Data Structures & Algorithms II

Contact Information

Nurul Liyana bt. Mohd. Zulkufli

Assistant Professor, Dr.

Department of Computer Science

Kulliyyah of Information & Communication Technology

Office: C4-22

Consultation hour: After 5pm (Mon-Thur, appointment)

Email: liyanazulkufli@iium.edu.my

Lectures

Section 1

- Time: **14.00 – 15.20 PM**
- Date: **Tuesday & Thursday**
- Location: **ICT TL-E5-02, Level 5E**

Section 2

- Time: **15.30 – 16.50 AM**
- Date: **Tuesday & Thursday**
- Location: **ICT TL-E5-02, Level 5E**

Objectives

- construct, understand and analyse data structures and algorithms
- write efficient computer programs using algorithm design techniques
- develop applied algorithms for computer networking, security and other areas

Required Reference

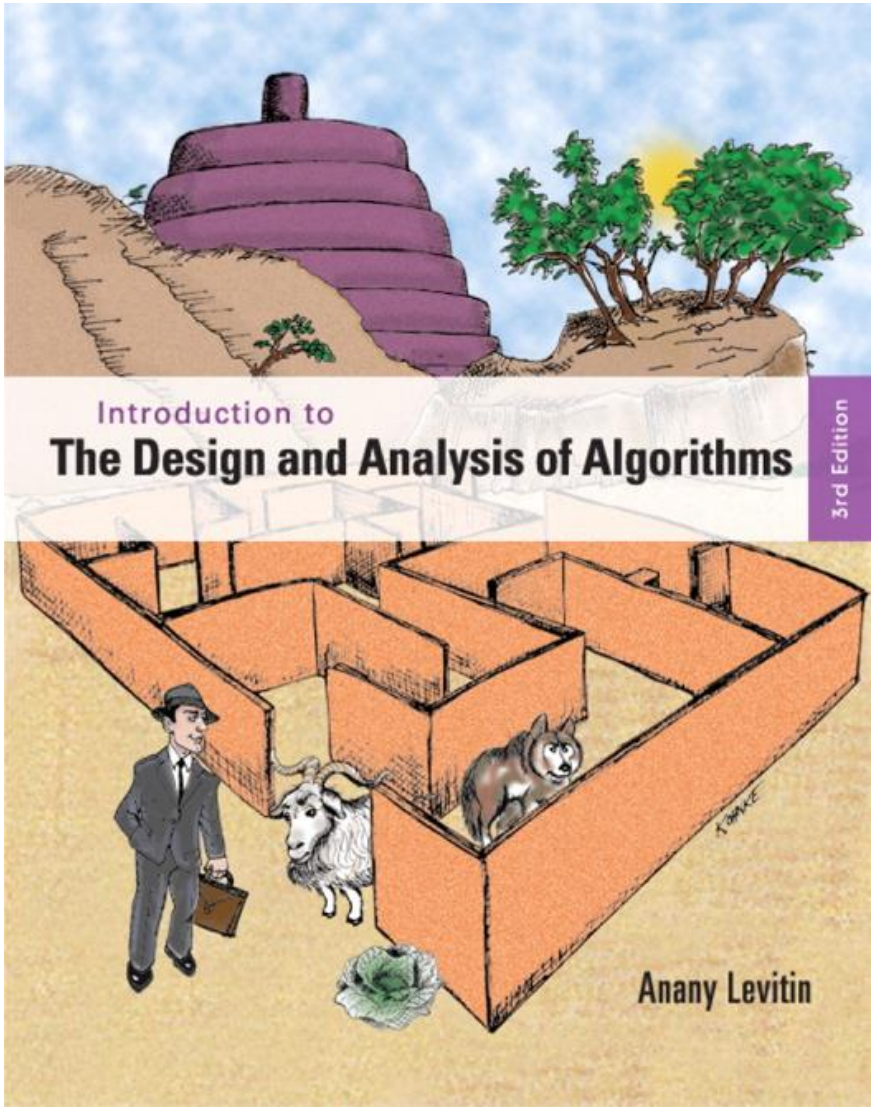
Levitin, A.

Introduction to the Design and Analysis of Algorithms.

3/E.

USA: Addison-Wesley.

