\, c > 0,\ n_0 > 0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$

We say $f(n) = \Omega(g(n))$ if $f(n)$ grows at least as fast as $g(n)$, up to constant factors.

Definition 3 (Big-Θ Notation)**.**

$$\Theta(g(n)) = \{f(n) : \exists\, c_1, c_2 > 0,\ n_0 > 0 \text{ such that } 0 \le c_1g(n) \le f(n) \le c_2g(n) \text{ for all } n \ge n_0\}.$$

We say $f(n) = \Theta(g(n))$ if $f(n)$ grows precisely at the same asymptotic rate as $g(n)$, up to constant factors.

Theorem 2 (Equivalence of Θ-notation)**.** *For functions $f(n)$ and $g(n)$,*

$$f(n) = \Theta(g(n)) \quad \Longleftrightarrow \quad f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

Definition 4 (Little-o Notation)**.**

$$o(g(n)) = \{f(n) : \forall c > 0,\ \exists n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0\}.$$

We say $f(n) = o(g(n))$ if $f(n)$ grows strictly slower than $g(n)$, i.e. becomes insignificant relative to $g(n)$ as $n \to \infty$.

Definition 5 (Little-ω Notation)**.**

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

We say $f(n) = \omega(g(n))$ if $f(n)$ grows strictly faster than $g(n)$, i.e. eventually dominates $g(n)$ as $n \to \infty$.

Examples.

- $4n^2 + 100n + 500 = O(n^2)$, since constants and lower-order terms can be absorbed.
- The same function is also $\Omega(n^2)$, hence $\Theta(n^2)$.
- $2n = o(n^2)$ because $2n/n^2 \to 0$ as $n \to \infty$.
- $n^2 = \omega(n)$ because $n^2/n \to \infty$.

Asymptotic Notation in Equations. We often use equality signs for membership or shorthand:

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

Interpretation: the term $\Theta(n)$ denotes some anonymous function of order n, and the final simplification captures the dominant order.

Properties of Asymptotic Notation. For asymptotically positive functions f, g, h:

- *Transitivity:* $f = \Theta(g)$ and $g = \Theta(h) \implies f = \Theta(h)$; similarly for O, Ω, o, ω.
- *Reflexivity:* $f = O(f)$, $f = \Omega(f)$, $f = \Theta(f)$.
- *Symmetry:* $f = \Theta(g) \iff g = \Theta(f)$.
- *Transpose symmetry:* $f = O(g) \iff g = \Omega(f)$, and $f = o(g) \iff g = \omega(f)$.

Non-comparability. Not all functions are asymptotically comparable. For example, n and $n^{1+\sin n}$ cannot be related by O, Ω, or Θ, since the exponent oscillates between 0 and 2.

Proper Abuses of Notation.

- Writing $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.
- Omitting the variable tending to infinity (e.g. $O(1)$).
- Writing $T(n) = O(1)$ for $n < 3$, interpreted as "bounded by a constant" for small n.

- Using asymptotic notation even when a function is defined only on a restricted domain (e.g. n a power of 2).

Concluding Remark. Asymptotic notation formalizes the intuitive act of discarding inessential details. Big notations give loose or tight bounds, while little notations capture strict dominance. The rules and conventions ensure consistency, while accepted abuses keep the notation concise without losing correctness.

3.3 Standard notations and common functions

. **Motivation.** When analyzing algorithms, certain functions arise repeatedly. This section reviews their definitions, properties, and asymptotic growth, providing a toolkit for later proofs and comparisons.

Monotonicity

Definition 6 (Monotonicity)**.** A function $f(n)$ is:

- *Monotonically increasing* if $m \le n \implies f(m) \le f(n)$.
- *Monotonically decreasing* if $m \le n \implies f(m) \ge f(n)$.
- *Strictly increasing* if $m < n \implies f(m) < f(n)$.
- *Strictly decreasing* if $m < n \implies f(m) > f(n)$.

