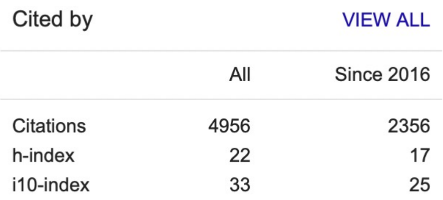
Introduction

Kyunghan Lee
Networked Computing Lab (NXC Lab)
Department of Electrical and Computer Engineering
Seoul National University
https://nxc.snu.ac.kr
kyunghanlee@snu.ac.kr

Networked Computing (NXC) Lab @ SNU

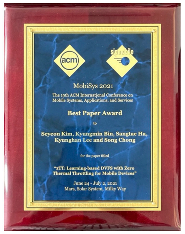
- □ Kyunghan Lee
  - ▪ Associate Professor, ECE



Cited by		VIEW ALL
	All	Since 2016
Citations	4956	2356
h-index	22	17
i10-index	33	25

- □ Community Roles
  - ▪ Editors for IEEE/ACM ToN, IEEE TVT, Computer Networks
  - ▪ General Co-chair for ACM MobiHoc 2022, TPC for MobiSys, CoNEXT, MobiHoc, Infocom, etc.
- □ Awards
  - ▪ ACM MobiSys Best Paper Award 2021
  - ▪ IEEE ComSoc William R. Bennett Prize 2016
  - ▪ IEEE ComSoc William R. Bennett Prize 2013

ACM MobiSys2021
The 19th ACM International Conference on Mobile Systems, Applications, and Services
June 24 - July 2, 2021 • Mars, Solar System, Milky Way



MobiSys 2021

The 19th ACM International Conference on Mobile Systems, Applications, and Services

Best Paper Award

to

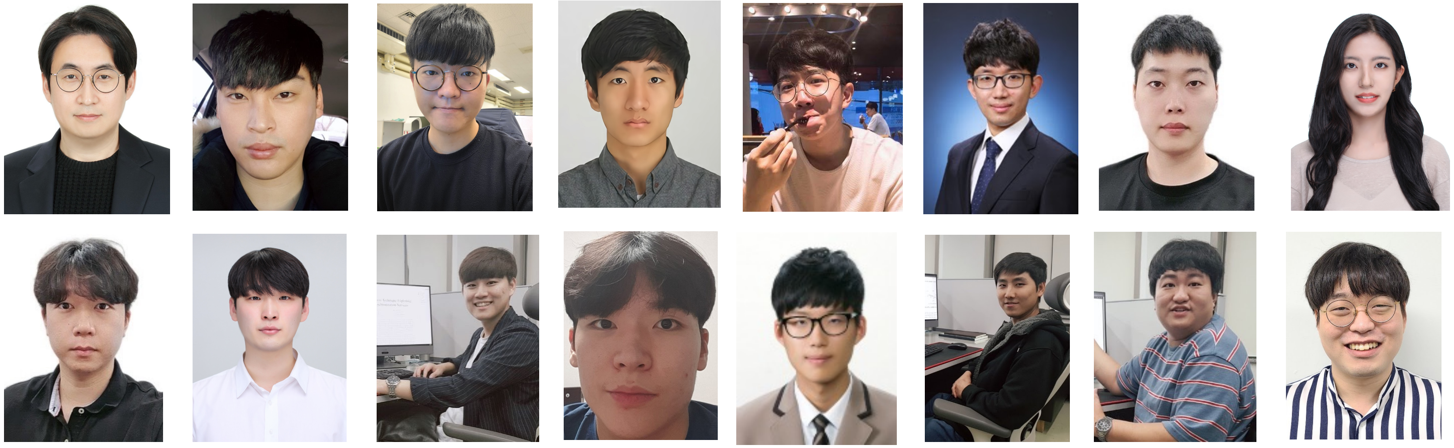
Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee and Song Chong

for the paper titled

"zTT: Learning-based DVFS with Zero Thermal Throttling for Mobile Devices"

