Algorithms

Asymptotic Analysis, Divide-and-Conquer, and Recurrence Relations

Week 2 — Lectures 5-8

I. Asymptotic Notation and Analysis

A. Theta Notation (Θ) - Asymptotic Equivalence

Formal Definition:

- **$\Theta(g(n))$:** $= \{f(n) \mid \exists c_1, c_2, n_0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$
- **Asymptotic Equivalence:** Functions with the same growth rate
- **Flexibility:** Can choose n_0 as large as needed; constants c_1, c_2 don't affect classification

Example Analysis: $f(n) = 8n^2 + 2n - 3$

Proving $f(n) \in \Theta(n^2)$:

- **Lower Bound:** Need $c_1 n^2 \leq 8n^2 + 2n - 3$
  - Choose $c_1 = 7$ and $n_0 \geq \sqrt{3}$
  - For $n \geq \sqrt{3}$: $2n - 3 \geq n^2$, so $8n^2 + 2n - 3 \geq 7n^2$
- **Upper Bound:** Need $8n^2 + 2n - 3 \leq c_2 n^2$
  - Choose $c_2 = 10$ and $n_0 \geq 1$
  - For $n \geq 1$: $2n \leq 2n^2$ and $-3 \leq 0$, so $8n^2 + 2n - 3 \leq 10n^2$

Disproving Other Classes:

- **$f(n) \notin \Theta(n)$:** Upper bound fails
  - Proof by contradiction: $\lim_{n\to\infty} \frac{8n^2+2n-3}{n} = \lim_{n\to\infty}(8n+2) = \infty$
- **$f(n) \notin \Theta(n^3)$:** Lower bound fails
  - Proof by contradiction: $\lim_{n\to\infty} \frac{8n^2+2n-3}{n^3} = \lim_{n\to\infty} \frac{8}{n} = 0$

B. Big-O and Omega Notation

Formal Definitions:

- **$O(g(n))$:** $= \{f(n) \mid \exists c, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
- **$\Omega(g(n))$:** $= \{f(n) \mid \exists c, n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$

Relationship: $f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$

C. Limit Rules for Asymptotic Analysis

Theta Rule:

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \text{ for some constant } c > 0, \text{ then } f(n) \in \Theta(g(n))$$

Big-O Rule:

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \text{ for some constant } c \geq 0, \text{ then } f(n) \in O(g(n))$$

Omega Rule:

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \text{ (positive constant or } \infty\text{), then } f(n) \in \Omega(g(n))$$

L'Hôpital's Rule:

- If $f(n)$ and $g(n)$ both approach 0 or both approach ∞:

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

D. Polynomial and Exponential Functions

General Polynomial Theorem:

- For polynomial $p(n) = \sum_{i=0}^{d} a_i n^i$ where $a_d > 0$: $p(n) \in \Theta(n^d)$
- **Example:** $f(n) = 2n^4 - 5n^3 - 2n^2 + 4n - 7 \in \Theta(n^4)$
- **Proof using limits:** $\lim_{n\to\infty} \frac{f(n)}{n^4} = \lim_{n\to\infty} \left(2 - \frac{5}{n} - \frac{2}{n^2} + \frac{4}{n^3} - \frac{7}{n^4}\right) = 2$

Exponential vs. Polynomial Lemma:

- For any positive constants $a > 1$, b, and c:

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 \quad \text{and} \quad \lim_{n\to\infty} \frac{\lg^b n}{n^c} = 0$$

Growth Hierarchy:

$$\text{Constant} < \text{Logarithmic} < \text{Linear} < \text{Linearithmic} < \text{Polynomial} < \text{Exponential} < \text{Factorial}$$

II. Divide-and-Conquer Methodology

A. Paradigm and Problem-Solving Strategy

Three-Step Process:

- **Divide:** Split problem into smaller subproblems
- **Conquer:** Solve subproblems recursively
- **Combine:** Merge solutions into global solution

When to Apply:

- After considering brute force approach
- Problem has optimal substructure
- Subproblems can be solved independently
- Combining step is efficient

B. MergeSort: Classic Divide-and-Conquer Algorithm

Algorithm Structure:

Algorithm 1 MergeSort

```
function MergeSort(A, p, r)
    if p < r then                              ▷ At least 2 elements
        q = ⌊(p + r)/2⌋
        MergeSort(A, p, q)                     ▷ Sort left half
        MergeSort(A, q + 1, r)                 ▷ Sort right half
        Merge(A, p, q, r)                      ▷ Combine results
    end if
end function
```

Merge Procedure:

