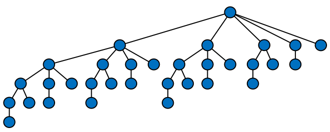
Lecture 5. Divide and Conquer, Solving Recurrences

CpSc 8400: Algorithms and Data Structures
Brian C. Dean



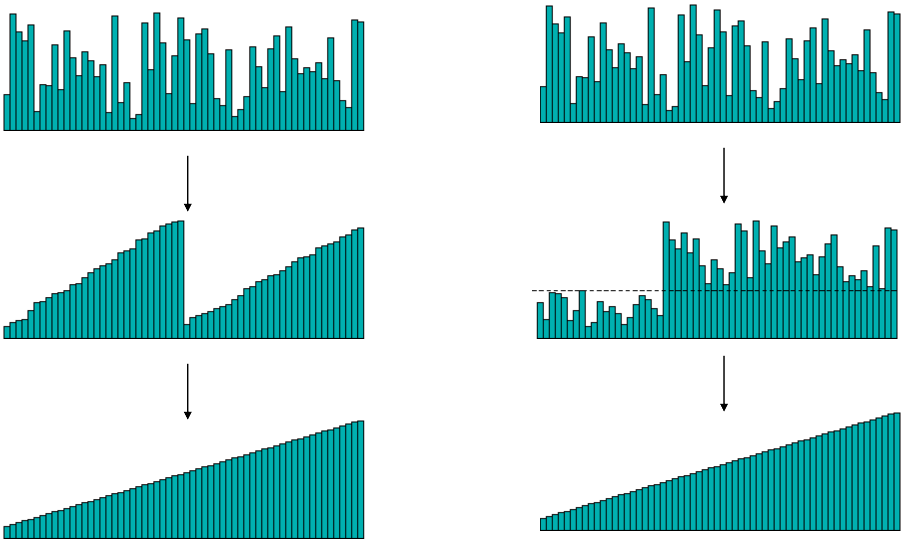
School of Computing
Clemson University
Spring, 2016

Divide and Conquer

- Extremely powerful and widely-applicable technique for designing algorithms:
  - **Divide** problem into "multiplicatively" smaller subproblems (e.g., 2 problems of size n/2) that have the same structure as the original.
  - **Recursively solve** the subproblems.
  - **Combine** their solutions to obtain a solution to the original problem.
- Today: how to analyze running times of such recursive algorithms using **recurrences.**

2

Divide and Conquer Examples



Merge Sort QuickSort

3

Recurrences

- To merge sort an array of size n…
  - We recursively sort two smaller arrays of size n/2.
  - Then we spend $\Theta(n)$ time merging the results.
- If T(n) denotes the running time of merge sort on an input of size n, we can therefore write a **recurrence** (recursive formula) for T(n):

  $T(n) = 2T(n/2) + \Theta(n)$

  As a base case, $T(n) = O(1)$ for $n = O(1)$.
- The goal is now to **solve** the recurrence to produce an explicit (non-recursive) formula for T(n):

  $T(n) = \Theta(n \log n)$.

4

Simplifying Recurrences

- Actual merge sort recurrence:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$$

- However, since we only care about an asymptotic solution, we can always ignore floors, ceilings, and any other small additive terms (e.g., T(n/2 + 7) in our recursive calls).
- We can also replace the Θ(n) with just n, as this will only change our solution by a constant factor.
- So we focus on solving T(n) = 2T(n/2) + n.

5

Algebraic Expansion

- A useful way to solve (or guess the solution to) <u>any</u> recurrence is to simply expand it out a few levels:

$$\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2[n/2 + 2T(n/4)] \\
&= n + n + 4T(n/4) \\
&= n + n + 4[n/4 + 2T(n/8)] \\
&= n + n + n + 8T(n/8) \\
&\ldots \\
&= \underbrace{n + n + n + \ldots + n}_{n \log n} + \underbrace{nT(1)}_{n\Theta(1)} = \Theta(n \log n).
\end{aligned}$$

6

Solving Simple Recurrences

- Let's start by considering "divide and conquer" recurrences of the form:

  $T(n) = aT(n/b) + f(n)$

  with $T(n) = O(1)$ for $n = O(1)$ as a base case.
- I.e., to solve a problem of size n, we recursively solve a subproblems of size n/b, and spend f(n) time during the "division" and "recombination" steps of the algorithm.
- The simple form above covers nearly all of the recurrences we'll encounter in this course…

