CE6029: Advanced Algorithm (Fall 2025)

Tutorial 1: Introduction

Date: Sept 4, 2025

Instructor: Dr. Hao-Tsung Yang | Noted By: Po-Cheng Chan

1 Week 1: Course Overview and Purpose

The first lecture focused on clarifying the course objectives and introducing the fundamental concept of algorithms.

Instructions. Show your work, justify asymptotics, and clearly label answers. Unless stated otherwise, you may use standard results covered in class.

1.1 Course Objectives

The primary goals of this course are:

- To develop a **broad, high-level understanding** of algorithmic thinking.
- To conduct demonstrations that help illustrate complex ideas.
- To understand the **barrier between the known and the unknown**, encouraging critical analysis of what can and cannot be solved algorithmically.

1.2 What is an Algorithm?

An **algorithm** can be defined as:

- A **method** to achieve a given *goal* or *objective*.
- A **procedure**, i.e., a finite sequence of well-defined instructions or orders.
- A **guarantee**, meaning the procedure provides an explicit and reliable outcome.

1.3 Key Characteristics of an Algorithm

An algorithm must be:

- **Explicit**: clearly and precisely stated.
- **Unambiguous**: leaving no room for multiple interpretations.
- **Mechanically executable**: every step can be carried out by a computational process without human intuition.

1.4 Example 1: Simple Decrement Algorithm

Input: n, where $n \in \mathbb{N}$.

Listing 1: Pseudo-code of Example 1

```
Algorithm:
  while n > 0:
      take one beer off the wall
      n <- n - 1
```

1.4.1 Time Complexity Analysis

We analyze the time cost of each operation in the algorithm.

Table 1: Time cost of each step in the algorithm

Step	**Time Complexity**	**Explanation**
Check condition $n > 0$	$O(1)$	Simple comparison of an integer with zero.
Action: take one beer off the wall	$O(1)$	Treated as a single atomic action (e.g., removing one bottle).
Assignment $n \leftarrow n - 1$	$O(1)$	Single subtraction and assignment operation.

Therefore, the cost of a **single iteration** is:

$$O(1) + O(1) + O(1) \approx O(1).$$

Since the `while` loop executes n times and each iteration costs $O(1)$, the overall running time is:

$$T(n) = n \times O(1) = O(n).$$

1.5 Example 2: Multiplication of Two Numbers

Input:

$$X = [x_1, x_2, \dots, x_m], \quad x_i \in [0, 9]$$
$$Y = [y_1, y_2, \dots, y_n], \quad y_i \in [0, 9]$$

Output:

$$X \times Y$$

Example:

$$3 \times 6 = 18$$

Algorithm (conceptual):

1. Interpret X and Y as integers represented by digit arrays.
2. Apply digit-wise multiplication (e.g., grade-school method or more advanced algorithms such as Karatsuba if m, n are large).
3. Accumulate the partial products and perform carries to obtain the final product.

1.5.1 Time Complexity of Multiplication

Let m and n be the number of digits of X and Y, respectively.

- Using the classical grade-school method:

  $$T(m, n) = O(m \times n).$$

  If $m \approx n \approx k$, this simplifies to $O(k^2)$. However, none of these achieve $O(1)$.

1.6 Example 3: Lattice Multiplication Algorithm

The lattice (or convolution-based) multiplication algorithm computes the product of two numbers X and Y digit by digit.

1.6.1 Mathematical Definition

Let

$$X = \sum_{i=0}^{m-1} x_i \, 10^i, \qquad Y = \sum_{j=0}^{n-1} y_j \, 10^j,$$

where $x_i, y_j \in [0, 9]$.

Then the product can be expressed as a double summation:

$$X \times Y = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_i \cdot y_j \cdot 10^{i+j}.$$

The number of digits of the product satisfies:

$$\text{len}(X \times Y) = m + n - 1.$$

1.6.2 Algorithm (Pseudo-code)

Listing 2: Lattice multiplication algorithm

```
Multi(X[0..m-1], Y[0..n-1]):
    hold <- 0
    for k from 0 to m + n - 1:
        for all pairs (i, j) such that i + j = k:
            hold <- hold + X[i] * Y[j]
        Z[k] <- hold mod 10
        hold <- hold // 10   # integer division
```

Example:

$$4 \times 9 = 36$$

Digit extraction:

- $Z[k] \leftarrow \text{hold mod } 10 \quad \rightarrow \; 3$
- $\text{hold} \leftarrow \lfloor \text{hold}/10 \rfloor \quad \rightarrow \; 6$

1.6.3 Time Complexity Analysis

For each k from 0 to $m + n - 1$, we examine all pairs (i, j) such that $i + j = k$. The total number of (i, j) pairs is exactly $m \times n$, because each digit x_i multiplies with each digit y_j once.

