Algorithm Design & Analysis

David Maier
Portland State University

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

Let's Solve a Problem

Longest Common Subsequence (LCS)

$S = a_1 a_2 \ . \ . \ . \ a_n$

A <u>subsequence</u> is

$a_{i_1} a_{i_2} \cdots a_{i_k} \qquad i_1 < i_2 < \cdots < i_k$

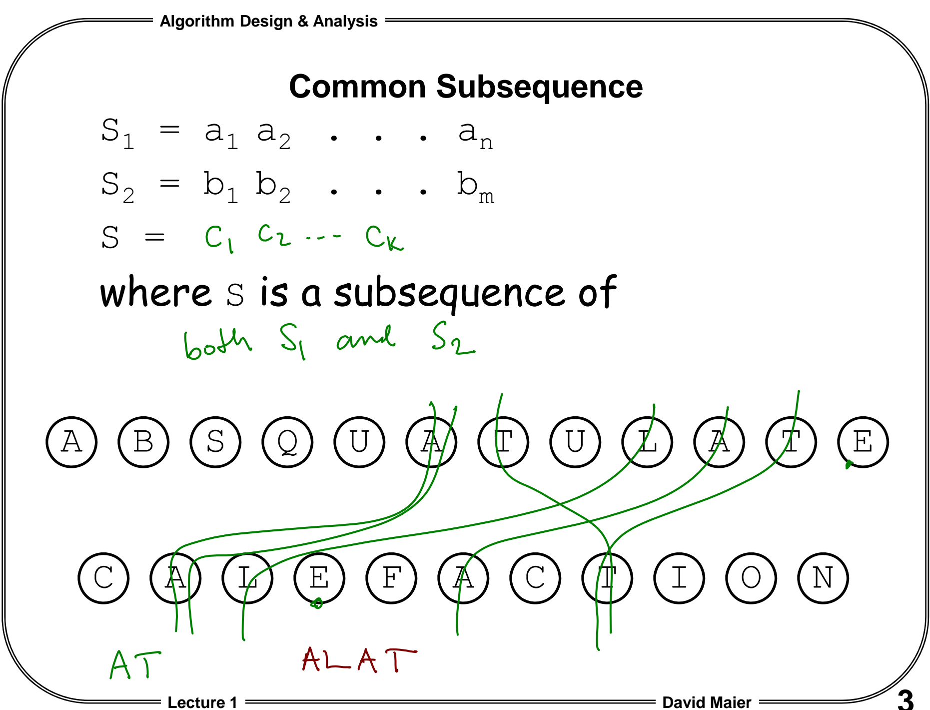
A B S Q U A T U L A T E

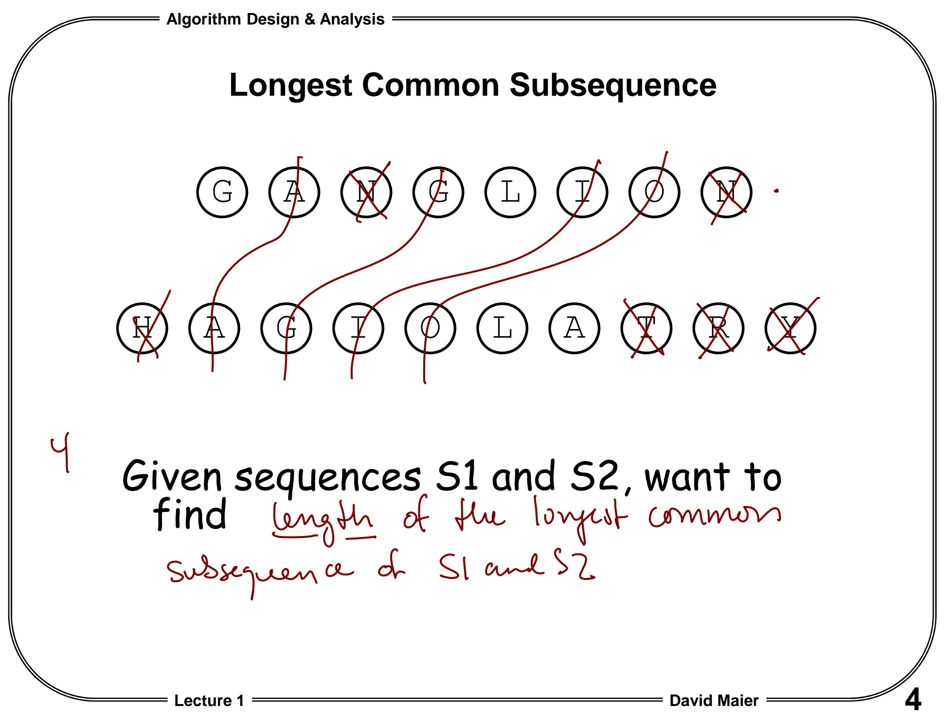
AQTUL

LATE

ε empty string

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





© 1994, 2003, 2006, 2008, 2009

David Maier

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 to shortest? quit at first one

