Introduction

CSE 326
Data Structures
Lecture 1

Administrative

- Instructor: Ethan Phelps-Goodman (ethanpg@cs)
- TA: Ian Simon (iansimon@cs)
- See http://www.cs.washington.edu/326 for:
  - Lecture, reading, and homeworks
  - Exams
  - Office hours
  - Grading criteria
- Mailing list: cse326@cs
  - Important course announcements—please read!
  - Feel free to use for class discussion, questions, thoughts, etc.

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

Course Topics

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

Reading

- Reading in *Data Structures and Algorithm Analysis in Java*, by Weiss
- For this week:
  - Chapter 1 – (review) Mathematical preliminaries
    - See especially page 10 for a good inductive proof.
  - Chapter 2 – Algorithm Analysis

Projects and Homeworks

- Weekly homeworks
  - Involve algorithms design and analysis
  - No coding
- 3 projects
  - Writing code
  - Validating and Experimenting
  - Writing

Psuedocode

- 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 psuedocode
  - Explain algorithm precisely, but without all the details needed for computer code

Psuedocode 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
```


Psuedocode 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() );
  }
```


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

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.
  - 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**.

```
sum(integer array v, integer n) returns integer
  let sum = 0
  for i = 1...n
    sum = sum + ith number
  return sum
```

Note the use of pseudocode

Programming via Recursion

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

```
sum(integer array v, integer n) returns integer
 if n = 0 then
   sum = 0
 else
   sum = nth number + sum of first n-1 numbers
 return sum
```

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.
- **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)= (by inductive hyp.)`
  `v[k]+(v[0]+v[1]+…+v[k-1])=`
  `v[0]+v[1]+…+v[k-1]+v[k]` ✓

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

$E = m c^2$

25 million megawatt-hours

Quantum mechanics:
Switching speed = $\pi h / (2 * \text{Energy})$

h is Planck's constant

5.4×10^{50} operations per second

Seth Lloyd, SCIENCE, 31 Aug 2000

Asymptotic Scaling