COMP 2119
Introduction to Data Structures and Algorithms

Lecturer: SM Yiu

Teaching Assistants: Meiqi He
Linru Zhang

Course web page
http://moodle.hku.hk
COMP2119 [Section 1A, 2016]

Course Structure

Lectures

Tutorials
(Review, On-demand)

Consultation
& Help channels

Assignments & Tests

Examination

 + 

~ 34 "lecture + tutorial" hours
(~3 hours per week)

+ *on-demand tutorials*

~ 3 written assignments (18%)
+ 1 programming assign (7%)
Two tests (15%)

Consultation hours
Emails
Discussion forum

This course is like a math course, don't wait until the last minute to ask questions.

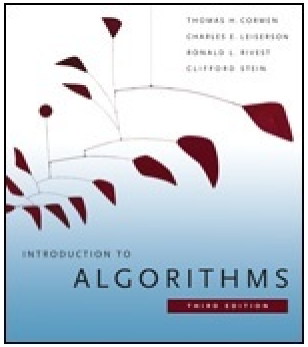
Coursework: 40%; Final Exam: 60%

Schedule

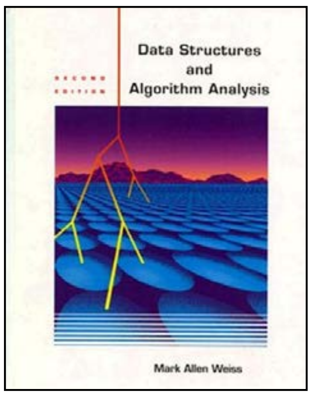
Mon	Tue	Wed	Thu	Fri
	14:30-15:30 [SMYiu, CB402] **15:30-16:20 [lecture/ tutorial]**		14:30-15:30 [SMYiu, CB402] **15:30-17:20 [lecture/ tutorial]**	

- Outside consultation hours: by appointments via emails.
- For TAs' consultation hours & locations, refer to webpage.

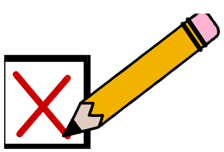
References:



(1)
Introduction to Algorithms, 3rd ed.
Thomas H. Cormen, Charles E. Leiserson,
Ronald L. Rivest, Clifford Stein
The MIT Press



(2)
Data structures and algorithm analysis
Mark Allen Weiss
Addison Wesley



Tentative Test Schedule

Test 1: 28 Oct (after the reading week)
Test 2: 29 Nov (last week)

About the Course

- Prerequisite: Programming
- Co-requisite: Programming technologies and tools

What is an algorithm?

Problem → Program

```
#include <iostream>
int main( ) {
        .....
        for (int I=1;I<10;I++){

}
```

Statements of the exact syntax of some prog. lang. specifying the precise steps to be carried out by the computer

e.g.
Write a program to solve the following problem: "Given a set T of numbers and a sum s, find a subset (of size <= x) of T such that the sum of the numbers in this subset equals s."

What is an algorithm?

Problem → ? → Program

```
#include <iostream>
int main( ) {
        .....
        for (int I=1;I<10;I++){
}
```

Statements of the exact syntax of some prog. lang. specifying the precise steps to be carried out by the computer

Steps:
Define the problem (e.g. input? Representation?)
Find a solution
.....

- Give essential steps on how to solve the problem
- May skip fine details
- Can be presented in pseudocodes (English-like step by step instructions) [Did you learn in programming course?]

* Programming language independent
* Machine independent

Roughly speaking, this is an algorithm

An algorithm is a finite sequence of well-defined instructions, which given an input, can be completed in a finite amount of time with a desired output.

If the lecturer is nice,
    we take the course
else
    we drop the course

not well-defined

has a clear meaning
(roughly speaking, you know how to translate it in program statements)

If the lecturer gives fewer than 3 assignments and no quiz,
    we take the course
else
    we drop the course

Example: Given a number x and an array A of n numbers, check if x is in the list of numbers

```
for i = 1 to n do
        if (A[i] = x) return "yes"
return "no"
```

c.f.
Search the array A (?)
Until you find x or
return "no"

Recall the problem:

Write a program to solve the following problem: Given a set T of numbers and a sum s, find a subset (of size <= x) of T such that the sum of the numbers in this subset equals s.

