Let’s Solve a Problem

Longest Common Subsequence (LCS)

\[ S = a_1 \ a_2 \ \ldots \ a_n \]

A subsequence is

\[ A B S Q U A T U L A T E \]
Common Subsequence

\[ S_1 = a_1 a_2 \ldots a_n \]
\[ S_2 = b_1 b_2 \ldots b_m \]
\[ S = \]

where \( S \) is a subsequence of

\[ \text{QUATULATE} \]
\[ \text{CALEFACTION} \]

Longest Common Subsequence

\[ \text{GANGLION} \]
\[ \text{HAGIOLATRY} \]

Given sequences \( S_1 \) and \( S_2 \), want to find
Enumeration Algorithm

LCS(S1, S2)
max ← 0
for every subseq. R1 of S1
  for every subseq. R2 of S2
    if R1 = R2 and length(R1) > max
      then max ← length(R1)
return(max)

time complexity =

Divide and Conquer

S1 . . . . . . . . . S2 . . . . . . . . .
S1' . . . S1'' . . . S2' . . . S2'' . . . .

LCS(S1, S2) =
Recursion

\[ R = a b b c d a \]
\[ S = a e b b a e e \]

\[ R[i] \neq S[j] \]
\[ \text{LCS}(i,j) = \max(\text{LCS}(\_), \text{LCS}(\_)) \]
\[ \text{LCS}(6,7) = \]

\[ R[i] = S[j] \]
\[ \text{LCS}(i,j) = \text{LCS}(\_)+1 \]
\[ \text{LCS}(6,5) = \]

Recursion 2

\[
\text{LCS}(i,j) =
\begin{align*}
\text{if } i=0 \text{ or } j=0 \text{ then } \text{LCS} & \leftarrow 0 \\
\text{else if } R[i] \neq S[j] \text{ then } & \\
\text{LCS} & \leftarrow \max(\text{LCS}(i-1,j), \text{LCS}(i,j-1)) \\
\text{else } & \\
\text{LCS} & \leftarrow \text{LCS}(i-1,j-1) + 1
\end{align*}
\]

Count comparisons

\[ c(\text{LCS}(i,j)) = \]
Dynamic Programming
Have matrix $\text{LCS}(0-m, 0-n)$ and fill it in

$R = a \ a \ b \ c \ a \ b$
$S = c \ a \ b \ b$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ri</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>Sj</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Algorithm Longest

1. Set $\text{LCS}[i, j] \leftarrow 0$ if $i=0$ or $j=0$
2. for $j = 1$ to $n$ do
   for $i = 1$ to $m$ do
     if $R[i] \neq S[j]$ then
       \[ \text{LCS}[i, j] \leftarrow \max(\text{LCS}[i-1, j], \text{LCS}[i, j-1]) \]
     else
       \[ \text{LCS}[i, j] \leftarrow \text{LCS}[i-1, j-1]+1 \]
3. return $\text{LCS}[]$

Time complexity
Logistics

Course web page is
http://www.cs.pdx.edu/~maier/cs584

Expect class email list to be
cs584@cs.pdx.edu
(Watch the web page for sure.)

Will try to post weekly lecture slides by the evening before Monday class

Class Activity and Quiz

Work in groups on a sample execution of the Freq algorithm

Will have a short quiz on it -
The quiz doesn’t count, but should help you assess if you are prepared for this class

I assume you know the material from CS350
Algorithm Design & Analysis

Freq(S)
0. If |S| = 0, return (null, 0)

1. Pick some element e ∈ S.

2. Split S into two subsets:
   \[ S_{\text{low}} \leftarrow \{ d \in S \mid d < e \} \]
   \[ S_{\text{high}} \leftarrow \{ d \in S \mid d > e \} \]

3. Set \( f \leftarrow |S| - |S_{\text{low}}| - |S_{\text{high}}| \)
   Set \((e_{\text{low}}, f_{\text{low}}) \leftarrow \text{Freq}(S_{\text{low}})\)
   Set \((e_{\text{high}}, f_{\text{high}}) \leftarrow \text{Freq}(S_{\text{high}})\)

4. If \( f > f_{\text{low}} \) and \( f > f_{\text{high}} \), return \((e, f)\)
   else if \( f_{\text{low}} > f_{\text{high}} \), return \((e_{\text{low}}, f_{\text{low}})\)
   else return \((e_{\text{high}}, f_{\text{high}})\)

