Introduction

- Me: Software engineer, taught at University of Michigan
- Office hours: MQH 215 Tuesdays 8-10am
Today's Outline

• Introductions
• **What is this course about?**
• Administrative info
• Review: Queues and stacks
• Asymptotic Analysis
Class Overview

• Introduction to many of the basic data structures used in computer science
  – Understand the data structures
  – Analyze the algorithms that use them
  – Know when to apply them
Class Overview

- Practice design and analysis of data structures
- Practice using these data structures by writing programs
- Make the transformation from programmer to computer scientist
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 can
  – Make good design choices as a developer, project manager, or system customer
Goal

- You will be able to
  - Justify your design decisions via formal reasoning
  - Communicate ideas about programs clearly and precisely
Roll Call
Today's Outline

- Introduction
- What is this course about?
- Administrative Info
- Review: Queues and stacks
Course Information

- **Greensheet**
- **Text:** Data Structures & Algorithm Analysis in Java, (Mark Allen Weiss)
- **Web page:**
Course Mechanics

• Written homeworks
  – Due at the start of class on due date

• Programming homeworks
  – In Java
  – Turned in electronically

• Work in teams only on explicit team projects
  – Appropriate discussions encouraged
  – Anytime you use someone else's work, it's cheating
Project and homework guidelines

- On the website – note especially,
  - Homeworks – psuedocode is ok, a human being is reading your homeworks
  - See website for pseudocode examples
  - Projects: execution is only a portion of your grade!
  - Spend time commenting your code as you write – it will help you be a better programmer
Homework for Today!!

• Project #1: Implement Stacks and Queues.
  – Due in one week.

• Reading in Weiss
  – Chapter 1 (review): Mathematics and Java
  – Chapter 3 (Project #1): Stacks and Queues
  – Chapter 2 (Homework #1): Algorithm Analysis

• 4) Homework #1 is based off of reading and will be released next class.
Project 1

• Soundblaster! Reverse a song
  – Implement a stack and a queue to make the “Reverse” program work

• Read the website
  – Detailed description of assignment
  – Detailed description of how programming projects are graded
Data Structures

- “Clever” ways to organize information in order to enable efficient computation
  - What do we mean by clever?
  - What do we mean by efficient?
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.
- 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.
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 ADT

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

- A stack is an ADT 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
Concepts vs Mechanisms

- Abstract or Concrete?
  - Pseudocode vs Specific programming language
  - Algorithm vs Program
  - ADT vs Data structure
Why so many data structures?

• Ideal data structure:
  – “fast”, “elegant”, memory efficient

• Generates tensions
  – Time vs space
  – Performance vs elegance
  – Generality vs simplicity
  – One operation's performance vs another's
Today's Outline

- Introduction
- What is this course about?
- Administrative Info
- Review: Queues and stacks
- Asymptotic Analysis
First Example: Queue ADT

- FIFO: First In, First Out
- Queue operations
  - create
  - destroy
  - enqueue
  - Dequeue
  - is_empty
Circular Array Queue Data Structure

enqueue(Object x) {
    Q[back] = x ;
    back = (back + 1) % size
}

dequeue() {
    x = Q[front] ;
    front = (front + 1) % size;
    return x ;
}

How test for empty list?

How to find K-th element in the queue?

What is complexity of these operations?

Limitations of this structure?
void enqueue(Object x) {
    if (is_empty())
        front = back = new Node(x)
    else
        back->next = new Node(x)
        back = back->next
}

bool is_empty() {
    return front == null
}
Circular Array vs. Linked List

- **Circular Array**
  - Too much space
  - Kth element accessed in O(1)
  - Not as complex
  - Could make array more robust

- **Linked List**
  - Can keep growing
  - No going back around to front
  - More complex code
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
Iterative Algorithm for Sum

- Find the sum of the first n integers stored in an array v

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

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

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

```java
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: $\text{sum}(v, 0) = 0$
- 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 $v[k] + \text{sum}(v, k) = (\text{by inductive hypothesis})$
  $v[k] + (v[0] + v[1] + \ldots + v[k-1]) = v[0] + v[1] + \ldots + v[k-1] + v[k]$
Today's Outline

• Introduction
• What is this course about?
• Administrative Info
• Review: Queues and stacks
• Asymptotic Analysis
What we want

- Rough Estimate
- Ignores Details
  - Or really: independent of details
- What are some details we should ignore?
Big-O Analysis

• What are some details we should ignore?
  – Speed of machine
  – Programming language used
  – Amount of memory
  – Order of input
  – Size of input
  – Compiler
Analysis of Algorithms

• Efficiency measure
  – How long the program runs (time complexity)
  – How much memory it uses (space complexity)

• Why analyze at all?
  – Decide which one to implement before going to the trouble
  – Given code, idea of where bottlenecks will be – without running and timing
Asymptotic Analysis

- One “detail” we won't ignore – problem size, # elements
- Complexity as a function of input size $n$
  - $T(n) = 4n+5$
  - $T(n) = 0.5n \log n – 2n + 7$
  - $T(n) = 2^n + n^3 + 3n$
- What happens as $n$ grows?
Why Asymptotic Analysis?

- Most algorithms are fast for small $n$
  - Time difference too small to be noticeable
  - External things dominate (OS, disk I/O, ...)
- BUT $n$ is often large in practice
  - Databases, internet, graphics, ...
- Time difference really shows up as $n$ grows!
Big-O Common Names

- constant: $O(1)$
- logarithmic: $O(\log n)$
- linear: $O(n)$
- quadratic: $O(n^2)$
- cubic: $O(n^3)$
- polynomial: $O(n^k)$ ($k$ is a constant)
- exponential: $O(c^n)$ ($c$ is a constant $> 1$)
Exercise

bool ArrayFind (int array[], int n, int key) {

}
Analyzing Code

- **Basic Java operations** - Constant Time
- **Consecutive statements** – Sum of times
- **Conditionals** – Larger branch plus test
- **Loops** – Sum of iterations
- **Function calls** – Cost of function body
- **Recursive functions** – Solve recurrence relation, Number of calls * work for each call
bool LinearArrayFind(int array[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (array[i] == key) {
            // Found it!
            return true;
        }
    }
    return false;
}