Data Structures

Yu-Tai Ching
Department of Computer Sceince
National Chiao Tung University

- Office: EC444, Phone: 31547,
- email: ytc@cs.nctu.edu.tw,
- Office Hour: Tuesday afternoon.
- Prerequisit- know C++ or any programming language, our sample codes will be in C++.
- Had experiences in solving small problems.
- Know C/C++ pointer.
- Book: Fundamentals of Data Structures in C++, 2nd Edition, by Ellis Horowitz, Sartaj Sahni, and Dinesh P. Mehta. Silicon Press.
- 10% quiz, 20% programming assignments,
- 20% for first mid-term, 25% for 2nd mid-term exam., and 25% final exam.

- English,
- Stop me any time if you have question.
- You may ask question in Mandarin.
- Coding is not the most important.
- Learn data structure in two approaches, OO approach and algorithmic approach.
- My approach is much more algorithmic, course covers
  - introduction, what is algorithm, time complexity,
  - simple data structures, array, linked list, stack and queue,
  - Tree, Forest,
  - sort and search,
  - Priority queues, Balance Trees.
  - graph.

Introduction

- Real world objects have to be stored as data objects in a computer so that they can be processed.
- Data Structure talks about the ways to organized the data so that
  - we can avoid troubles while developing large scale software
  - we can design efficient algorithm to solve problem.
- Two issues to be discussed
  - Software Engineering, develop very large scale software system.
  - Algorithms, design algorithms and evaluate performance of algorithms.

System Life Cycle

- A system undergos a development process, called system life cycle,
  - Requirements, specification to define the purposes.
  - Analysis, break the problem down to manageable pieces.
  - Design, design the data type the programs needed and the operations performed on them.
  - Refinement and Coding, choose representations for the data objects and write algorithms for each operation on them.
  - Verification, corectness proof of the program, testing the program with variety of input data, and remove errors.

Software Life Cycle

- Requirement analysis,
- Design,
- Development
- Verification
- Maintenance
- obsolete

The one costs the most is the maintenance. Carefully requirement analysis and design could save a lot of money in development and maintenance. OO design could also reduce the maintenance cost.

Object-Oriented Design

- develop complex software system: using the philosophy of divide and conquer.
- break up a complex software design project into a number of simpler subprojects.
  - Algorithmic Decomposition, structural programming.
  - Object-Oriented decomposition, design a set of objects that interact with each other. Encourage the reuse of software.

Object-Oriented Programming

Definition An *Object* is an entity that performs computations and has a local states. It may therefore be view as a combination of data and procedural elements.

Definition *Object-oriented Programming* is a method of implementation in which

1. Objects are the fundamental building blocks.
2. Each object is an instance of some type (or class).
3. Classes are related to each other by inheritance relationships.

Definition A language is said to be an *object-oriented* language if

1. It supports objects.
2. It requires objects to belong to class.
3. It supports inheritance.

C++ is one of the OO programming language.

Data Abstraction and Encapsulation

A DVD player as an example,

- Interactions with DVD player are through buttons on the control panel, such as PLAY, STOP, or PAUSE. DVD player is packaged so that we cannot directly interact with circuitry inside. So the *internal representation* of the DVD player is *hidden* from the user, - principal of *encapsulation*.
- Instruction manual tells us *what* the DVD player is suppose to do if we press a particular button. It does not tell us *how* this function is implemented. Separate *what* the DVD player does and *how* it does it, -principal of *abstraction*.

Data Abstraction and Encapsulation

- **Definition** *Data Encapsulation* or *Information Hiding* is the consealing of the implementation details of a data object from the outside world.
- **Definition***Data Abstraction* is the separation between the *specification* of a data object and its *implementation*.
- **Definition** A *data type* is a collection of *objects* and a set of *operations* that act on the those objects.
- **Definition** An *abstract data type (ADT)* is a data type that is organized in the way that specification (of objects and operations) is separated from the representation of the objects and implementation of the operations.
- In C++, the header file (*.h) and the definition file (*.cpp). These concepts are important in software development, and result in better quality programs.

Algorithm

- Core of computer science
- An *algorithm* is a finite set of instructions that, if followed, accomplish a particular task. It satisfies the following criteria:
  1. Input, zero or more quantities are externally supplied.
  2. Output, at least one quantity is produced.
  3. Definiteness, each instruction is clear and unambiguous.
  4. Finiteness, it terminates after finite number of steps for all cases.
  5. Effectiveness, instructions are basic enough to be carried out.

Algorithm

- Algorithm and program
- Specification of algorithms
  - Flow chart
  - pseudo code, combination of English and C++

Selection Sort

- Devise a program that sorts a collection of $n \geq 1$ integers.
- *For those integers that are currently unsorted, find the smallest and place it next in the sorted list.*
- The pseudo code.

for(int i=0;i<n;i++){
    examine $a[i]$ to $a[n-1]$ and
        suppose the smallest integer is at $a[j]$;
    interchange $a[i]$ and $a[j]$;
}

```
void SelectionSort (int *a, const int n)
    for (int i=0; i < n; i++)
    {
        int j = i;
        for(int k = i + 1; k < n; k++)
            if (a[k] < a[j]) j = k
        swap (a[i], a[j]);
    }
```

Binary Search, Pseudo Code

int *BinarySearch* (int *a, const int x, const int n)
{
    Initialize $left$ and $right$;
    while (there are more elements)
    {
        Let $middle$ be the middle element;
        if ($x < a[middle]$) set $right$ to $middle - 1$;
        else if ($x > a[middle]$) set $left$ to $middle + 1$;
        else return $middle$;
    }
    Not found;
}

Binary Search, C++ code

```
int BinarySearch (int *a, const int x, const int n)
{
    int left =0, right = n - 1;
    while (left < right)
    {
        int middle = (left + right)/2;
        if (x < a[middle]) right = middle - 1;
        else if ((x > a[middle]) left = middle + 1;
        else return middle;
    }
    return -1;
}
```

Recursion

- Define a function by using the same function having smaller parameters.
  - $f(n) = n \cdot f(n-1)$, $f(1) = 1$,
  - $f(n) = f(n-1) + f(n-2)$, $f(1) = f(2) = 1$.
- Solve a problem by solving the same problem but having smaller problem size.
- Need to define the boundary condition, i.e., when the problem size is a constant.

