

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

## Algorithm Design & Analysis

David Maier  
Portland State University

All material © 1994, 2003,  
2006, 2008, 2009  
David Maier

Lecture 1 David Maier 1

Algorithm Design & Analysis

### Let's Solve a Problem

## Longest Common Subsequence (LCS)

$S = a_1 a_2 \dots a_n$

A subsequence is

$a_{i_1} a_{i_2} \dots a_{i_k} \quad i_1 < i_2 < \dots < i_k$

A B S Q U A T U L A T E

A Q T U L  
L A T E  
ε empty string

Lecture 1 David Maier 2

Algorithm Design & Analysis

### Common Subsequence

$S_1 = a_1 a_2 \dots a_n$   
 $S_2 = b_1 b_2 \dots b_m$   
 $S = c_1 c_2 \dots c_k$

where  $S$  is a subsequence of  
*both  $S_1$  and  $S_2$*

Lecture 1 David Maier **3**

Algorithm Design & Analysis

### Longest Common Subsequence

4

Given sequences  $S_1$  and  $S_2$ , want to find length of the longest common subsequence of  $S_1$  and  $S_2$

Lecture 1 David Maier **4**

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

### 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)
    
```

*longest k shortest 2 quit at first one*

*prune out letters in just one sequence*

*check if R1 subseq of S2*

$length(S_1) = n \quad length(S_2) = m$

time complexity =  $2^n \cdot m$

*comparisons of letters*

Lecture 1 David Maier 5

Algorithm Design & Analysis

### Divide and Conquer ~~Capitulate~~

$S_1 \dots S_2$

$S_1' \dots S_1'' \dots S_2' \dots S_2''$

$$LCS(S_1, S_2) = LCS(S_1', S_2') + LCS(S_1'', S_2'')$$

*Doesn't work*

*abc: cde*  
*afg: phi*  
 1 0

Lecture 1 David Maier 6

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

*LCS(4,5)*

R	a	b	b	c	d	a
S	a	e	b	b	a	e e

*R[i] ≠ S[j]*

$LCS(i, j) = \max(LCS(i-1, j), LCS(i, j-1))$

$LCS(6, 7) = \max(LCS(5, 7), LCS(6, 6)) = 4$

*3 abb                      4 abba*

*R[i] = S[j]*

$LCS(i, j) = LCS(i-1, j-1) + 1$

$LCS(6, 5) = LCS(5, 4) + 1$

*3 + 1 = 4*

$LCS(0, j) = LCS(i, 0) = 0$

**Recursion**

$L(i, j) =$   
length of LCS of  
 $R[1..i]$   
 $S[1..j]$

Lecture 1
David Maier
7

Algorithm Design & Analysis

R

S

**Recursion 2**

$LCS(i, j)$

**if**  $i=0$  or  $j=0$  **then**  $LCS \leftarrow 0$

**else if**  $R[i] \neq S[j]$  **then**

$LCS \leftarrow \max(LCS(i-1, j), LCS(i, j-1))$

**else**  $\{R[i] = S[j]\}$

$LCS \leftarrow LCS(i-1, j-1) + 1$

↑

↑

**Count comparisons**

*worst case*

$c(LCS(i, j)) = 1 + c(LCS(i-1, j)) + c(LCS(i, j-1))$

*2<sup>n</sup> behavior*

*2<sup>n</sup> twice as many of the subproblems*

Lecture 1
David Maier
8

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

### Dynamic Programming # Comparisons

Have matrix  $LCS(0-m, 0-n)$  and fill it in  $\sim m \cdot n$

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

*LCS[i,j]*

Ri		a	a	b	c	a	b
Sj	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
c	1	0	0	0	1	1	1
a	2	0	1	1	1	2	2
b	3	0	1	2	2	2	3
b	4	0	1	2	2	2	3

*a b b*

Lecture 1 David Maier **9**

Algorithm Design & Analysis

### Algorithm Longest

1. Set  $LCS[i,j] \leftarrow 0$  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**  
              $LCS[i,j]$   
              $\leftarrow \max(LCS[i-1,j], LCS[i,j-1])$   
         **else**  
              $LCS[i,j] \leftarrow LCS[i-1,j-1]+1$
3. **return**  $LCS[m, n]$

Time complexity  $m \cdot n$  comparisons

Lecture 1 David Maier **10**

## Logistics

Course web page is

<http://www.cs.pdx.edu/~maier/cs584>

Expect class email list to be

[cs584@cs.pdx.edu](mailto: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  
CS310

## Study of Algorithms

- Design techniques  
*divide + conquer, dynamic programming*
- Specific algorithms  
*building blocks geometry*
- Proofs of correctness
- Complexity analysis
  - measure *- code size*
  - time it takes, space it consumes  
*as a function of input size*