Hence the running time is:

$$T(m, n) = O(m \times n).$$

If $m \approx n \approx k$, this simplifies to:

$$T(k) = O(k^2).$$

1.6.4 Core Conclusion: Time Complexity of Lattice Multiplication

The lattice (or convolution-based) multiplication algorithm performs one elementary digit–digit multiplication for every pair (i, j) where $i \in [0, m-1]$ and $j \in [0, n-1]$. Therefore, the total number of digit multiplications is

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} 1 = m \times n.$$

Each such digit multiplication and addition of carries is considered an $O(1)$ operation under the RAM model. Consequently, the overall running time satisfies

$$T(m, n) = O(m \times n).$$

When $m \approx n \approx k$, this simplifies to

$$T(k) = O(k^2).$$

Key insight: The lattice algorithm is quadratic in the number of digits. It has the same asymptotic complexity as the classical grade-school method. More advanced algorithms such as Karatsuba ($O(k^{\log_2 3}) \approx O(k^{1.585})$) or Schönhage–Strassen FFT-based multiplication ($O(k \log k \log \log k)$) can asymptotically outperform the lattice algorithm, but the lattice algorithm itself remains $O(k^2)$.

1.7 Example 4: Big-O Definition and Growth Rate

Definition (Big-O). Let $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. We say $f(n) \in O(g(n))$ if there exist constants $M > 0$ and $n_0 \geq 0$ such that

$$\forall n \geq n_0 : \quad f(n) \leq M\, g(n).$$

Graphical interpretation. The figure illustrates the Big-O relationship $f(x) \in O(g(x))$. Although $f(x)$ may initially exceed $g(x)$ for $x < x_0$, there exists a constant M such that for all $x \geq x_0$, $f(x) \leq Mg(x)$. The shaded area indicates the region we may ignore asymptotically. This geometric view explains why constant factors and finite hardware effects only affect M or x_0, but not the Big-O class.

Intuition. Big-O compares **growth rates**. Once n is large enough ($n \geq n_0$), g (scaled by a constant) upper-bounds f. Implementation details or **hardware limits** only affect the constant factor M or the threshold n_0; they do not change the asymptotic class.

Example. $O(5n^2 + n) = O(3n^2)$. Let $f(n) = 5n^2 + n$ and $g(n) = 3n^2$. For all $n \geq 1$,

$$5n^2 + n \leq 5n^2 + n^2 = 6n^2 \leq 2 \cdot 3n^2 = 2\, g(n).$$

Thus we can choose $M = 2$ and $n_0 = 1$, which proves

$$5n^2 + n \in O(3n^2).$$

Equivalently, constants do not matter: $O(5n^2 + n) = O(n^2) = O(3n^2)$.

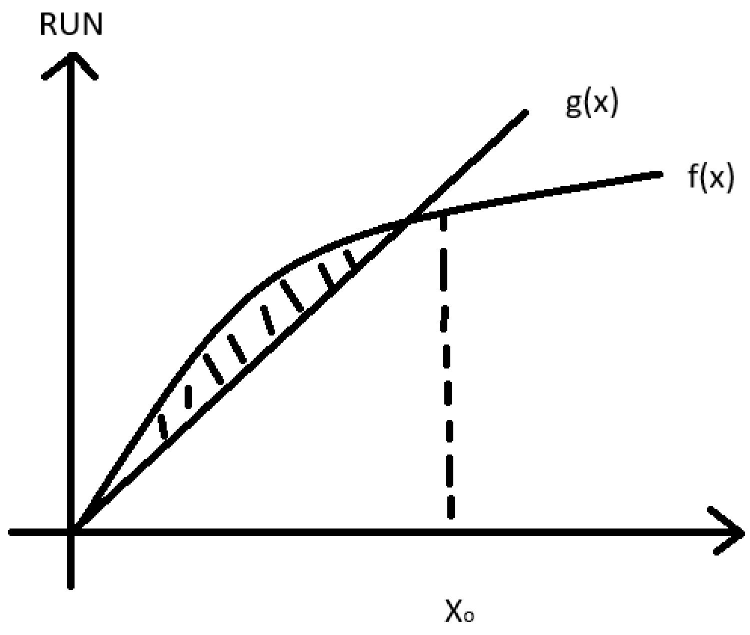

Figure 1: Geometric intuition of $f(n) \in O(g(n))$: beyond n_0, $f(n)$ is bounded above by a constant multiple $M \cdot g(n)$ (shaded area).

