CS483 Design and Analysis of Algorithms*

Fei Li

August 28, 2007

*This lecture note is based on Introduction to The Design and Analysis of Algorithms by Anany Levitin and Jyh-Ming Lie’s cs483 notes.
Overview

- Introduction to algorithms
- Course syllabus
What is an algorithm?

An **algorithm** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
Procedure of solving a problem on a computer

- Analyze and model a real problem as a computational problem
- Get the intuition
- **Design an algorithm**
  - Prove its correctness
- **Analyze the solution**, i.e., time efficiency, space efficiency, optimality, etc.
  - Can we get an improved solution?
  - Can we generalize our solution?
- Code an algorithm
Example of a computational problem

Statement of problem:
Rank students based on their grades

Input: A sequence of \( n \) numbers \(<a_1, a_2, \ldots, a_n>\)

Output: A reordering of the input sequence \(<a'_1, a'_2, \ldots, a'_n>\)
so that \( a'_i \leq a'_j \) whenever \( i < j \)

Algorithms:
- Selection sort
- Insertion sort
- Merge sort
- (many others)
Selection Sort

**Input:** An array $a[1], \ldots, a[n]$

**Output:** An array sorted in non-decreasing order

**Algorithm:**

```
for $i = 1$ to $n$
    swap $a[i]$ with smallest of $a[i], \ldots, a[n]$
```

**Example:** $<5,3,2,8,3> \rightarrow <2,3,3,5,8>$
An algorithm

Recipe, process, method, technique, procedure, routine,… with following requirements:

- **Finiteness**
  - terminates after a finite number of steps

- **Definiteness**
  - rigorously and unambiguously specified

- **Input**
  - valid inputs are clearly specified

- **Output**
  - can be proved to produce the correct output given a valid input

- **Effectiveness**
  - steps are sufficiently simple and basic
Why study algorithms?

- Theoretical importance
  - The core of computer science

- Practical importance
  - A practitioner’s toolkit of known algorithms
  - Framework for designing and analyzing algorithms for new problems
Example 1 – String Matching (Chap. 3 and 7)

- A string is a sequence of characters from an alphabet.
- **Problem:** search strings in a text

**Input:**
- a string of m characters called the pattern
- a string of n characters called the text

**Output:**
- a substring of the text that matches the pattern.
Example 2 – Travelling Salesman Problem (TSP) (Chapter 3)

**Problem:** Find the shortest tour through a given set of cities, which a salesman visits each city exactly once before returning to the starting city.

**Input:**
- A map of n cities
- Starting city

**Output:**
- The shortest tour which has all the cities
Travelling Salesman Problem

Weighted graph

Image from Wolfram MathWorld
Travelling Salesman Problem

A → C → G → F → B → H → D → E → A

Image from Wolfram MathWorld
Example 3 – Path Finding (Chap. 9)

Problem: Find the optimal path from the origin to the destination subject to certain objectives

Input:
- A weighted graph
- Origin and destination

Output:
- Optimal path
Example 4 – Interval Scheduling (Chap. 8 and 9)

**Problem:** Maximize the *maximum number or possible size* of requests.

**Input:**
- A shared resource used by one person at one time
- A bunch of requests
  - User i: Can I reserve the resource (classroom, book, supercomputer, microscope, ..) from time $s_i$ to $f_i$?

**Output:**
- A selection of requests with assigned resource
Example 5 – Stable Marriage (Chap. 10)

A set of marriages is *stable* if there are no two people of opposite sex who would both rather have each other than their current partners.

**Problem:** Find a stable marriage matching for given men and women to be paired off in marriages.

**Input:**
- n men and n women
- Each person has ranked all members of the opposite sex with a unique number between 1 and n in order of preference

**Output:**
- A matching
Basic issues related to algorithms

- How to design algorithms
- How to express algorithms
- Proving correctness
- Efficiency
  - Theoretical analysis
  - Empirical analysis
- Optimality and improvement
Greatest Common Divisor Problem

Problem: Find \( \text{gcd}(m,n) \), the greatest common divisor of two nonnegative, not both zero integers \( m \) and \( n \)

