Algorithms

Computational Models, Complexity Analysis, and Design Principles

Week 1 — Lectures 1-3

I. Fundamental Concepts of Algorithm Design

A. Definition and Mathematical Foundation

Formal Definition:

- **Algorithm:** A well-defined computational procedure that takes input and produces output
- **Mathematical Abstraction:** Algorithms are mathematical objects, not programs
- **Deterministic:** Given the same input, always produces the same output

Input-Output Specification:

$$\text{Algorithm} : \mathcal{I} \to \mathcal{O}$$

where $\mathcal{I}$ is the input domain and $\mathcal{O}$ is the output domain.

Example: Sorting Algorithm

- **Input:** Sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers
- **Output:** Permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$
- **Property:** Output is a sorted rearrangement of the input

B. Importance in Computer Science and Programming

Multi-Level Design Considerations:

- **Macro Issues:** Coordinating efforts of multiple programmers on large software systems
  - Module interfaces and data structure design
  - System architecture and component interaction
  - Scalability and maintainability concerns
- **Micro Issues:** Optimizing critical code sections
  - **80/20 Rule:** 80% of execution time spent in 20% of code
  - Focus optimization efforts on algorithmically critical sections
  - Data structure selection for performance-critical operations

Common Anti-Pattern:

- **Premature Implementation:** Using poor algorithms/data structures, then optimizing
- **Why This Fails:** Fine-tuning cannot overcome fundamental algorithmic inefficiency
- **Correct Approach:** Design good algorithms before implementation

Example: Search Problem

- **Poor Choice:** Linear search in sorted array: $O(n)$
- **Good Choice:** Binary search in sorted array: $O(\log n)$
- **Impact:** For $n = 1,000,000$: Linear requires $\sim 500,000$ operations on average, Binary requires ~ 20 operations

II. Computational Models and Complexity Analysis

A. Random Access Machine (RAM) Model

Mathematical Abstraction:

- **Memory Model:** Infinitely large random-access memory
- **Processor Model:** Single-processor, no parallelism
- **Data Types:** Simple types (integers, booleans) with unit-cost operations
- **Operations:** Arithmetic $(+, -, \times, \div)$, comparison $(<, >, =)$, memory access

Unit-Cost Assumption:

$$T_{\text{operation}} = O(1) \text{ for all primitive operations}$$

Why RAM Model:

- Simplifies analysis while maintaining practical relevance
- Abstracts away machine-specific details
- Enables focus on algorithmic structure rather than implementation details
- Provides foundation for asymptotic analysis

B. Runtime Analysis Methodology

Input Size Parameterization:

- **Input Size n:** Problem-dependent measure of input magnitude
- Examples: Array length, graph vertices, matrix dimensions
- **Running Time:** $T(n)$ = number of primitive operations as function of n

Analysis Types:

- **Worst-Case Analysis:** $T_{\text{worst}}(n) = \max_{\text{input } I, |I|=n} T(I)$
  - Guarantees upper bound on performance
  - Most commonly used in theoretical analysis
  - Conservative but provides reliability guarantees
- **Average-Case Analysis:** $T_{\text{avg}}(n) = \mathbb{E}[T(I)]$ over input distribution
  - Requires probabilistic model of inputs
  - More realistic for typical performance
  - Harder to analyze mathematically

Focus on Worst-Case:

- Provides performance guarantees
- Simpler mathematical analysis
- Often close to average-case for well-designed algorithms

III. Case Study: 2D Maxima Problem

A. Problem Definition and Mathematical Formulation

Formal Problem Statement:

- **Input:** Set of points $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^2$
- Each point $p_i = (x_i, y_i)$ with integer coordinates
- **Output:** Set of maximal points $M \subseteq P$

Dominance Relation:

$$p_i \text{ dominates } p_j \iff (x_i \geq x_j) \land (y_i \geq y_j) \land (p_i \neq p_j)$$