Study of Algorithms

- Design techniques
- Specific algorithms
- Proofs of correctness
- Complexity analysis
  - measure
  - time it takes, space it consumes
  - worst case
  - average case
Study of Problems

- What's the best any algorithm can do?
- Relatedness
- Intractability of problems

Complexity of an Algorithm

Amount of time and space required as a function of size of input

\[ T(n) = \text{maximum time required} \]

\[ S(n) = \text{maximum space required} \]

Input size      Time measures
Time Complexity of a Problem

Minimum time over all algorithms

to get
  upper bound:
  lower bound:

Asymptotic Complexity

order notation
  \( f(n) \) has \( \mathcal{O}(g(n)) \) if
  \( f(n) \leq g(n) \)

\[ f(n) = n^2 + 3n + 4 \]
\( f(n) \) has
\( f(n) \)
Asymptotic Notation

\[ O(g(n)) = \{ \text{there are positive } c \text{ and } n_0 \text{ such that} \} \]
\[ n^2 + 3n + 4 = O(n^2) \]

means

\[ \Theta(g(n)) = \{ f(n) \mid \text{there are positive } c_1, c_2 \text{ and } n_0 \text{ such that} \} \]

\[ n^2 + 3n + 4 \]

Asymptotic Notation 2

Note

\[ n^2 + 3n + 4 \in O(n^3) \]
\[ n^2 + 3n + 4 \in \Theta(n^3) \]
\[ \Theta(n^3) \subseteq O(n^3) \]

\[ \Omega(g(n)) = \{ f(n) \mid \text{there are positive } c \text{ and } n_0 \text{ such that} \} \]

asymptotic

- upper bound
- tight bound
- lower bound
Algorithm Design & Analysis

Asymptotic Notation 3

Theorem
\[ f(n) \in O(g(n)) \quad \text{and} \quad f(n) \in \Omega(g(n)) \]
imply

Other facts
\[ f(n) \in \Theta(h(n)) \quad \text{and} \quad h(n) \in \Theta(g(n)) \]
imply
\[ f(n) \in \Theta(g(n)) \]
implies

Heapsort

Heap: binary tree where all leaves at depth

node labels are

value at node all values in subtrees
Algorithm Design & Analysis

Example Heap

Heapsort Algorithm

Heapsort
1. remove
2. move
3. delete
4. remake

Implicit data structure
A[1..n]
A[1] is
left and right children of A[i] are
Example Structure

![Tree Diagram]

Restriction:

Swap
Pretend array runs

Remake heap:
exchange value with

Remove Root Value

![Tree Diagram]
Formally

Input: $a_1 \ a_2 \ \ldots \ an$ to sort
Initialize $A[i] \leftarrow$
build-max-heap
  want $A[i]$
  for $i$ between $1$ and $n/2$

Heap Construction

Assume:
Want:
max-heapify$(i,j)$
  if $i$ not a leaf and
    a child of $i$ has a greater value
  then
    let $k$ be child with larger value

build-max-heap
  for $i = n$ downto $1$ do
Example of Build-max-heap

Complexity of Build-max-heap

Theorem: build-max-heap has $O(n)$ time complexity

Define $T(h)$ as the time to max-heapify a node of height $h$

$T(h) \leq T(h)$

Build-max-heap calls max-heapify once per node. How many nodes of height $i$?
Heapsort Algorithm

heapsort
    build-max-heap
    for i = n to 2 do

Sorted list ends up in \( A[1] \ldots A[n] \)

Time complexity:

Heapsort Example

```
5
/  \
15  3
/  \
6  13
/  \
4  23
```

Sorted list ends up in \( A[1] \ldots A[n] \)
Lower Bounds on Sorting

Counting comparisons
Assume $A[1], A[2], \ldots, A[n]$ are distinct
Represent algorithm by a decision tree

Fact: Binary tree of height $h$ has at most $2^h$ leaves

Consequence

Corollary: Any algorithm for sorting by comparisons needs compares for some $c > 0$ and large $n$.

$$n! \geq n(n-1) \ldots (n/2) \geq (n/2)^{n/2}$$

$$n \log n \geq \log n! \quad \text{since}$$