Algorithm Design & Analysis

David Maier
Portland State University

Let's Solve a Problem

Longest Common Subsequence (LCS)

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

A <u>subsequence</u> is

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

Common Subsequence

$S_1 = a_1\ a_2\ .\ .\ .\ a_n$

$S_2 = b_1\ b_2\ .\ .\ .\ b_m$

S =

where S is a subsequence of

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

C A L E F A C T I O N

Longest Common Subsequence

G A N G L I O N

H A G I O L A T R Y

Given sequences S1 and S2, 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

S_1 . . . . S_2 . . . .

S_1' . . . . S_1'' . . . . S_2' . . . . S_2'' . . . .

$LCS(S_1, S_2) =$

Recursion

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

R[i] ≠ S[j]
   LCS(i,j) = max(LCS(      ),LCS(     ))
   LCS(6,7) =

R[i] = S[j]
   LCS(i,j) = LCS(     ) + 1)
   LCS(6,5) =
```


Recursion 2

```
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
    LCS ← LCS(i-1,j-1)
```

Count comparisons

c(LCS(i,j)) =

Dynamic Programming

Have matrix `LCS(0-m,0-n)` and fill it in

```
R = a a b c a b
S = c a b b
```

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

Algorithm Design & Analysis

Algortithm <u>Longest</u>