June 24 - July 2, 2021
Mars, Solar System, Milky Way

- □ NXC Memebers
  - ▪ 8+3 Ph.D students, 4 MS/Ph.D students



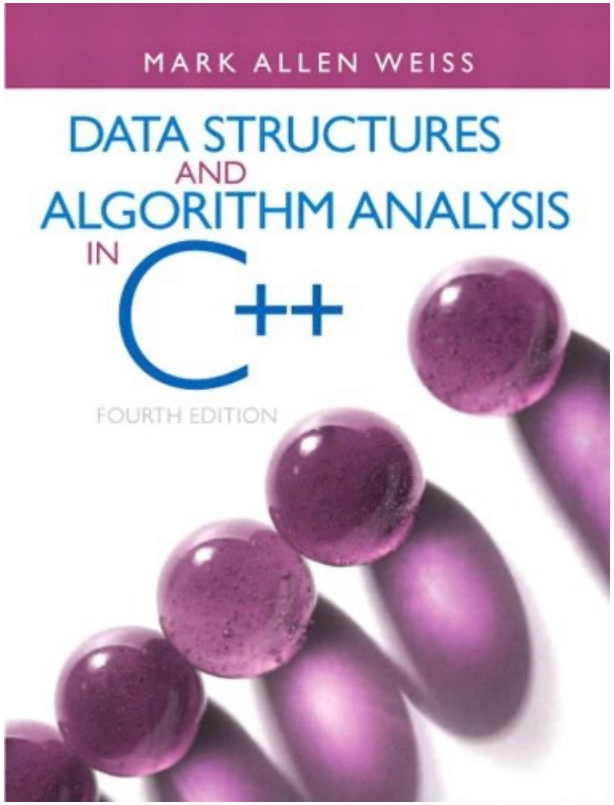
Introduction to Data Structures

Course No.	430.217	Lecture No.	001	Course Title (Subtitle)	Introduction to Data Structures	Credit	3
Instructor	Name	Kyunghan Lee (post.: Assoc. Proessor.)			Homepage	https://nxc.snu.ac.kr	
	E-mail	kyunghanlee@snu.ac.kr			Phone No.	02-880-1672	
	Office Hours : Tue/Thu 2pm - 4pm, Building 301, Room 1006						
Prerequisite	Programming Methodology						
* 1.Course Goals	Data structures constitute the basis of computation, providing how to organize, manage, read, or write the data when building up computer programs. In this class, we will be covering many different elemental data structures, ranging from stack, queue, list, tree to graphs, as well as covering basic algorithms with those data structures. All students are expected to complete multiple programming assignments asking to implement relevant data structures (and algorithms) in C++.						
* 2.Materials and Reference	Primary: Lecture slides (to be provided) Reference: Data Structures and Algorithm Analysis in C++ (4th Edition), Mark Allen Weiss						
* 3.Evaluation Method	Attendance	Assignments & Mid-term	Final	Quiz	Attitude	Other	Total
	5%	40%	40%	10%	5	0	100%
	Attendance Policy :	Students who are absent for over 1/3 of the class will receive a grade of 'F' or 'U' for the course. (Exceptions can be made when the cause of absence is deemed unavoidable by the course instructor.)					
	Remarks:	The evaluation method may change depending on the lecture progress.					

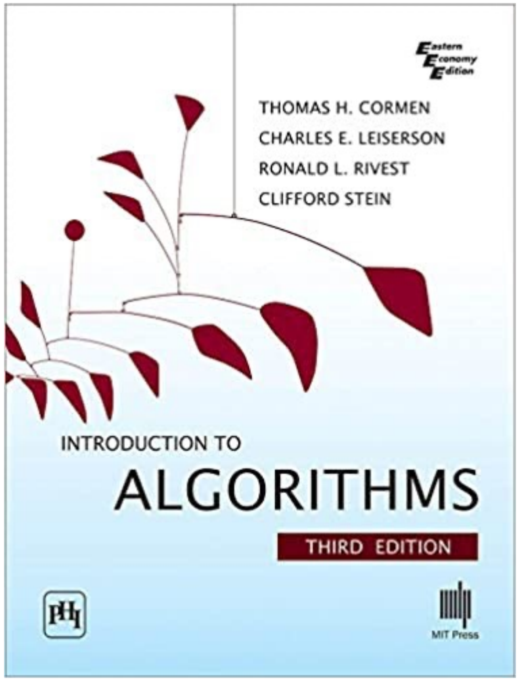
# of term projects and the project topics may change depending on the situations.