Binary Search- Recursion

```
int BinarySearch (int *a, const int x, const int left, const int right)
{
    if (left <= right) {
        int middle = (left + right)/2;
        if (x < a[middle])
            return BinarySearch(a, x, left, middle - 1);
        else if (x > a[middle])
            return BinarySearch(a, x, middle + 1, right);
        return middle;
    }
    return -1;
}
```

Performance Analysis

- Space Complexity: Memory Space required when running a program (the program is an implementation of an algorithm). It is reasonable to measure in bytes. (NOTE: we actually don't use "byte" to measure the space requirement.)
- Time Complexity: Computing time required when running a program (the program is an implementation fo an algorithm). Measure the time in "second"? Different machine gives you different measuring. Different input needs different time. Thus using "second" is not good idea for measuring the time complexity.

Performance Analysis

Define the time or space required as a "function of input size". Suppose that input size is n, the space required or time required is $f(n)$. Furthermore, we only look at the order of growth of $f(n)$.

- Given a code which is an implementation of an algorithm.
- $t_p(n) = C_a ADD(n) + C_s SUB(n) + C_m MUL(n) + C_d DIV(n) + \cdots$.
- C_a, C_s, C_m, and C_d: time required for ADD, SUB, MUL, and DIV.

- C_a, C_s, C_m, and C_d can be measure in “second”, but that depends on machine (and usually with a constant factor).
- Or they can be measure by “clock cycle”, for different machine, the number of clock cycles needed is fixed for an operations.
- C_m and C_d are generally larger than C_a and C_s (within a constant factor).
- $t_p(n) \leq$
  $c_m \cdot (ADD(n) + SUB(n) + MUL(n) + DIV(n) + \cdots)$
  $\leq c \cdot f(n)$ where $f(n)$ is the total number of operations.
- The constant c depends on the machines, thus $f(n)$ determines the performance of algorithm.

- $t_p(n) = f(n)$, the time is a function of the input size.
- Given an algorithm, we count the number of steps.
- If a “macro” consists of some basic enough operators, but can be bounded by a constant, we say that is a step. $swap(a, b)$ in selection sort is an example.
- Furthermore we look at the order of growth.

Asymptotic Notation, Big O, O

- $f(n) = O(g(n))$
- $O(g(n))$ is a set of functions that $\exists c$ and n_0, s.t., every function $f(n)$ in the set, $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$.
- $O(g(n)) = \{f(n) | \exists$ positive constant c, n_0, s.t. $f(n) \leq c \cdot g(n)\}$.
- $g(n)$ is the upper bound to the set of functions $f(n)$.
- Examples of binary search and insertion sort.

Asymptotic Notation, Omega, Ω

- $f(n) = \Omega(g(n))$
- $\Omega((n))$ is a set of functions that $\exists c$ and n_0, s.t. for every function $f(n)$ in the set, $f(n) \geq c \cdot g(n)$, $\forall n \geq n_0$.
- $\Omega(g(n)) = \{f(n) | \exists$ positive constant c, n_0, s.t. $f(n) \geq c \cdot g(n)\}$.
- $g(n)$ is the lower bound to the set of functions $f(n)$.

Asymptotic Notation, Theta, Θ

- $f(n) = \Theta(g(n))$.
- $\Theta(g(n))$ is a set of functions that $\exists c_1$, c_2, and n_0 s.t. for every function $f(n)$ in the set, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.
- $\{f(n) | \exists$ positive constant c_1, c_2 and n_0 s.t., $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$.
- $g(n)$ is the exact bound to the set of functions $f(n)$,
- An example of selection sort.

- Selection Sort, $O(n^2)$ or $\Theta(n^2)$?
- Binary Search $O(\log n)$ or $\Theta(\log n)$?
- A view from recursive binary search, $T(n) = 1 + T(n/2)$.

Practical Complexity

n	$\log n$	$n \log n$	n^2	n^3	2^n
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4284963286
1024	10	10240	1000000	1000000000	$*$

n	$\Theta(n)$	$\Theta(n \log n)$	n^2	n^4	n^{10}	2^n
10	$.01\mu s$	$0.03\mu s$	$.1\mu s$	$10\mu s$	$10s$	$1\mu s$
20	$.02\mu s$	$.09\mu s$	$.4\mu s$	$160\mu s$	$2.82h$	$1ms$
30	$.03\mu s$	$.15\mu s$	$.9\mu s$	$810\mu s$	$6.83d$	$1s$
40	$.04\mu s$	$.21\mu s$	$1.6\mu s$	$2.56ms$	$121d$	$18m$
50	$.05\mu s$	$.28\mu s$	$2.5\mu s$	$6.25ms$	$3.1y$	$13d$
100	$.1\mu s$	$.66\mu s$	$10\mu s$	$100ms$	$3171y$	$4 \cdot 10^{13}y$
10^3	$1\mu s$	$9.96\mu s$	$1ms$	$16.67m$	$3.17 \cdot 10^{13}y$	$32 \cdot 10^{283}y$