  - worst case
  - average case

## Study of Problems

- What's the best any algorithm can do?  
*lower bound - usually hard*  
*upper bound - exhibiting an algorithm*
- Relatedness *- use algorithms for one problem to solve another*  
*transitive closure + boolean matrix mult.*
- Intractability of problems  
*There's no efficient algorithm*

## Complexity of an Algorithm

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

$T(n)$  = maximum time required  
*over all inputs of size  $n$*

$S(n)$  = maximum space required  
*over all inputs of size  $n$*

**Input size**  
*# of elements*  
*# of bits*

**Time measures**  
*count instructions*  
*count a particular op*  
*e.g. comparisons*

## Time Complexity of a Problem

Minimum time over all algorithms

to get

upper bound: *exhibit alg.*

lower bound: *ad hoc argument*  
*reduction from another problem*

## Asymptotic Complexity

order notation

$f(n)$  has <sup>order</sup>  $O(g(n))$  if there is a  $c > 0$   
 $f(n) \leq c \cdot g(n)$  for all  $n > n_0$   
 for some  $n_0$

$$f(n) = n^2 + 3n + 4$$

$f(n)$  has  $O(n^2)$

$$f(n) \leq 2 \cdot g(n)$$

for  $n \geq 5$   
 $\uparrow n_0$

## Asymptotic Notation

$O(g(n)) = \{f(n) \mid \text{there are positive } c \text{ and } n_0$   
 such that  $f(n) \leq c \cdot g(n) \text{ when } n > n_0\}$   
 $n^2 + 3n + 4 = O(n^2)$

means

$$n^2 + 3n + 4 \in O(n^2) \quad n^2 + 3n + 4 \in O(n^3)$$

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

$$n^2 + 3n + 4 \in O(n^3)$$

$$\notin \Theta(n^3)$$

## Asymptotic Notation 2

### Note

$$n^2 + 3n + 4 \in O(n^3)$$

$$n^2 + 3n + 4 \notin \Theta(n^3)$$

$$\Theta(n^3) \subseteq O(n^3)$$

*Omega*  $\Omega(g(n)) = \{f(n) \mid \text{there are positive } c \text{ and } n_0 \text{ such that } c \cdot g(n) \leq f(n) \text{ for } n > n_0\}$

### asymptotic

- upper bound  $O$
- tight bound  $\Theta$
- lower bound  $\Omega$

## Asymptotic Notation 3

### Theorem

$$f(n) \in O(g(n)) \text{ and}$$

$$f(n) \in \Omega(g(n))$$

imply

$$f(n) \in \Theta(g(n))$$

### Other facts

$$f(n) \in \Theta(h(n)) \text{ and } h(n) \in \Theta(g(n))$$

$$\text{imply } f(n) \in \Theta(g(n))$$

$$f(n) \in \Theta(g(n))$$

implies

$$g(n) \in \Theta(f(n))$$

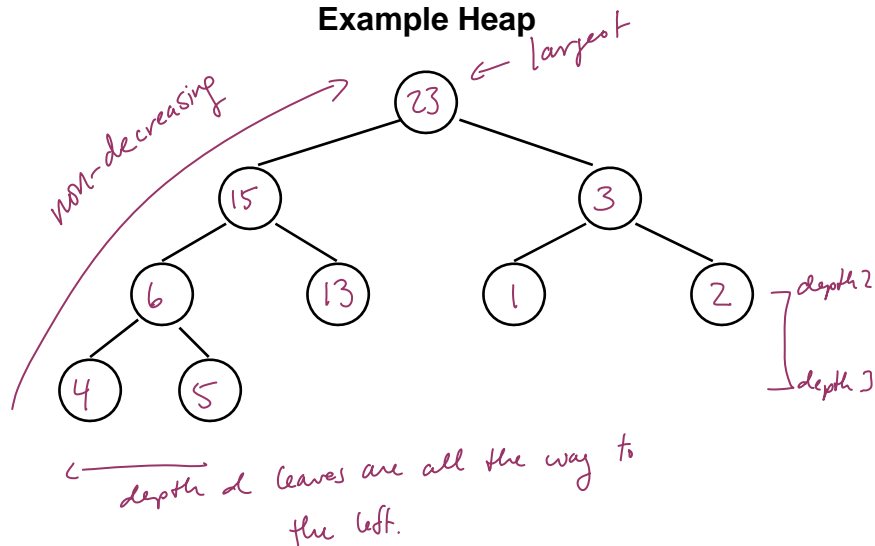
## Heapsort

Heap: binary tree where all leaves at depth  $d-1$  or  $d$

node labels are items to sort

value at node  $\geq$  all values in subtrees

## Example Heap



# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

## Heapsort Algorithm

**Heapsort** *(make heap)*

1. remove *label of root*
2. move *label of some leaf to root*
3. delete *that leaf*
4. remake *heap*

**Implicit data structure**

$A[1..n]$  — *array of items to sort*  
 $A[1]$  is *root*  
 left and right children of  $A[i]$  are at  
 $A[2i]$     $A[2i+1]$

Lecture 1 David Maier **23**

Algorithm Design & Analysis

## Example Structure

$i$	1	2	3	4	5	6	7	8	9
$A[i]$	23	15	3	6	13	1	2	4	5

**Restriction:** *Leaves at depth  $d$  are to left.*

Lecture 1 David Maier **24**

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

### Remove Root Value

Swap  $A[1] \leftrightarrow A[n]$

Pretend array runs  $A[1..n-1]$

~~5~~ 2 3 15 3 6 13 1 2 4 ... 2 3  
~~5~~

Remake heap: *at root*  
 exchange value with  
 the larger of its  
 two children  
 (if one is larger)  
*recursively*

Lecture 1 David Maier 25

Algorithm Design & Analysis

### Formally

Input:  $a_1 a_2 \dots a_n$  to sort

Initialize  $A[i] \leftarrow a_i$

build-max-heap  
 want  $A[i] \geq A[2i], A[2i+1]$   $1 \leq i \leq \frac{n}{2}$   
 for  $i$  between 1 and  $n/2$

Lecture 1 David Maier 26

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

## Heap Construction

Assume:  $A[i+1] \dots A[j]$  have heap property  
Want:  $A[i]$  to be root of heap