Introduction to Data Structures

Lecture slides base on Douglas W. Harder's notes



Data Structures & Algorithm Analysis in C++ (4th Edition)
Mark Allen Weiss
Pearson



Introduction to Algorithms (3rd Edition)
Thomas H. Cormen et al.
MIT Press

Data Structures

- □ In this course, we will look at:
  - Data structures for efficiently storing, accessing, and modifying data
  - Some Algorithms for solving problems efficiently

- □ We will see that all data structures have trade-offs
  - There is no ultimately good data structure…
  - The choice depends on your requirements

Good Data Structures? Bad?

- □ Consider accessing the kth entry in an array or linked list
  - ▪ In an array, we can access it using an index array[k]
    - Fast
  - ▪ In a linked list, we must step through the first k – 1 nodes
    - Slow

- □ Consider searching for an entry in a sorted array or linked list
  - ▪ In a sorted array, we use a fast binary search
    - Very fast
  - ▪ In a linked list, we must step through all entries less than the entry we're looking for
    - Slow

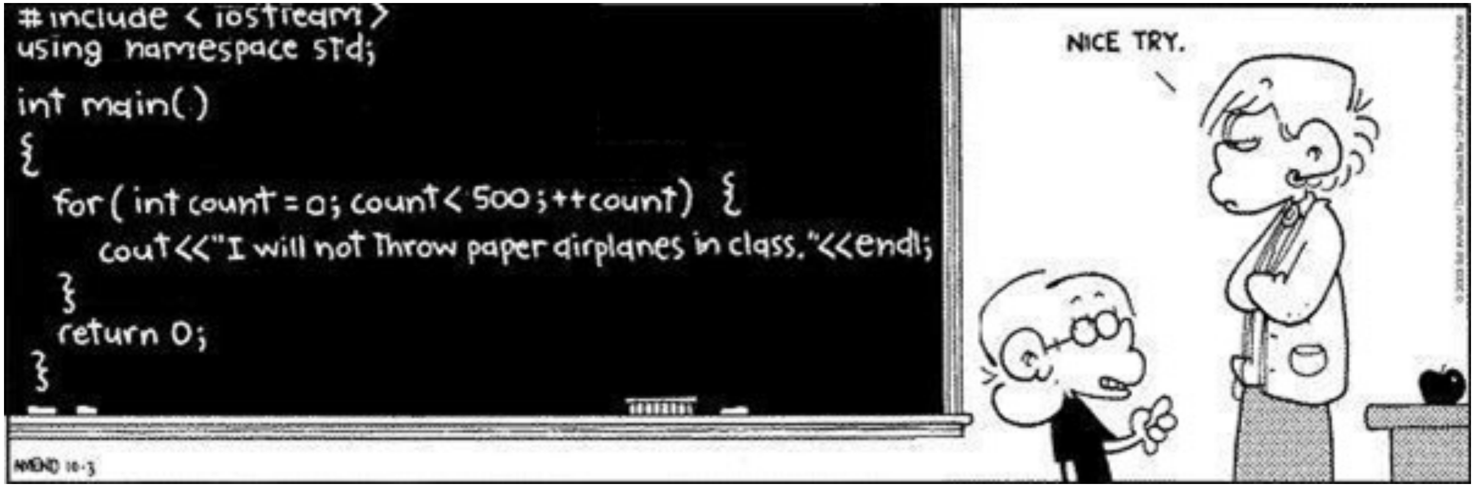
Topics to be covered

- The course is divided into numerous topics
  - Basics
    - C++
    - Algorithms
    - Time complexity and space analysis
  - List/Stack/Queue
  - Tree
  - Hash
  - Priority Queue
  - Sorting
  - Graph

- Assignments/Projects
  - C++
  - Linux

C++

□ You will be using the C++ programming language in this course



```
#include <iostream>
using namespace std;
int main()
{
  for(int count=0; count<500;++count) {
     cout<<"I will not Throw paper airplanes in class."<<endl;
  }
  return 0;
}
```

