EECS 114:
Engineering Data Structures and Algorithms
Lecture 1

Brian Demsky

eecs114@newport.eecs.uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine
Lecture 1: Overview

• Course administration
  • Course web pages

• Getting started
  • Obtain your UCI netID
  • Obtain an account on the EECS servers
  • Log into the server

• Algorithm Analysis
Course Administration

- Course web pages online at http://eee.uci.edu/11f/18090/
  - Instructor information
  - Course description and contents
  - Course policies and resources
  - Course schedule
  - Homework assignments
  - Course communication
    - Mailing list (announcements)
    - Email (administrative issues)
Getting Started

• Obtain an account on the EECS servers
  • Your working account in EECS
Getting Started

- Log into the server
  - Terminal with SSH protocol (secure shell)
  - EECS servers
    - ladera.eecs.uci.edu
    - malibu.eecs.uci.edu
- User name, password
Java compilation

- Programming assignments should be completed in Java
- To compile a file into a class file
  javac file.java
- To execute a class file
  java file.class
- Java documentation available at http://java.sun.com/
Alternate Programming Environments

• You can use any platform you wish to write course assignments
• You can install Java on your own machine (MS Windows, Macintosh, Linux)
• You can use any text editor to write your code in
• But check that your assignments run on the Sun machines before turning them in
What is an Algorithm?

- An algorithm:
  - Takes an input (a value or set of values)
  - Produces an output (a value of set of values)
  - Terminates
  - Output satisfies some correctness property (the output of a sorting algorithm is sorted)
Why take this class?

• Fundamental - cross cutting across all areas of computer science
• Analysis aspect - need to know how long an algorithm takes to execute (will your code work with 1 million entries, 1 billion?), how to classify the difficult of problems
• Provides many solutions for a given problems
• Many applications of a given solutions
Other Reasons

• One in eight people in California is unemployed => You need to be competitive
• Interviewers for Computers Engineering jobs typically ask algorithms questions
• Why?
  • Easy to ask
  • Consider knowledge important in the work force
Example Algorithm: Sorting n integers

- Problem statement:
  - Input: An array $A=\{a_1, a_2, ..., a_n\}$
  - Output: An array $A'=\{a'_1, a'_2, ..., a'_n\}$ such that $a_i \leq a_{i+1}$ for $1 \leq i < n$.

- Many different possible algorithms to solve this problem
  - Different algorithms can have very different runtimes
  - Important to understand behavior of algorithm (can it handle large inputs)?
Analysis of Execution Time

• Use algorithm analysis to characterize behavior of algorithms

• Assumptions:
  • RAM (random access memory) model- all memory accesses are constant time
  • Sequential instruction execution (single processor)
  • Basic instructions are constant time (add, multiple, divide, subtract, compares, ...)

Algorithm Runtime

• Could measure it, but want a formula $T(n)$ where $n$ is the problem size so we can predict it
• Want to factor out machine details as scaling factors
• Worst case, best case, average case
search(A, key)
1. for i ← 1 to length[A]
2. if A[i]=key
3. then return i

Searches for key in the array A and returns the index of the key
### Best Case Algorithm Runtime

**search(A, key)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Cost</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>for i ← 1 to length[A]</td>
<td>$c_1$</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>if A[i]=key</td>
<td>$c_2$</td>
<td>1</td>
</tr>
<tr>
<td>3.</td>
<td>then return i</td>
<td>$c_3$</td>
<td>1</td>
</tr>
</tbody>
</table>

$T(n) = c_1 + c_2 + c_3$
Worst Case Algorithm Runtime

search(A, key)  cost  times
1. for i ← 1 to length[A]  $c_1$  n
2. if A[i]=key  $c_2$  n
3. then return i  $c_3$  1

$T(n) = n(c_1+c_2)+c_3$
Average Case Algorithm Runtime

search(A, key)  
cost  
times  
1. for i ← 1 to length[A]  \(c_1\)  \(n/2\)  
2. if A[i]=key  \(c_2\)  \(n/2\)  
3. then return i  \(c_3\)  1  

\[T(n) = \frac{n}{2}(c_1+c_2)+c_3\]
Asymptotic Notation

• The coefficients $c_1, c_2, \ldots$ depend on details of the machine

• Typically we just care about how fast the runtime grows with increasing input size
  • Coefficients aren’t important
  • Lower order terms aren’t important
Big-O Notation

• Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants $c$ and $n_0$ such that $f(n) \leq cg(n)$ for $n \geq n_0$.

• Informally, if $f(n)$ if $O(g(n))$, $f(n)$ grows no faster than $g(n)$.
Big-O Notation for Polynomials

- If \( f(n) \) is a polynomial, then \( f(n) \) is \( O(n^d) \) where \( d \) is the polynomial degree of \( f(n) \)
  - Drop lower-order terms
  - Drop constant factors
- Example
  - \( 3n^2+2n \) is \( O(n^2) \)
Other notations

• big-Omega (lower bound)
  • $f(n) \in \Omega(g(n))$ if there are constants $c>0$ and $n_0 \geq 1$ such that $f(n) \geq cg(n)$ for $n \geq n_0$

• big-Theta (tight bound)
  • $f(n) \in \Theta(g(n))$ if there are constants $c>0$, $c'>0$, and $n_0 \geq 1$ such that $cg(n) \leq f(n) \leq c'g(n)$ for $n \geq n_0$

• little-o (strict upper bound)
  • $f(n) \in o(g(n))$ if for any constant $c>0$ there is a constant $n_0 \geq 0$ such that $f(n) \leq cg(n)$ for $n \geq n_0$

• little-omega (strict lower bound)
  • $f(n) \in \omega(g(n))$ if for any constant $c>0$ there is a constant $n_0 \geq 0$ such that $f(n) \geq cg(n)$ for $n \geq n_0$