```
max-heapify(i, j) uses j
  if i not a leaf and
    a child of i has a greater value
  then
    let k be child with larger value
     $A[i] \leftrightarrow A[k]$ 
    max-heapify(k, j)
build-max-heap n/2
  for i = n downto 1 do
    max-heapify(i, n)
```

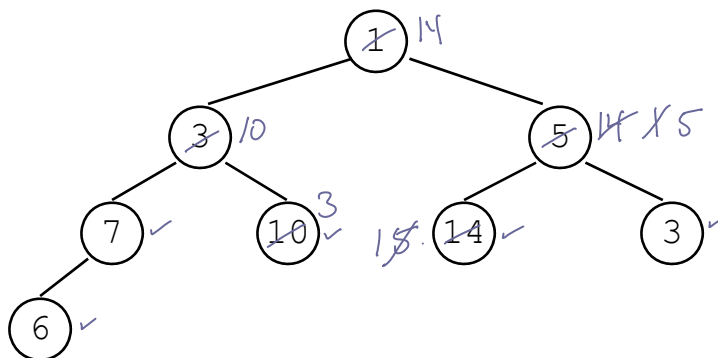
Lecture 1

David Maier

27

Algorithm Design & Analysis

## Example of Build-max-heap



Lecture 1

David Maier

28

Algorithm Design & Analysis

### 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  $h$  *distance above a leaf*

$$T(h) \leq T(h-1) + c$$

$$T(h) \leq h \cdot c \in O(h)$$

Build-max-heap calls max-heapify once per node.  
How many nodes of height  $i$ ? *limit of series*

*at most*  $\frac{n}{2^{i+1}}$

*log n max height*  $c \cdot n \sum_{i=1}^{\log n} \frac{1}{2^{i+1}}$

$\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} \dots = O(n)$

Lecture 1 David Maier **29**

Algorithm Design & Analysis

### Heapsort Algorithm

**heapsort**

$O(n)$  - build-max-heap

