CCES – College of Computer Engineering and Sciences

Prince Sattam Bin Abdulaziz University

CS3401 – Algorithm Design and Analysis

Chapter 2

Algorithm Analysis & Mathematical Background

Academic Year 2026 – 2027

Dr. Khalil Chebil

Dr. Esraa Eldesouky

Chapter Overview

Section 1

Theoretical Analysis of Time Efficiency

Objectives

After this chapter you will be able to:

- Estimate the running time of an algorithm *without* running it
- Apply **Big-O**, **Big-Omega**, and **Big-Theta** notation correctly
- Use the *limit method* to compare growth rates
- Identify worst-case, best-case, and average-case complexities
- Apply the *Master Theorem* to divide-and-conquer recurrences
- Analyse iterative and recursive algorithms systematically

Why does this matter?

Before writing a single line of code, analysis tells you whether your algorithm will finish in milliseconds — or centuries.

Theoretical Framework

Running Time Model

$$T(n) \approx c_{\text{op}} \cdot C(n)$$

c_{op} = execution time of one basic operation $\quad$ $C(n) = \#$ times the basic operation executes on input of size n

Input size measure examples:

Problem	Input size
Searching in a list	n = list length
Matrix multiplication	$n \times n$ dimension
Primality test	# bits of n
Graph traversal	$\lvert V\rvert + \lvert E\rvert$

Basic operation examples:

Problem	Basic op.
Searching	key comparison
Sorting	element comparison
Matrix multiply	multiplication
Polynomial eval.	multiplication

Worst, Best, and Average Case

Three complexity measures

- $C_{\text{worst}}(n)$ — maximum over all inputs of size n
  Provides an upper bound guarantee
- $C_{\text{best}}(n)$ — minimum over all inputs of size n
  Often unrealistically optimistic
- $C_{\text{avg}}(n)$ — *expected* count assuming a probability distribution over inputs
  Harder to compute; most realistic

Important note

$C_{\text{avg}}(n)$ is **not** the average of worst and best case — it is a probabilistic expectation.

Example: Sequential Search

Input: Array $A[0..n-1]$, key K

```
1: i ← 0
2: while i < n and A[i] ≠ K
3:     i ← i + 1
4: end while
5: if i < n then return i
6: else return −1
7: end if
```

Best: 1 comparison

Worst: n comparisons

Average: $\frac{n+1}{2}$ comparisons

Section 2

Asymptotic Notation

Big-O Notation — Upper Bound

Definition

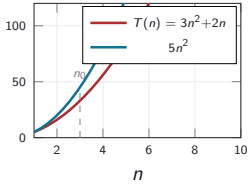
$T(n) = \mathcal{O}(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \leq c\, f(n) \quad \text{for all } n \geq n_0$$

Intuition: $T(n) = \mathcal{O}(f(n))$ means T grows *no faster than* f.

Examples:

- $3n^2 + 2n = \mathcal{O}(n^2)$
- $100 \log n = \mathcal{O}(\log n)$
- $2^n + n^{100} = \mathcal{O}(2^n)$



Common mistakes

Do not write $\mathcal{O}(2n^2)$ or $\mathcal{O}(n^2 + n)$. Drop constants and lower-order terms. The correct form is $\mathcal{O}(n^2)$.

Big-Ω and Big-Θ Notation

Big-Ω — Lower Bound

$T(n) = \Omega(g(n))$ if $\exists\, c, n_0 > 0$ such that

$$T(n) \;\geq\; c\, g(n) \quad \forall\, n \geq n_0$$

Intuition: T grows *at least as fast* as g.

Example: $n^4 = \Omega(n^3)$

Big-Θ — Tight Bound

$T(n) = \Theta(h(n))$ iff $T(n) = \mathcal{O}(h(n))$ **and** $T(n) = \Omega(h(n))$

Intuition: T and h grow at the *same rate*.

Key fact: If $T(n)$ is a degree-k polynomial, then $T(n) = \Theta(n^k)$.

