1/26/16

Homework 3 due Thursday

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

$\forall \exists \; c, n_0$

$f(n) \geq g(n) * c$

where $n \geq n_0$

example: any function same or smaller than $O(n^2)$ [upperbound]

"sorting can be done in order of n^2" is correct since the n^2 is an upperbound

[lowerbound] $\Omega(n^2)$ any function asymptotically larger than n^2 (or equal)

$n^3 = \Omega(n^2)$

$n^2/100 = \Omega(n^2)$

$T_{sorting}(n) = \Omega(n)$ (lowerbound for the problem since must look @ each element)

to improve a lowerbound → increase it

to improve an upperbound → decrease it

↳ hopefully they match - optimal algorithm

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

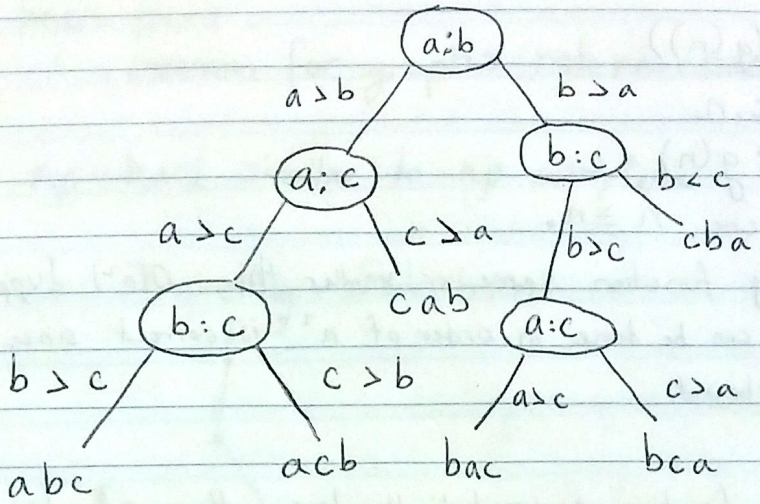
$f(n)$ and $g(n)$ are in the same order (optimal)

ex: prove lowerbound of sorting

model of computation: comparison/exchange model

- must establish a specific model

assume all numbers are unique (makes analysis simpler)



Decision Tree: a model of computation

- number of leaves are all sorted orders (# leaves $\geq n!$)
- this is a compact tree b/c no redundancy
  but redundancy is possible for other trees
- any algorithm can be modelled w/ a Decision Tree
- runtime is length of longest path

any binary tree w/ n leaves: height is log(n) or, rather,

height $\geq \log(n!)$ [worst case]

$T(n) \geq \log(n!)$

prove by induction that height of binary tree is log(leaves)

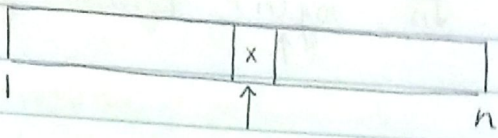
$T(n) \geq n\log(n)$ for comparison/exchange model

$T_{sorting}(n) = \Omega(n \log n)$

this is correct as is our previous statement since we proved both

Binary Search

given a sorted list (in non-decreasing order)



looking for k

if $k < x$, only work w/ left side

if $k > x$, only work w/ right side

otherwise k does not exist in the list

$T(n) = T(\frac{n}{2}) + c$ → equivalent to $O(1)$

similar analysis to merge sort

$T(n) = O(\log n)$

$T_{\text{binary search}}(n) = \Omega(1)$ [very trivial]

improve lowerbound?

using c/E model

use Decision Tree to prove lowerbound to be $\log(n)$

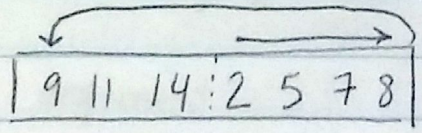
need a node that is the possibility k does not exist

# of leaves = n

height of tree = $\log(n)$ [only true for balanced]

Midterm Problem: Circulary Sorted List

Smallest number is somewhere in the list, the next smallest is next to it. when you run out of space put the next smallest number all the way to the left



time complexity to find k?

find an efficient algorithm

trivial : $O(n)$ [compare each element to k]

possible targets: $\sqrt{n}$, $\log(n)$, 1

$\log(n)$: w/ a few comparisons → get rid of half the list

look at first number and the median, what conclusions can you draw?

using these two numbers, you can get rid of half the list

median = middle of the list or $\frac{n}{2}$