Assignment #1

Goals of this assignment:
• Introduce the ADTs (abstract data types) for lists and sparse vectors, motivated by an application to information retrieval.
• Show the connection between the empirical runtime scaling of an algorithm and formal asymptotic complexity
• Gain experience with the Unix tools g++, make, gnuplot, csh, and awk.
• Learn how to use templates in C++.
  – We will use new g++ version 3.0 compiler – does templates right!

Implementing in C++

Create separate classes for
– Node (contains a pointer to the first node)
– List Iterator (specifies a position in a list; basically, just a pointer to a node)

Pro: syntactically distinguishes uses of node pointers
Con: a lot of verbage! Also, is a position in a list really distinct from a list?

Implementing Linked Lists Using Arrays

1  2  3  4  5  6  7  8  9  10
Data  F  O  A  R  N  R  T
Next  3  4  6  7  5  10  2
First = 2

“Cursor implementation” Ch 3.2.8
Can use same array to manage a second list of unused cells

Structure Sharing

L = (a b c)  M = (b c)

• Important technique for conserving memory usage in large lists with repeated structure
• Used in many recursive algorithms on lists

Implementing Linked Lists in C

(struct node{
  Object element;
  struct node * next;
} Everything else is a pointer to a node!
typedef struct node * List;
typedef struct node * Position;
List ADT
Polynomial ADT

\( A_i \) is the coefficient of the \( x^i \) term:

\[
\begin{align*}
5 + 2x + 3x^2 & : (5 2 3) \\
7 + 8x & : (7 8) \\
3 + x^2 & : (3 0 2)
\end{align*}
\]

Problem?

Sparse List Data Structure:
\( 4 + 3x^{2001} \)
\((<4 0> <2001 3>)\)

Addition of Two Polynomials?
\( 15 + 10x^{10} + 3x^{1200} \)
\( 5 + 30x^{50} + 4x^{100} \)

To ADT or NOT to ADT?

- Issue: when to bypass / expand List ADT?
- Using general list operations:

```
reverse(x) {
    y = new list;
    while (!x.empty()){
        /* remove 1st element from x, insert in y */
        y.insert_after_kth( x.kth(1), 0);
        x.delete_kth(1);
    }
    return y;
}
```

Disadvantages?
Analysis of Algorithms

- Analysis of an algorithm gives insight into how long the program runs and how much memory it uses
  - time complexity
  - space complexity
- Why useful?
  - Input size is indicated by a number \( n \)
  - sometimes have multiple inputs, e.g. \( m \) and \( n \)
- Running time is a function of \( n \)
  \[ n, \ n^2, \ n \log n, \ 18 + 3n(\log n^3) + 5n^4 \]

Simplifying the Analysis

- Eliminate low order terms
  \[ 4n + 5 \Rightarrow 4n \]
  \[ 0.5 n \log n - 2n + 7 \Rightarrow 0.5 n \log n \]
  \[ 2^n + n^3 + 3n \Rightarrow 2^n \]
- Eliminate constant coefficients
  \[ 4n \Rightarrow n \]
  \[ 0.5 n \log n \Rightarrow n \log n \]
  \[ \log n^2 = 2 \log n \Rightarrow \log n \]
  \[ \log_3 n = (\log_2 2) \log n \Rightarrow \log n \]

Order Notation

- BIG-O \( T(n) = O(f(n)) \)
  Upper bound
  Exist constants \( c \) and \( n' \) such that
  \[ T(n) \leq c f(n) \] for all \( n \geq n' \)
- OMEGA \( T(n) = \Omega(f(n)) \)
  Lower bound
  Exist constants \( c \) and \( n_0 \) such that
  \[ T(n) \geq c f(n) \] for all \( n \geq n_0 \)
- THETA \( T(n) = \Theta(f(n)) \)
  Tight bound
  \[ O(n) = \Omega(n) = \Theta(n) \]

Examples

\[ n^2 + 100 n = O(n^2) \] because
\[ (n^2 + 100 n) \leq 2 n^2 \] for \( n \geq 10 \)
\[ n^2 + 100 n = \Omega(n^2) \] because
\[ (n^2 + 100 n) \geq 1 n^2 \] for \( n \geq 0 \)
Therefore:
\[ n^2 + 100 n = \Theta(n^2) \]