Modified for C++ from http://www.foxtrot.com/

C++

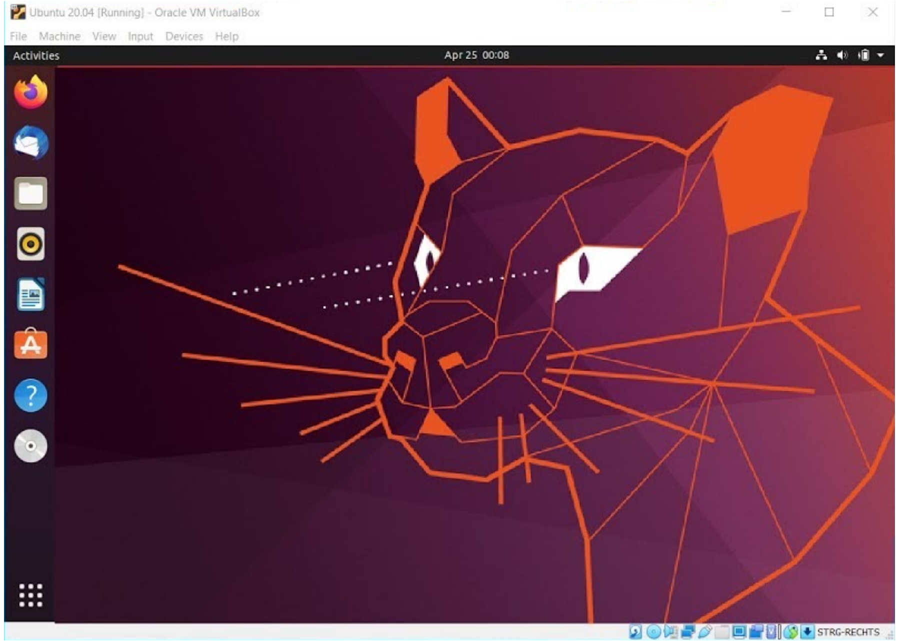
- □ This course does not teach C++ programming
  - ▪ You will use C++ to demonstrate your knowledge in this course

- □ Again, this course assumes that you are familiar with C++

- □ Other sources of help in C++ are:
  - ▪ TAs
  - ▪ YouTube tutorials
  - ▪ Online tutorials: http://www.cplusplus.com/

Linux

- □ You will be exposed to the Linux environment
  - ▪ All the projects will be marked in Unix using the g++ compiler
  - ▪ We will help you get familiar with Linux
  - ▪ Will be providing a short tutorial on using VirtualBox
    - VirtualBox allows you to run (virtualized) Linux on Windows and Mac



Plagiarism

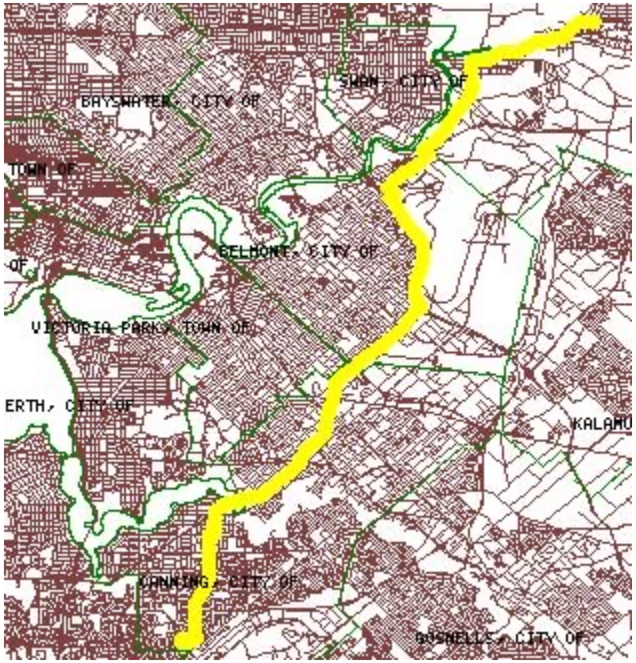
- All projects must be done individually:
  - You should not copy code directly from any other source
  - If you saw another code (from books or lecture notes), you must include a reference in your project
    - Leave a comment!
  - You should not share your code with any other students by transmitting completed functions to your peers
    - Both students (who shared or copied) will get the same penalty
  - You may discuss projects together and help another student debug his or her code; however, you cannot give the exact solution (all the discussion/debugging must stay in high level)
    - All such collaborations must be documented in your source code
  - Your misbehavior may lead you to getting F, and the cases will be reported to the student council

