Introduction

CSE 373
Data Structures & Algorithms
Ruth Anderson
Spring 2008

Staff

- Instructor
  - Ruth Anderson (rea@cs)
- TAs
  - Tian Sang (sang@cs)
  - Eric McCambridge (ericm6@cs)
  - Devy Pranowo (devynp@cs)

UNIVERSITY of VIRGINIA

Me (Ruth Anderson)

- Grad Student at UW (Programming Languages, Compilers, Parallel Computing)
- Taught Computer Science at the University of Virginia for 5 years
- Grad Student at UW (Educational Technology, Pen Computing)
- Just defended my PhD last fall!!

Web Page

- All info is on the web page for CSE 373
  (or at least will be once things are a bit further along…)
  - http://www.cs.washington.edu/373
  - also known as
    http://www.cs.washington.edu/education/courses/373/08sp
- Look there for schedules, contact information, assignments, links to discussion boards and mailing lists, etc.

Office Hours

- Ruth Anderson– 360 CSE (Allen Center)
  - M 12:30-1:30pm, T 1:30-2:30pm, or by appointment
- Tian Sang - TBA
- Eric McCambridge – TBA
- Devy Pranowo - TBA

CSE 373 E-mail List

- If you are registered for the course you will be automatically registered. Otherwise, subscribe by going to the class web page
- E-mail list is used for posting important announcements by instructor and TAs
- You are responsible for anything sent here

CSE 373 Discussion Board

- The course will have a Catalyst Go-Post message board
- 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

- Math Sciences Computer Center
  - http://depts.washington.edu/aslab/
- Programming language: Java 5
  - Java 6 is also fine
  - Java 1.4 is ok for some things, but we will use generics which were introduced in Java 5.0

Programming Tools

- Eclipse, DrJava, Textpad, whatever…
  - Also may need JavaDoc, JUnit, which are easy to access from most tools
- We're not religious about this as long as your code is standard Java
  - But stay away from code-generating "wizards"
- Sun Java for Windows/Linux, Java 5 for OS X, and most tools are freely available on the web – easy to set up at home

Textbook

- *Data Structures and Algorithm Analysis in Java,* Mark Weiss, 2nd edition, Addison-Wesley, 2007.

Grading

Estimated Breakdown:

- Midterms 30% (15% each)
- Final 20%
  - 2:30-4:20pm Wednesday, Jun. 11, 2008
- Assignments 50%
  - Weights may differ to account for relative difficulty of assignments
  - Assignments will be a mix of shorter written exercises and longer programming projects

Deadlines & Late Policy

- Assignments generally due Thursday evenings via the web
  - Exact times and dates will be given for each assignment
- Late policy:
  - 25% off per 24hrs late
  - Note: ALL parts of the assignment must be received by that time (may require you to make an electronic version of written assignments).
    - (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 (group work), if any, will be clearly announced
- Sharing solutions, doing work for or accepting work from others will be penalized
- Referring to solutions from this or other courses from previous quarters is cheating.
- 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!!

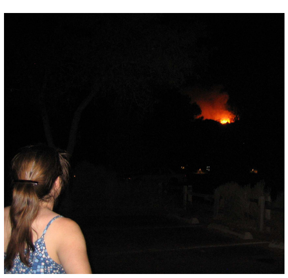
1) **Assignment #1:** (posted in the next day or so)
2) **Preliminary Survey**: fill out by evening of Tuesday April 1st
3) **Information Sheet**: bring to lecture on Wednesday April 2nd
4) **Reading** in Weiss (see next slide)

Reading

- Reading in *Data Structures and Algorithm Analysis in Java*, by Weiss
- For this week:
  - › Chapter 1 – (review) Mathematics and Java
  - › Chapter 3 – (Assign #1) Lists, Stacks, & Queues
  - › Chapter 2 – (Topic for Friday) Algorithm Analysis

Bring to Class on Wednesday:

- Name
- Email address
- Year (1,2,3,4)
- Major
- Hometown
- Interesting Fact or what I did over summer/break.



Class Overview

Introduction to many of the basic data structures used in computer software

- › Be exposed to a variety of data structures
- › Know when to use them
- › Practice mathematical techniques for analyzing the algorithms that use them
- › Practice implementing and using them by writing programs

Goal:

be able to make good design choices as a developer, project manager, or system customer

Data Structures

"Clever" ways to organize information in order to enable efficient computation.

Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues (mostly review)
- Search Algorithms and Trees – particularly balanced trees
- Hashing and Heaps, Dictionaries
- Sorting
- Disjoint Sets
- Graph Algorithms

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

Data Structures: What?

- Need to organize program data according to problem being solved
- Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  - › List ADT with operations **`insert`** and **`delete`**
  - › Stack ADT with operations **`push`** and **`pop`**
- Note similarity to Java classes
  - › private data structure and public methods

Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - › Implementation of some operations may become easier or harder
  - › Speed of program may dramatically decrease or increase
  - › Memory used may increase or decrease
  - › Debugging may be become easier or harder

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 ADT with operations **`insert`** and **`delete`**
  - › Stack ADT with operations **`push`** and **`pop`**

Terminology

- Abstract Data Type (ADT)
  - › Mathematical description of an object with set of operations on the 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

Terminology examples

- A stack is an *abstract data type* supporting push, pop and isEmpty operations
- A stack *data structure* could use an array, a linked list, or anything that can hold data
- One stack *implementation* is found in java.util.Stack

Algorithm Analysis: Why?

- Correctness:
  - › Does the algorithm do what is intended.
- Performance:
  - › What is the running time of the algorithm.
    - (In terms of what?)
  - › How much storage does it consume.
- Different algorithms may correctly solve a given task
  - › Which should we use? When?

Iterative Algorithm for Sum

- Find the sum of the first **num** integers stored in an array **v**.

```
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;
}
```

Note the use of pseudocode

Programming via Recursion

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

```
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
```


Pseudocode

- In the lectures algorithms will (often) be presented in “pseudocode”.
  - › Common in the computer science literature
  - › Pseudocode is usually easily translated 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:** sum(v,0) = 0. ✓
- **Inductive Hypothesis (n=k):** Assume sum(v,k) correctly returns sum of first k elements of v, i.e. `v[0]+v[1]+…+v[k-1]`
- **Inductive Step (n=k+1):** sum(v,n) returns `v[k]+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