```
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[      ]
```

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 CS310

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)` = maximum time required

`S(n)` = maximum space required

Input size

Time measures

Time Complexity of a <u>Problem</u>

Minimum time over <u>all</u> algorithms

to get

upper bound:

lower bound:

Asymptotic Complexity

order notation

$f(n)$ has $O(g(n))$ if

$f(n) \le \quad g(n)$

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

$f(n)$ has

$f(n)$

Asymptotic Notation

$O(g(n)) = \{$ there are positive c and n_0 such that $\}$

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

means

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

$n^2 + 3n + 4$

Asymptotic Notation 2

Note

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

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

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

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

asymptotic

- upper bound
- tight bound
- lower bound

Asymptotic Notation 3

Theorem

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

Other facts

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

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

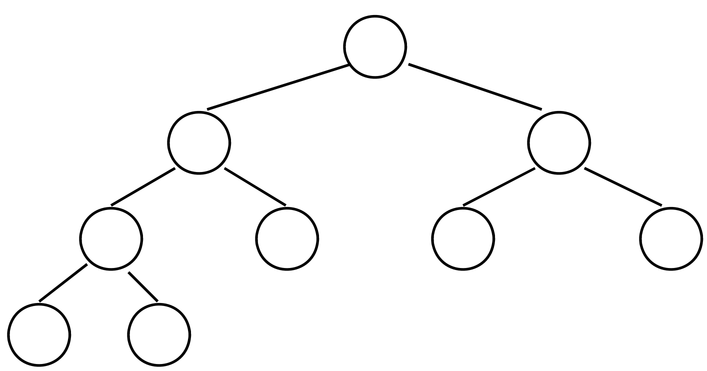
Heapsort

Heap: binary tree where all leaves at depth

node labels are

value at node all values in subtrees

Example Heap



Algorithm Design & Analysis

Heapsort Algorithm

Heapsort

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

Implicit data structure

A[1..n]

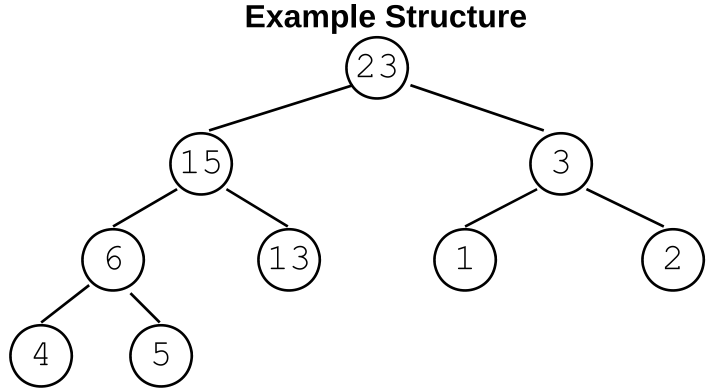
A[1] is

left and right children of A[i] are

Lecture 1 David Maier 23

Algorithm Design & Analysis

Example Structure



i	1	2	3	4	5	6	7	8	9
A[i]									

Restriction:

Lecture 1 David Maier 24

Remove Root Value

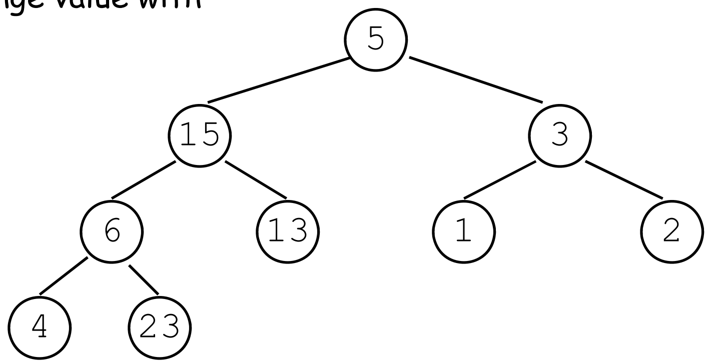
Swap

Pretend array runs

`23 15 3 6 13 1 2 4 5`

Remake heap:

exchange value with



Formally