Maximal Point Definition:

$$p_i \in M \iff \nexists p_j \in P \text{ such that } p_j \text{ dominates } p_i$$

Geometric Interpretation:

- Maximal points form the "northeast boundary" of the point set
- No point lies in the northeast quadrant of any maximal point
- Forms a monotonic decreasing sequence when sorted by x-coordinate

B. Brute Force Algorithm and Analysis

Algorithm Design:

Algorithm 1 Brute Force 2D Maxima

```
function BruteForceMaxima(P)
    M = ∅                                          ▷ Initialize maximal set
    for each point p_i ∈ P do
        isMaximal = true
        for each point p_j ∈ P do
            if p_j dominates p_i then
                isMaximal = false
                break
            end if
        end for
        if isMaximal then
            M = M ∪ {p_i}
        end if
    end for
    return M
end function
```

Correctness Analysis:

- **Invariant:** Algorithm correctly identifies all maximal points processed so far
- **Completeness:** Every point is tested against all others
- **Soundness:** Only non-dominated points are added to result set

Complexity Analysis:

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} O(1) = \sum_{i=1}^{n} O(n) = O(n^2)$$

Detailed Operation Count:

- **Outer loop:** n iterations
- **Inner loop:** n iterations per outer iteration
- **Dominance check:** $O(1)$ per iteration
- **Total:** $n \times n \times O(1) = O(n^2)$

IV. Summation Analysis Techniques

A. Converting Loops to Summations

General Principle: Nested loops with k levels create k-fold summations:

$$\text{for } i_1 = a_1 \text{ to } b_1 \text{ do } \cdots \text{ for } i_k = a_k(i_1, \ldots, i_{k-1}) \text{ to } b_k(i_1, \ldots, i_{k-1})$$

$$\Rightarrow \sum_{i_1=a_1}^{b_1} \sum_{i_2=a_2(i_1)}^{b_2(i_1)} \cdots \sum_{i_k=a_k(i_1,\ldots,i_{k-1})}^{b_k(i_1,\ldots,i_{k-1})} \text{Cost of innermost operation}$$

Standard Examples:

- **Simple nested loop:**

```
for i = 1 to n
   for j = 1 to n
       // O(1) work
```

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=1}^{n} O(1) = O(n^2)$$

- **Triangular nested loop:**

```
for i = 1 to n
   for j = i to n
       // O(1) work
```

$$\Rightarrow \sum_{i=1}^{n} \sum_{j=i}^{n} O(1) = \sum_{i=1}^{n} (n-i+1) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = O(n^2)$$

B. Essential Summation Formulas

Arithmetic Series:

$$\sum_{i=1}^{n} 1 = n$$

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

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

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \Theta(n^4)$$

Geometric Series:

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1} \text{ for } r \neq 1$$

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \text{ for } |r| < 1$$

Harmonic Series:

$$\sum_{i=1}^{n} \frac{1}{i} = H_n = \Theta(\log n)$$

Logarithmic Series:

$$\sum_{i=1}^{n} \log i = \log(n!) = \Theta(n \log n)$$

C. Advanced Example: Triple Nested Loop Analysis

Complex Loop Structure:

```
for i = 1 to n
   for j = i to n
       for k = 1 to j
           // O(1) work
```

Summation Setup:

$$T(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=1}^{j} O(1) = \sum_{i=1}^{n} \sum_{j=i}^{n} j$$

Step-by-Step Solution:

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n} \sum_{j=i}^{n} j \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} j - \sum_{j=1}^{i-1} j \right) && \text{(Telescoping)} \\
&= \sum_{i=1}^{n} \left(\frac{n(n+1)}{2} - \frac{(i-1)i}{2} \right) \\
&= \sum_{i=1}^{n} \frac{n(n+1) - i(i-1)}{2} \\
&= \frac{1}{2} \left[n^2(n+1) - \sum_{i=1}^{n} i(i-1) \right] \\
&= \frac{1}{2} \left[n^2(n+1) - \sum_{i=1}^{n} (i^2 - i) \right] \\
&= \frac{1}{2} \left[n^2(n+1) - \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right] \\
&= \Theta(n^3)
\end{aligned}$$

V. Solving Summations: Methods and Techniques

A. Closed Form Solutions

Definition: A closed form is an exact formula without summation notation.