Takeaway.

- **Growth rate dominates:** polynomial degree, logarithms, exponentials determine the class.
- **Hardware/implementation** shifts constants (M) and thresholds (n_0) but not the asymptotic order.

1.8 Example 5: Duplication and Mediation (Egyptian/Russian Peasant Multiplication)

Definition. Given $x, y \in \mathbb{N}$, define

$$x \cdot y = \begin{cases} 0, & \text{if } x = 0, \\ \lfloor x/2 \rfloor \cdot (2y), & \text{if } x \text{ is even}, \\ \lfloor x/2 \rfloor \cdot (2y) + y, & \text{if } x \text{ is odd}. \end{cases}$$

This corresponds to repeatedly halving x and doubling y; whenever the current x is odd, we add the current y to the answer.

1.8.1 Pseudo-code

Listing 3: Duplication and Mediation

```
PeasantMultiply(x, y):
    ans <- 0
    while x > 0:
        if x is odd: ans <- ans + y
        x <- floor(x/2)     # halve
        y <- y + y          # double
    return ans
```

Worked example. Select rows where A is odd and sum the corresponding B.

A	**B**	**Prod. contribution**
123	456	456
61	912	912
30	1824	0
15	3648	3648
7	7296	7296
3	14592	14592
1	29184	29184
	sum	123×456

1.8.2 Running Time

Unit-cost RAM model (fixed-size integers). The loop halves x each iteration, so the number of iterations is

$$\Theta(\log_2 x).$$

Each step does $O(1)$-cost integer operations; hence

$$T(x, y) = \Theta(\log x).$$

Arbitrary-precision model (digit cost). Let $m = \lceil \log_{10} x \rceil$ and $n = \lceil \log_{10} y \rceil$ be the decimal digit lengths. There are $\Theta(\log x) = \Theta(m)$ iterations. Within each iteration we do additions on numbers whose length grows up to

$$\lceil \log_{10}(x \cdot y) \rceil = \Theta(m + n).$$

Thus the total time is

$$T(m, n) = \Theta\Big(m \cdot (m + n)\Big) = \Theta(mn + m^2).$$

W.l.o.g. assume $x \le y$ (so $m \le n$); then $mn + m^2 = \Theta(mn)$, hence

$$\boxed{T(m, n) = \Theta(mn)}.$$

This matches the lattice/grade-school multiplication's asymptotic cost.

Notes on logarithm bases. For any bases $a, b > 1$,

$$\log_a n = \frac{\log_b n}{\log_b a},$$

a constant-factor change. Therefore

$$O(\log_{10} n) = O(\log_2 n) = O(\log_{1000} n).$$

Intuition. The algorithm performs $\lfloor \log_2 x \rfloor + 1$ rounds of *halve-and-double*. Counting true costs of big-integer additions makes each round cost proportional to the current digit length (up to $m + n$), yielding the tight bound $O(mn)$ when $m \le n$.

1.8.3 Comparison with Lattice Multiplication

It is instructive to compare the duplication–mediation (peasant) algorithm with the lattice (grade-school) multiplication introduced earlier.

Table 2: Complexity comparison of two multiplication algorithms

Algorithm	**Asymptotic cost**	**Main cost source**
Lattice (Example 3)	$O(mn)$	Computes every digit pair $x_i y_j$ ($m \times n$ pairs) once.
Duplication–Mediation (Example 5)	$O(mn)$ (tight)	$\Theta(\log x) = \Theta(m)$ iterations, each adding numbers of length $\Theta(m + n)$.

Discussion.

- In the RAM model with fixed-size integers, peasant multiplication is $O(\log x)$ because each halving/doubling is constant time. Lattice multiplication is $O(mn)$ even on a unit-cost RAM if digits are unbounded.
- Under an *arbitrary-precision* model where cost grows with digit length, both algorithms have tight complexity $O(mn)$ when $m \leq n$.
- Constants and logarithm bases only affect the hidden constant factor, because $\log_a n = \frac{\log_b n}{\log_b a}$ implies

$$O(\log_{10} n) = O(\log_2 n) = O(\log_{1000} n).$$

Conclusion. Both algorithms ultimately require time proportional to the product of the input digit lengths when big-integer arithmetic is considered:

$$\boxed{T(m, n) = \Theta(mn)}.$$

The duplication–mediation method achieves this bound by iteratively halving x and doubling y, while the lattice method achieves it by directly computing all $m \times n$ digit-wise products.