```

for i = n to 2 do
    A[1] ↔ A[i]
    max-heapify(1, i-1)
    
```

*pretending the array runs from 1 to i-1*

Sorted list ends up in  $A[1] \dots A[n]$

Time complexity:  $\max\text{-heapify}(1, i) \in O(\lg i)$

$\log n + \log n-1 + \log n-2 + \dots + \log 2 + (\log 1) \in O(n \lg n)$

\* comparisons is  $O(n \lg n)$

Lecture 1 David Maier **30**

Algorithm Design & Analysis

### Heapsort Example

Lecture 1 David Maier **31**

Algorithm Design & Analysis

### Lower Bounds on Sorting

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

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

Prm: Any decision tree that sort  $n$  distinct elements has height of least  $\lg n!$   
 At least  $n!$  leaves

Lecture 1 David Maier **32**

## Consequence

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

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

$$\begin{aligned} \lg n! &\geq \lg (n/2)^{n/2} = \frac{n}{2} \lg \frac{n}{2} = \frac{n}{2} (\lg n - \lg 2) \\ &\geq \frac{1}{4} n \lg n \quad \text{if } n \geq 4 \end{aligned}$$

$n \log n \geq \log n!$  since

$$n^n \geq n!$$

## Dynamic Programming

Two elements

Bellman's principle

- Correctness
- Optimal substructure: solution contains optimal solutions to subproblems  
 $LCS(7,7)$  (as sequence) contains  $LCS(6,6)$  if  $a_7 = b_7$
- Utility
- Overlapping subproblems: if done recursively  
 $LCS(6,5)$  used for  $LCS(7,5)$ ,  
 $LCS(6,6)$   
 $LCS(7,6)$

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

### Matrix Chain Product

Multiplying an  $n \times p$  matrix and a  $p \times m$  matrix takes  $n \cdot p \cdot m$  scalar mults.

Lecture 1 David Maier **35**

Algorithm Design & Analysis

### Recursive Approach

Have matrices  $M_1 * M_2 * \dots * M_n$

Let  $Mult[i, j]$  be cost to multiply  $M_i * \dots * M_j$

Let  $p_0, p_1, \dots, p_n$  be dimensions such that  $M_i$  has dimension  $p_{i-1} \times p_i$

Assume  $Mult[i, i] = 0$

Then  $Mult[i, j] =$

$$\min_{i \leq k < j} ( \underbrace{Mult[i, k]}_{p_{i-1} \times p_k} + \underbrace{Mult[k+1, j]}_{p_k \times p_j} + p_{i-1} p_k p_j )$$

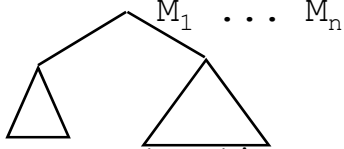
Lecture 1 David Maier **36**

# CS 410/584, Algorithm Design & Analysis: Lecture 1

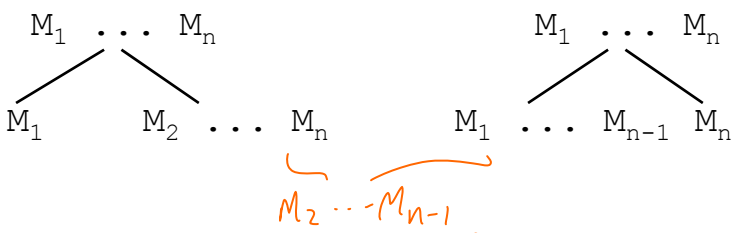
Algorithm Design & Analysis

### Has Necessary Properties

- optimal substructure



- overlapping subproblems



