CSE 201 3-3-20

Lower Bounds

Let P be a problem. We have 2 goals:

(1) find an algorithm for P, and find a fcn. $f(n)$ for which algorithm runs in time $O(f(n))$. We seek to reduce $f(n)$ as much as possible.

(2) find a lower bound $g(n)$ on runtime of <u>all</u> algorithms solving P, i.e. Prove any such algorithm runs in time $\Omega(g(n))$. We seek to increase $g(n)$ as much as possible.

We're happy if:

$$f(n) = g(n)$$

or $f(n) \sim g(n)$

or $f(n) = \Theta(g(n))$.

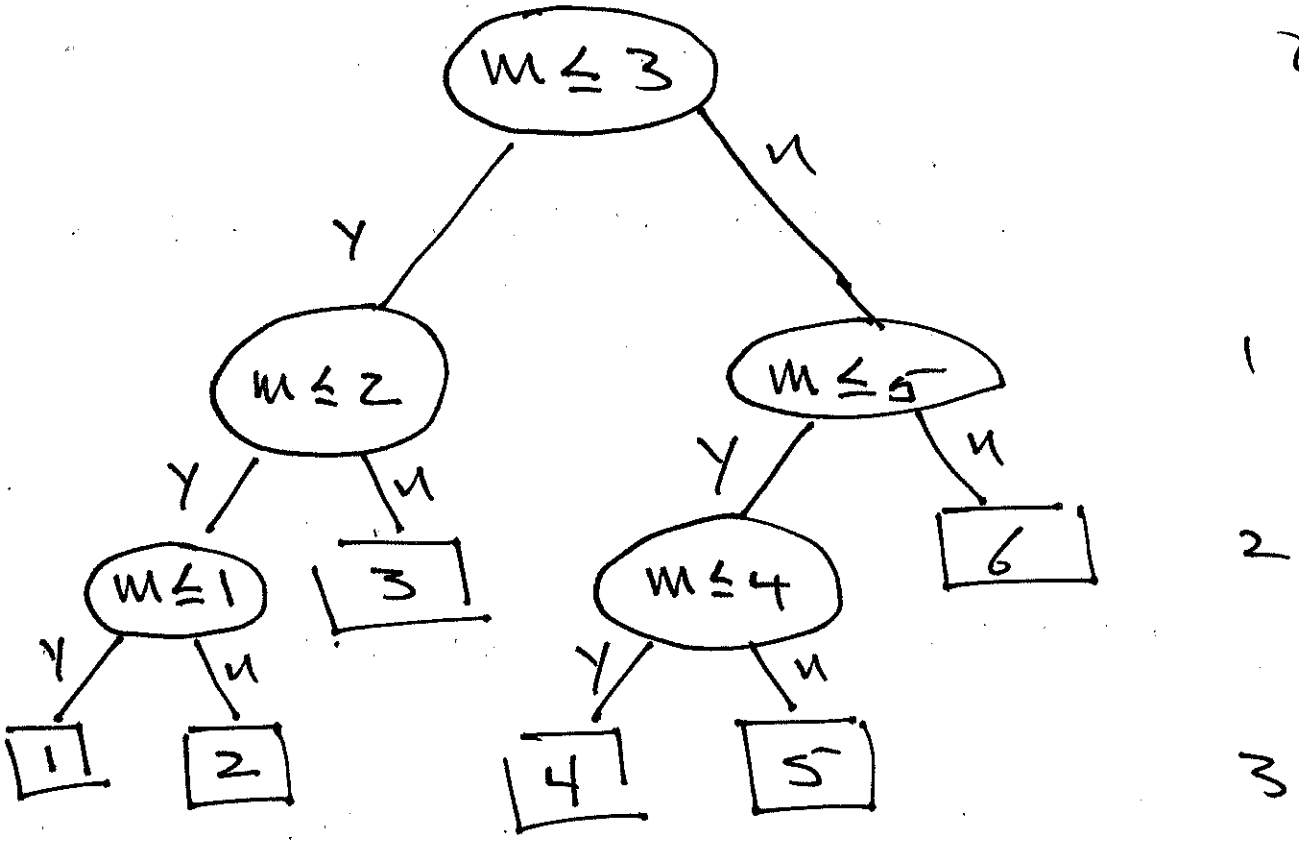
Two Kinds of L.B.

- Decision tree
- Adversary arguments

Ex. Game

- You think of $m \in \{1, 2, 3, 4, 5, 6\}$.
- I try to guess m by asking yes/no questions

Binary search: as a decision tree.