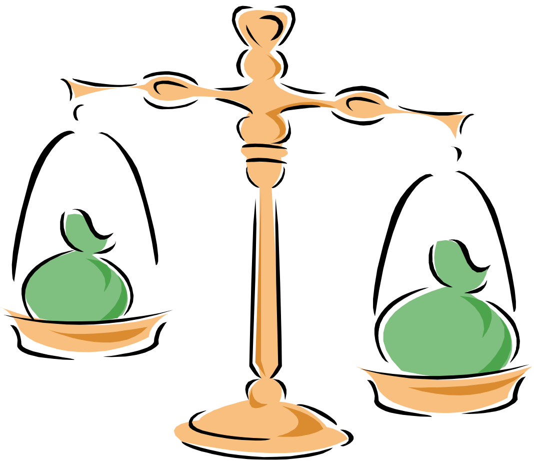
How to form these subsets? We should expand this to make it well defined.

Q: Do you think the following solution (algorithm) is ok?

```
for i = 1 to x {
        form all subsets of T with size  i
        check if the sum of the numbers in each subset = s
        if yes, return that subset and stop
}
return "no such subset"
```

Example

Assuming that we have 9 coins of which one of them is counterfeit and is lighter than others, you are given a balance which can only tell which side of the balance is heavier. How can you identify the counterfeit coin?



One solution (algorithm):

- Pick 4 coins randomly and weight against another 4 coins.
- If they have equal weight, the remaining coin is counterfeit.
- Otherwise, divide the group of lighter weight into 2 groups (each with 2 coins), weight these groups against each other
- For the lighter group, weight the two coins against each other, the lighter one is counterfeit.

At most 3 weightings are needed in the worst case!

Problem → Design an algorithm to solve the problem → Translate algorithm into program

1 Algorithm Design Techniques

Examples:
Greedy approach
Divide-and-conquer
....

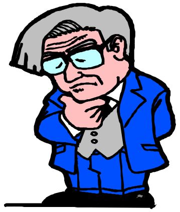
Most of the materials will be taught in anothoer course (COMP3250)

We will learn some of these in this course, e.g. Divide-and-Conquer

1 Algorithm Design Techniques

2 Analysis of algorithms Why?

We may have > 1 algorithms solving the same problem, so have to decide which one is better?



What do you mean by better?

Two aspects

Time: which one will run faster?

Space (memory): which one will use less memory?

Remark: these two issues are important when the volume of data is large.

e.g. If you are going to sort 10 numbers, you can do it by hand or computer; and you can do it instantly no matter which algorithms are used.

How about 1,000,000,000 numbers?

Or you want to locate a keyword from billions of webpages

Well, how to measure which algorithm is better?

Can we do this?
Implement the programs using the algorithms and compare their running time and memory consumption.

May have the following problems:

- programmer skill level
- what programming language is used
- what OS
- what compiler
- what machine (speed of CPU, memory size, memory speed)
- time consuming
- different input

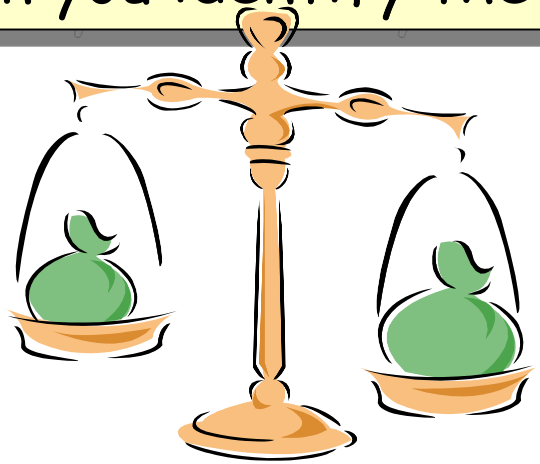
=> We will learn how to measure the complexity of algorithms in a theoretical sense (using mathematics) in this course

1. Algorithm Design Techniques
2. Analysis of algorithms
3. Is our algorithm the best possible?

"Lower Bound" for the problem

Recall this example:

> Assuming that we have 9 coins of which one of them is counterfeit and is lighter than others, you are given a balance which can only tell which side of the balance is heavier. How can you identify the counterfeit coin?



One solution:

- Pick 4 coins randomly and weight against another 4 coins.
- If they have equal weight, the remaining coin is counterfeit.
- Otherwise, divide the group of lighter weight into 2 groups (each with 2 coins), weight these groups against each other
- For the lighter group, weight the two coins against each other, the lighter one is counterfeit.

In the worst case, it will use 3 weightings. Is this the best possible algorithm? Can you do it in 2 weightings?

