CSE 326: Data Structures
Lecture #1
Introduction

Henry Kautz
Winter Quarter 2002

Today's Outline

- Administrative Stuff
- Overview of 326
- Introduction to Abstract Data Types: Lists
- Introduction to Complexity

Course Information

- Instructor: Henry Kautz <kautz@cs.washington.edu>
  Office hours: Monday 2:30-3:30 and by appointment
  417 Sieg Hall
- TA's: Nick Diebel <jdeibel@cs.washington.edu>
  Hannah Tang <htang@cs.washington.edu>
  Office hours *TBA*
  Meet in Sieg 226B
- Text: *Data Structures & Algorithm Analysis in C++*, 2nd edition, by Mark Allen Weiss
- Final: Monday, March 18th; Midterm date TBA

Course Policies

- Weekly assignments – mix of programming and mathematical analysis
  - 10% a day penalty for late assignments
  - *Learning from each other and discussing the assignments is great, but plagiarism is not. When in doubt just check with me or the TA's.*
- Grading
  - Homework: 65%
  - Exams: 35%
  - Class participation: 5%

  105%

Course Mechanics

- http://www/education/courses/326/02wi
- Mailing list: cse326@cs.washington.edu
  You should get a test mailing by next class; if not, send email to Nick!
- Course labs are 232 and 329 Sieg Hall
  - lab has NT machines w/X servers to access UNIX
- All programming projects graded on UNIX using g++ compiler

What is this Course About?

Clever ways to organize information in order to enable efficient computation

- What do we mean by clever?
- What do we mean by efficient?

Clever? Efficient?

Data Structures	Algorithms
Lists, Stacks, Queues	Sparse Matrix Multiply
Heaps	Merge
BST/AVL/Splay Trees	Binary Search
Hash Tables	Double Hashing
Graphs	A* Search
Up Trees	Union/Find
Quad Trees	Nearest Neighbor

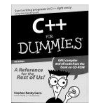
Data Structures *Algorithms*

Theory Graphics AI

Systems Applications

Used Everywhere!

Mastery of this material separates you from:



Anecdote #1

- N^2 "pretty print" routine nearly dooms knowledge-based product configuration system at AT&T
  - Written in Franz Lisp
  - 10 MB data = 10 days (100 MIPS) to save to disk!
  - Whoever wrote the compiler must have skipped this course…

Guaranteed Non-Obsolescence

- Much is passing…
  - B, DOS, .com's…
- Won't our notions of "efficiency" change?
  Moore's Law: computer capacity doubles every 18 months
- Anecdote #2: *Drum Computers*
  - in 1960's, expertise on laying out data on drum became obsolete with invention of magnetic core memory

Asymptotic Complexity

Our notion of efficiency:

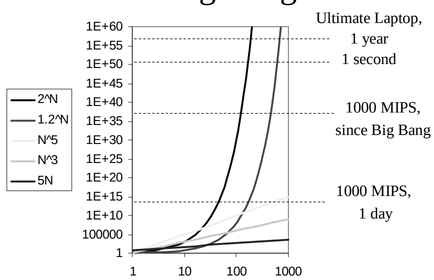
How the running time of an algorithm scales with the size of its input

- several ways to further refine:
  - worst case
  - average case
  - amortized over a series of runs

The Apocalyptic Laptop

Seth Lloyd, SCIENCE, 31 Aug 2000

Big Bang



Specific Goals of the Course

- Become familiar with some of the fundamental data structures in computer science
- Improve ability to solve problems abstractly
  - data structures are the building blocks
- Improve ability to analyze your algorithms
  - prove correctness
  - gauge (and improve) time complexity
- Become modestly skilled with the UNIX operating system (you'll need this in upcoming courses)

Why Are We All Here?

- My interest: Artificial intelligence
  - What are the ***theoretical limitations*** of difference algorithms for logical and probabilistic ***inference***?
  - How can a AI system ***learn*** to reason more efficiently, by ***analyzing*** it's past performance?
  - How can an AI system ***augment*** the reasoning capability of a person suffering from a cognitive disorder?
- What about computing interests you?

AI Graphics Systems

Theory Hardware Languages/Software

One Preliminary Hurdle

A little mathematics …

Interactive Survey:

CSE 321 completed?

