Introduction

CSE 326
Data Structures
Lecture 1

Administrative

- Instructor
  - Richard Ladner
  - ladner@cs.washington.edu
- TA
  - Gyanit Singh
- Class info is on the web site
- Message Board
  - [https://catalysttools.washington.edu/gopost/board/ladner/3909/](https://catalysttools.washington.edu/gopost/board/ladner/3909/)
  - For discussion and announcements

Office Hours

- Richard Ladner – CSE 632
  - Tuesdays 12-1, Thursday 11-12 or by appointment
- TA Office Hours
  - TBA

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 (1)

- 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

Goal (2)

- Be able to:
  - Reason formally about algorithms
  - Communicate ideas about programs clearly and precisely
- Homeworks are mostly written
  - Need more than “correct” answer—need to effectively communicate the ideas
Weekly Assignments

• Weekly homeworks
  › Involve algorithms design and analysis
  › No coding
  › Pseudocoding is preferred
  › Due on Wednesdays

Pseudocode

• The algorithms you design in homework will be read by a person, not a computer
• The No Code Rule:
  › Do not turn in Java or C code when asked for pseudocode
  › Explain algorithm precisely, but without all the details needed for computer code

Pseudocode example (good)
• reversePrint( string s )
  Create an empty stack A
  For each character c in s
    Push c onto A
  While A is not empty
    Pop c from A
    Print c

Pseudocode example (bad)
• void reversePrint( String s ) {
  Stack A = new Stack();
  for (int i = 0; i < s.length(); i++) {
    A.push( s.get(i) );
  }
  While ( ! A.isEmpty() ) {
    Print( A.pop() );
  }

Projects

• First project will be done in C in Linux
  › TA will provide preparation in section
• Other projects are in Java.
• Projects involve
  › Writing code
  › Experimenting
  › Writing

Assignments, Projects, Exams

• Assignments 25%
  › Due on Wednesdays, no late assignments
• Projects 25%
  › 3 programming projects
• Midterm 20%
  › Friday, Wednesday 13, 2008
• Final 30%
  › 8:30 – 10:20 Thursday, March 20, 2008
Course Topics

- Introduction to Algorithm Analysis
- Sorting
- Memory Hierarchy
- Search Algorithms and Trees
- Priority Queues
- Hashing
- Disjoint Sets
- Graph Algorithms
- Computational Geometry

Reading

- Reading in *Data Structures and Algorithm Analysis in Java*, by Weiss
  - Chapter 1 – Mathematical preliminaries
  - Chapter 2 – Algorithm Analysis
  - Chapter 7 - Sorting
    - Insertion Sort
    - Quicksort
    - Mergesort

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 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.
  - How well does the algorithm complete its goal
- 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 $n$ integers stored in an array $v$.

$\text{sum}(\text{integer array } v, \text{ integer } n) \text{ returns integer}
\begin{align*}
\text{let } \text{sum} &= 0 \\
\text{for } i &= 1 \ldots n \\
\text{sum} &= \text{sum} + v[i] \\
\text{return sum}
\end{align*}$

Programming via Recursion

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

$\text{sum}(\text{integer array } v, \text{ integer } n) \text{ returns integer}
\begin{align*}
\text{if } n &= 0 \\
\text{sum} &= 0 \\
\text{else} \\
\text{sum} &= v[n] + \text{sum}(v,n-1) \\
\end{align*}$

Pseudocode

- In the lectures I will be presenting algorithms in pseudocode.
  - This is very common in the computer science literature.
  - Pseudocode is usually easily translated to real code.
  - This is what I’m used to.
- Pseudocode should also be used for homework.

Proof by Induction

- **Basis Step**: The algorithm is correct for a base case or two by inspection.
- **Inductive Hypothesis**: Assume that the algorithm works correctly for the first $n-1$ cases.
- **Inductive Step**: Given the hypothesis above, show that the $n$-th case will be calculated correctly.

Program Correctness by Induction

- **Basis Step**: $\text{sum}(v,0) = 0$.
- **Inductive Hypothesis**:
  - Assume $\text{sum}(v,n-1)$ correctly returns sum of first $n-1$ elements of $v$, i.e.
- **Inductive Step**:
  - $\text{sum}(v,n) = v[n]+\text{sum}(v,n-1)$ (by program)
  - $v[n]+(v[1]+\ldots+v[n-1])$ (by inductive hyp.)
  - $v[1]+\ldots+v[n-1]+v[n]$ (by algebra)

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
Defining Efficiency

- Asymptotic Complexity - how running time scales as function of size of input
  - Order of magnitude notation
  - \( O(n^2) \) is better than \( O(n^3) \) in the long run
- Why is this a reasonable definition?
  - Definition is independent of any possible advances in computer technology

The Apocalyptic Laptop

Speed \( \propto \) Energy Consumption
\[ E = mc^2 \]
25 million megawatt-hours
Quantum mechanics:
Switching speed = \( \pi \frac{h}{2 \times \text{Energy}} \)
\( h \) is Planck’s constant
5.4 \( \times 10^{30} \) operations per second
Seth Lloyd, SCIENCE, 31 Aug 2000

Asymptotic Scaling

- Apocalyptic laptop, 1 year
- Apocalyptic laptop, 1 second
- PC, since Big Bang
- PC, 1 day