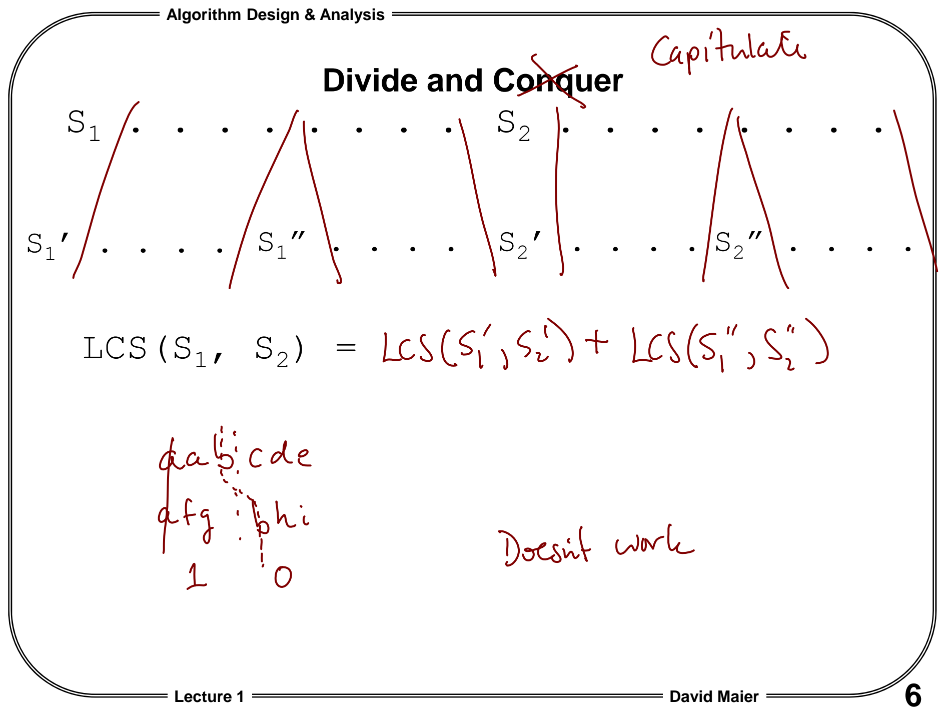
prune out letters in just one sequence

checks if R1 subseq of S2

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

time complexity = $2^n \cdot m$

comparisons of letters



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

Recursion

LCS(4,5)

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

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

```
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 2

R

S

```
LCS(i,j)
   if i=0 or j=0 then LCS ← 0
   else if R[i] ≠ S[j] then
      LCS ← max(LCS(i-1,j),LCS(i,j-1))
   else  (R[i] = S[j])
      LCS ← LCS(i-1,j-1) +1
```

Count comparisons

worst case

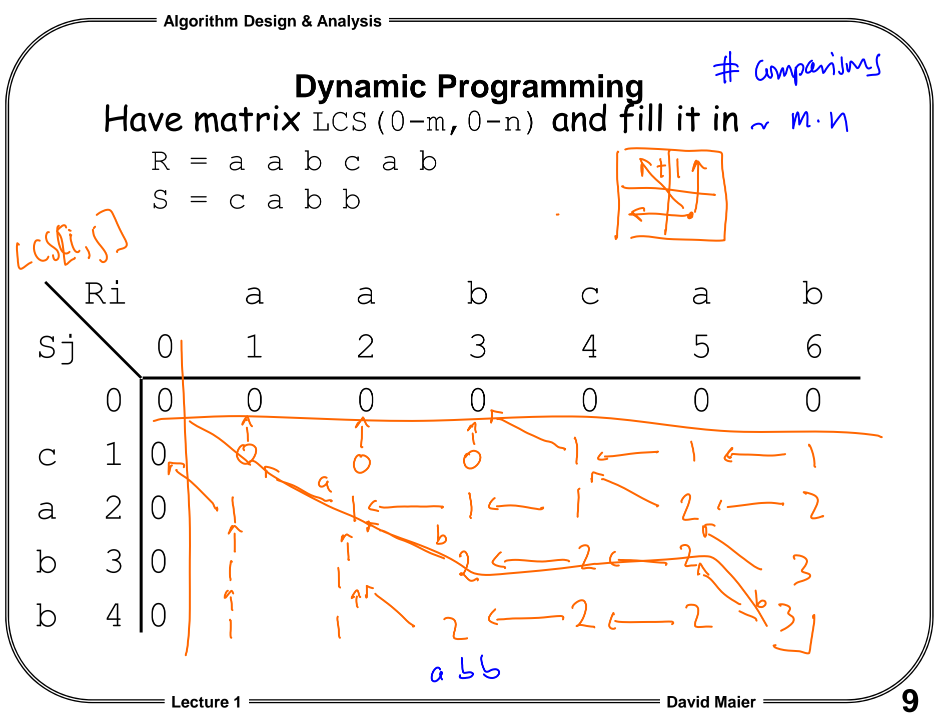
2 twice as many of the subproblem

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

2^n behavior

© 1994, 2003, 2006, 2008, 2009

David Maier



Algorithm Longest