1.8.4 Additional Example: 15×4

To further illustrate the duplication–mediation method, consider

$$15 \times 4.$$

Stepwise computation. We repeatedly halve the first number and double the second:

A	**B**	**Contribution**
15	4	4
7	8	8
3	16	16
1	32	32

Add all B's corresponding to odd A's:

$$15 \times 4 = 32 + 16 + 8 + 4.$$

Alternative chained expression. This can also be written as

$$15 \times 4 = 7 \times 8 + 4 = 3 \times 16 + 8 + 4 = 1 \times 32 + 16 + 8 + 4.$$

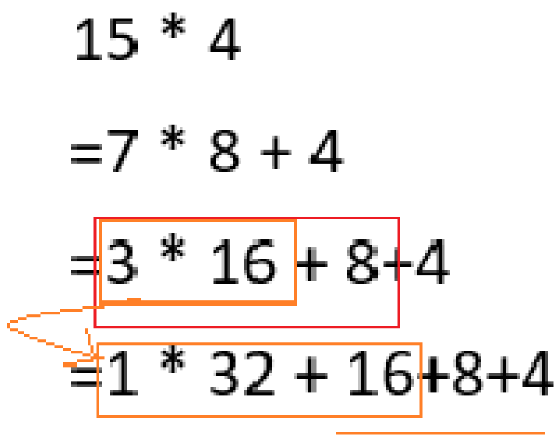

Figure 2: Visual representation of the duplication–mediation computation for 15×4.

Illustration. This example clearly shows how the algorithm repeatedly halves and doubles, accumulating the doubled values corresponding to odd rows.

1.9 Example 6: Compass and Straightedge Construction

Definition. We describe a geometric construction using only an ideal compass and straightedge. Given points A, B, C, D in the plane, define:

1. $\alpha \leftarrow \text{Orth}\big(A, \overline{AC}\big)$
   (the line through A perpendicular to $\overline{AC}$).
2. $E \leftarrow \text{Intersection}\big(C(AB), \alpha\big)$
   (intersection of the circle with center A and radius AB, and the line α).
3. $F \leftarrow \text{Intersection}\big(C(AD), \alpha\big)$
   (intersection of the circle with center A and radius AD, and the line α).
4. $\beta \leftarrow \text{Line}\big(E, F\big)$
   (the line through E and F).
5. $\gamma \leftarrow \text{Orth}\big(B, F\big)$
   (the line through B perpendicular to F).
6. The final point of interest is

$$P \leftarrow \text{Intersection}\big(\gamma, \overline{AC}\big).$$

Illustration.

Interpretation. This procedure formalizes a classical straightedge-and-compass construction:

- Circles $C(AB)$ and $C(AD)$ supply equal-length distances from A, ensuring the constructed points E and F are at known distances.

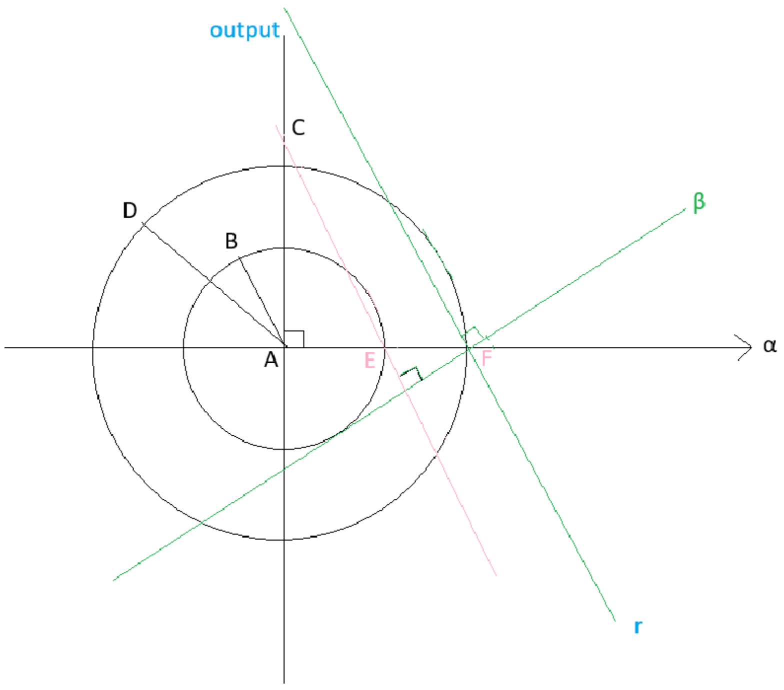

Figure 3: Compass and straightedge construction of point P. Points E and F are intersections of circles $C(AB)$ and $C(AD)$ with the perpendicular line α. Line β is defined by E and F, and γ is the perpendicular through B to F. The final point P is the intersection of γ with $\overline{AC}$.