Floors and Ceilings

Definition 7 (Floor and Ceiling)**.** For real x:

$$\lfloor x \rfloor = \max\{m \in \mathbb{Z} : m \le x\}, \quad \lceil x \rceil = \min\{m \in \mathbb{Z} : m \ge x\}.$$

Properties:

$$\lfloor x \rfloor \le x \le \lceil x \rceil, \qquad \lfloor x \rfloor + \lceil x \rceil = \lfloor 2x \rfloor \text{ or } \lfloor 2x \rfloor + 1.$$

Modular Arithmetic

Definition 8 (Modulo)**.** For integers a and $n > 0$, the value $a \bmod n$ is the remainder of a/n.

We say $a \equiv b \pmod{n}$ if $n \mid (a - b)$, i.e. a and b leave the same remainder upon division by n.

Polynomials

A polynomial of degree d is

$$p(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_0, \quad a_d \ne 0.$$

If $a_d > 0$, then $p(n)$ is asymptotically positive and

$$p(n) = \Theta(n^d).$$

A function is *polynomially bounded* if $f(n) = O(n^k)$ for some constant k.

Exponentials

For $a > 0$ and integers m, n:

$$a^0 = 1, \quad a^1 = a, \quad a^{-1} = \tfrac{1}{a}, \quad (a^m)^n = a^{mn}, \quad a^m a^n = a^{m+n}.$$

If $a > 1$, a^n is monotonically increasing in n.

Proposition 3. *For all real $a > 1$ and b, we have*

$$n^b = o(a^n).$$

Thus, every exponential outgrows every polynomial.

Logarithms

We use:

$$\lg n = \log_2 n, \quad \ln n = \log_e n, \quad \lg^k n = (\lg n)^k, \quad \lg \lg n = \lg(\lg n).$$

Properties:

$$\log_b a = \frac{\log_c a}{\log_c b}, \quad \text{for } a, b, c > 0,\ b, c \neq 1.$$

Thus, base choice only changes constants in asymptotic analysis.

Definition 9 (Polylogarithmic bound)**.** A function is polylogarithmically bounded if $f(n) = O(\lg^k n)$ for some constant k.

Factorials

$$n! = 1 \cdot 2 \cdot 3 \cdots n, \quad 0! = 1.$$

A weak bound: $n! \leq n^n$. Stronger bounds use Stirling's approximation:

$$n! \sim \sqrt{2\pi n}\left(\tfrac{n}{e}\right)^n.$$

Hence, $n!$ grows faster than polynomial but slower than n^n.

Functional Iteration

For function $f(n)$ and nonnegative integer i:

$$f^{(0)}(n) = n, \quad f^{(i)}(n) = f(f^{(i-1)}(n)).$$

Example: if $f(n) = 2n$, then $f^{(i)}(n) = 2^i n$.

Iterated Logarithm

Definition 10 (Iterated logarithm)**.**

$$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}.$$

Values: $\lg^* 2 = 1$, $\lg^* 4 = 2$, $\lg^* 16 = 3$, $\lg^* 65536 = 4$. Even for astronomically large n, $\lg^* n \leq 5$.

Fibonacci Numbers

Defined recursively:

$$F_0 = 0, \quad F_1 = 1, \quad F_i = F_{i-1} + F_{i-2} \quad (i \geq 2).$$

Sequence: $0, 1, 1, 2, 3, 5, 8, 13, \ldots$
Relation to Golden Ratio. Let $\varphi = \frac{1+\sqrt{5}}{2}$ and $\hat{\varphi} = \frac{1-\sqrt{5}}{2}$. Then

$$F_i = \frac{\varphi^i - \hat{\varphi}^i}{\sqrt{5}}.$$

This shows F_i grows asymptotically as $\Theta(\varphi^i)$.
Concluding Remark. These standard functions — polynomials, exponentials, logarithms, factorials, and Fibonacci numbers — provide the basic comparative vocabulary for asymptotic growth. Their properties, especially monotonicity and bounding relationships, are essential in algorithm analysis.