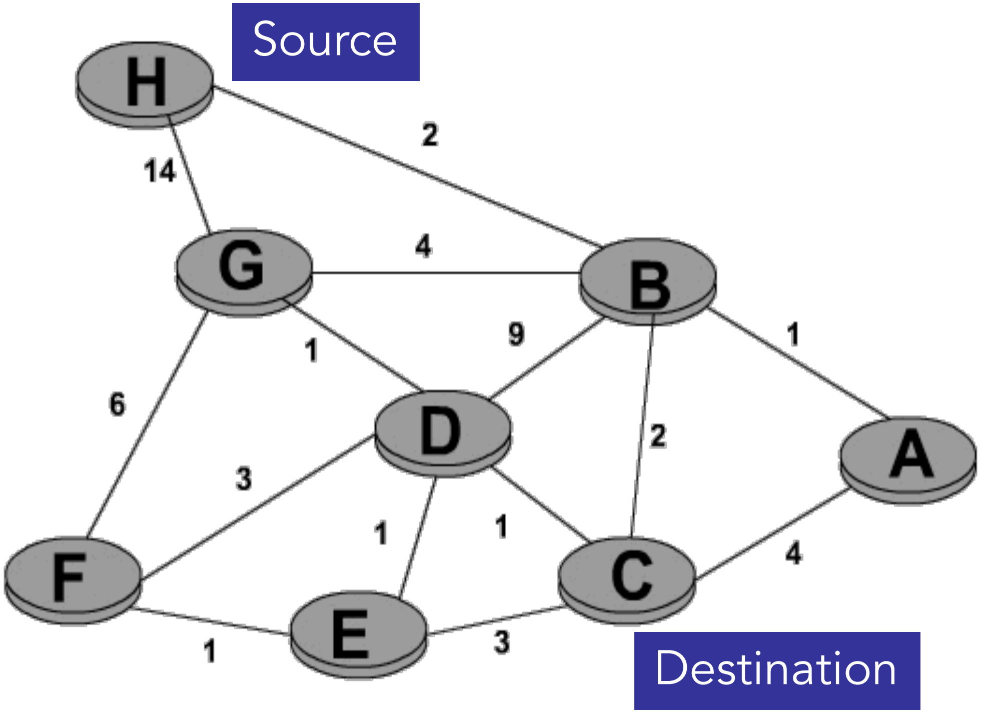
Plagiarism

- The best way to avoid plagiarism is:
  - review the C++ tutorial
  - read the project as soon as it is available
  - start the project so that there is sufficient time to contact the T.A. or myself if you have difficulty
  - do not give your code to anyone

Why to Study Algorithms?

- □ Simply, it is cool to invent something that solves a problem
- □ Example: Shortest Path Problem and Algorithm
  - ▪ Used in Google Maps