Example: $2n^4 = \Theta(n^4)$

Little-o and Little-ω

$T(n) = o(f(n))$: T grows *strictly slower* than f ($T(n) = \mathcal{O}(f(n))$ but $T(n) \neq \Theta(f(n))$)

$T(n) = \omega(f(n))$: T grows *strictly faster* than f

Composition rules

If $T_1 = \mathcal{O}(f)$ and $T_2 = \mathcal{O}(g)$, then:

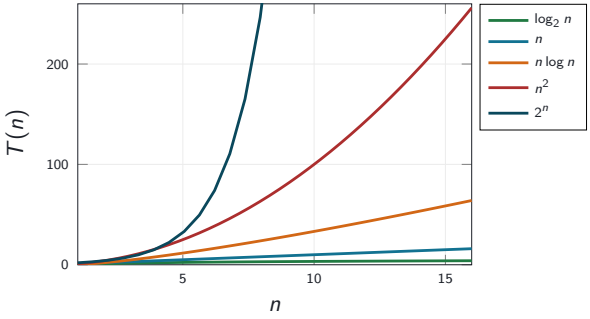
- $T_1 + T_2 = \mathcal{O}(\max(f, g))$
- $T_1 \cdot T_2 = \mathcal{O}(f \cdot g)$

Example: $\mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2)$

Typical Growth Rates

Function	Name
c	Constant
$\log n$	Logarithmic
$\log^2 n$	Log-squared
n	Linear
$n \log n$	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
$n!$	Factorial

(Ordered by increasing growth rate)



Establishing Order of Growth: The Limit Method

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \Rightarrow T(n) = o(g(n)) \quad \text{(T grows slower)} \\ c > 0 & \Rightarrow T(n) = \Theta(g(n)) \quad \text{(same rate)} \\ \infty & \Rightarrow T(n) = \omega(g(n)) \quad \text{(T grows faster)} \end{cases}$$

Example 1: Compare $10n$ vs n^2

$$\lim_{n\to\infty} \frac{10n}{n^2} = \lim_{n\to\infty} \frac{10}{n} = 0 \implies 10n = o(n^2)$$

Example 2: Compare $\frac{n(n-1)}{2}$ vs n^2

$$\lim_{n\to\infty} \frac{n(n-1)/2}{n^2} = \frac{1}{2} \lim_{n\to\infty} \left(1 - \frac{1}{n}\right) = \frac{1}{2}$$

$$\implies \frac{n(n-1)}{2} = \Theta(n^2)$$

Practice

Use the limit method to show: $n^3 = o(n^4)$ and $5n^2 + 3n = \Theta(n^2)$

Case Study: Leaderboard Ranking

Problem

A gaming platform stores n player scores. After each match, find the *top-k* players and check whether a given player is in the top-k.

Three approaches:

Algorithm	Preprocessing	Per query
Linear scan	none	$\mathcal{O}(n)$
Sort once	$\Theta(n \log n)$	$\mathcal{O}(1)$
Max-heap (top-k)	$\Theta(n \log k)$	$\mathcal{O}(\log k)$

Classifying the sort cost $T(n) = 5n \log_2 n + 3n + 7$:

$$\lim_{n \to \infty} \frac{5n \log_2 n + 3n + 7}{n \log n} = 5 \implies \Theta(n \log n)$$

Lower-order terms $3n$ and constant 7 are *absorbed*.

Scaling to 10 million players:

	$n{=}10^4$	$n{=}10^6$
Linear / query	0.1 ms	10 ms
After sort	0.000 ms	0.000 ms

Lesson

Asymptotic notation ignores constants and lower-order terms — what matters is the *dominant term*:

- $5n \log n + 3n + 7 = \Theta(n \log n)$
- $7 = \Theta(1)$ (constant $\neq$ free)
- $n^2 + 10^6 n = \Theta(n^2)$

Section 3

Running Time Calculation

Calculating Running Time — A Simple Example

```
1 int sum(int n) {                    //
      1 op
2   int i, partialSum;                //
        0 ops (declarations)
3   partialSum = 0;                   //
        1 op
4   for (i = 1; i <= n; i++)          //
        1+n+1+n = 2N+2
5     partialSum += i*i*i;            //
          4 ops * N = 4N
6   return partialSum;                //
        1 op
7 }
```