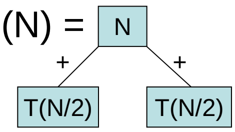
7

Tree Expansions

- $T(N) = N + 2T(N/2)$

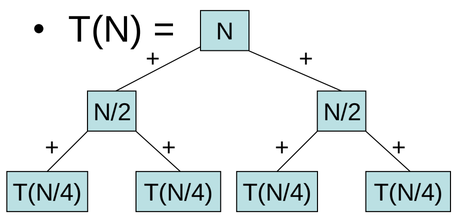
Tree Expansions

- T(N) =



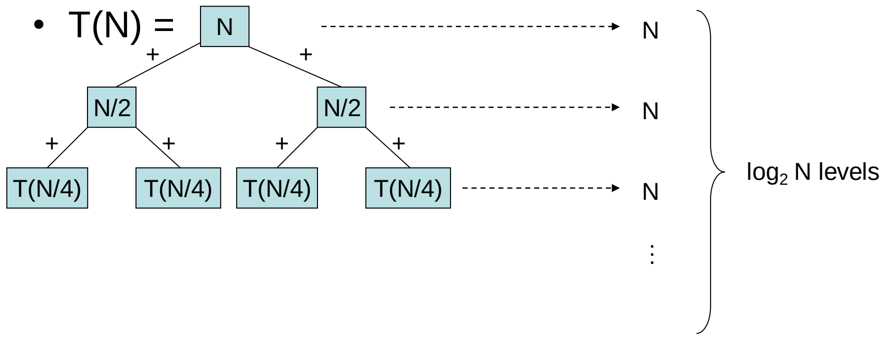
Tree Expansions

- T(N) =



Tree Expansions

- T(N) =

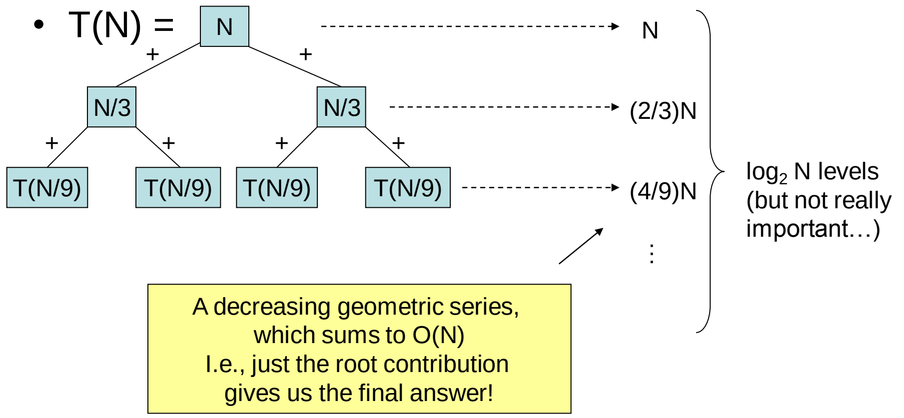


What about…

- T(N) = N + 2T(N/3)

Tree Expansions

- T(N) =



Quick Aside: Asymptotic Behavior of a Geometric Series

- Anyone remember how to sum a geometric series?

$$S = n^2 + (3/4)\,n^2 + (3/4)^2\,n^2 + \ldots + (3/4)^k\,n^2$$
$$= n^2\,[1 + (3/4) + (3/4)^2 + \ldots + (3/4)^k]$$
$$\leq n^2\,[1 + (3/4) + (3/4)^2 + \ldots]$$

Quick Aside: Asymptotic Behavior of a Geometric Series

- Anyone remember how to sum a geometric series?

$$S = n^2 + (3/4)\, n^2 + (3/4)^2\, n^2 + \ldots + (3/4)^k\, n^2$$
$$= n^2\, [1 + (3/4) + (3/4)^2 + \ldots + (3/4)^k]$$
$$\leq n^2\, [1 + (3/4) + (3/4)^2 + \ldots]$$
$$= 4n^2.$$

- A decreasing geometric series behaves asymptotically just like its 1st term: $S = \Theta(n^2)$.
- By symmetry, if S were increasing, it would behave asymptotically like its final term:

$$T = n^2 + 2n^2 + \ldots + 2^k n^2 = \Theta(2^k n^2).$$

Tree Expansions

- Consider $T(n) = aT(n/b) + n^\alpha$
- Let's expand it algebraically as before, only in a more useful way --- as a tree!
- If we add up the contribution of each level in the tree, we always obtain a geometric series!
- So we just need to know if this series is:
  - Decreasing: solution $T(n) = \Theta(\text{root contribution})$
  - Increasing: solution $T(n) = \Theta(\text{leaf contribution})$
  - Unchanging: solution $T(n) = \Theta(L \log n)$, where L is the contribution on each of the log n levels.

