CPSC 320: Intermediate Algorithm Design & Analysis

Asymptotic Notation: (O, Ω, Θ, o, ω)

Steve Wolfman

Analysis of Algorithms

- Analysis of an algorithm gives insight into how long the program runs and how much memory it uses
  - time complexity
  - space complexity
- Analysis can provide insight into alternative algorithms
- Input size is indicated by a number *n* (sometimes there are multiple inputs)
- Running time is a function of *n* ($\mathbf{Z}^+ \rightarrow \mathbf{R}^+$) such as
  - $T(n) = 4n + 5$
  - $T(n) = 0.5 \, n \log n - 2n + 7$
  - $T(n) = 2^n + n^3 + 3n$
- But...

We'll be fast and loose on the "**R+**" part... If the function goes negative, let's assume it's actually some small positive constant.

Types of asymptotic analysis

- bound flavor
  - upper bound (O, o)
  - lower bound (Ω, ω)
  - asymptotically tight (θ)
- analysis case
  - worst case (adversary)
  - average case
  - expected case
  - best case
  - "common" case
  - "amortized"
- analysis quality
  - loose bound (any true analysis)
  - tight bound (no better bound which is asymptotically different)

Order Notation

- $T(n) \in O(f(n))$ if there are constants c and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$

Big-O Practice

- Prove that $n^2/2 + 5n/2 \in O(n^2)$
- Prove that $n^2/2 + 5n/2 \notin O(n)$
- Prove that $3^n \notin O(2^n)$

Asymptotic Analysis Hacks

- Eliminate low order terms
  - $4n + 5 \Rightarrow 4n$
  - $0.5\, n \log n - 2n + 7 \Rightarrow 0.5\, n \log n$
  - $2^n + n^3 + 3n \Rightarrow 2^n$
- Eliminate coefficients
  - $4n \Rightarrow n$
  - $0.5\, n \log n \Rightarrow n \log n$
  - $n \log (n^2) = 2\, n \log n \Rightarrow n \log n$

Justifying the Hacks

More Order Notation

- $T(n) \in O(f(n))$ iff there are constants $c \in \mathbf{R}^+$ and $n_0 \in \mathbf{Z}^+$ such that $T(n) \le c\, f(n)$ for all $n \ge n_0$
- $T(n) \in \Omega(f(n))$ iff there are constants $c \in \mathbf{R}^+$ and $n_0 \in \mathbf{Z}^+$ such that $f(n) \le c\, T(n)$ for all $n \ge n_0$
- $T(n) \in \theta(f(n))$ iff $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$
- $T(n) \in o(f(n))$ iff for all constants c, there is a constant n_0 such that $T(n) < c\, f(n)$ for all $n \ge n_0$
- $T(n) \in \omega(f(n))$ iff for all constants c, there is a constant n_0 such that $T(n) > c\, f(n)$ for all $n \ge n_0$

For my 221ers...
I defined o and ω blatantly wrong!
Sorry ☹

Limits Definition of o/ω/Θ

- $f(n) \in o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \Theta(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c$
- $f(n) \in \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

L'Hôpital's Rule Reminder

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{0}{0}$ or $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\infty}{\infty}$

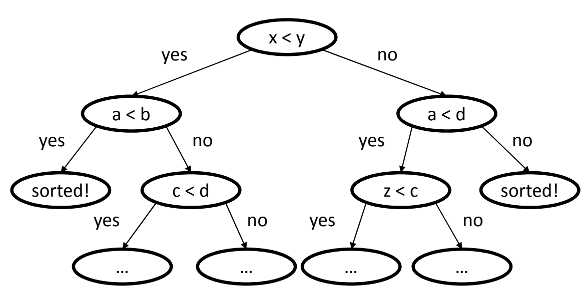
Then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\frac{\mathrm{d}f(n)}{\mathrm{dn}}}{\frac{\mathrm{d}g(n)}{\mathrm{dn}}}$

Limits Practice

- Prove that $2^{2^n} \in \omega(n^n)$
- Prove that $\lg n \in o(n^{0.5})$

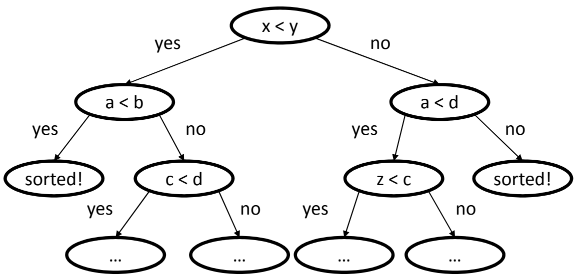
Complexity of Sorting Using Comparisons as a Problem

Each comparison is a "choice point" in the algorithm. You can do one thing if the comparison is true and another if false. So, the whole algorithm is like a binary tree…



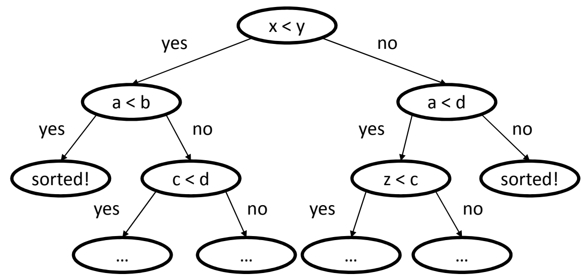
Complexity of Sorting Using Comparisons as a Problem

The algorithm spits out a (possibly different) sorted list at each leaf. What's the maximum number of leaves?



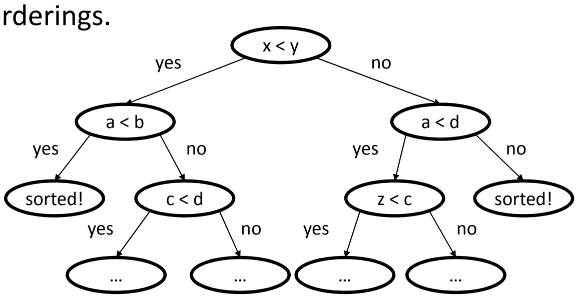
Complexity of Sorting Using Comparisons as a Problem

There are n! possible permutations of a sorted list (i.e., input orders for a given set of input elements). How deep must the tree be to distinguish those input orderings?



Complexity of Sorting Using Comparisons as a Problem

If the tree is *not* at least lg(n!) deep, then there's some pair of orderings I could feed the algorithm which the algorithm does not distinguish. So, it must not successfully sort one of those two orderings.



Complexity of Sorting Using Comparisons as a Problem

QED: The complexity of sorting using comparisons is Ω(lg (n!)) in the worst case, *regardless of algorithm*!

In general, we can *lower-bound* but not *upper-bound* the complexity of problems.

(Why not? Because I can give as crappy an algorithm as I please to solve any problem.)

Ω Practice: Find a Good Bound for lg (n!)

More Practice: Find Θ-Bound for lg (n!)

(This isn't tightly related to our decision tree proof, except that it shows our Ω-bound is tight.)

What's Next?

- Divide and Conquer Algorithms
- Recurrence Relations
- CLRS Chapter 4

On Your Own

- PRACTICE PRACTICE PRACTICE with proofs about asymptotic complexity (there are many practice problems in the textbook and on old exams)