Introduction and Course Overview
CSE 373

Yang Li
University of Washington
Autumn 2007

September 26, 2007
Staff

• Instructor
  › Yang Li
    (yangli@cs.washington.edu)

• TAs
  › Cam Thach Nguyen
    (ncthach@cs.washington.edu)
  › Sierra Michels-Slettvet
    (sierra@cs.washington.edu)
Yang Li

• Currently a Research Associate at **UW CSE**
  › Ubiquitous Computing & Pen-based Computing
  › http://www.cs.washington.edu/homes/yangli

• Previously a Postdoc at **UC Berkeley EECS**
  › Ubiquitous Computing & Pen-based Computing

• Acquired a PhD from **Chinese Academy of Sciences**
  › Computer Science
  › Pen-based Computing
Data Structures: Why?

We need clever ways to organize information in order to enable efficient computation.
Using Appropriate Abstractions is the Key!
Course Website

- http://www.cs.washington.edu/373

- All the information for the course
  - Contact information & announcements
  - Assignments
  - Schedules & lectures
  - Links to discussion boards and mailing lists
  - Handouts
  - Links to computing resources
Office Hours

- Yang Li – CSE212 (Allen Center)
  › Monday & Wednesday, 2:00-3:00
  › Or by appointment

- Cam Thach Nguyen – CSE218
  › Tuesday & Thursday, 9:30 to 10:20

- Sierra Michels-Slettvet – TBA
  › Thursday, 3:30
CSE 373 E-mail List

- Automatically subscribed if you are registered for the course
  - Otherwise, subscribe via the class web page

- Use
  - Posting announcements by instructor & TAs
CSE 373 Discussion Board

• Subscribe through the course website

• Use
  › General discussion of class contents
  › Hints and ideas about assignments
    • but not detailed code or solutions
  › Other topics related to the course
Computer Lab

• College of Arts & Sciences Instructional Computing Lab
  › http://depts.washington.edu/aslab/

• Programming language: Java 5 or 6
Programming Tools

• **Eclipse**
  › The best IDE I’ve ever used!
  › Or whatever editor that allow you to type in code!

• Stay away from code-generating “wizards”

• Most tools are freely available on the web
  › Easy to set up at home
Textbook

Grading & Estimated Breakdown

- Two Midterms 30% (15% each)

- Final 20%
  - 10:30-12:20 pm, Wednesday, Dec 12

- Assignments 50%
  - Weights differ to account for difficulty of assignments
  - A mix of written exercises and programming projects
Deadlines & Late Policy

- Assignments generally due Thursday evenings
  › Turnin via the web
  › Exact times/dates will be given for each assignment

- Late policy: NONE
  › As in, no late assignments accepted
  › Talk to the instructor if something truly outside your control causes problems here
Academic (Mis-) Conduct

• You are expected to do your own work
  › Exceptions will be clearly announced

• Misconducts will be penalized
  › Sharing solutions
  › Doing work for or accepting work from others

*Integrity is a fundamental principle in the academic world (and elsewhere) – we and your classmates trust you; don’t abuse that trust*
Policy on Collaboration

“Gilligan’s Island” rule

› You may discuss problems with your classmates to your heart's content.

› After you have solved a problem, discard all written notes about the solution.

› Go watch TV for a ½ hour (or more). Preferably Gilligan's Island.

› Then write your solution.
Homework for Today

- Assignment #1
  - Posted in the next day or so

- Reading in Weiss
  - Chapter 1 – Mathematics and Java
  - Chapter 2 – Algorithm Analysis
  - Chapter 3 – Lists, Stacks, & Queues
Class Overview

• Be exposed to a variety of data structures
• Know when to use them
• Apply mathematical techniques for analysis
• Practice implementing them by writing programs

Goal:
Be able to make good design choices as a developer, project manager, or system customer
Good Designs

Program design depends crucially on how data is structured for use by the program