2012



Recommended References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (**2009**) **Introduction to Algorithms**. 3/E. Cambridge, Massachusetts & London, England: The MIT Press.

2. Sedgewick, R., Wayne, K. (**2011**) **Algorithms**. 4/E. NJ: Addison-Wesley.

3. Cormen, T.H. (**2013**) **Algorithms Unlocked**. Cambridge, Massachusetts & London, England: The MIT Press.

Lecture Notes (Slides)

http://italeem.iium.edu.my/
(or others)

- **Course Information**
- **Lecture Slides/Notes**
- **Home assignments**
- **Announcements, Discussions, Q&A, etc.**

Course Assessments & Marking

METHOD	MARKING (%)
Assignments	20
Quizzes	20
Mid-term examination	20
Final examination/assessment	40

Course Outline

	Algorithm Analysis
	Greedy Method
	Divide & Conquer
	Dynamic Programming

Course Outline

	Balanced Trees
	Graphs
	Shortest Paths
	Minimum Spanning Trees
	Fast Sorting & Searching
	String Matching
	Network Flow & Matching

Important Notes

- **Attendance** is **compulsory**
- **University dress code**
- **No gadgets…** (power off or mute mode, except when requested)
- **No late homework**
- **No make-up exams/quizzes**
- **DO NOT BE LATE**

Please fill up this form

- https://goo.gl/forms/XOLnjs8QyS4Tb1LV2
- (can find in i-Taleem)

Padlet

- https://padlet.com/sem2_2018_2019/dsa2

Has a pet cat (at home etc)	Lives in different mahallah	Has a car	Likes Python language
Lives outside campus	Prefers C/C++	From other country than you	Different gender
Has travelled to another country	Has a bike	Does programming almost every day	Has met a new muslim
Prefers Java	Has programmed an Arduino/ Rasp Pi	Has a Mac computer or Ipad/Iphone	Play sports every week

INTRODUCTION

Why to Study Algorithms?

If you want to be a **computer professional, programmer**, there are some reasons:

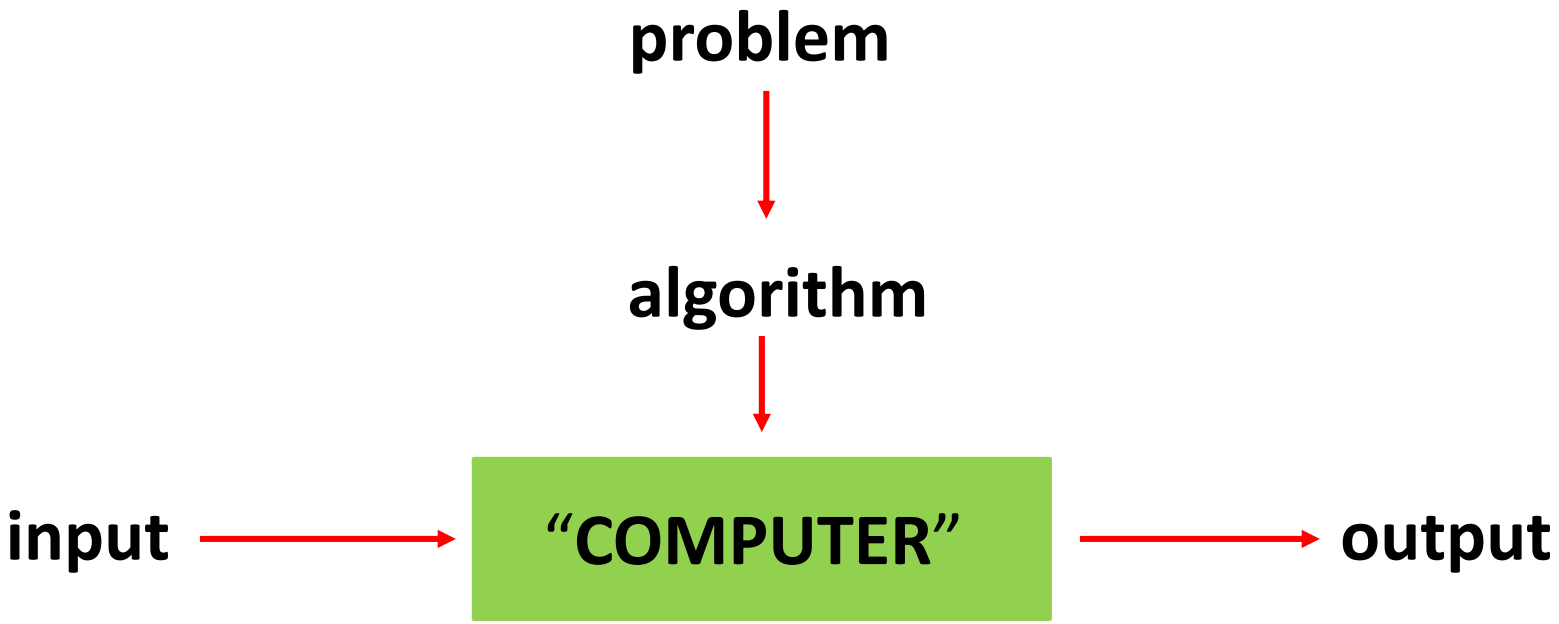
- **Practical**:
  - to **know** a standard set of important algorithms
  - to **design** new algorithms
  - to **analyze** the correctness & efficiency of algorithms
- **Theoretical**:
  - the study of algorithms is the **core of Computer Science**.