- Orthogonal and intersection operations generate right angles and precise intersection points.
- The final intersection P is uniquely determined by the given data and Euclidean postulates.

1.10 Example 7: Voting and Fair Districting

Background. The figure illustrates a voting-district problem where the initial shape of the districts can lead to an *unfair* election outcome. The unfairness arises from the way boundaries are drawn, which can artificially favor or disfavor certain groups of voters.

Idea. Joe Mitchell proposed that by combining two neighboring regions into a single voting district and letting them vote together, the election result becomes more *fair*. This idea is sometimes referred to as "artitictual" (to emphasize an artificial yet structural adjustment of boundaries).

Conceptual Explanation.

- Separate small or oddly shaped districts can cause **gerrymandering effects**, i.e., disproportionate representation.
- By merging two such regions into one larger district and aggregating the votes, the overall outcome better reflects the actual voter proportions, hence **fairness**.

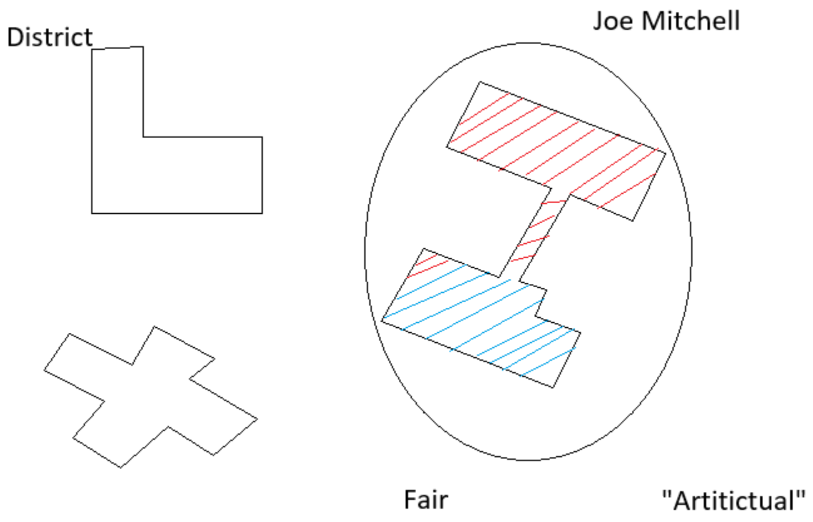

Figure 4: A schematic of the district-merging concept. The left shapes represent two separate regions (“District”) whose independent votes may give unfair representation. The right oval shows the merged configuration suggested by Joe Mitchell, where red and blue hatched areas indicate balanced influence, yielding a more equitable combined vote.

Conclusion. This example demonstrates how geometric reasoning (merging areas, adjusting shapes) can be used to achieve fairer voting districts. The proposal by Joe Mitchell shows that sometimes careful redistricting—rather than increasing the number of votes— is the key to achieving equity in representation.

2 Summary: Standard Components of an Algorithm Report

When presenting or analyzing an algorithm, it is useful to follow a structured template. This ensures both human readability and mathematical rigor.

2.1 1. Problem Definition

- Clearly describe the input and output specifications.
- Formalize any constraints or assumptions.
- State the computational goal (e.g., find shortest path, compute product).

2.2 2. Algorithm Description (Human Readable)

- Provide a high-level idea or intuition.
- Present pseudo-code or well-structured steps.
- Keep only the *necessary* details: focus on essential operations that determine correctness and complexity.

2.3 3. Correctness Proof

- Argue that the algorithm always produces the desired output.

- Techniques include:
  - Invariants or loop invariants.
  - Inductive arguments.
  - Reduction to known correct procedures.
- Discuss edge cases and termination conditions.

2.4 4. Running-Time Analysis

- Define the cost model (e.g., unit-cost RAM or digit-cost big integers).
- Express the total cost as a function of input size n or digit lengths m, n.
- Use asymptotic notation (Big-O, Θ, Ω) to summarize growth rate:

$$T(n) = O(\cdots), \quad T(m, n) = \Theta(\cdots).$$

- Explain whether lower-order terms or constant factors are significant in practice.

Takeaway. This four-part structure—**Problem Definition**, **Algorithm Description**, **Correctness**, and **Running Time**— provides a consistent framework for communicating algorithms in both theoretical and applied contexts.