```
1. Set LCS[i,j] ← 0 if i=0 or j=0
2. for j = 1 to n do
     for i = 1 to m do
       if R[i] ≠ S[j] then
       LCS[i,j]
         ← max(LCS[i-1,j],LCS[i,j-1])
       else
         LCS[i,j] ← LCS[i-1,j-1]+1
3. return LCS[m , n ]
```

Time complexity m.n comparisons

© 1994, 2003, 2006, 2008, 2009

David Maier

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 CS310

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

Study of Algorithms

- Design techniques

  divide + conquer, dynamic programming
- Specific algorithms

  building blocks geometry
- Proofs of correctness
- Complexity analysis
  - measure — codesize
  - 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

© 1994, 2003, 2006, 2008, 2009

David Maier

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

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

Asymptotic Complexity

order notation

f(n) has order O(g(n)) if there is a c > 0

f(n) ≤ c · 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) ≤ 2 · g(n) for n ≥ 5

c ← (the 2); n_0 ← (the 5)

Asymptotic Notation

O(g(n)) = { f(n) there are positive c and n_0 such that f(n) ≤ c · g(n) when n > n_0 }

n^2 + 3n + 4 = O(n^2)

means

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

Theta Θ(g(n)) = {f(n) | there are positive c_1, c_2 and n_0 such that $c_1 g(n) \le f(n) \le c_2 g(n)$, $n > n_0$}

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

$\notin \Theta(n^3)$

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

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

asymptotic

- upper bound O
- tight bound Θ
- lower bound Ω

Asymptotic Notation 3

Theorem

$f(n) \in O(g(n))$ and
$f(n) \in \Omega(g(n))$
imply
$f(n) \in \Theta(g(n))$

Other facts

$f(n) \in \Theta(h(n))$ and $h(n) \in \Theta(g(n))$
imply $f(n) \in \Theta(g(n))$

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