Better algorithm:

1. Divide the coins into 3 groups (each with 3 coins). Randomly pick two groups and weight against each other.
2. Case a - If they have equal weight, the counterfeit coin is in the 3rd group.
   Case b – Otherwise, the counterfeit coin is in the lighter group.

   In both cases, take the group with the counterfeit coin. Randomly pick two coins from it and weight them against each other.

   Case a – If they have equal weight, the 3rd coin is fake
   Case b – The one with the lighter weight is fake

Is it the best possible solution, can we do it in one weighting?

Take-home exercises:
(No need to hand in)

1)Assuming that we have 3^k ($k \geq 1$) coins of which one of them is counterfeit and is lighter than others, you are given a balance which can only tell which side of the balance is heavier. Describe an algorithm to identify the counterfeit coin using only k weightings?

2) Do you think you can solve the problem with 9 coins using only 1 weighting? If yes, describe the algorithm. If not, argue that it is not possible. [This one is difficult, no worry if you do NOT know how to do it, we will teach you later!]

3)
Another version:

Assume that we have 12 coins of which one of them is counterfeit and is either heavier or lighter than others.

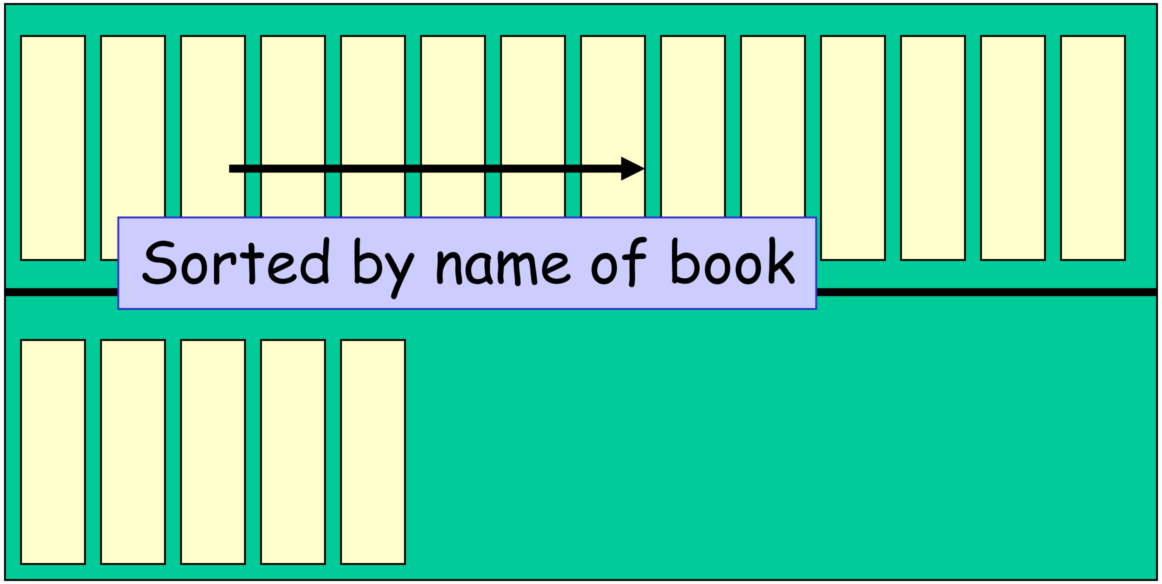
You are given a balance, which can only tell which side of the balance is heavier or indicate that both sides are of equal weight.

Design an algorithm using only 3 weightings to identify the counterfeit coin and tell whether it is heavier or lighter.
[This one is a little bit difficult too, try it if you have time.]

Data Structure: a way to store and organize data in order to facilitate access and modifications depending on the applications

A simple real-life example:

You got a lot of books to be put in the bookshelf, you usually search your books by "name of the book".