The “Master Method”

- Consider $T(n) = aT(n/b) + n^{\alpha}$.
- There are $\log_b n$ levels in the tree, and our branching factor is a, so the total # of leaves is $a^{\log_b n} = n^{\log_b a}$.
- Since each leaf contributes $\Theta(1)$, the total leaf contribution is $\Theta(n^p)$, where $p = \log_b a$.
- We can now say that T(n) =

$\Theta(n^{\alpha})$	if $\alpha > p$	(decreasing series)
$\Theta(n^{\alpha} \log n)$	if $\alpha = p$	(unchanging series)
$\Theta(n^p)$	if $\alpha < p$	(increasing series)

- This formula is sometimes called the **master method** for solving a divide-and conquer recurrence. (don’t forget the tree expansion intuition though!)

17

Practice

- Let’s solve the following recurrences:

$T(n) = 2T(n/2) + \Theta(n)$

$T(n) = T(n/2 - 6) + T(n/2 + 10) + \Theta(n)$

$T(n) = 3T(n/2) + \Theta(n)$

$T(n) = 3T(n/2) + \Theta(n^2)$

$T(n) = 4T(n/2) + \Theta(n^2)$

$T(n) = 8T(n/3) + \Theta(n^2)$

$T(n) = 81T(n/3) + \Theta(n^4)$

$T(n) = 1023T(n/2) + \Theta(n^{10})$

$T(n) = 2T(n/2) + \Theta(n \log n)$

18

Example: Maximum Sum Subarray

- Given an array A[1…n] of numbers, find a subarray A[i…j] whose elements have maximum sum.

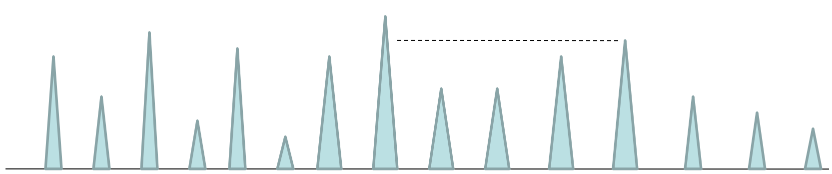
Example: Maximum Sum Subarray

- Given an array A[1…n] of numbers, find a subarray A[i…j] whose elements have maximum sum.
- Trivial approach: spend O(n) time checking all $\binom{n}{2}$ subarrays, for a total of O(n^3).
- Slightly faster: use prefix sums to check each subarray in O(1) time, for a total of O(n^2).
- Better yet: Use divide and conquer -- O(n log n).
- Best: Dynamic programming gives an even simpler O(n) algorithm, which we'll discuss later in the semester.

Example: Cyclic Shift Testing

- A sorted array A[1…n] of numbers has been subject to a right cyclic shift by k positions.
- Given the contents of the shifted array, please determine k quickly.

Example: Longest Line of Sight



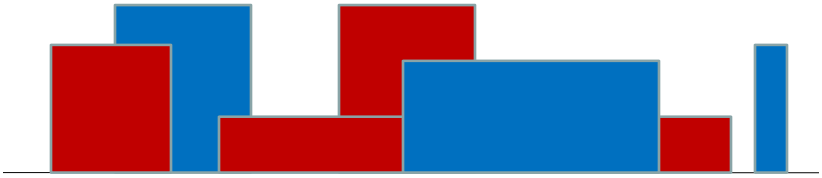
- Given the heights of N individuals standing in a line.
- **Goal:** find the length of the longest line of sight (difference in indices between two people between whom everyone else is strictly shorter).
- Divide and conquer gives us an O(n log n) solution (and there is an even faster solution using fancy data structures!)