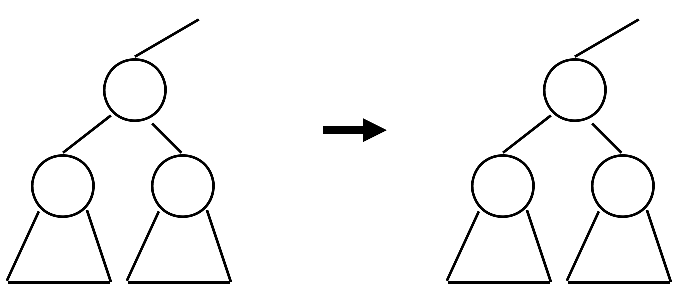
Input: `a1 a2 ... an` to sort

Initialize `A[i]` ←

build-max-heap

want `A[i]`

for `i` between `1` and `n/2`



Heap Construction

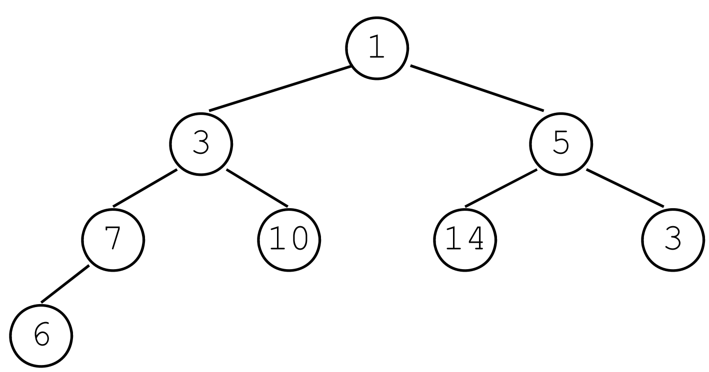
Assume:

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)` ≤

`T(h)`

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

How many nodes of height `i`?

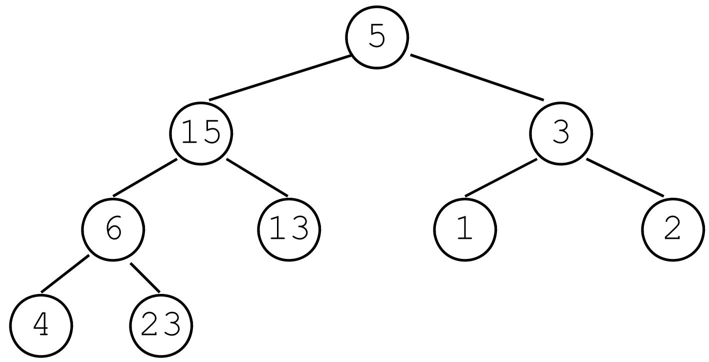
Heapsort Algorithm

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

Sorted list ends up in `A[1] ... A[n]`

Time complexity:

Heapsort Example

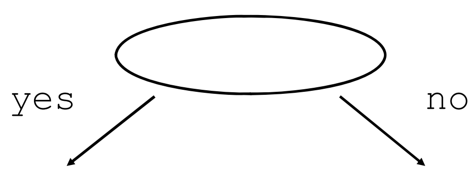


Lower Bounds on Sorting

Counting comparisons

Assume `A[1], A[2], ..., A[n]` are distinct

Represent algorithm by a decision tree



Fact: Binary tree of height `h` has at most   leaves

Consequence

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

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

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

Dynamic Programming

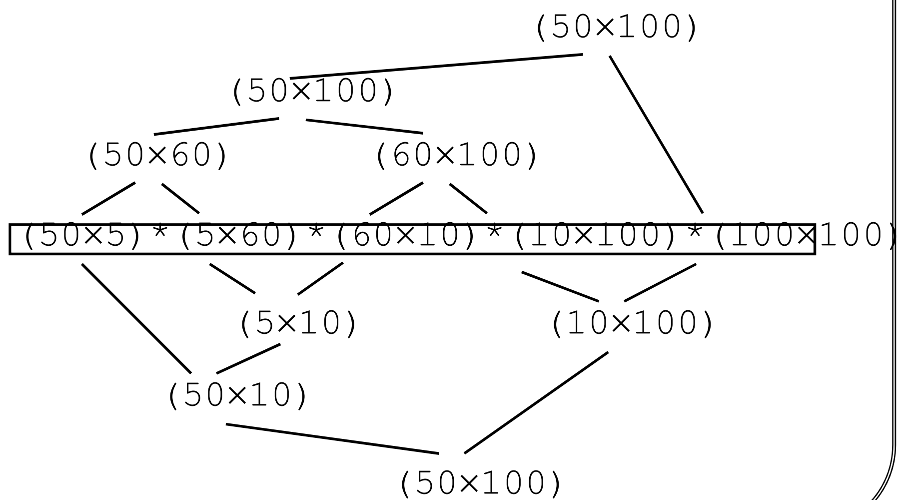
Two elements

- Optimal substructure: solution contains optimal solutions to subproblems

- Overlapping subproblems: if done recursively

Matrix Chain Product

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



Recursive Approach

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

Let `Mult[i,j]` be cost to multiply

Let $p_0, p_1, ..., p_n$ be dimensions such that M_i has dimension

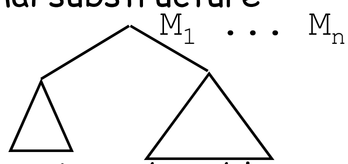
Assume `Mult[i,i] =`

Then `Mult[i,j] =`

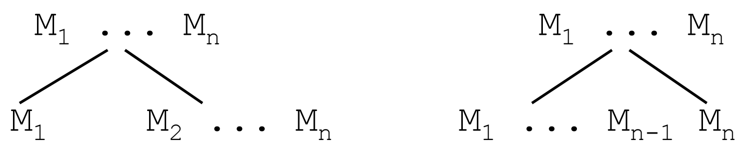
`min (Mult[ , ]+Mult[ , ]+ )`

Has Necessary Properties

- optimal substructure



- overlapping subproblems



How many subproblems?

Algorithm Design & Analysis

Build a Table

p_0	p_1	p_2	p_3	p_4	p_5
50	5	60	10	100	100

i \ j	5	4	3	2	1
1					
2					
3					
4					
5					

In 1000's of mults.

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
```

Mult[i,j] Values 3

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

Complexity