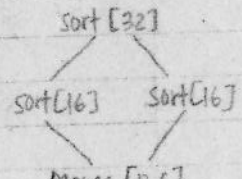
CSCI 303 Lecture
1/26/10

Topics
- Sorting
- Data Analysis
- Divide and Conquer (Reduction)
- Recursive Algorithm
- Time Analysis

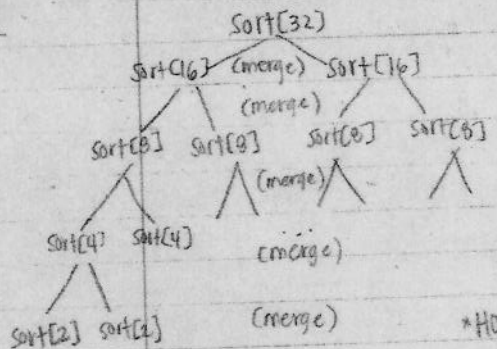
Ex: Shuffling Cards

sort A[1:32]
↓
sort (A[1:16]) (A[17:32]) } Reduce 1 problem into two of same type
- rearranged to B[1:16] such that B[i] ≤ B[i+1] and C[1:16] such that C[i] ≤ C[i+1]
- Two sorted sets, how do we put them together?
  1) Compare each card in 1st set w/ each card in 2nd
- started w/ sorting problem of size 32
- split into 2 sorting problems of size 16 } Reduction subroutines
- Merged (B,C]



→ Merge
Input B,C such that B,C sorted
Output sorted list of B ∪ C

could also do



Divide & conquer · keep reducing problem size til we get to "base" case

*How do we do time analysis?

# of comparisons:
Sort[2] = ≤ 2 × 32/2
Sort[4] = ≤ 4 × 32/4
Sort[8] = ≤ 8 × 32/8
Sort[16] = ≤ 16 × 32/16
Sort[32] = ≤ 32 × 1

⇒ 32+32+32+32+32 = 32×5

$\log_2 32 = 5$

recurrence: $n \cdot \frac{n}{n} + \frac{n}{2} \cdot \frac{n}{n/2}$ = $\boxed{n \log n}$ for MergeSort

- Is this algorithm faster?
  Yes, nLogn vs n^2

In General, (split time)

$T_{MS}(n) = 2T_{MS}(\frac{n}{2}) + n + n$

$\Rightarrow T_{MS}(n) = 2T_{MS}(\frac{n}{2}) + 2n$

Another way to think about complexity:

Worst Case Analysis

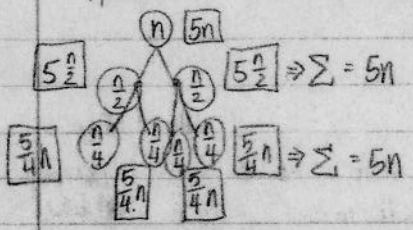
First way: $T_{MS} = T(16) + T(16) + T_M(16,16)$ //recurrence

$\leq 2T_{MS}(16) + 32 \Rightarrow 2T_{MS}(\frac{n}{2}) + 2n \Rightarrow n\log n$

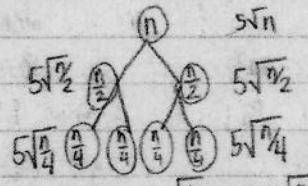
Last Time: $T(n) = T(n-1) + (n-1) \Rightarrow T_L(n) = n(n-1)/2 \Rightarrow n^2$

Solving Recurrences

suppose: $T(n) + 2T(\frac{n}{2}) + 5n$



$T(n) = 2T(\frac{n}{2}) + 5\sqrt{n}$



$5\sqrt{n} + 5\cdot 2\sqrt{n/2} + 5\cdot 4\sqrt{n/4} \ldots$

$= 5\sqrt{n}\,[1 + \sqrt{2} + \sqrt{4} + \ldots \sqrt{n/2}]$

$= 5\sqrt{n}\,[1 + \sqrt{2} + (\sqrt{2})^2 + (\sqrt{2})^3 + (\sqrt{2})^4 \ldots] \leq O(n)$

Suppose $T(n) = 2T(\frac{n}{2}) + n^{1-\varepsilon}$ where $\varepsilon = 0.0001$

$= n^{1-0.0001}\,[1 + 2^{0.0001} + (2^{0.0001})^2 + (2^{0.0001})^3 + \ldots (2^{0.0001})^n]$

$= O(n)$ //as long as exponent is small enough, n will be dominant

Another Sorting Algorithm

Given a collection of data A[1], A[2] ... A[n] (more stable than avg)

What do we want to know? min, max, Avg, median, Mode, Std dev

1st Moment $\Sigma A[i]$

2nd Moment $\Sigma A^2[i]$

3rd Moment $\Sigma A^3[i]$

Max = O(n)

Min = O(n)

Avg = O(n)

$T_{median}(n) = ?$

$T_{sort}(n) = 2T_{sort}(\frac{n}{2}) + n + T_{median}(n)$

$T_{median}(n) = \Theta(n)$ for next time!