CSCI 104
Overview

Mark Redekopp
Aaron Cote
Sandra Batista
Administrivia

• Online (via Zoom) for Summer 2021
• Exams: 1 midterms and 1 final
• Six assignments.
  – Each assignment has a written component due earlier in the HW cycle and a programming component due later.
• Lectures will be recorded and should automatically be posted on Blackboard under USC Zoom Pro on the left menu of our course's BB section
• All other content is on our website (https://bytes.usc.edu/cs104)
• Class should be interactive. Speak up directly (I don't mind being interrupted), raise your hand, or write in chat to ask questions.
Administrative Issues

• Preparation
  – Basic if, while, for constructs
  – Arrays, linked-lists
  – Structs, classes (constructors, destructors, operator overloading, copy semantics)
  – Dynamic memory allocation and pointers

• Syllabus
  – https://bytes.usc.edu/cs104

• Expectations
  – I'll give you my best, you give me yours...
    • Attendance, participation, asking questions, academic integrity, take an interest
  – Treat CS104 right!
  – Let's make this fun
Some Helpful Links

Remedial modules
http://ee.usc.edu/~redekopp/csmodules.html

Class website
http://bytes.usc.edu/cs104/
Organizing Your Data

• Intentionally vague question: "Should you always sort your data?"
  – No. What are the tradeoffs?
  – An Insert operation becomes more expensive, but a Lookup operation becomes less expensive
  – In a backup system, you are constantly inserting information, and you rarely (hopefully never) performing lookups on that information.

• How should you organize your data? What is the best data structure?
  – The answer is, invariably, “it depends.”
  – Otherwise, this class would be called “Data Structure” (singular), I’d teach it to you today, and everyone would go home and get an A.
  – Demo...Need 2 volunteers
Data Structure Consideration

• Some questions to consider:
  – Will you search the data often?
  – Will data be added in small, frequent chunks?
  – Will data be added in large, infrequent chunks?
• Besides Insert and Lookup, what other operations are common?
  – Remove and Update
• Which of these operations you need, and how frequently you need each one, will dictate which data structure you select!
  – There is a data structure called a “Heap” which is really good at all of these operations... except Lookup!
  – Others, such as AVL Trees, are able to do all 4 operations fairly well (but they are worse than Heaps on every operation except Lookup!)
  – Yet others, such as Hash Tables, are usually lightning fast, but are probabilistic and occasionally produce very bad runtimes.
Why Data Structures Matter?

- Modern applications process vast amount of data
- Adding, removing, searching, and accessing are common operations
- Various data structures allow these operations to be completed with different time and storage requirements

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Insert</th>
<th>Lookup</th>
<th>Get-Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted List</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>AVL Tree</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Heap</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

Recall $\Theta(n)$ indicates that the actual run-time is bounded by some expression $a*n$ for some $n > n_0$ (where $a$ and $n_0$ are constants)
## Importance of Complexity

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Bit operations used</th>
</tr>
</thead>
<tbody>
<tr>
<td>n =</td>
<td>log n</td>
</tr>
<tr>
<td>10</td>
<td>3 x 10⁻¹¹ s</td>
</tr>
<tr>
<td>10²</td>
<td>7 x 10⁻¹¹ s</td>
</tr>
<tr>
<td>10³</td>
<td>10⁻¹⁰ s</td>
</tr>
<tr>
<td>10⁴</td>
<td>1.3 x 10⁻¹⁰ s</td>
</tr>
<tr>
<td>10⁵</td>
<td>1.7 x 10⁻¹⁰ s</td>
</tr>
<tr>
<td>10⁶</td>
<td>2 x 10⁻¹⁰ s</td>
</tr>
</tbody>
</table>
Abstract Data Types

• Programming students tend to focus on the code and less on the data and its organization
• More seasoned programmers focus first on
  – What data they have
  – How it will be accessed
  – How it should be organized
• An **abstract data type** describes what data is stored and what operations are to be performed
• A **data structure** is a specific way of storing the data implementing the operations
• Example **ADT**: **List**
  – Data: items of the same type in a particular order
  – Operations: insert, remove, get item at location, set item at location, find
• Example **data structures** implementing a **List**:
  – Linked list, array, etc.
Our First ADT

• **add(key, value)**
  – The key is a unique identifier that we can use to find the value in the future.
  – `add(6, "Super Mario Bros.")`

• **remove(key)**
  – `remove(11)`, to remove "Pac Man".

• **Lookup(key)**
  – `Lookup(9)`, to find "Tetris".

• This ADT is known as a map. We implemented the above map using a sorted list. So, is a sorted list a map?
  – No! The sorted list is the data structure. The map is the ADT.
Course Goals

01
Learn basic and advanced techniques for implementing data structures and analyzing their efficiency
• Will require mathematical analysis from CS 170

02
Learn how to identify the best data structure for your needs.

03
Learn object-oriented design principles that make your code readable, modular, and extensible