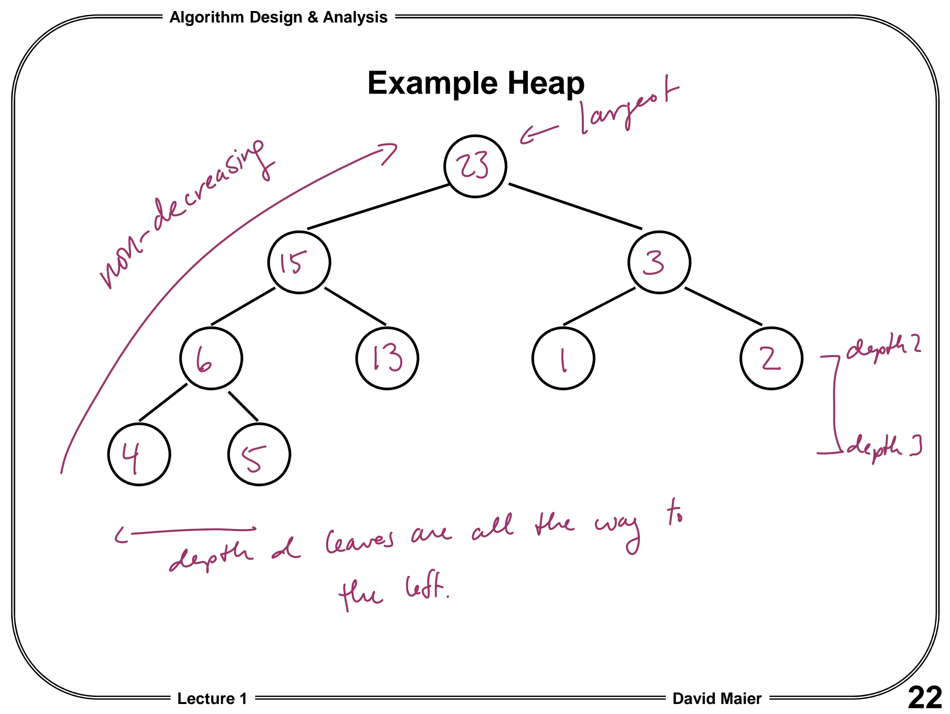
© 1994, 2003, 2006, 2008, 2009
David Maier

Heapsort

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

node labels are items to sort

value at node ≥ all values in subtrees



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

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

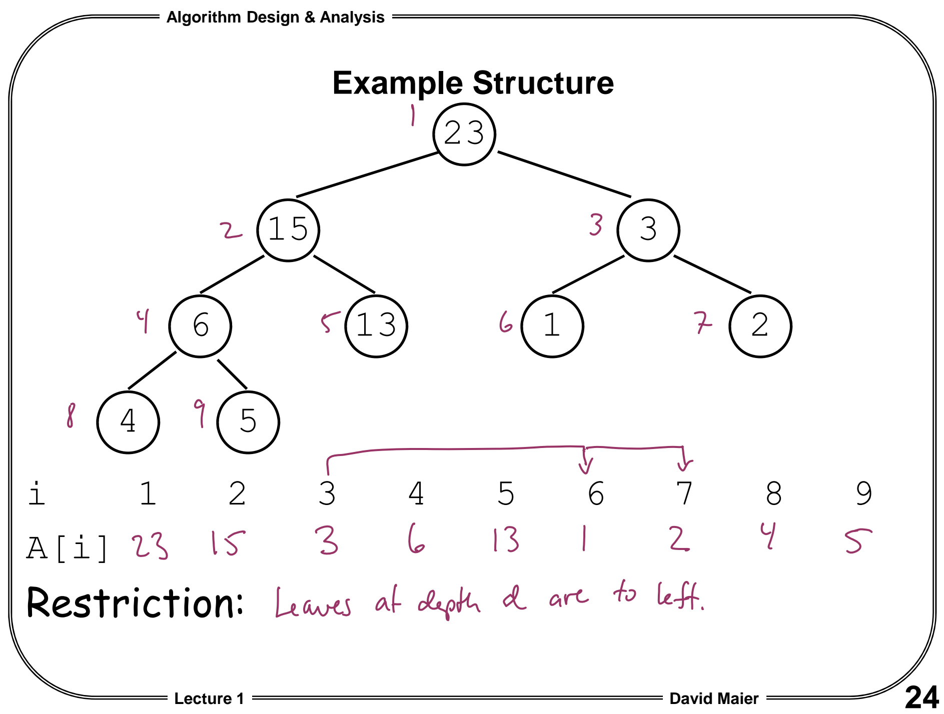
Implicit data structure

A[1..n] — array of items to sort

A[1] is root

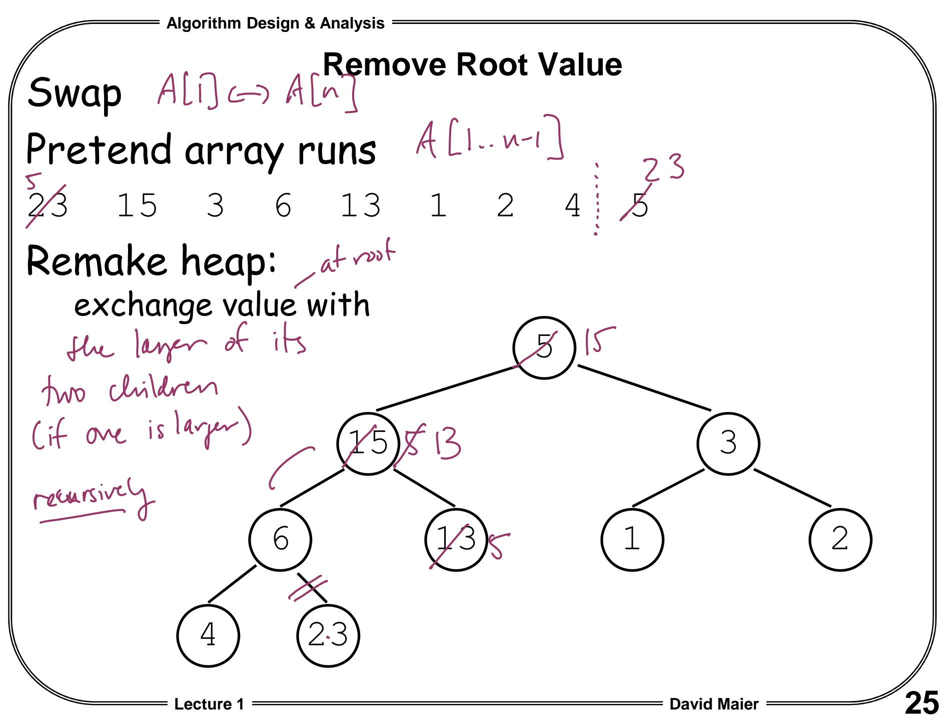
left and right children of A[i] are at

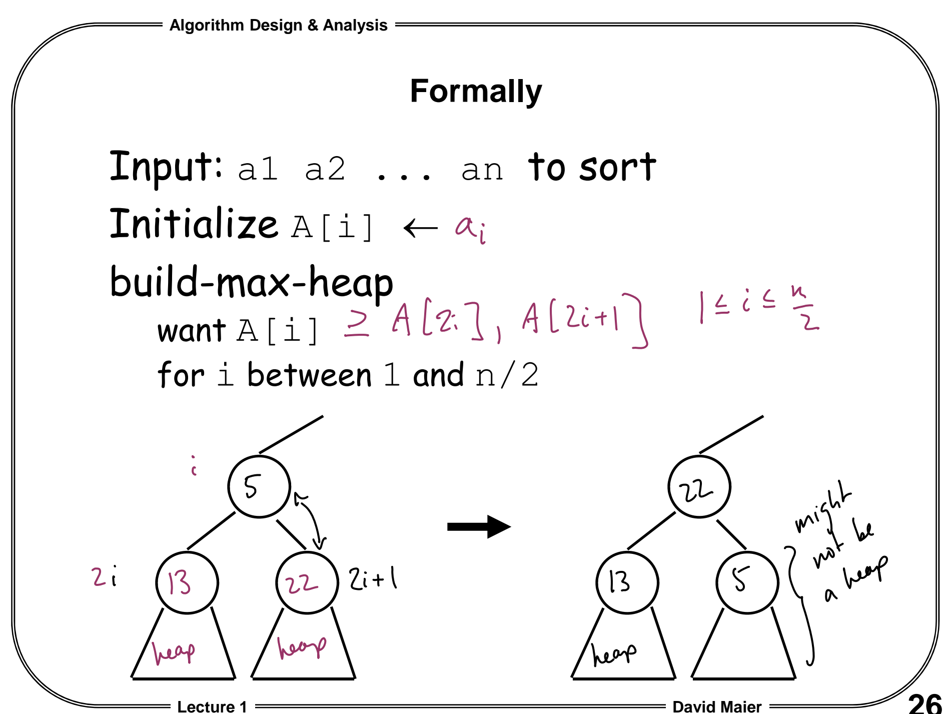
A[2i] A[2i+1]



© 1994, 2003, 2006, 2008, 2009

David Maier





© 1994, 2003, 2006, 2008, 2009

David Maier

Heap Construction

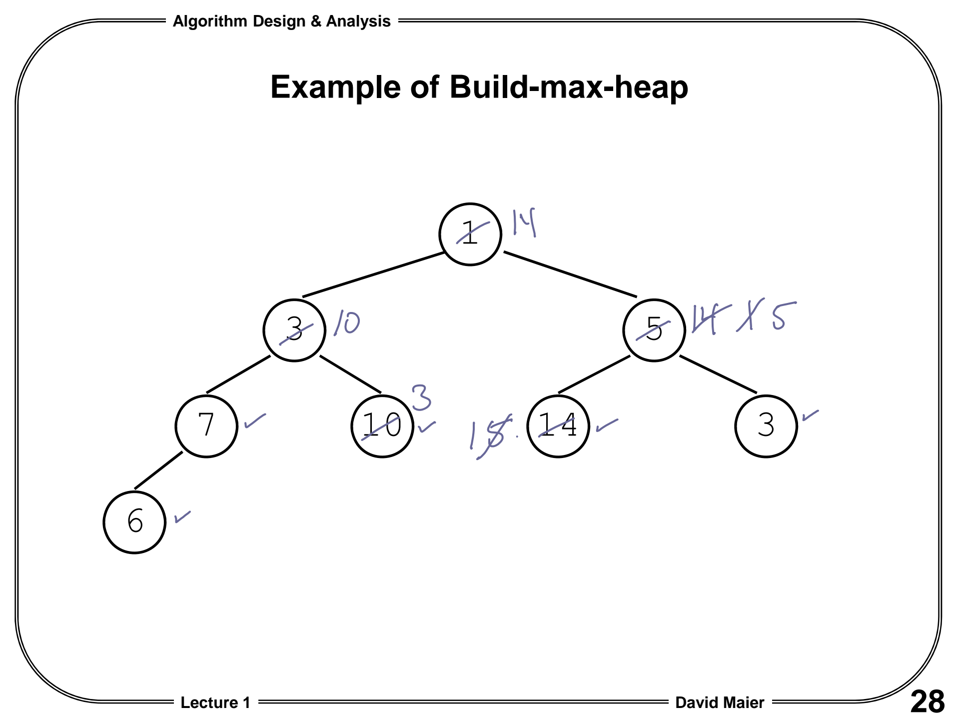
Assume: A[i+1] ... A[j] have heap property

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] <-> A[k]
      max-heapify(k,j)

build-max-heap
   for i = n/2 downto 1 do
      max-heapify(i,n)
```