Method 1: Pattern Recognition

- Match summation to known formula
- Use algebraic manipulation to transform into standard form
- Apply appropriate closed form formula

Example: $\sum_{i=5}^{n} i$

$$\begin{aligned}\sum_{i=5}^{n} i &= \sum_{i=1}^{n} i - \sum_{i=1}^{4} i && \text{(Telescoping)} \\ &= \frac{n(n+1)}{2} - \frac{4\cdot 5}{2} \\ &= \frac{n(n+1)-20}{2}\end{aligned}$$

B. Alternative Methods When Closed Form Is Difficult

Method 2: Crude Bounds

- Find simple upper and lower bounds
- Useful when exact formula is complex
- Often sufficient for asymptotic analysis

Example: $\sum_{i=1}^{n} \sqrt{i}$

$$\begin{aligned}\sum_{i=1}^{n} 1 \leq \sum_{i=1}^{n} \sqrt{i} &\leq \sum_{i=1}^{n} \sqrt{n} \\ n \leq \sum_{i=1}^{n} \sqrt{i} &\leq n\sqrt{n} \\ \Rightarrow \sum_{i=1}^{n} \sqrt{i} &= \Theta(n^{3/2})\end{aligned}$$

Method 3: Integral Approximation

- Replace summation with definite integral
- Valid when function is monotonic
- Provides asymptotic bounds

Integral Test: For monotonic function $f(x)$:

$$\int_{1}^{n+1} f(x)dx \leq \sum_{i=1}^{n} f(i) \leq f(1) + \int_{1}^{n} f(x)dx$$

Example: $\sum_{i=1}^{n} \log i$

$$\int_{1}^{n} \log x\, dx = [x\log x - x]_1^n = n\log n - n + 1 = \Theta(n\log n)$$

Method 4: Constructive Induction

- Guess the form of the solution
- Prove correctness by mathematical induction

- Useful when pattern is suspected but not obvious

Example: Prove $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
Base case: $n = 1$: $1^2 = 1 = \frac{1\cdot 2\cdot 3}{6} = 1$
Inductive step: Assume true for $n = k$, prove for $n = k+1$:

$$\begin{aligned}
\sum_{i=1}^{k+1} i^2 &= \sum_{i=1}^{k} i^2 + (k+1)^2 \\
&= \frac{k(k+1)(2k+1)}{6} + (k+1)^2 && \text{(Inductive hypothesis)} \\
&= \frac{k(k+1)(2k+1) + 6(k+1)^2}{6} \\
&= \frac{(k+1)[k(2k+1) + 6(k+1)]}{6} \\
&= \frac{(k+1)(2k^2 + k + 6k + 6)}{6} \\
&= \frac{(k+1)(2k^2 + 7k + 6)}{6} \\
&= \frac{(k+1)(k+2)(2k+3)}{6}
\end{aligned}$$

C. Practical Guidelines for Summation Analysis

Algorithm for Analyzing Summations:

1. **Identify the summation structure** from loop nesting
2. **Attempt pattern recognition** with standard formulas
3. **Apply algebraic manipulation** (telescoping, factoring)
4. **If closed form is difficult:** Use bounds or approximation
5. **Verify result** using asymptotic notation

Common Pitfalls:

- **Index Range Errors:** Carefully track summation limits
- **Off-by-One Errors:** Distinguish between $\sum_{i=1}^{n}$ and $\sum_{i=0}^{n-1}$
- **Nested Dependency:** Inner summation limits may depend on outer variables