Operation count:

- Initialization: 1 (partialSum)
- Loop setup: 1 (i=1), $N+1$ tests, N increments $= 2N+2$
- Loop body: $4N$ (two multiplications, one addition, one assignment)
- Return: 0 (ignored)

$$\text{Total} = 1 + 2N + 2 + 4N + 1 = 6N + 4$$

$$\implies T(n) = \mathcal{O}(N)$$

General Rules for Running Time

Rule 1 — For loop

$T_{\text{loop}} = (\text{loop body cost}) \times (\#\text{ iterations})$

`for (i=1; i<=n; i++) k++;` $\rightarrow \mathcal{O}(n)$

Rule 2 — Nested loops

Multiply the costs of all loops.

```
1: for i ← 0 to n−1
2:     for j ← 0 to n−1
3:         k += 1
4:     end for
5: end for
```

$\rightarrow \mathcal{O}(n^2)$

Rule 3 — Consecutive statements

Take the maximum (the dominant term).

$\mathcal{O}(n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$

Rule 4 — Conditional

The running time of `if-else` is bounded by the *larger* of the two branches.

$\max(\mathcal{O}(S_1),\ \mathcal{O}(S_2))$

Two useful identities

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \in \Theta(N^2)$$

$$\sum_{i=1}^{N} i^2 = \frac{N(2N+1)(N+1)}{6} \in \Theta(N^3)$$

Loop Analysis: Two Worked Examples

Example A — $\mathcal{O}(N^2)$

```
1: sum ← 0
2: for j ← 0 to n−1
3:     for k ← 0 to j−1
4:         sum += 1
5:     end for
6: end for
```

$$\sum_{j=0}^{N-1} \sum_{k=0}^{j-1} 1 = \sum_{j=0}^{N-1} j = \frac{(N-1)N}{2} \in \Theta(N^2)$$

Example B — $\mathcal{O}(N^3)$

```
1: sum ← 0
2: for j ← 0 to n−1
3:     for k ← 0 to n²−1
4:         sum += 1
5:     end for
6: end for
```

$$\sum_{j=0}^{N-1} \sum_{k=0}^{N^2-1} 1 = \sum_{j=0}^{N-1} N^2 = N^3 \in \Theta(N^3)$$

Key point

Always simplify the inner sum *before* multiplying. Look at the upper limit of k carefully — it may depend on j or on n.

Section 4

Logarithms in Running Time

$\mathcal{O}(\log n)$ Algorithms

When does $\log n$ appear?

An algorithm runs in $\mathcal{O}(\log n)$ if it *halves* the problem size at each step in $\mathcal{O}(1)$ work.

Fast Exponentiation: X^N

$$X^N = \begin{cases} 1 & N = 0 \\ X & N = 1 \\ (X^{N/2})^2 & N \text{ even} \\ (X^{(N-1)/2})^2 \cdot X & N \text{ odd} \end{cases}$$

Recurrence:

$T(N) = T(N/2) + c \implies T(N) = \mathcal{O}(\log N)$

Derivation of $\mathcal{O}(\log N)$:

$$\begin{aligned} T(N) &= T(N/2) + c \\ &= T(N/4) + 2c \\ &= T(N/2^k) + kc \end{aligned}$$

stop when $N/2^k = 1 \implies k = \log_2 N$

$$T(N) = c \log_2 N \in \mathcal{O}(\log N)$$

X^{1024} requires only **10 multiplications**!

Section 5

Analysis of Recursive Algorithms

Recursive Analysis: Factorial

Algorithm $F(n) = n!$

Input: Non-negative integer n
Output: $n!$
1: **if** $n = 0$ **then return** 1
2: **end if**
3: **return** $F(n-1) \cdot n$

Recurrence for multiplications:

$$M(n) = M(n-1) + 1, \quad M(0) = 0$$

Solving by telescoping:

$$\begin{aligned} M(n) &= M(n-1) + 1 \\ &= M(n-2) + 1 + 1 \\ &= M(n-3) + 3 \\ &\vdots \\ &= M(0) + n = n \end{aligned}$$

$$\boxed{M(n) = n \in \Theta(n)}$$

Linear — one multiplication per level of recursion.