Example of Build-max-heap



© 1994, 2003, 2006, 2008, 2009

David Maier

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)` ≤ $T(h-1) + c$

`T(h)` $\le h \cdot c \in O(h)$

Build-max-heap calls max-heapify once per node.

How many nodes of height `i`?

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

(log n = max height)

$cn \sum_{i=1}^{\log n} i \cdot \frac{1}{2^{i+1}}$ — $\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} \ldots$ *(limit of series)* $c \cdot n \cdot k = O(n)$

Heapsort Algorithm

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

O(n) - build-max-heap

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] ... A[n]`

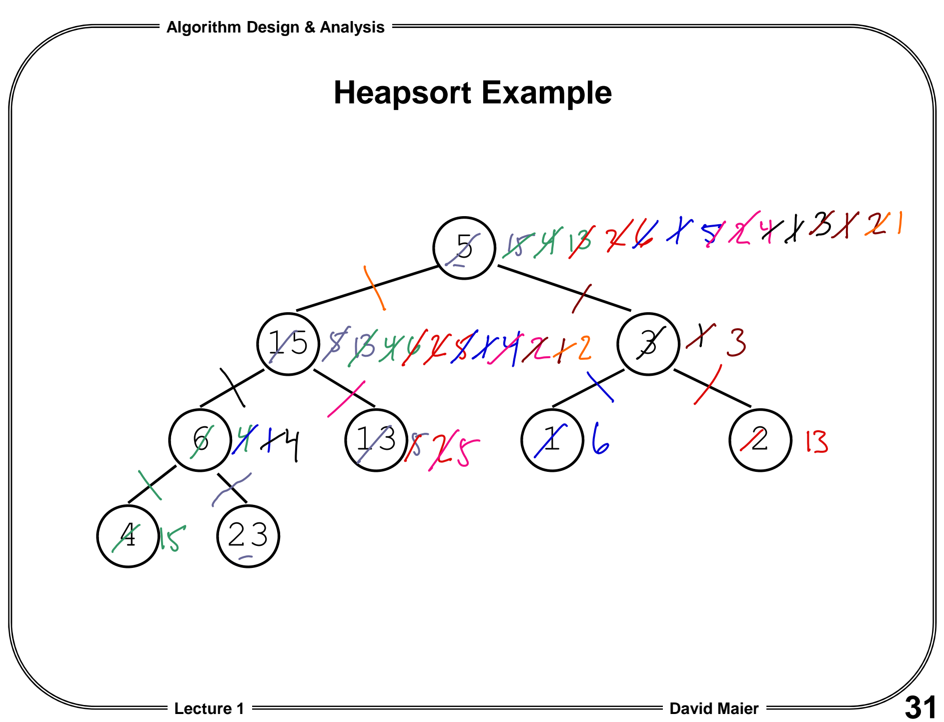
Time complexity: *max-heapify (1, i)* $O(\lg i)$

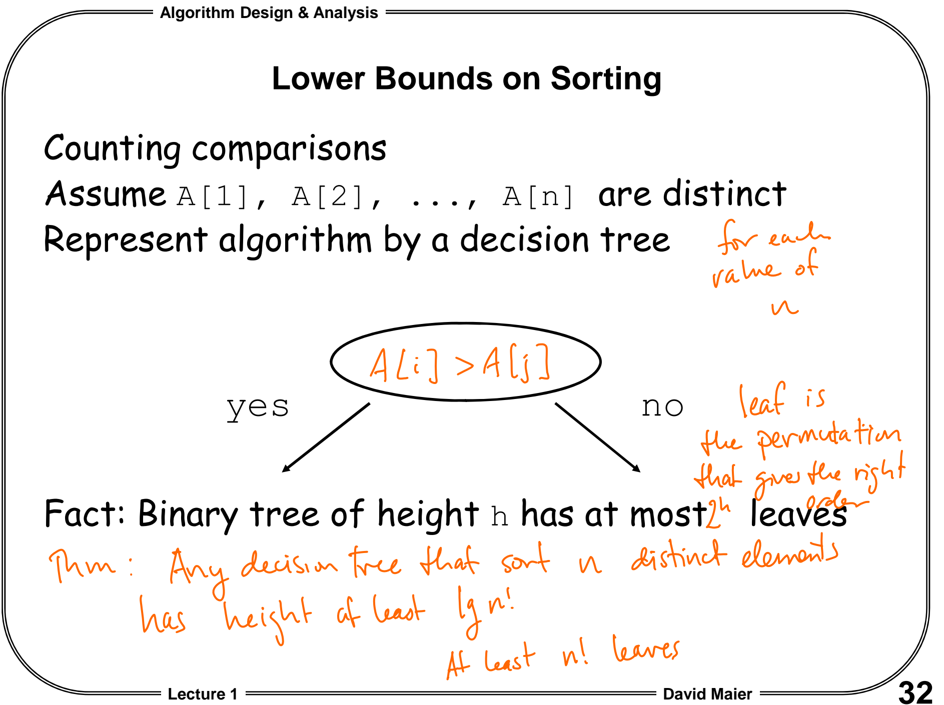
$\log n + \log n-1 + \log n-2 + \cdots + \log 2 + \log 1 \in O(n \lg n)$