The Upshot
ROBO RECRUITING
Can an Algorithm Hire Better Than a Human?
MANAGEMENT
Algorithm That Tells the Boss Who Might Quit
Wal-Mart, Credit Suisse Crunch Data to See Which Workers
Leave or Stay
If Algorithms Know All, How Much Should Humans Help?
ALGORITHMS TAKE CONTROL OF WALL STREET
Prisons turn to computer algorithms for deciding who to parole
Can maths find you love? eHarmony's love algorithm
love? The dating
claim to have
PERSONALITY TESTS
This Algorithm Knows You Better Than Your Facebook Friends Do
New Google algorithm elevates facts; critics worry 'dissidents' will be quashed
29 comments
The Algorithm Economy Heads To Amazon
1,796
THE SATURDAY ESSAY
Bitcoin and the Digital Currency Revolution
For all bitcoin's growing pains, it represents money and global finance.
THE WALL STREET JOURNAL. TECH
TECHNOLOGY
At UPS, the Algorithm Is the Driver
Turn right, turn left, turn right: inside Orion, the 10-year effort to squeeze every penny
By STEVEN ROSENBUSH and LAURA STEVENS
Feb. 16, 2015 8:28 p.m. ET
87 COMMENTS

For fun and profit.

What is an Algorithm?

An **ALGORITHM** is a sequence of **unambiguous** (deterministic) **instructions** for solving a problem, i.e., for obtaining a required **output** for any **legitimate input** in a **finite amount of time** using **finite amount of space**.



Problem

- A rectangular floor measures 60 inch x 24 inch.
- What is the largest square tiles that can be used to cover the floor exactly?

Problem: Greatest Common Divisor

Problem: *Find* $\gcd(m, n)$, **the greatest common divisor** *of two nonnegative, not-both-zero integers* m *and* n.

- Brute force
- Prime factorization (common prime factor)
- Consecutive integer checking
- Euclid's algorithm

Brute Force

Brute force algorithm for computing $\gcd(m, n)$

Step 1: Find the divisors of m: divide m with every number until m

Step 2: Find the divisors of n.

Step 3: Find the greatest number in common.

Prime Factors

Prime factors' algorithm for computing $\gcd(m, n)$

Step 1: Find the prime factors of m.

Step 2: Find the prime factors 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 $\gcd(m, n)$.

Consecutive Integer Checking

Consecutive integer checking algorithm for computing $\gcd(m, n)$

Step 1: Assign the value of $\min\{m, n\}$ to t.

Step 2: Divide m by t. If the remainder of this division is 0, go to **Step 3**.

Step 3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise go to **Step 4**.

Step 4: Decrease the value of t by 1; go to **Step 2**.

Euclid's Algorithm

Problem: *Find* $\gcd(m,n)$, **the greatest common divisor** *of two nonnegative, not-both-zero integers* m *and* n.

Examples:

$$\gcd(60,24) = 12, \quad \gcd(60,0) = 60, \quad \gcd(0,0) = ?$$

Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0.