Algorithm 2 Merge

```
function Merge(A, p, q, r)
    Copy A[p..q] and A[q + 1..r] to temporary arrays
    i = p, j = q + 1, k = p
    while i ≤ q and j ≤ r do
        if A[i] ≤ A[j] then
            A[k++] = A[i++]                    ▷ Copy from left
        else
            A[k++] = A[j++]                    ▷ Copy from right
        end if
    end while
    Copy remaining elements
    Copy back to original array
end function
```

Algorithm Properties:

- **Stability:** Preserves relative order of equal elements
- **Space Complexity:** $O(n)$ auxiliary space required
- **Optimization:** Stop divide-and-conquer for small inputs (e.g., $n \leq 20$)
- **Practical Advantage:** Excellent for already-sorted data

III. Recurrence Relations and Analysis Methods

A. Recurrence Formulation

Definition:

- **Recurrence:** Recursively defined function expressing algorithm's running time
- **Components:** Base case(s) and recursive case(s)

MergeSort Recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{otherwise} \end{cases}$$

Pattern Recognition Through Computation:

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 2T(1) + 2 = 2 + 2 = 4 \\ T(4) &= 2T(2) + 4 = 8 + 4 = 12 \\ T(8) &= 2T(4) + 8 = 24 + 8 = 32 \\ T(16) &= 2T(8) + 16 = 64 + 16 = 80 \end{aligned}$$

Insight from Powers of 2:

$$\frac{T(1)}{1} = 1, \quad \frac{T(2)}{2} = 2, \quad \frac{T(4)}{4} = 3, \quad \frac{T(8)}{8} = 4, \quad \frac{T(16)}{16} = 5$$

Pattern: $\frac{T(2^k)}{2^k} = k + 1 = \log_2(2^k) + 1$

B. Logarithmic Relationships

Change of Base Formula:

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

Key Property: All logarithms differ by constant factors, so $\log_b n \in \Theta(\log_a n)$

C. Solution Methods

Method 1: Induction Verification

- **Hypothesis:** $T(n) = n \log n + n$ for MergeSort
- **Base Case:** $T(1) = 1 \log 1 + 1 = 1$ ✓
- **Inductive Step:** Assume true for $T(n/2)$, prove for $T(n)$:

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\left(\frac{n}{2} \log \frac{n}{2} + \frac{n}{2}\right) + n \\ &= n(\log n - 1) + n + n \\ &= n \log n + n \quad \checkmark \end{aligned}$$

Method 2: Iteration Method

- Repeatedly expand recurrence until pattern emerges
- **Example:** $T(n) = 3T(n/4) + n$

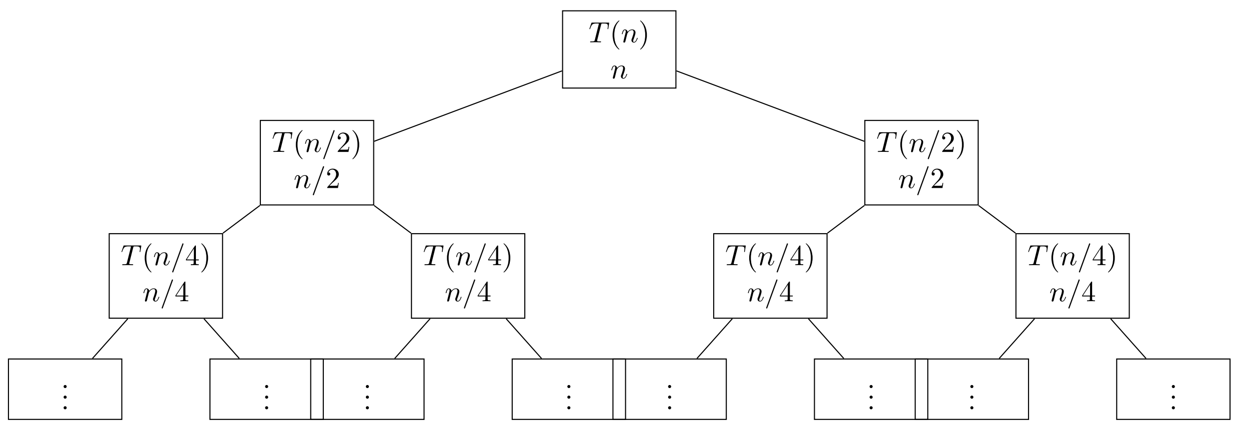
$$\begin{aligned} T(n) &= 3T(n/4) + n \\ &= 3[3T(n/16) + n/4] + n = 9T(n/16) + 3n/4 + n \\ &= 9[3T(n/64) + n/16] + 3n/4 + n = 27T(n/64) + 9n/16 + 3n/4 + n \end{aligned}$$

- Pattern: $T(n) = 3^k T(n/4^k) + n \sum_{i=0}^{k-1} (3/4)^i$

IV. Recursion Trees and Visualization

A. MergeSort Recursion Tree

Tree Structure for $T(n) = 2T(n/2) + n$:



