Recap: A useful mathematical fact

$$x^{\log_b y} = y^{\log_b x}$$

We really will use it today!

The Master Theorem for Divide-and-Conquer Recurrence Relations

Derive the resultd

Technique 4: catalog some general equations and their solutions

- General Divide and Conquer Example:
- $f_n = Af_{n/B} + h(n)$, where $h(n)$ is $O(n^k)$, and where $A \geq 1$ and $B > 1$
- Solution:

$$f_n = \begin{cases} O\left(n^{\log_B A}\right) & if \quad A > B^k \\ O\left(n^k \log n\right) & if \quad A = B^k \\ O\left(n^k\right) & if \quad A < B^k \end{cases}$$

On to the derivation of this solution…

Why will we do this long derivation of this solution?

- We'll use the result a few times
- The derivation will reinforce a technique (telescoping) that works for solving many other recurrence relations
- Some of the detailed steps are the kinds of things you'll be doing again, and repetition does not hurt!

Proof of a special case of this general divide-and-conquer recurrence relation

- $f_n = Af_{n/B} + h(n)$,
  where $h(n)$ is $O(n^k)$,
  and $A \geq 1$ and $B > 1$
- Solution:

$$f_n = \begin{cases} O\left(n^{\log_B A}\right) & if \quad A > B^k \\ O\left(n^k \log n\right) & if \quad A = B^k \\ O\left(n^k\right) & if \quad A < B^k \end{cases}$$

- Special case:
  - Assume that n is a power of B ($n=B^M$), that $f_1=1$, and ignore the constant factor in $O(n^k)$:
    [i.e., use $h(n) = n^k$]
- The recurrence relation becomes

$$f_{B^M} = A \quad f_{B^{M-1}} + B^{kM}$$

Continue the derivation

- From last slide: $$f_{B^M} = A \quad f_{B^{M-1}} + B^{kM}$$
- Divide both sides by A^m: $$\frac{f_{B^M}}{A^M} = \frac{f_{B^{M-1}}}{A^{M-1}} + \left(\frac{B^k}{A}\right)^M$$
- Replace M by other numbers: $$\frac{f_{B^{M-1}}}{A^{M-1}} = \frac{f_{B^{M-2}}}{A^{M-2}} + \left(\frac{B^k}{A}\right)^{M-1}$$

$$\frac{f_{B^1}}{A^1} = \frac{f_{B^0}}{A^0} + \left(\frac{B^k}{A}\right)^1$$

- Add the terms, see some disappear, simplify.

Continue the derivation

- Simplify to get

$$f_n = f_{B^M} = A^M \sum_{i=0}^{M} \left(\frac{B^k}{A}\right)^i$$

- If A > B^k
- If A = B^k
- If A < B^k

$$\boxed{f_n = \begin{cases} O\left(n^{\log_B A}\right) & if \quad A > B^k \\ O\left(n^k \log n\right) & if \quad A = B^k \\ O\left(n^k\right) & if \quad A < B^k \end{cases}}$$

Use this to analyze a few algorithms

- $f_n = Af_{n/B} + h(n)$, where h(n) is $O(n^k)$, and where A≥1 and B>1
- **Solution:**

$$f_n = \begin{cases} O\left(n^{\log_B A}\right) & if \quad A > B^k \\ O\left(n^k \log n\right) & if \quad A = B^k \\ O\left(n^k\right) & if \quad A < B^k \end{cases}$$

Analyze binary search, merge sort, max subsequence sum, binary search of an unsorted array.

Sorting

Sorting Outline

- Sorting overview
- Review of elementary sorts
- Lower bound for the worst case of comparison-based sorting algorithms
- Non-comparison-based sorting algorithms
- Quicksort and its analysis

Sorting is ubiquitous

- In the classic book series *The Art of Computer Programming*, Donald Knuth devoted a whole volume (about 700 pages) to sorting and searching
- He claimed that about 70% of all CPU time is spent on these two activities

"Sorting" is a funny word for this concept!

- Not quite like normal English usage
- Is there a normal English usage?
- From Knuth:
  - He was sort of out of sorts from **sorting** that sort of data.
- Could "ordering" be a better word?
- Knuth again:
  - My boss ordered me to order [more memory] so that we could order our data several orders of magnitude faster
  - Actually in Knuth's (dated) statement, it was "tape drive" instead of "more memory"

Elementary Sorting Methods

- Name several of them
- How does each work?
- Running time for each (sorting N items)?
  - best
  - worst
  - average
  - Extra space requirements
- Spend 10 minutes with a group of three, answering these questions. Then we will summarize

Elementary Sorting Methods

- Some possible answers **(Collect them on the board)**
  - Bubble sort **(Don't say the b–word!)**
  - Insertion sort **Like sorting files in manila folders**
  - Selection sort **Select the largest, then the second largest, …**
  - Merge sort **Split, recursively sort, merge**
  - Binary tree sort **Insert all into BST, then inOrder traversal**
  - (Quicksort) **Not so elementary. We'll do it in detail**
    - http://students.ceid.upatras.gr/~pirot/java/Quicksort/
  - (Heapsort) **We'll also do this one in detail**
  - (Shellsort) **Interesting variation on insertion sort**
  - (Radix sort) **Another one that we'll consider in some detail**
  -

Best, worst, average time?
Extra space requirements?

A Lower Bound for Sorting Algorithms' Worst-case Run Time

- Lower bound for best case?
  - A particular algorithm that achieves this?
- Lower bound for worst case
  - This is the one we really care about
  - It's tricky:
    - We want to be able to find a function f(N) such that the worst case running time for all sorting algorithms is Ω(f(N))
    - The problem is, how do we get a handle on "all sorting algorithms"?

Lower bound for sorting algorithms running time

- The problem is, how do we get a handle on all sorting algorithms?
- We can't list all sorting algorithms and analyze all of them
  - Why not?
- But we can find a uniform representation of any sorting algorithm that is based on **comparing** elements of the array to each other

This "uniform representation" idea is exploited in a big way in Theory of Computation, to demonstrate the unsolvability of the "Halting Problem"

First of all…

- The problem of sorting N elements is at least as hard as determining their ordering
  - e.g., determining that $a_3 < a_4 < a_1 < a_5 < a_2$
- So any lower bound on all "order-determination" algorithms is also a lower bound on "all sorting algorithms"

Sort Decision Trees

- Let A be any comparison-based algorithm for sorting an array of distinct elements
  - What do we mean by comparison-based?
- Note that sorting is asymptotically equivalent to determining the correct order of the originals. Because once we have determined the correct order, a linear algorithm will do the actual sorting
- For any given N, we can draw an EBT that corresponds to the comparisons that will be used by A to sort an array of N elements
  - [This is just an on-paper EBT. Not a data structure to implement]
  - Do it for three elements and selection sort
  - Clearly, different algorithms will have different trees
- The worst-case number of comparisons for A is the _______ of the Sort Decision Tree