Math Review

CSE 373
Data Structures & Algorithms
Ruth Anderson
Autumn 2011

Today’s Outline

• Announcements
  – Assignment #1 due Thurs, Oct 6 at 11pm
  – Midterm date? Please fill out poll by end of day today.

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

When did you take cse 143?

<table>
<thead>
<tr>
<th>Numeric values</th>
<th>Answer</th>
<th>Frequency</th>
<th>Percenta ge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 - summer 11</td>
<td>6</td>
<td>6.52%</td>
</tr>
<tr>
<td>2</td>
<td>1 - spring 11</td>
<td>8</td>
<td>8.70%</td>
</tr>
<tr>
<td>3</td>
<td>2 - winter 11</td>
<td>21</td>
<td>22.83%</td>
</tr>
<tr>
<td>4</td>
<td>3 - autumn 10</td>
<td>13</td>
<td>14.13%</td>
</tr>
<tr>
<td>5</td>
<td>4 - summer 10</td>
<td>2</td>
<td>2.17%</td>
</tr>
<tr>
<td>6</td>
<td>5 - spring 10</td>
<td>16</td>
<td>17.39%</td>
</tr>
<tr>
<td>7</td>
<td>6 - before spring 10</td>
<td>19</td>
<td>20.65%</td>
</tr>
<tr>
<td>8</td>
<td>7 - I did not take CSE 143 at UW (AP or transfer credit)</td>
<td>5</td>
<td>5.43%</td>
</tr>
<tr>
<td>9</td>
<td>Other:</td>
<td>2</td>
<td>2.17%</td>
</tr>
<tr>
<td></td>
<td>Total responses (N): 92</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Did not respond: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Homework 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: Thurs, Oct 6, 2011
Submit via catalyst drop box before: 11pm

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 “base case” or “basis.”
2. If \( P(k) \), then \( P(k+1) \) - this is called the “induction step” or “inductive case”

Note: We prove 2. by assuming \( P(k) \) is true.
Putting these together, we show that \( P(n) \) is true.

Example

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

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

in other words:

\[
1 + 2 + 4 + \ldots + 2^{n-1} = 2^n - 1.
\]
Example Proof by Induction

Theorem: \( P(n) \) holds for all \( n \geq 1 \)
Proof: By induction on \( n \)
- Base case, \( n=1 \):
  \[ 2^1 = 1 \]
- Induction step:
  - Inductive hypothesis: Assume the sum of the first \( k \) powers of 2 is \( 2^k - 1 \)
  - Given the hypothesis, show that:
    - the sum of the first \( k+1 \) powers of 2 is \( 2^{k+1} - 1 \)
  From our inductive hypothesis we know:
  \[
  1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1
  \]
  Add the next power of 2 to both sides:
  \[
  1 + 2 + 4 + \ldots + 2^{k-1} + 2^k = 2^k + 1
  \]
  We have what we want on the left; massage the right a bit:
  \[
  1 + 2 + 4 + \ldots + 2^{k-1} + 2^k = 2(2^k - 1) + 1
  \]
  \[
  = 2^{k+1} - 1
  \]
Therefore if the equation is valid for \( n = k \), it must also be valid for \( n = k+1 \).

Summary: Our theorem is valid for \( n=1 \) (base case) and by the induction step it is therefore valid for \( n=2, n=3, \ldots \)
Thus, it is valid for all integers greater than or equal to 1.

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?

\[
\begin{array}{c|c|c}
\text{# Bits} & \text{Patterns} & \text{# of patterns} \\
\hline
1 & 000000000101011 & \\
2 & & \\
\end{array}
\]

Unsigned binary numbers
- For \textit{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 \( \sum_{i=0}^{n-1} a_i \cdot 2^i \)
- Each bit position represents a power of 2 with \( a_i = 0 \) or \( a_i = 1 \)

\[
\begin{array}{c|c|c}
\text{Java:} & & \\
\text{an \textit{int} is 32 bits and signed, so “max int” is “about 2 billion”} & & \\
\text{a \textit{long} is 64 bits and signed, so “max long” is 2^{64}-1} & & \\
\end{array}
\]

Powers of 2
- A bit is 0 or 1
- A sequence of \( n \) bits can represent \( 2^n \) distinct things
  - For example, the numbers 0 through \( 2^n - 1 \)
  - \( 2^{10} \) is 1024 ("about a thousand", kilo in CSE speak)
  - \( 2^{20} \) is "about a million", mega in CSE speak
  - \( 2^{30} \) is "about a billion", giga in CSE speak

\[
\begin{array}{c|c|c}
\text{Java:} & & \\
\text{an \textit{int} is 32 bits and signed, so “max int” is “about 2 billion”} & & \\
\text{a \textit{long} is 64 bits and signed, so “max long” is 2^{64}-1} & & \\
\end{array}
\]
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 $\log_n n$ 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
  - $\log_2 256 = 8$

Therefore...

- Could give a unique id to...
  - Every person in the U.S. with 29 bits
  - Every person in the world with 33 bits
  - Every person to have ever lived with 38 bits (estimate)
  - Every atom in the universe with 250-300 bits

So if a password is 128 bits long and randomly generated, do you think you could guess it?

Signed Numbers?

- Since so much is binary in CS, $\log$ almost always means $\log_2$
- Definition: $\log_2 x = y$ if $x = 2^y$
- So, $\log_2 1,000,000 = \text{“a little under 20”}$
- Just as exponents grow very quickly, logarithms grow very slowly

See Excel file for plot data – play with it!

Logarithms and Exponents

- Since so much is binary in CS, $\log$ almost always means $\log_2$
- Definition: $\log_2 x = y$ if $x = 2^y$
- So, $\log_2 1,000,000 = \text{“a little under 20”}$
- Just as exponents grow very quickly, logarithms grow very slowly

See Excel file for plot data – play with it!
Logarithms and Exponents

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

Floor and Ceiling

Facts about Floor and Ceiling

Properties of logs
### Other log properties

- \( \log A/B = \log A - \log B \)
- \( \log (A^B) = B \log A \)
- \( \log \log X \leq \log X \leq X \) for all \( X > 0 \)
  - \( \log \log X = Y \) means:
    - \( \log x \) grows more slowly than \( x \)
    - called a “sub-linear” function
    - \( (\log x)/(\log x) \) is written \( \log^2 x \) (aka “log-squared”)
  - Note: \( \log \log X \neq \log^2 X \)

### A log is a log is a log

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

### Log base doesn’t matter (much)

“Any base \( B \) log is equivalent to base 2 log within a constant factor”
- And we are about to stop worrying about constant factors!
- In particular, \( \log_2 x = 3.22 \log_10 x \)
- In general, we can convert log bases via a constant multiplier
- To convert from base \( B \) to base \( A \):
  \[
  \log_A x = \left( \frac{\log_B x}{\log_B A} \right)
  \]

### Arithmetic Sequences

- \( N = \{0, 1, 2, \ldots \} \) = natural numbers
- \( \{0, 1, 2, \ldots \} \) is an infinite arithmetic sequence
- \( \{a, a+d, a+2d, \ldots \} \) is a general infinite arith. sequence.

There is a constant difference between terms.

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

### Algorithm Analysis Examples

- Consider the following program segment:
  
  ```
  x:= 0;
  for i = 1 to N do
    for j = 1 to i do
      x := x + 1;
  ```

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

### Analyzing the Loop

- Total number of times \( x \) is incremented is executed =
  
  \[
  1+2+3+\ldots+N=\sum_{i=1}^{N}i = \frac{N(N+1)}{2}
  \]

- 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 ??