Example: The "Skyline" Problem

- Find the total area of a skyline defined by a union of rectangular buildings with a common base:

Example: The "Skyline" Problem

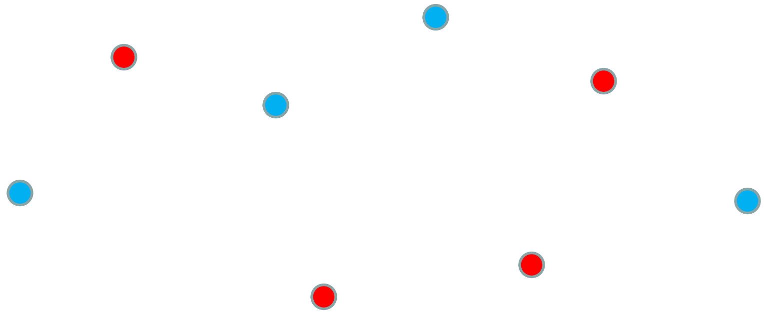
- Find the total area of a skyline defined by a union of rectangular buildings with a common base:



- All you need is merge! (the slightly lesser known hit Beatles song…)

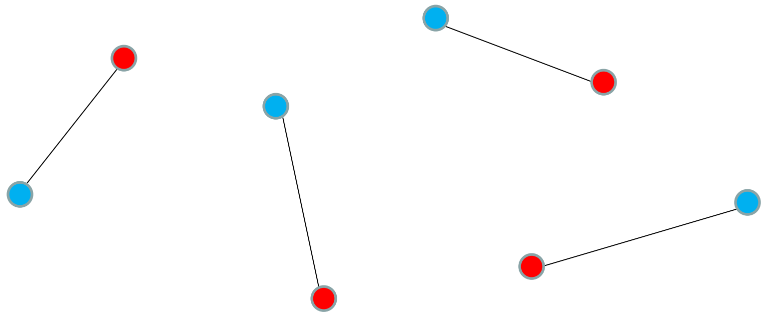
Bi-Chromatic Matching

- Given N red points and N blue points in the plane, match them up with non-crossing line segments…



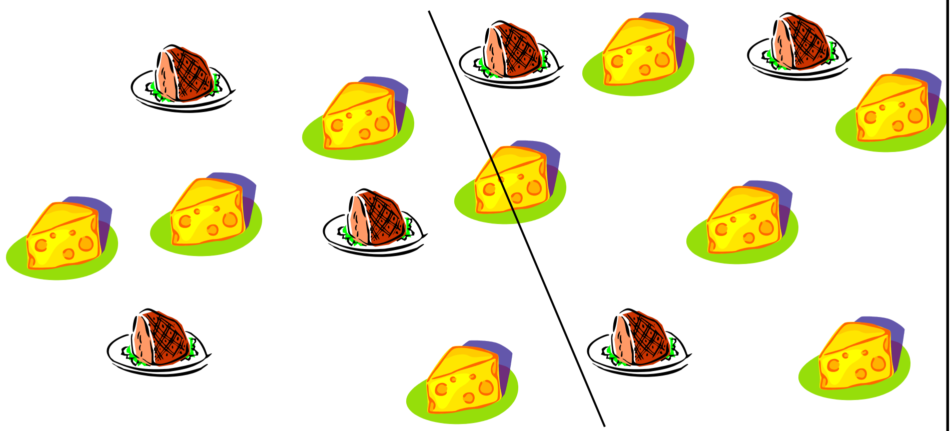
Bi-Chromatic Matching

- Given N red points and N blue points in the plane, match them up with non-crossing line segments…



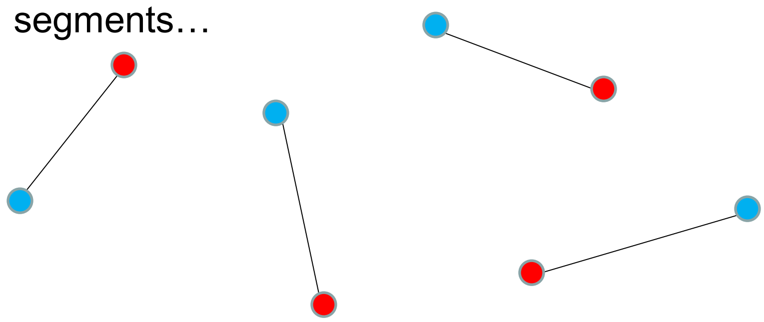
Ham Sandwich Cuts

- It's always possible to find a line in O(n) time that equally subdivides both the ham and the cheese!



Bi-Chromatic Matching

- Given N red points and N blue points in the plane, match them up with non-crossing line segments…



- Easy to solve recursively after partitioning with a ham sandwich cut!