Recurrences

Adapted from the CLRS book slides

Divide-and-Conquer

- **Divide** the problem into one or more subproblems that are smaller instances of the same problem.
- **Conquer** the subproblems by solving them recursively.
  - ***Base case:*** If the subproblems are small enough, just solve them by brute force.
- **Combine** the subproblem solutions to form a solution to the original problem.

Use a recurrence to characterize the running time of a divide-and-conquer algorithm. Solving the recurrence gives us the asymptotic running time.

A ***recurrence*** is a function is defined in terms of

- one or more base cases, and
- itself, with smaller arguments.

ALGORITHMIC RECURRENCES

Interested in recurrences that describe running times of algorithms.

A recurrence $T(n)$ is ***algorithmic*** if for every sufficiently large ***threshold*** constant $n_0 > 0$:

For all $n < n_0$, $T(n) = \Theta(1)$. *[Can consider the running time constant for small problem sizes.]*

For all $n \geq n_0$, every path of recursion terminates in a defined base case within a finite number of recursive invocations. *[The recursive algorithm terminates.]*

CONVENTIONS

Will often state recurrences without base cases. When analyzing algorithms, assume that if no base case is given, the recurrence is algorithmic.

Ceilings and floors in divide-and-conquer recurrences don't change the asymptotic solution $\Longrightarrow$ often state algorithmic recurrences without floors and ceilings.

Example:

recurrence for merge sort is really

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

CONVENTIONS (continued)

Some recurrences are inequalities rather than equations.

Example:

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

gives only an upper bound on $T(n)$, so state the solution using *O*-notation rather than Θ-notation.

EXAMPLES OF RECURRENCES

An algorithm that breaks a problem of size n into one subproblem of size $n/3$ and another of size $2n/3$, taking $\Theta(n)$ time to divide and combine: $T(n) = T(n/3) + T\left(\frac{2n}{3}\right) + \Theta(n)$.

- Solution: $T(n) = \Theta(n \lg n)$.

An algorithm that breaks a problem of size n into one subproblem of size $n/5$ and another of size $7n/10$, taking $\Theta(n)$ time to divide and combine: $T(n) = T(n/5) + T\left(\frac{7n}{10}\right) + \Theta(n)$.

- Solution: $T(n) = \Theta(n)$.

Subproblems don't always have to be a constant fraction of the original problem size. ***Example:*** recursive linear search creates one subproblem and it has one element less than the original problem. Time to divide and combine is $\Theta(1)$, given $T(n) = T(n-1) + \Theta(1)$.

- Solution: $T(n) = \Theta(n)$.

METHODS FOR SOLVING RECURRENCES

Substitution method: Guess the solution, then use induction to prove that it's correct.

Recursion-tree method: Draw out a recursion tree, determine the costs at each level, and sum them up. Useful for coming up with a guess for the substitution method.

Master method: A cookbook method for recurrences of the form $T(n) = aT(n/b) + f(n)$, where $a > 0$ and $b > 1$ are constants, subject to certain conditions. Requires memorizing three cases, but applies to many divide-and-conquer algorithms.

Akra-Bazzi method: A general method for solving divide-and-conquer recurrences. Requires calculus, but applies to recurrences beyond those solved by the master method.