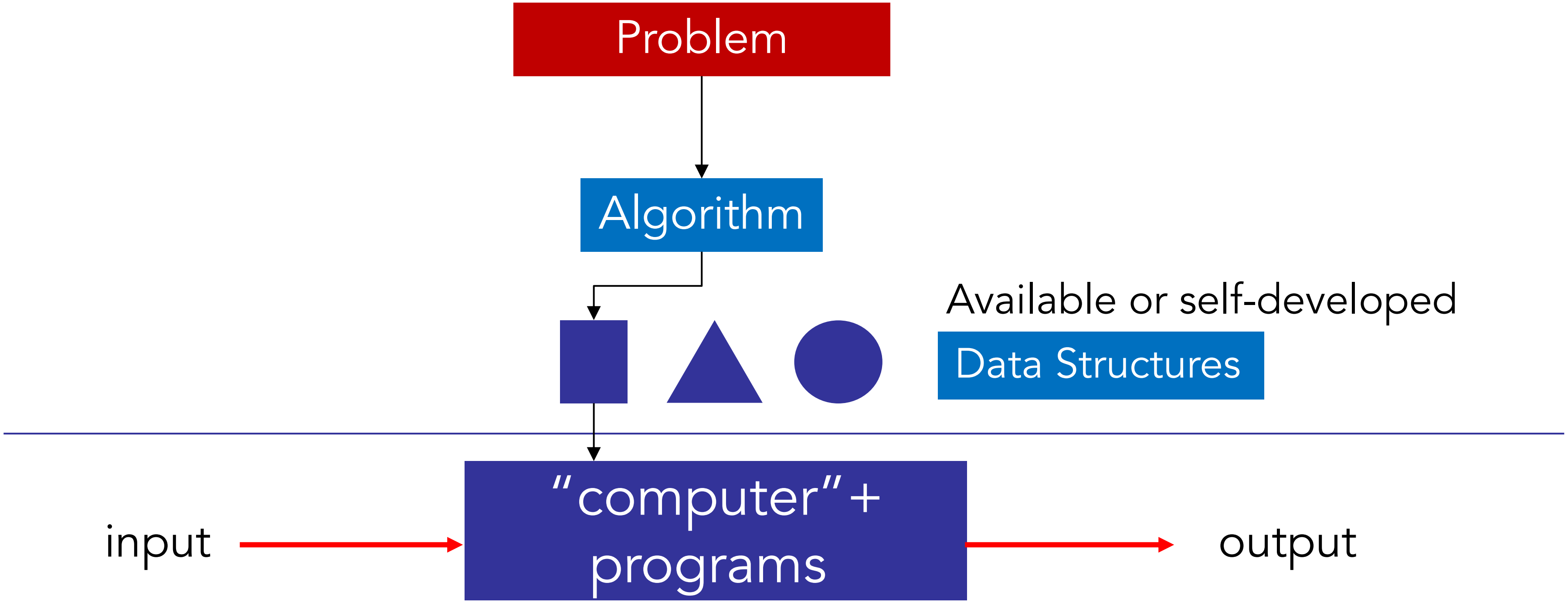


Source

Destination

Algorithm and Data Structure

- An algorithm is a sequence of unambiguous instructions/ operations for solving a problem
  - i.e., for obtaining a required output for any legitimate input in a finite (shorter) amount of time.



The Nature of This Course

- □ This is one of the most important courses of computer science
  - It plays a central role in both the science and the practice of computing
  - It tells you how to design a program to solve important problems efficiently, effectively and professionally
  - The knowledge in this course differentiates a 'real' computer-major student from other students

- □ What you are going to learn is independent of any specific software
  - You can implement the data structures and algorithms you learned in this course by any computing languages, such as C, C++, Java, Python, Go, etc.

Example: Sorting

- Statement of problem:
  - Input: A sequence of n numbers $< a_1, a_2, \cdots, a_n >$
  - Output: A reordering of the input sequence $< a'_1, a'_2, \cdots, a'_n >$
  - so that $a'_i \leq a'_j$ whenever $i < j$
- Instance: A sequence $< 5,3,2,8,3 >$
- Algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
  - (many others)

Selection Sort

□ Input: An array $a[1], a[2], \cdots, a[n]$

□ Output: An array $a[1 \cdots n]$ sorted in non-decreasing order

□ Algorithm:

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

Some Important Points

□ Each step of an algorithm should be unambiguous

□ The range of inputs has to be specified carefully

□ The same algorithm can be represented in different ways

□ The same problem may be solved by different algorithms

□ Different algorithms may take different time to solve the same problem – we may prefer one to the other

Fundamentals of Algorithmic Problem Solving

1. Understand the problem
2. Ascertain the capabilities of a computational device
3. Choose between exact and approximate problem solving
4. Decide on appropriate data structure
5. Apply algorithm design techniques
6. Specify an algorithm with a pseudocode

   Pseudocode (for, if, while, //, ←, indentation…)
7. Prove the correctness of the algorithm (e.g., mathematical induction)
8. Analyze the algorithm (e.g., complexity, efficiency, modularity)
9. Code an algorithm with a language

   (Compiler will let computer understand your code)

In general

- A good algorithm is typically a result of repeated effort and rework
- People have spent long time to devise an algorithm with
  - Better data structure
  - Better time or space efficiency
  - Better convenience in implementation

- There are well known Problems and Solutions
  - Sorting
  - Searching
  - Shortest paths in a graph
  - Minimum spanning tree
  - Traveling salesman problem
  - Knapsack problem

In general

- Conventional algorithm design techniques work well in most problems
  - Brute force
  - Divide and conquer
  - Decrease and conquer
  - Transform and conquer
  - Greedy approach
  - Dynamic programming
  - Backtracking and branch and bound
  - Space and time tradeoffs

Reading Assignment #1 – Chapter 1