Recursive Analysis: Tower of Hanoi

Problem: Move n disks from peg A to peg C using peg B. Rule: never place a larger disk on a smaller one.

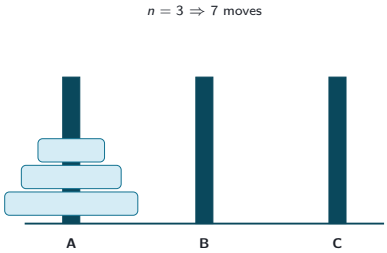
Recurrence:

$$M(n) = 2M(n-1) + 1, \quad M(1) = 1$$

Solution by telescoping:

$$\begin{aligned} M(n) &= 2M(n-1) + 1 = 2[2M(n-2) + 1] + 1 \\ &= 2^2 M(n-2) + 2 + 1 \\ &= 2^{n-1} M(1) + 2^{n-1} - 1 \\ &= 2^n - 1 \end{aligned}$$

$$\boxed{M(n) = 2^n - 1 \in \Theta(2^n)}$$



Combinatorial explosion

$n = 64$ requires $2^{64} - 1 \approx 1.8 \times 10^{19}$ moves. At 1 move/sec that is **584 billion years**.

Recursive Analysis: Counting Binary Digits

BinRec(n)

Input: positive integer n
Output: # binary digits of n
1: **if** $n = 1$ **then return** 1
2: **end if**
3: **return** $\textsc{BinRec}(\lfloor n/2 \rfloor) + 1$

Recurrence:

$$A(n) = A(\lfloor n/2 \rfloor) + 1, \quad A(1) = 0$$

Solution: Assume $n = 2^k$:

$$\begin{aligned} A(2^k) &= A(2^{k-1}) + 1 \\ &= A(2^{k-2}) + 2 \\ &= \ldots = A(2^0) + k = k \end{aligned}$$

Since $k = \log_2 n$:

$$\boxed{A(n) = \lfloor \log_2 n \rfloor \in \Theta(\log n)}$$

Each recursion halves $n \to$ depth $= \log_2 n$ levels.

Section 6

Useful Summation Formulas

Essential Summations for Analysis

Constant sum:

$$\sum_{i=a}^{b} c = c\,(b - a + 1)$$

Arithmetic series:

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \in \Theta(N^2)$$

$$\sum_{i=0}^{N-1} i = \frac{N^2 - N}{2}$$

Quadratic series:

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \in \Theta(N^3)$$

Geometric series:

$$\sum_{i=0}^{N} a^i = \frac{a^{N+1} - 1}{a - 1} \qquad (a \neq 1)$$

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1 \in \Theta(2^N)$$

Harmonic series:

$$\sum_{i=1}^{N} \frac{1}{i} \approx \ln N \in \Theta(\log N)$$

Application

These formulas are the core tools for converting summations from loop analyses into closed-form complexities.

Chapter 2 Summary

Key concepts:

- $T(n) \approx c_{\text{op}} \cdot C(n)$: count basic operations
- $\mathcal{O}(f)$: upper bound; $\Omega(g)$: lower bound; $\Theta(h)$: tight bound
- Use the *limit method* to establish growth order
- Four general rules for iterative analysis
- Recurrences: telescoping, Master Theorem
- Typical growth hierarchy: $\log n \ll n \ll n \log n \ll n^2 \ll 2^n \ll n!$

Next chapter

Chapter 3: Brute Force & Exhaustive Search — the simplest strategy: try everything. We analyse selection sort, bubble sort, string matching, and combinatorial problems.

Self-check questions

1. What is Θ for $T(n) = 5n^3 - 2n^2 + 100$?
2. Solve: $T(n) = T(n-1) + n,\ T(1) = 1$
3. Is $n! = \mathcal{O}(n^n)$? Prove it.