Example:

$$\gcd(60,24) = \gcd(24,12) = \gcd(12,0) = 12.$$

Structured Description

Euclid's algorithm for computing $\gcd(m, n)$

Step 1: If $n = 0$, return the value of m and stop; otherwise, proceed **Step 2**.

Step 2: Divide m by n and assign the value of the remainder to r.

Step 3: Assign the value of n to m and the value of r to n. Go to **Step 1**.

Pseudocode

Algorithm $\text{Euclid}(m, n)$

// Computes $\gcd(m, n)$ by Euclid's algorithm

// **Input**: two nonnegative, not-both-zero integers m and n

// **Output**: Greatest common divisor of m and n

while $n \neq 0$ **do**

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

Important Points

- The **nonambiguity** requirement cannot be compromised
- The **range of inputs** has to be specified carefully
- The same algorithm can be represented in **several different ways**
- The same problem can be solved by **different algorithms** based on different ideas with **drastically different speeds**

Algorithm Design & Analysis Process

1. Understanding the problem
2. Ascertaining the capabilities of the computing device
3. Choosing between exact and approximate problem solving
4. Algorithm design techniques
5. Designing an algorithm and data structures
6. Methods of specifying an algorithm
7. Proving the correctness of an algorithm
8. Analyzing an algorithm
9. Coding an algorithm

Understanding the Problem

- Understand a given problem **completely**
- Do a few small **examples** (by hand!)
- Think about **special cases**
- Ask **questions**
- Identify the **type** of the problem
- Specify exactly the set of **instances** of the problem

Computational Means

- Computer architecture:
  - **sequential** (random-access machine model): instructions are executed one after another, one operation at a time (sequential algorithms)
  - **parallel** (not random-access machine model) instructions are executed concurrently, i.e., in parallel (parallel algorithms)
- Speed
- Memory

Exact vs. Approximate Solving

- **Exact algorithms**
  - sorting, searching, solving linear equations, etc.
  - polynomial time and space
- **Approximation algorithms**
  - extracting square roots, solving nonlinear equations, etc.
  - exact algorithms are slow (very large number of choices)

Algorithms Design Techniques

An **algorithm design technique** (strategy, paradigm) is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing:

- Brute force
- Decrease and conquer
- Divide and conquer
- Transform and conquer
- Dynamic programming
- Greedy technique
- Iterative improvement
- Space and time tradeoffs
- Backtracking
- Branch and bound

Algorithm Design & Data Structures

- Algorithm design techniques are general approaches for algorithmic problem solving.
- Designing an algorithm for a specific problem is generally challenging task:
  - design techniques cannot be directly applicable
  - several techniques need to combined
- Choose data structures appropriate for the operations performed by algorithms:
  - array, list, stack, queue, or trees?
  - structure or restructure data

Methods of Specifying an Algorithm

- Structured description (a step-by-step form): a natural language
  - less precise, less succinct, ambiguous
- Flowchart: a collection of connected geometric shapes containing descriptions of algorithm's steps
  - inconvenient
- Pseudocode: a mixture of a natural language and programming language like constructs
  - more precise, more succinct
  - **for**, **if** **while**, ←, etc.

Proving an Algorithm's Correctness

- After the specification of an algorithm, we need to prove its correctness: prove that the algorithm gives the required result for every legitimate input in a finite amount of time.
- Proof techniques: mathematical induction, loop invariants
- For approximation algorithms we show that the error produced by the algorithm does not exceed a predefined limit.

Analyzing an Algorithm

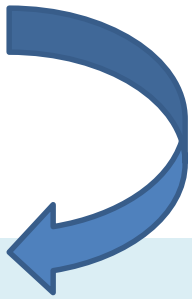
Algorithms qualities:

- time efficiency: how fast the algorithm runs
- space efficiency: how much the algorithm uses extra memory
- simplicity: easier to understand, easier to program, fewer bugs
- generality:
  - generality of the problem
  - the sets of inputs

