Administrivia- Introduction

CSE 373

Data Structures

Staff

- Instructor
  - Jean-Loup Baer, baer@cs
- TA's
  - Gary Yngve, gyngve@cs
  - Jean Wu, jeaneis@cs

Office Hours

- Jean-Loup Baer – 474 Allen Center
  - M 1:30 – 2:30, Th 11:00 – 12:00 or by appointment
- Gary Yngve – TBD
- Jean Wu – TBD

Computer Lab

- Math Sciences Computer Center
  - http://www.ms.washington.edu/
- Of course you can use your own computer
- Project must be done in Java
  - Use Java 5 (aka Java 1.5)
  - See Website for how to download the software

Textbook

- *Data Structures & Algorithms in Java* (4th Ed)

  by Michael Goodrich and Roberto Tamassia
- The book is accompanied by an "extensive website"

  http://java.datastructures.net
  - Java source code
  - Hints to exercises
  - Slides, Java applets etc…

Grading

- Assignments and programming projects 50%
- Midterm 1 15%
  - Probably April 21 in class
- Midterm 2 15%
  - Probably May 17 in class
- Final 20%
  - 2:30-4:20 p.m. Wednesday, June 7, 2006

E-mail

- If you are registered you are already on the e-mail list
- Otherwise subscribe by going to the class web page
- Used for important messages and announcements by course staff
- You are responsible for anything sent there

Discussion Board

- There is a Catalyst e-post message board
- Used for
  - › general discussion about the class
  - › Hints and ideas about assignments (but see policy on collaboration)
  - › Topics related to CSE 373

Assignments

- Electronic turnin due Wednesdays at 11:00 am
- Paper assignments due Wednesdays at beginning of class
- No late assignment accepted unless you talked to the instructor first (with an excellent excuse)

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.

Class Overview

- Introduction to many of the basic data structures used in computer software
  - Understand the data structures
  - Analyze the algorithms that use them
  - Know when to apply them
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.
- Data structures are the plumbing and wiring of programs.

Goal

- You will understand
  - what the tools are for storing and processing common data types
  - which tools are appropriate for which need
- So that you will be able to
  - make good design choices as a developer, project manager, or system customer

Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues (very fast!)
- Trees
- Heaps and Priority Queues
- Hashing
- Balanced search trees
- Sorting
- Disjoint Sets
- Graph Algorithms

Reading

- Chapter 3 Sections 1 and 2
  - This is basic review of CSE 142 and CSE 143 material
- Chapter 4
  - Mathematical tools we will use
  - The Big-Oh notation (super important)

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`** **`(among others)`**
  - 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

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

Algorithm Analysis: Why?

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

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);
}
```

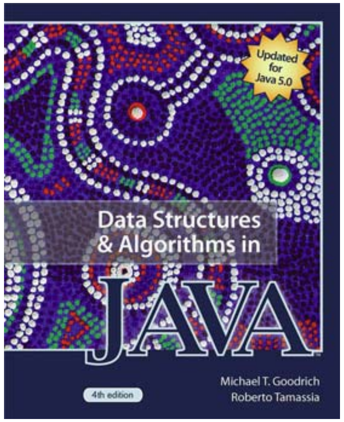
Pseudocode

- In the lectures algorithms will be presented in pseudocode.
  - This is very common in the computer science literature
  - Pseudocode is usually easily translated to real code.
  - This is programming language independent
- Pseudocode should also be used for “paper” homework

Teaching Philosophy

- Old style Don Knuth

- New Style Java-base



- This course: pseudo-code

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