How many subproblems?  $\sim \frac{n^2}{2}$

Lecture 1 David Maier **37**

Algorithm Design & Analysis

### Build a Table

$P_0$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$
50	5	60	10	100	100

	$j$	5	4	3	2	1	<i>in 1000's</i>
$i$	1	83	33	5.5	15	0	
	2	58	8	3	0		
	3	160	60	0			
	4	100	0				
	5	0					

In 1000's of mults.

Lecture 1 David Maier **38**

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

### Mult[i,j] Values

<u>1,3</u>	<sup>0</sup> <u>1,1</u>	<sup>3</sup> + <u>2,3</u>	<sup>25</sup> + <u>50*5*10</u>	<sup>5.5</sup>
	1,2	+ 3,3	+ 50*60*10	45
	<sup>15</sup>	<sup>0</sup>	<sup>30</sup>	
2,4	<sup>0</sup> <u>2,2</u>	+ <sup>60</sup> <u>3,4</u>	+ 5* <sup>30</sup> 60*100	90
	2,3	+ 4,4	+ 5*10*100	8 ←
	<sup>3</sup>	<sup>0</sup>	<sup>5</sup>	
3,5	<sup>0</sup> <u>3,3</u>	+ <sup>100</sup> <u>4,5</u>	+ 60* <sup>60</sup> 10*100	160 ←
	<u>3,4</u>	+ <u>5,5</u>	+ 60*100*100	660
	<sup>60</sup>	<sup>0</sup>	<sup>600</sup>	

Lecture 1 David Maier **39**

Algorithm Design & Analysis

### Mult[i,j] Values 2

1,4	<sup>0</sup> <u>1,1</u>	<sup>8</sup> + <u>2,4</u>	<sup>25</sup> + 50* <sup>5</sup> *100	<sup>33</sup> ✓
	1,2	+ 3,4	+ 50* <sup>300</sup> 60*100	
	1,3	+ 4,4	+ 50* <sup>50</sup> 10*100	
2,5	<sup>0</sup> <u>2,2</u>	+ <sup>160</sup> <u>3,5</u>	+ 5* <sup>30</sup> 60*100	
	<sup>3</sup> <u>2,3</u>	+ <sup>100</sup> <u>4,5</u>	+ 5* <sup>5</sup> 10*100	
	<sup>8</sup> <u>2,4</u>	+ <u>5,5</u>	+ 5* <sup>50</sup> 100*100	<sup>58</sup> ✓

Lecture 1 David Maier **40**

# CS 410/584, Algorithm Design & Analysis: Lecture 1

Algorithm Design & Analysis

## Mult[i,j] Values 3

$$\begin{array}{l} 1,5 \\ 1,5 \end{array} \quad \begin{array}{l} 0 \\ 1,1 \end{array} + \begin{array}{l} 58 \\ 2,5 \end{array} + 50 * \begin{array}{l} 25 \\ 5 \end{array} * 100 = 83 \checkmark$$
$$\begin{array}{l} 1,5 \\ 1,2 \end{array} \quad \begin{array}{l} 1,5 \\ 1,5 \end{array} + \begin{array}{l} 1,60 \\ 3,5 \end{array} + 50 * \begin{array}{l} 300 \\ 60 \end{array} * 100$$
$$\begin{array}{l} 1,5 \\ 1,3 \end{array} \quad \begin{array}{l} 5,5 \\ 1,3 \end{array} + \begin{array}{l} 1,100 \\ 4,5 \end{array} + 50 * \begin{array}{l} 10 \\ 50 \end{array} * 100$$
$$\begin{array}{l} 1,5 \\ 1,4 \end{array} \quad \begin{array}{l} 3,3 \\ 1,4 \end{array} + \begin{array}{l} 0 \\ 5,5 \end{array} + 50 * \begin{array}{l} 500 \\ 100 \end{array} * 100$$

Complexity

$$O(n^3)$$

Lecture 1

David Maier

41