› Speed of program may dramatically decrease or increase
› Memory used may increase or decrease
› Implementation of some operations may become easier or harder
› Debugging may be become easier or harder
Course Topics

• Introduction to Algorithm Analysis
• Lists, Stacks, Queues (mostly review)
• Search Algorithms & Trees
• Hashing & Heaps
• Sorting
• Disjoint Sets
• Graph Algorithms
Picking the best Data Structure for the job

• The data structure you pick needs to support the operations you need

• Ideally it supports the operations you will use most often in an efficient manner

• Examples of operations
  › List with operations insert and delete
  › Stack with operations push and pop
Background

• Prerequisite is CSE 143
• Topics you should have a basic understanding of
  › Variables, conditionals, loops, methods (functions),
    fundamentals of defining classes and inheritance,
    arrays, single linked lists, simple binary trees,
    recursion, some sorting and searching algorithms,
    basic algorithm analysis, e.g., $O(n)$ vs. $O(n^2)$ and
    similar things.
• We can fill in gaps as needed, but if any topics
  are new, plan on some extra studying
Terminology

- **Abstract Data Type (ADT)**
  - Mathematical description of a computational object
  - Useful building block

- **Algorithm**
  - A high level, language independent, description of a step-by-step process

- **Data structure**
  - A specific family of algorithms for implementing an abstract data type

- **Implementation of data structure**
  - A specific implementation in a specific language
A Terminology Example

• A stack is an *abstract data type (ADT)*
  › Supporting push, pop and isEmpty operations

• A stack *data structure*
  › Use an array or a linked list
  › Or anything that can hold data

• One stack *implementation*
  › See java.util.Stack
Why Algorithm Analysis

• Correctness
  › Does the algorithm do what is intended

• Performance
  › What is the running time of the algorithm
  › How much storage does it consume

• Choose among different data structures
  › All correctly solves a given task
  › Which should we use? When?
Iterative Algorithm for Sum

Problem: Find the sum of the first \textit{num} integers stored in an array \textit{v}.

\begin{verbatim}
sum(v[ ] : integer array, num: integer): integer 
{
    temp_sum: integer ;
    temp_sum := 0;
    for i = 0 to num - 1 do
        temp_sum := v[i] + temp_sum;
    return temp_sum;
}
\end{verbatim}
Programming via Recursion

Problem: Write a recursive function to find the sum of the first num integers stored in array v.

\[
\text{sum (v[ ]: integer array, num: integer): integer}
\{
    \text{if num = 0 then}
    \text{    return 0}
    \}
\text{base case}
\text{else}
\text{    return v[num-1] + sum(v,num-1);}
\}
\text{recursive case}
\]
Pseudocode

Algorithms will (often) be presented in “pseudocode”

› Common in the computer science literature
› Easy to translate to real code
› Independent of particular programming language
› Informal but precise: there is no “official” language definition for pseudocode
Proof by Induction

- **Basis Step**
  - The algorithm is correct for a base case or two by inspection

- **Inductive Hypothesis (n=k)**
  - Assume that the algorithm works correctly for the first k cases, for any k

- **Inductive Step (n=k+1)**
  - Given the hypothesis above, show that the k+1 case will be calculated correctly
Program Correctness by Induction

• Basis Step
  › \( \text{sum}(v,0) = 0 \) ✓

• Inductive Hypothesis (n=k)
  › Assume \( \text{sum}(v,k) \) correctly returns sum of first \( k \) elements of \( v \), i.e. \( v[0]+v[1]+\ldots+v[k-1] \)

• Inductive Step (n=k+1)
  › \( \text{sum}(v,n) \) returns \( v[k]+\text{sum}(v,k) \) which is the sum of first \( k+1 \) elements of \( v \). ✓
Algorithms vs. Programs

• Proving correctness of an algorithm is very important
  › a well designed algorithm is guaranteed to work correctly and its performance can be estimated

• Proving correctness of a program (an implementation) is fraught with weird bugs
  › Abstract Data Types are a way to bridge the gap between mathematical algorithms and programs