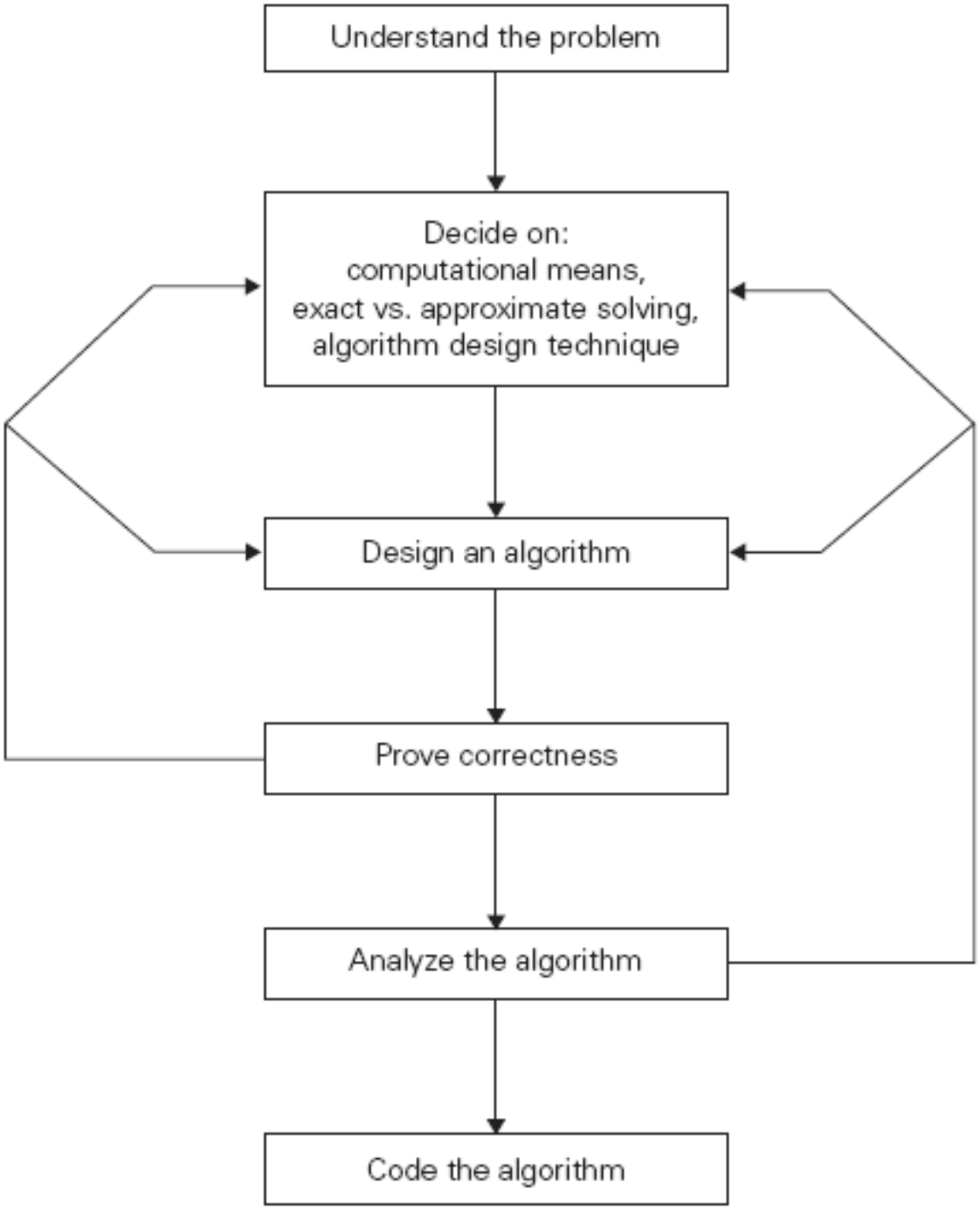
Coding an Algorithm

- Program implementation of an algorithm – peril & opportunity – may be incorrect or inefficient
- Correctness/validity by formal verification (mathematical) or by testing
- Code optimization mode (by modern compilers)
- Standard tricks for computing loop's invariant (an expression that does not change its value) outside the loop, collecting the common subexpressions, etc.
- Such improvements can speed up only by a constant factor

```
do {
        item = 10;
        value = value + item;
        } while(value<100);
```



```
Item = 10;
do {
        value = value + item;
        } while(value<100);
```


Understand the problem
Decide on:
computational means,
exact vs. approximate solving,
algorithm design technique
Design an algorithm
Prove correctness
Analyze the algorithm
Code the algorithm