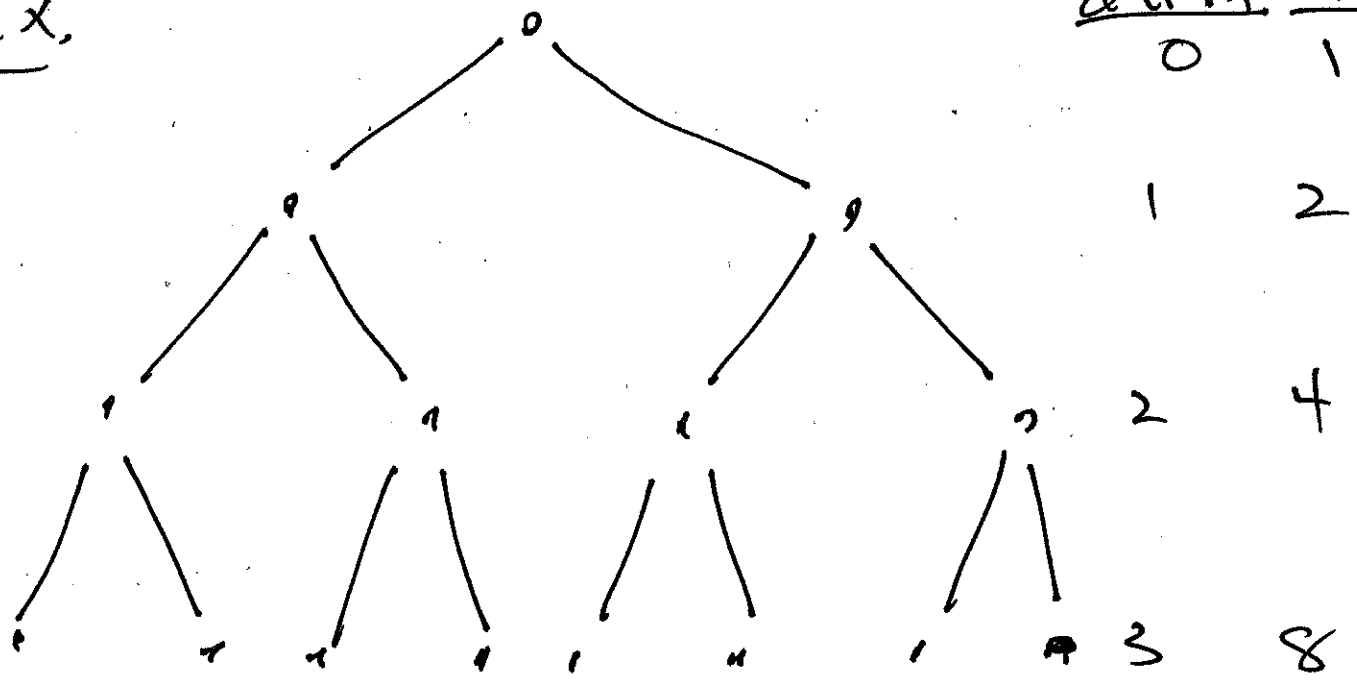
Defn

A k-ary tree is a rooted tree in which all internal nodes have at most k children.

A complete Binary Tree (CBT) is a B.T. in which

- all leaves at same depth
- all internal nodes have exactly 2 children.

Ex.



depth	#
0	1
1	2
2	4
3	8

$$(\#\text{nodes at depth } d) = 2^d$$

Defn

The height of a rooted tree is maximum leaf depth: h

The # leaves in a CBT of height h is

$$\#\text{leaves} = 2^h$$

also

$$h = \lg(\#\text{leaves})$$

Theorem

Let T be a B.T. with n leaves & height h. Then

$$h \geq \lceil \lg(n) \rceil$$

Proof:

Let $L(T)$, $H(T)$ denote #leaves an height of T, resp.

Proceed by induction on $h = H(T)$.

I. if $h=0$, then T has only one node, namely root. Then $h \geq \lceil \lg n \rceil$ becomes

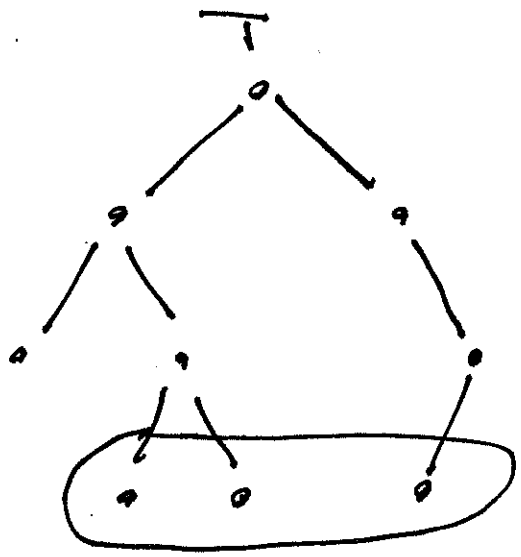
$$0 \geq 0$$

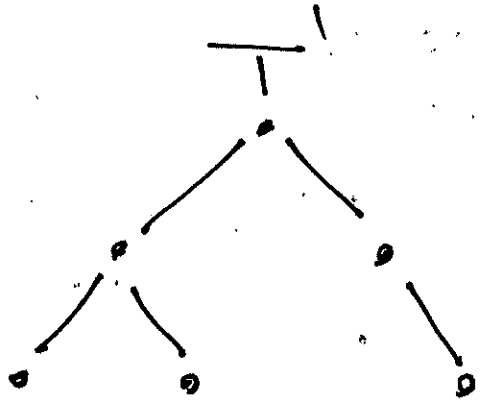
which is true.