Examples: \( \text{gcd}(60,24) = 12 \), \( \text{gcd}(60,0) = 60 \)
Solution 1

Observation: $\gcd(m, n) \leq \min\{m, n\}$

Consecutive integer checking algorithm

- **Step 1** Assign the value of $\min\{m, n\}$ to $t$
- **Step 2** Divide $m$ by $t$. If the remainder is 0, go to Step 3; otherwise, go to Step 4
- **Step 3** Divide $n$ by $t$. If the remainder is 0, return $t$ and stop; otherwise, go to Step 4
- **Step 4** Decrease $t$ by 1 and go to Step 2
Solution 2

Middle-school procedure

- **Step 1** Find the prime factorization of \( m \)
- **Step 2** Find the prime factorization of \( n \)
- **Step 3** Find all the common prime factors
- **Step 4** Compute the product of all the common prime factors and return it as \( \gcd(m, n) \)

Example: \( \gcd(60, 24) \)

- \( m = 60 = 2 \times 2 \times 3 \times 5 \)
- \( n = 24 = 2 \times 2 \times 2 \times 3 \)

\[ \gcd(m, n) = \gcd(60, 24) = 2 \times 2 \times 3 = 12 \]

Not an algorithm! Prime factorization
Prime Factorization

**Input:** Integer $n \geq 2$

**Output:** A sequence of prime numbers $S$, whose multiplication is $n$.

**Algorithm:**

1. Find a list of prime numbers $P$ that are smaller than $n$
2. $i \leftarrow 2$
3. While $i < n$ do
   - If $n \% i = 0$
     - Then $s \leftarrow i$, $n \leftarrow n / i$
   - Else $i \leftarrow$ next prime number
Sieve

**Input:** Integer \( n \geq 2 \)

**Output:** List of primes less than or equal to \( n \)

**Algorithm:**

```plaintext
for \( p \leftarrow 2 \) to \( n \) do
    \( A[p] \leftarrow p \)

for \( p \leftarrow 2 \) to \( \lfloor n \rfloor \) do
    if \( A[p] \neq 0 \)  //\( p \) hasn’t been previously eliminated from the list
        \( j \leftarrow p \times p \)
        while \( j \leq n \) do
            \( A[j] \leftarrow 0 \) //mark element as eliminated
            \( j \leftarrow j + p \)
```

```
Sieve (cont.)

Example

2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
2  3  5  7  9  11  13  15  17  19
2  3  5  7  11  13  17  19
2  3  5  7  11  13  17  19
Solution 3 - Euclid’s Algorithm

Euclid’s algorithm is based on repeated application of equality
\[ \text{gcd}(m,n) = \text{gcd}(n, m \mod n) \]
until the second number becomes 0, \( \text{gcd}(m, 0) = 0 \).

Example: \( \text{gcd}(60,24) = \text{gcd}(24,12) = \text{gcd}(12,0) = 12 \)

Algorithm

\[
\begin{align*}
\text{while } n \neq 0 & \text{ do} \\
& \quad r \leftarrow m \mod n \\
& \quad m \leftarrow n \\
& \quad n \leftarrow r \\
\text{return } m 
\end{align*}
\]
Algorithm design techniques/strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound
Analysis of algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency

- Does there exist a better algorithm?
  - Simplicity
  - Generality
  - lower bounds
  - optimality
Syllabus

Lecture time
- Tue & Thu 3:00-4:15pm

Office Hour
- Tue & Thu 4:30-5:30pm
- Office: 443 ST II

Course webpage:
- www.cs.gmu.edu/~lifei/teaching/cs483_fall07/
Syllabus (cont.)

TA: Yanyan Lu

- Email: ylu4@gmu.edu
- Office hour: Wed & Friday 4:00pm – 5:00pm
- Room 437 STII

Required Textbook:

Syllabus (cont.)

Topics

- Analysis of Algorithm Efficiency
- Brute Force
- Divide (decrease) and Conquer
- Transform and Conquer
- Greedy Techniques
- Dynamic Programming
- Iterative Improvement
- Limitations of Algorithm Power and Coping with Limitations
Syllabus (cont.)

Grading (tentative)
- Biweekly assignment (40%)
  - Work on your assignments independently.
  - List all the resources such as web, books and other students that may have helped with your solution.
  - Hand in hard copies.
  - One late submission (up to one week past the due date) per person per semester is permitted.
- Midterm exam (25%)
- Final exam (35%)
  - Two pages (letter size) of notes are allowed for both exams.
Some Suggestions

- Start working on assignments early
- Review notes and textbook after class
- Ask questions!
Before next class
   Read Chapter 1.1, 1.2, 1.4 and Appendix A.

Next class
   Algorithm analysis
   Recursion