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

© 1994, 2003, 2006, 2008, 2009

David Maier





© 1994, 2003, 2006, 2008, 2009

David Maier

Consequence

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

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

$$\lg n! \geq \lg (n/2)^{n/2} = \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - \log 2)$$

$$\geq \frac{1}{4} n \lg n \quad \text{if } n \geq 4$$

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

$$n^n \geq n!$$

Dynamic Programming

Two elements Bellman's principle

Correctness
- Optimal substructure: solution contains optimal solutions to subproblems

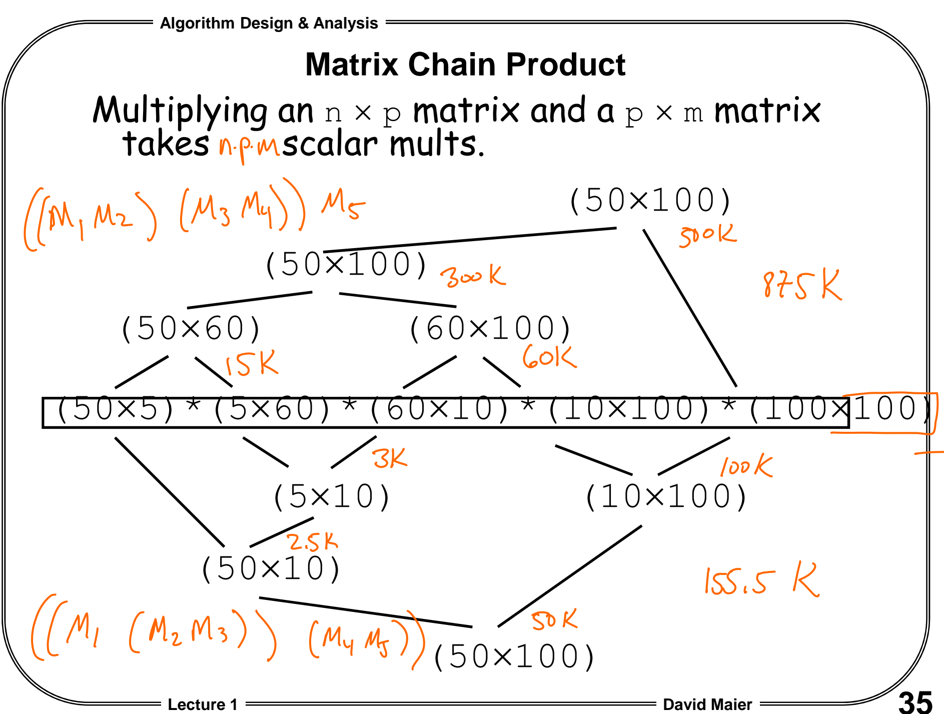
  LCS(7,7) (as sequence) contains LCS(6,6) if $a_7 = b_7$

Utility
- Overlapping subproblems: if done recursively

  LCS(6,5) used for LCS(7,5), LCS(6,6), LCS(7,6)