Notation Gotcha

- Order notation is not symmetric; write
  \[ 2n^2 + 4n = O(n^2) \]
  but never
  \[ O(n^3) = 2n^2 + 4n \]
  right hand side is a redudndification of the left
Likewise
\[ O(n^3) = O(n^3) \]
\[ \Omega(n^3) = \Omega(n^3) \]
Mini-Quiz

1. $5n \log n = O(n^2)$
2. $5n \log n = \Omega(n^2)$
3. $5n \log n = O(n)$
4. $5n \log n = \Omega(n)$
5. $5n \log n = \theta(n)$
6. $5n \log n = \theta(n \log n)$

Silicon Downs

<table>
<thead>
<tr>
<th>Post #1</th>
<th>Post #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^3 + 2n^2$</td>
<td>$100n^2 + 1000$</td>
</tr>
<tr>
<td>$n^{0.1}$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>$n + 100n^{0.1}$</td>
<td>$2n + 10 \log n$</td>
</tr>
<tr>
<td>$5n^5$</td>
<td>$n!$</td>
</tr>
<tr>
<td>$n^{100/100}$</td>
<td>$1000n^{15}$</td>
</tr>
<tr>
<td>$g \log n$</td>
<td>$3n^2 + 7n$</td>
</tr>
</tbody>
</table>

Race I

$n^3 + 2n^2$ vs. $100n^2 + 1000$

Race II

$n^{0.1}$ vs. $\log n$

Race III

$n + 100n^{0.1}$ vs. $2n + 10 \log n$

Race IV

$5n^5$ vs. $n!$
Race V
\[ n^{-15}2^n/100 \text{ vs. } 1000n^{15} \]

Race VI
\[ g^{2\log(n)} \text{ vs. } 3n^7 + 7n \]

The Losers Win

<table>
<thead>
<tr>
<th>Post #1</th>
<th>Post #2</th>
<th>Better algorithm!</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n^3 + 2n^2 )</td>
<td>( 100n^2 + 100 )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>( n^{3/2} )</td>
<td>( \log n )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>( n + 100n^{5/4} )</td>
<td>( 2n + 10 \log n )</td>
<td>( TIE: O(n) )</td>
</tr>
<tr>
<td>( 5n^3 )</td>
<td>( n^3 )</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>( n^{1+2/100} )</td>
<td>( 1000n^{15} )</td>
<td>( O(n^{15}) )</td>
</tr>
<tr>
<td>( g^{2\log n} )</td>
<td>( 3n^7 + 7n )</td>
<td>( O(n^9) )</td>
</tr>
</tbody>
</table>

Common Names

- Constant: \( O(1) \)
- Logarithmic: \( O(\log n) \)
- Linear: \( O(n) \)
- Log-linear: \( O(n \log n) \)
- Superlinear: \( O(n^{1+c}) \) (c is a constant > 0)
- Quadratic: \( O(n^2) \)
- Polynomial: \( O(n^k) \) (k is a constant)
- Exponential: \( O(c^n) \) (c is a constant > 1)

Analyzing Code

- C++ operations - constant time
- Consecutive stms - sum of times
- Conditionals - sum of branches, condition
- Loops - sum of iterations
- Function calls - cost of function body
- Recursive functions - solve recursive equation

Above all, use your head!

Conditionals

- Conditional
  \[
  \text{if } C \text{ then } S_1 \text{ else } S_2
  \]
  \[
  \text{time} \leq \text{time}(C) + \text{MAX( time}(S_1), \text{time}(S_2))
  \]
Nested Loops

for i = 1 to n do
for j = 1 to n do
sum = sum + 1

Nested Loops

for i = 1 to n do
for j = 1 to n do
sum = sum + 1

\[ \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = n^2 \]

Nested Dependent Loops

for i = 1 to n do
for j = i to n do
sum = sum + 1

Nested Dependent Loops

for i = 1 to n do
for j = i to n do
sum = sum + 1

1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1

\[ \sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n} (n-i) = \sum_{i=1}^{n} (n+1) - \sum_{i=1}^{n} i = \]

\[ \sum_{i=1}^{n} (n-i) = \sum_{i=1}^{n} (n+1) - \sum_{i=1}^{n} i = \]

\[ n(n+1) - \frac{n(n+1)}{2} = \frac{n(n+1)}{2} = O(n^2) \]

To Do

- Finish reading Ch 1 and 2
- Start reading Ch 3
- Get started on assignment #1!