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) + 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						

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 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 $(e\_low, f\_low) \leftarrow$ Freq(S_low)
   Set $(e\_high, f\_high) \leftarrow$ 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 <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)` $\leq$ `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) | there are positive c_1, c_2
and n_0 such that

}

$n^2 + 3n + 4$

Asymptotic Notation 2

Note

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

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

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

$\Omega(g(n))$ = {f(n) | 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

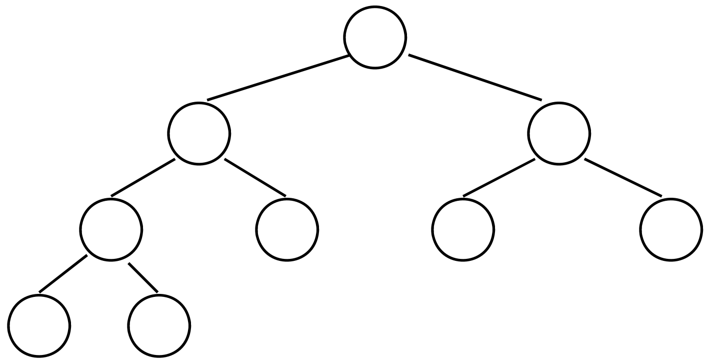
Heapsort

Heap: binary tree where all leaves at depth

node labels are

value at node   all values in subtrees

Example Heap



Heapsort Algorithm

Heapsort

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

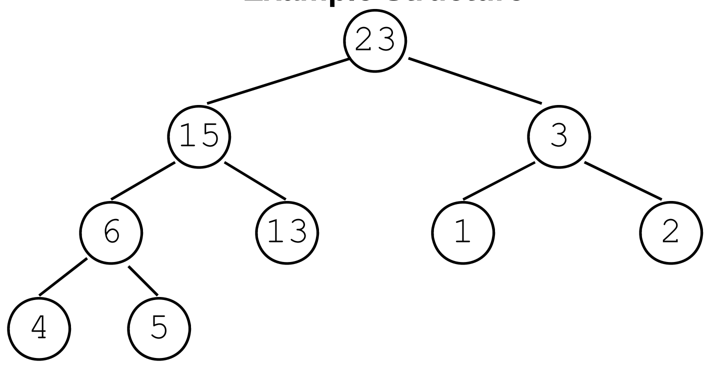
Implicit data structure

`A[1..n]`

`A[1]` is

left and right children of `A[i]` are

Example Structure



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

Restriction:

Remove Root Value

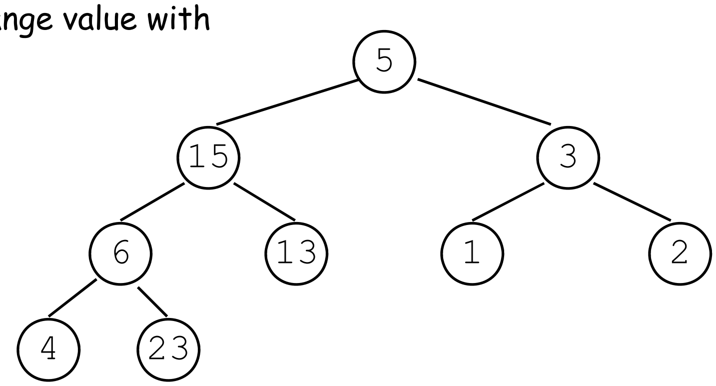
Swap

Pretend array runs

23 15 3 6 13 1 2 4 5

Remake heap:

exchange value with



Formally

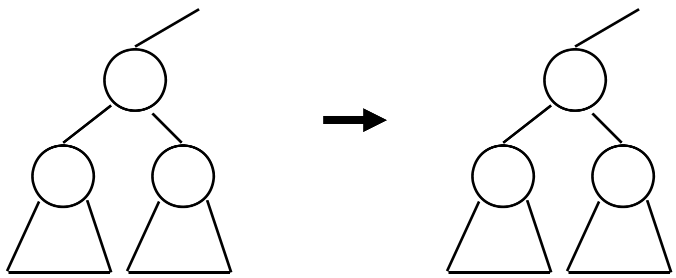
Input: `a1 a2 ... an` to sort

Initialize `A[i]` ←

build-max-heap

want `A[i]`

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



Heap Construction

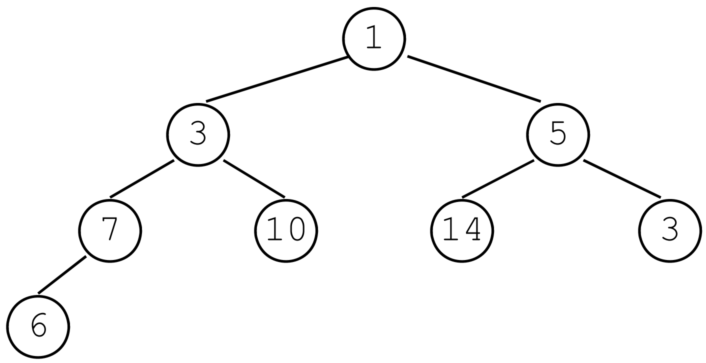
Assume:

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?

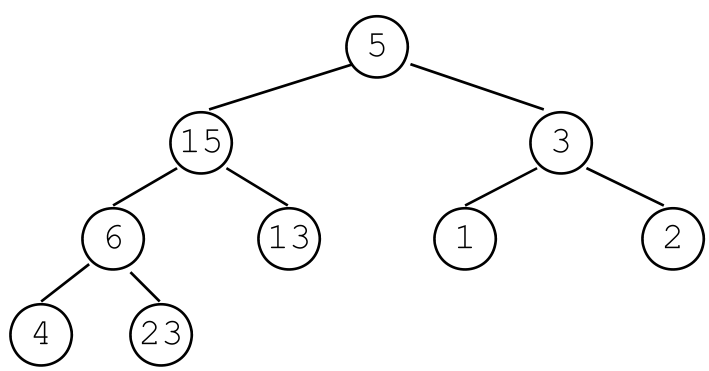
Heapsort Algorithm

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

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

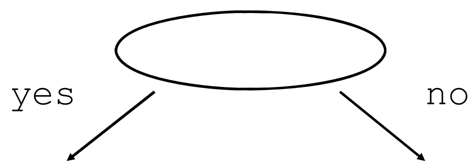
Time complexity:

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