II. let $h > 0$. Assume for any B.T. T' with $H(T') = h-1$, that $H(T') \geq \lceil \lg(L(T')) \rceil$. we must show that if T is a tree with $H(T) = h$ and $L(T) = n$, then $h \geq \lceil \lg(n) \rceil$.

Suppose T is a B.T. with $H(T)=h$ and $L(T)=n$.

Let T' be the B.T. obtained from T by deleting all nodes at depth h (along with incident edges.)





$H(T)=3$

$H(T')=2$

delete

Observe $H(T') = h-1$. So by the ind. hyp. we have

$$H(T') \geq \lceil \lg(L(T')) \rceil$$

Also, since T is a binary tree: $L(T) \leq 2L(T')$, hence

$$L(T') \geq \frac{L(T)}{2}.$$

$\Rightarrow$

$$\begin{aligned} h-1 &= H(T') \\ &\geq \lceil \lg(L(T')) \rceil \end{aligned}$$

by ind. hyp.

$$\geq \lceil \lg\left(\frac{L(T)}{2}\right) \rceil$$

$$= \lceil \lg(L(T)) - 1 \rceil$$

$$= \lceil \lg(L(T)) \rceil - 1$$

$$= \lceil \lg(n) \rceil - 1$$

∴ $h \geq \lceil \lg(n) \rceil$ ∎

Theorem

Let T be a k-ary tree with n leaves and height h. Then

$$h \geq \lceil \log_k(n) \rceil .$$

Exercise: Prove this -

To find a lower bound for algorithms solving P: consider all algorithms that solve P by asking k-ary questions (Probes), i.e. having k poss. answers. Let n be size of an instance of P, and $t(n)$ the # of verdicts as fcn. of input size.

Any algorithm of this kind can be represented by a k-ary tree.

internal nodes: Probes of data,
children: next Probe to execute.
leaves: Verdicts

a descending Path from root to leaf is a particular seq. of Probes, leading to a particular Verdict.

note: #verdicts ≤ #leaves

i.e. $f(n) \leq L(T)$

By last Theorem

worst case # Probes (k-ary) $= H(T)$

$\geq \lceil \log_k(L(T)) \rceil$

$\geq \lceil \log_k(f(n)) \rceil$

This proves:

Theorem

Suppose $\mathcal{P}$ has $f(n)$ Poss. verdicts on input of size n.

Then any algorithm solving

↳ by asking only k-ary questions

must ask at least

$$\lceil \log_k(f(n)) \rceil$$

such questions in worst case.

Hence asymp. run time

$$\lceil \log_k(f(n)) \rceil = \Theta(\log(f(n)))$$

Ex. • Pick $m \in \{1, 2, 3, 4, 5, 6\}$

• guess m asking yes/no questions

$n = 6$

$f(n) = 6$

$k = 2$

must ask at least

$$\lceil \log_2(6) \rceil = 3$$

questions

Ex.

- Pick $m \in \{1, \ldots, n\}$
- guess m asking k-ary questions

$$L.B = \lceil \log_k(n) \rceil$$

(n = f(n))

if $n = 10^6$, $k = 2$

$$L.B = \lceil \lg(10^6) \rceil = 20$$

if $n = 10^6$, $k = 3$

$$L.B. = \lceil \log_3(10^6) \rceil = 13$$

Theorem

Any comparison based sorting algorithm (i.e. $A[i] < A[j]$) must perform at least

$$\lceil \lg(n!) \rceil = \Theta(n \log n)$$

comparisons in worst case, on input arrays of length n.

Proof

$f(n) = n!$ since $A[1 \ldots n]$ has $n!$ permutations. $k = 2$, so follows from last Theorem.

Summarize

(1) Determine K, max # of outcomes to any probe of input data

(2) determine $f(n)$, the # of predicts possible on input of size n.

(3) conclude any algorithm for P doing probes of type (1) must do

- $\lceil \log_K(f(n)) \rceil$ probes (worst case)
- $\log_K(f(n))$ probes (avg. case.)

Warning:

nothing here implies existence of algorithms as few probes as stated. Only non-existence of algorithms that do better.

Adversary arguments

Ex same game

A : Picks $m \in \{1, \ldots, 6\}$

B : guesses m by asking only yes/no questions.

Now A cheats!

A is adversary or daemon.