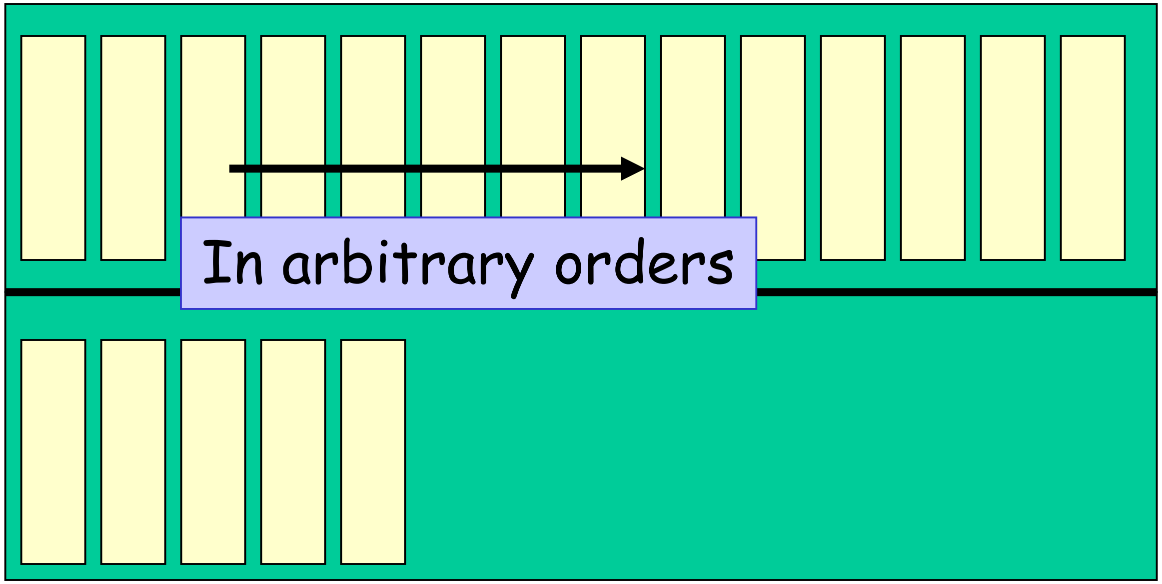


Problem: if you need to add a new book, you probably have to move a lot of books

If you seldom buy books, then it is ok.

If you only have a few books, then it is ok.

But if you have a lot of books and buy books frequently, then....



Problem: Now, adding a new book is easy, but to locate a book is difficult

If you seldom read these books, then it is ok.
If you only have a few books, then it is ok.
But if you have a lot of books and need to read these books frequently, then....

Can you have a better arrangement of books so as to facilitate both adding new books and searching books from the bookshelf?

Another example:

Dr X asked you to write a program to maintain a data set so that the user can either (1) insert a new number to your data set; or (2) give you a number and the program should return whether the number is inside your data set or not.

Solution 1

Store the numbers in a big array, when a new number arrives, append it at the end. If you need to search for a number, check the numbers one by one from the beginning of the array.

Solution 2

Store the numbers in a big array, but keep them sorted. When a new number arrives, insert it in the right position. If you need to search for a number, use “binary search”.

Pros and cons

Insert a number:	Solution 1 is better
Search for a number:	Solution 2 is better

Depends on which operation is needed more frequently & also we should try to find solution 3 (better than both solutions)

When we are talking about data structure, we should know:

1) What are the data we need to organize?
2) What operations (functions) to be supported by data structure?

Then, you should tell people:

1) How you organize the data?
2) How to achieve the operations required using your suggested organization?

related to algorithms

On the other hand, when you design an algorithm, you may need to design a data structure to help handling the data.

Remark:

Program (or a solution to a computational problem) = data structures + algorithms

1 Design a data structure

Learn from existing data structures.

Examples:

Stacks, Queues, Linked Lists, Trees etc.

2 Analysis of how good a data structure is?

1) Whether the corresponding operations are feasible and efficient (time)
2) How much space is required for storing this data structure

Reminder: a good data structure is particularly important if the volume of data is large

Example:
Databases also uses some data structures (B-trees or its variations) to store the records

Topics to be covered (tentative list)

Algorithm Analysis (running time calculation etc.)

Recursion (Divide-and-Conquer)

Hashing

Searching algorithms

Sorting algorithms

Tree and tree searching algorithms

Sorting lower bound

Sorting in linear time

Other topics (if time allows)

Outcome-based learning (OBL)

OBL – a process to help improving teaching/learning with the focus on students (how much you have learned)

* Define a set of outcomes
* Evaluate if students can achieve the outcomes via assignments, tests, and questionnaires

How you help?

- provide comments on the expected outcomes
- feedback on whether we can help you to achieve outcomes

Goal: to provide a better course for you.

Course outcomes: [What we expect you to achieve]

On successful completion of the course, students should be able to:
[O1: Mathematics foundation] understand the concept of time, space complexity and analyze the time and space complexities of an algorithm and a data structure.

[O2: Data structures] understand well-known generic data structures such as stack, queue, tree and related algorithms and apply them to solve problems.

[O3: Problem solving] design new data structures and algorithms to solve problems.

[O4: Implementation] implement selected data structures and algorithms.