

# 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,  
2011 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 B S Q U A T U L A T E

Lecture 1

David Maier

2

1

© 1994, 2003, 2006, 2008, 2009,  
2011 David Maier

Algorithm Design & Analysis

**Common Subsequence**

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

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

Lecture 1 David Maier 3

Algorithm Design & Analysis

**Longest Common Subsequence**

G A N G L I O N  
H A G I O L A T R Y

Given sequences  $S_1$  and  $S_2$ , want to find

Lecture 1 David Maier 4

## 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

$S_1 \dots S_2 \dots$

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

$LCS(S_1, S_2) =$

## 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) + 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

	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					
a	2	0					
b	3	0					
b	4	0					

## 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**  $LCS[ \quad ]$

Time complexity

## 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  
CS350

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

1. Pick some element  $e \in S$ .

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

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

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

## 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

$T(n)$  = maximum time required

$S(n)$  = 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  $O(g(n))$  if

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

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

$f(n)$  has

$f(n)$

## Asymptotic Notation

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

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

means

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

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

## Asymptotic Notation 2

Note

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

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

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

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

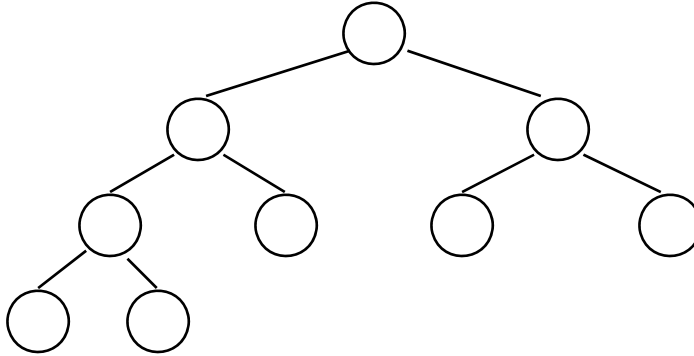
## Heapsort

Heap: binary tree where all leaves at  
depth

node labels are

value at node    all values in subtrees

## Example Heap



## 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

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

Algorithm Design & Analysis

### Example Structure

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

**Restriction:**

Lecture 1 David Maier **25**

Algorithm Design & Analysis

### Remove Root Value

**Swap**  
Pretend array runs  
23 15 3 6 13 1 2 4 5

**Remake heap:**  
exchange value with

Lecture 1 David Maier **26**

## Formally

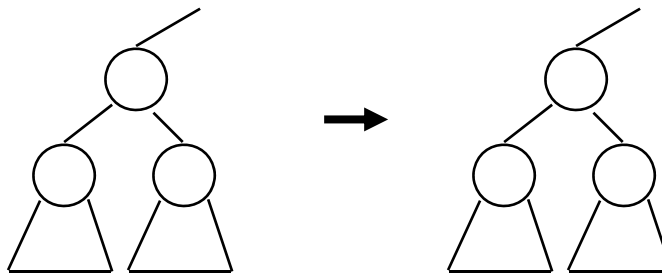
Input:  $a_1 a_2 \dots a_n$  to sort

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

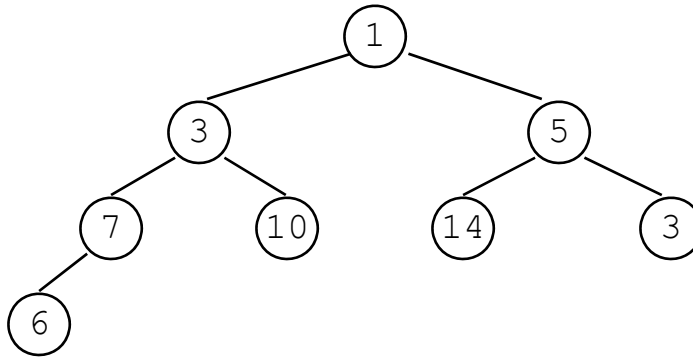
**then**

    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  $i$ ?

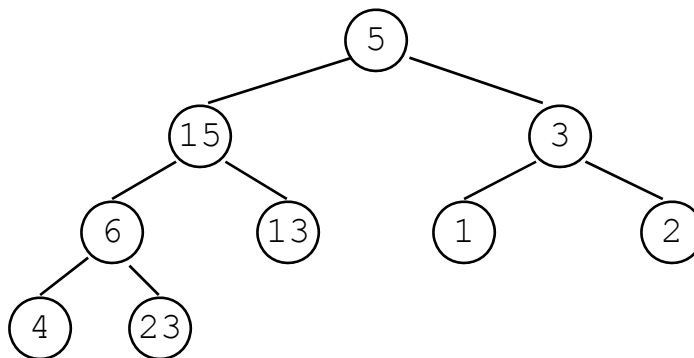
## Heapsort Algorithm

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

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

Time complexity:

## Heapsort Example

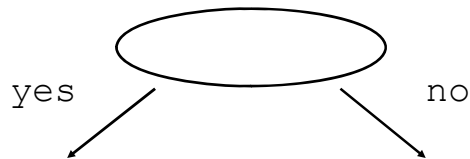


## Lower Bounds on Sorting

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

## Consequence

Corollary: Any algorithm for sorting by comparisons needs  $\Omega(n \log n)$  compares for some  $c > 0$  and large  $n$ .

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

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