Today’s Outline

• Announcements
  – Project #1 due Wed, Jan 13 at 11pm, come to section tomorrow with questions.
  – Two emails sent to cse326 mailing list – did you get them?
  – Have you installed Eclipse (or whatever environment you will be using this quarter) and Java yet?
  – Please fill out survey and bring Info sheet to class on Friday.

• Queues and Stacks

• Math Review
  – Proof by Induction
  – Powers of 2
  – Binary numbers
  – Exponents and Logs

---

Project 1 – Sound Blaster!

Play your favorite song in reverse!

Aim:
1. Implement stack ADT two different ways
2. Use to reverse a sound file

Due: Wed, Jan 13, at 11pm via Catalyst

---

Mathematical Induction

Suppose we wish to prove that:
For all \( n \geq n_0 \), some predicate \( P(n) \) is true.

We can do this by proving two things:
1. \( P(n_0) \) --- this is called the “basis.”
2. If \( P(k) \) then \( P(k+1) \) -- this is called the “induction step.”

---

Example: Basis Step

Prove for all \( n \geq 1 \), sum of first \( n \) powers of 2 = \( 2^n - 1 \)

\[
2^0 + 2^1 + 2^2 + \ldots + 2^{n-1} = 2^n - 1.
\]

in other words: \[
1 + 2 + 4 + \ldots + 2^{n-1} = 2^n - 1.
\]

Proof by induction:

Basis with \( n_0 = 1 \):

(left hand side)
(right hand side)

So true for \( n_0 = 1 \)
Example: Inductive Step

- **Induction hypothesis:** (Assume this is true)
  \[ 1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1 \]
- **Induction step:**
  \[ 1 + 2 + 4 + \ldots + 2^{k-1} + 2^k = 2^{k+1} - 1 \]

Therefore if the equation is valid for \( n = k \), it must also be valid for \( n = k+1 \).

- **Summary:** It is valid for \( n=1 \) (basis) and by the induction step it is therefore valid for \( n=2, n=3, \ldots \) It is valid for all integers greater than 0.

---

Powers of 2

- Many of the numbers we use in Computer Science are powers of 2
- Binary numbers (base 2) are easily represented in digital computers
  - each "bit" is a 0 or a 1
  - an \( n \)-bit wide field can represent how many different things?

<table>
<thead>
<tr>
<th># Bits</th>
<th>Patterns</th>
<th># of patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unsigned binary numbers

- For **unsigned** numbers in a fixed width field
  - the minimum value is 0
  - the maximum value is \( 2^n - 1 \) where \( n \) is the number of bits in the field
  - The value is \( \sum_{i=0}^{n-1} a_i 2^i \)
  - Each bit position represents a power of 2 with \( a_i = 0 \) or \( a_i = 1 \)
Signed Numbers?

Logarithms and Exponents

- Definition: \( \log_2 x = y \) if and only if \( x = 2^y \)
  - \( 8 = 2^3 \), so \( \log_2 8 = 3 \)
  - \( 65536 = 2^{16} \), so \( \log_2 65536 = 16 \)
- Notice that \( \log_2 n \) tells you how many bits are needed to distinguish among \( n \) different values.
  - 8 bits can hold any of 256 numbers, for example: 0 to \( 2^8 - 1 \), which is 0 to 255
  - \( \log_2 256 = 8 \)

One function that grows very quickly, One that grows very slowly

One function that grows very quickly, One that grows very slowly

What is the minimum height of a binary tree with \( N \) nodes?

Floor and Ceiling

\[
\left\lfloor X \right\rfloor \quad \text{Floor function: the largest integer} \leq X
\]

\[
\left\lceil X \right\rceil \quad \text{Ceiling function: the smallest integer} \geq X
\]

\[
\left\lfloor 2.7 \right\rfloor = 2 \quad \left\lceil -2.7 \right\rceil = -3 \quad \left\lfloor 2 \right\rfloor = 2
\]

\[
\left\lfloor -2.3 \right\rfloor = 3 \quad \left\lceil -2.3 \right\rceil = -2 \quad \left\lceil 2 \right\rceil = 2
\]
Facts about Floor and Ceiling

1. \( x - 1 \leq \lfloor x \rfloor \leq x \)
2. \( x \leq \lfloor x \rfloor < x + 1 \)
3. \( \lfloor x/2 \rfloor + \lfloor x/2 \rfloor = n \) if \( n \) is an integer

Properties of logs

- We will assume logs to base 2 unless specified otherwise.
- \( 8 = 2^3 \), so \( \log_2 8 = 3 \), so \( 2^{\log_2 8} = \) ________

Show:
- \( \log (A \cdot B) = \log A + \log B \)
  
- \( A = 2^{\log_2 A} \) and \( B = 2^{\log_2 B} \)
  
- \( A \cdot B = 2^{\log_2 A + \log_2 B} \)
  
- So: \( \log_2 AB = \log_2 A + \log_2 B \) !!

Other log properties

- \( \log A/B = \log A - \log B \)
- \( \log (A^B) = B \log A \)
- \( \log X < \log Y < X \) for all \( X > 0 \)
  
  - \( \log \log Y < \log Y < Y \) for all \( Y > 0 \)
  
  - called a “sub-linear” function

Note: \( \log \log X \neq \log (\log X) \)

A log is a log is a log

- “Any base \( B \) log is equivalent to base 2 log within a constant factor.”

Algorithm Analysis Examples

- Consider the following program segment:
  
  \[
  x := 0; \\
  \text{for } i = 1 \text{ to } N \text{ do} \\
  \quad \text{for } j = 1 \text{ to } i \text{ do} \\
  \quad \quad x := x + 1; \\
  \]

  What is the value of \( x \) at the end?

- Total number of times \( x \) is incremented is executed =

  \[
  1 + 2 + 3 + \ldots + N = \sum_{k=1}^{N} k = \frac{N(N+1)}{2} \\
  \]

  An Arithmetic Sequence

  - Congratulations - You’ve just analyzed your first program!
  
  - Running time of the program is proportional to \( N(N+1)/2 \) for all \( N \)
  
  - Big-O ?!
Asymptotic Analysis

What we want

• Rough Estimate
• Ignores Details

Big-O Analysis

• Ignores “details”

Analysis of Algorithms

• Efficiency measure
  – how long the program runs time complexity
  – how much memory it uses space complexity
    • For today, we’ll focus on time complexity only

• Why analyze at all?

Asymptotic Analysis

• Complexity as a function of input size \( n \)
  \[ T(n) = 4n + 5 \]
  \[ T(n) = 0.5 n \log n - 2n + 7 \]
  \[ T(n) = 2^n + n^3 + 3n \]

• 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)