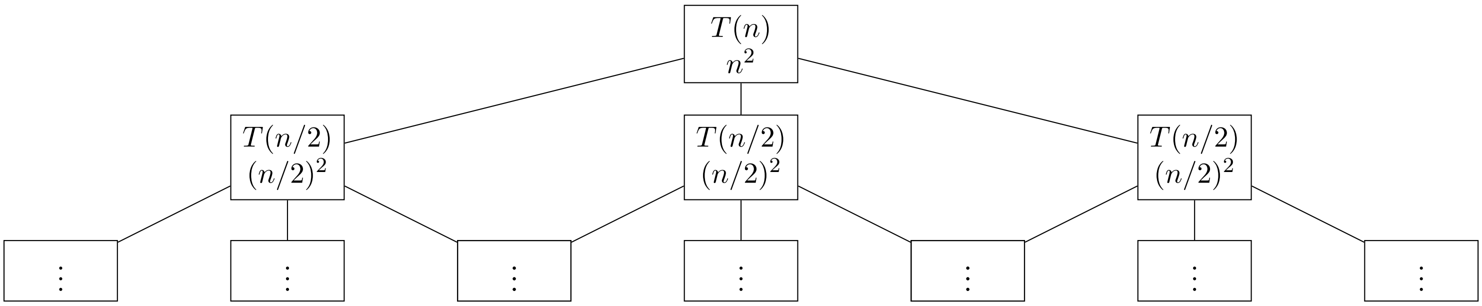
Level Analysis:

- **Level 0:** Cost $= n$
- **Level 1:** Cost $= 2(n/2) = n$
- **Level 2:** Cost $= 4(n/4) = n$
- **Level i:** Cost $= 2^i (n/2^i) = n$
- **Height:** $\lg n + 1$ levels

Total Cost: $n(\lg n + 1) = n \lg n + n$

B. Complex Recursion Tree Example

For $T(n) = 3T(n/2) + n^2$:



Level Analysis:

- **Level 0:** Cost $= n^2$
- **Level 1:** Cost $= 3(n/2)^2 = 3n^2/4$
- **Level 2:** Cost $= 9(n/4)^2 = 9n^2/16$
- **Level i:** Cost $= n^2(3/4)^i$

Geometric Series Sum:

$$T(n) = n^2 \sum_{i=0}^{\lg n} \left(\frac{3}{4}\right)^i = n^2 \cdot \frac{1 - (3/4)^{\lg n + 1}}{1 - 3/4} = \Theta(n^2)$$

V. Master Theorem

A. Theorem Statement

For recurrences of the form $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$:

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$

B. Examples and Applications

Example 1: MergeSort

- $T(n) = 2T(n/2) + n$
- $a = 2, b = 2, f(n) = n$
- $\log_b a = \log_2 2 = 1$
- $f(n) = n = \Theta(n^1) \to$ **Case 2**
- $T(n) = \Theta(n \lg n)$

Example 2: Binary Search Tree Operations

- $T(n) = T(n/2) + 1$
- $a = 1, b = 2, f(n) = 1$
- $\log_b a = \log_2 1 = 0$
- $f(n) = 1 = \Theta(n^0) \to$ **Case 2**
- $T(n) = \Theta(\lg n)$

Example 3: Matrix Multiplication (Standard)

- $T(n) = 8T(n/2) + n^2$
- $a = 8, b = 2, f(n) = n^2$
- $\log_b a = \log_2 8 = 3$
- $f(n) = n^2 = O(n^{3-1}) \to$ **Case 1**
- $T(n) = \Theta(n^3)$

Example 4: Efficient Algorithm

- $T(n) = 2T(n/2) + n^2$
- $a = 2, b = 2, f(n) = n^2$
- $\log_b a = \log_2 2 = 1$
- $f(n) = n^2 = \Omega(n^{1+1})$ and regularity condition holds $\to$ **Case 3**
- $T(n) = \Theta(n^2)$

C. Master Theorem Application Strategy

Step-by-Step Process:

1. Identify a, b, and $f(n)$ from recurrence
2. Calculate critical exponent: $\log_b a$
3. Compare $f(n)$ with $n^{\log_b a}$:
   - If $f(n)$ is polynomially smaller $\to$ Case 1
   - If $f(n)$ is asymptotically equal $\to$ Case 2
   - If $f(n)$ is polynomially larger $\to$ Case 3 (check regularity)
4. Apply appropriate case formula

Common Pitfalls:

- **Gap Between Cases:** Not all recurrences fit Master Theorem
- **Regularity Condition:** Must verify for Case 3
- **Polynomial Difference:** Logarithmic factors don't count as polynomial difference