© 1994, 2003, 2006, 2008, 2009

David Maier



Algorithm Design & Analysis

Recursive Approach

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

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

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

Assume Mult[i,i] = 0

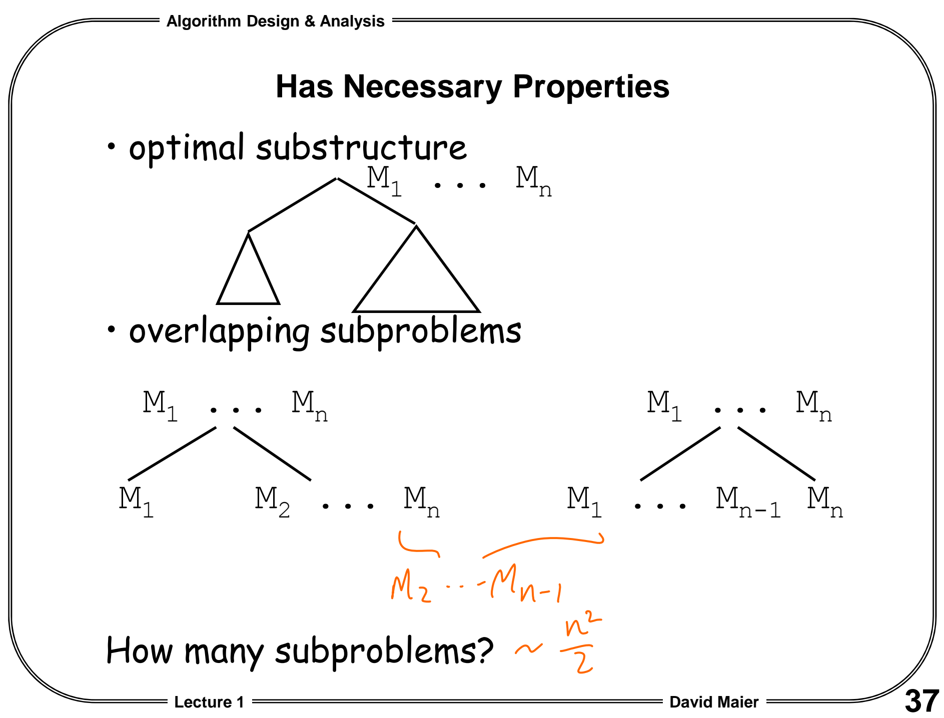
Then Mult[i,j] =

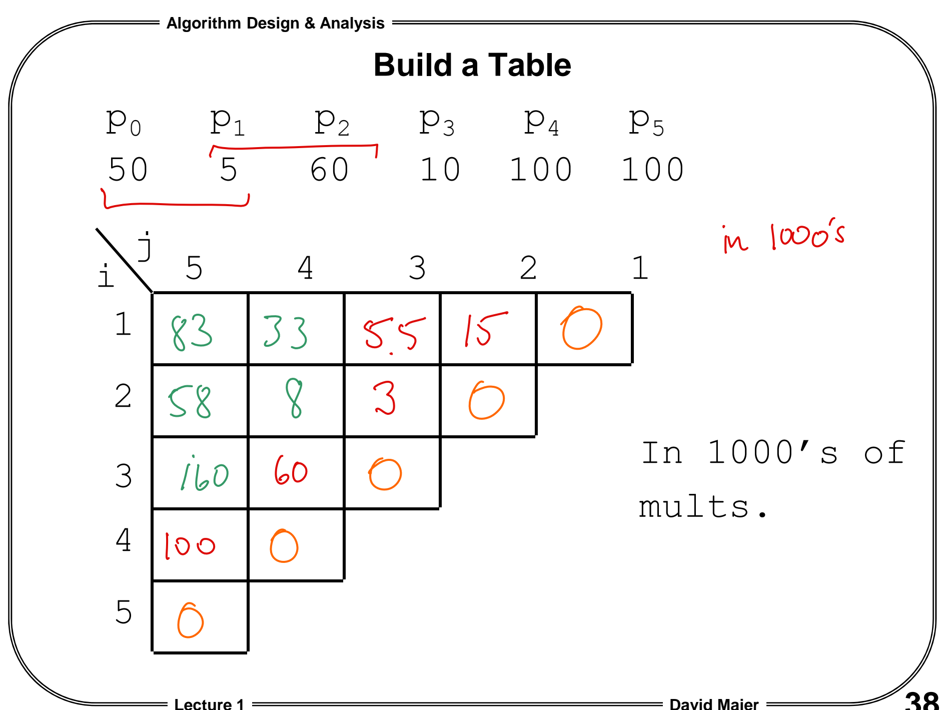
$$\min_{i \le k < j} (\text{Mult}[\underbrace{i, k}_{p_{i-1} \times p_k}] + \text{Mult}[\underbrace{k+1, j}_{p_k \times p_j}] + p_{i-1} p_k p_j)$$

Lecture 1 David Maier 36

© 1994, 2003, 2006, 2008, 2009

David Maier





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

Mult[i,j] Values

```
1,3     1,1 + 2,3 + 50*5*10
        1,2 + 3,3 + 50*60*10

2,4     2,2 + 3,4 + 5*60*100
        2,3 + 4,4 + 5*10*100

3,5
```


Mult[i,j] Values 2

```
1,4     1,1 + 2,4 + 50*   *100
        1,2 + 3,4 + 50*   *100
        1,3 + 4,4 + 50*   *100

2,5     2,2 + 3,5 + 5*   *100
        2,3 + 4,5 + 5*   *100
        2,4 + 5,5 + 5*   *100
```


© 1994, 2003, 2006, 2008, 2009

David Maier

Algorithm Design & Analysis

Mult[i,j] Values 3

```
1,5   1,1 + 2,5 + 50* 5 *100
      1,2 + 3,5 + 50* 60 *100
      1,3 + 4,5 + 50* 10 *100
      1,4 + 5,5 + 50* 100 *100
```

Handwritten annotations: 0, 58, 25 (= 83 ✓); 1.5, 160, 300; 5.5, 100, 50; 33, 0, 500

Complexity

$O(n^3)$

Lecture 1 — David Maier — 41

© 1994, 2003, 2006, 2008, 2009

David Maier