$\sum_{i=1}^{n} i$?

$f(0) = a;\ f(n) = f(n/2) + c$?

$O(n)$ versus $\Omega(n)$ versus $\theta(n)$?

Proof of program correctness?

A Second Hurdle

- Unix

  Experience 1975 all over again!
  - Still the OS used for most cutting-edge research in computer science
  - Robust, stable, simple
  - Not just the OS and compiler, but a set of incredibly handy tools for running experiments and manipulating data – csh, awk, gnuplot
    - Also grep, perl
    - CYGWIN – simulates UNIX under Windows – handy way to develop code on your (non-Linux) laptop!

A Third Hurdle: Templates

```
class Set_of_ints {
   public:
      insert( int x );
      boolean is_member( int x ); … }

template <class Obj> class Set {
   public:
      insert( Obj x );
      boolean is_member( Obj x ); … }
Set <int> SomeNumbers;
Set <char *> SomeWords;
```

See notes on course web page on using templates in g++ !

Handy Libraries

- From Weiss:
```
vector < int > MySafeIntArray;
vector < double > MySafeFloatArray;
string MySafeString;
```
- Like arrays and char*, but provide
  - Bounds checking
  - Memory management
  - Okay to use
- STL (Standard Template Library)
  - most of CSE 326 in a box
  - ***don't use; we'll be rolling our own!***

Interactive Survey, Continued

- C++ ?
- Templates ? Defining new iterators?
- Unix ?
- Linked lists ? Stretchy arrays?
- Recursive vs. iterative computation of Fibonacci numbers?

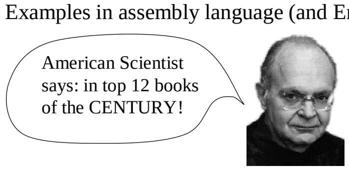
C++ $\neq$ Data Structures

One of the all time great books in computer science:

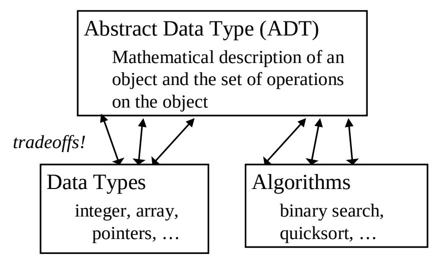
The Art of Computer Programming (1968-1973)

by Donald Knuth

Examples in assembly language (and English)!



Abstract Data Types



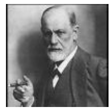
List ADT

- List properties $(\ A_1\ A_2\ \ldots\ A_{n-1}\ A_n\ )$ `length = n`
  - A_i precedes A_{i+1} for $1 \leq i < n$
  - A_i succeeds A_{i-1} for $1 < i \leq n$
  - Size 0 list is defined to be the **empty list**
- Key operations
  - Kth(integer) = item
  - Find(item) = position
  - Insert(item, position)
  - Delete(position)
  - Iterate through elements of the lists
- What are some possible data structures?

Which is Best?

	Linked list	Array	Sorted array
Kth()			
Find			
Insert			
Delete			
Iterate			

Why Analysis?



- Proofs of correctness guarantee that our code actually does what we intended it to do
- Complexity analysis makes our intuitions about efficiency concrete and precise

Summing an Array Recursively

```
int sum(int v[], int n)
{



}
```

Summing an Array Recursively

```
int sum(int v[], int n)
{
  if (n==0) return 0;
  else return v[n-1]+sum(v,n-1);
}
```

Inductive Proof of Correctness

```
int sum(int v[], int n)
{
  if (n==0) return 0;
  else return v[n-1]+sum(v,n-1);
}
```

Need to prove: sum(v,n) correctly returns sum of 1st n elements of array v for any n.

Basis Step: Program is correct for n=0; returns 0. ➨

Inductive Hypothesis (n=k): Assume sum(v,k) returns sum of first k elements of v.

Inductive Step (n=k+1): sum(v,k+1) returns v[k]+sum(v,k), which is the same of the first k+1 elements of v. ➨

To Do

- Get started on homework # 1
  - Log on to Unix servers
  - Bring questions to section!
- Read Weiss chapters 1 and 2