

RECURRENCES

This handout summarizes highlights of *CLRS* Chapter 4 and Appendix A (*CLR* Chapters 3 & 4).

Two-Step Strategy for Analyzing Algorithms

1. Characterize running-time (space, etc.) of algorithm by a recurrence equation.
2. Solve the recurrence equation, expressing answer in asymptotic notation.

Recurrence Equations

A **recurrence equation** is just a recursive function definition. It defines a function at one input in terms of its value on smaller inputs.

We use recurrence equations to characterize the running time of algorithms. $T(n)$ typically stands for the running-time (usually worst-case) of a given algorithm on an input of size n .

Recurrence equations for divide-and-conquer algorithms typically have the form:

General Case ($n \geq 1$)

$$T(n) = [\# \text{ of subproblems}] T([\text{size of each subproblem}]) + [\text{cost of divide \& glue}]$$

Base Case ($n < 1$)

$$T(n) = O$$

Because the base case is always the same, it is usually omitted when defining $T(n)$.

Deriving Recurrence Equations: Examples

Insert an element into a sorted list of length n .	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Insertion-sort a list of length n .	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Binary search on an array of n elements.	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Find the maximum leaf of a balanced binary tree with n numeric leaves.	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Merge-sort a list of n elements.	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Given a balanced binary tree with n non-negative leaves summing to S , find a leaf such that all leaves to its left and all leaves to its right both have sums $\leq S/2$.	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$
Towers of Hanoi	$T(n) = \underline{\hspace{1cm}} T(\underline{\hspace{1cm}}) + \underline{\hspace{1cm}}$

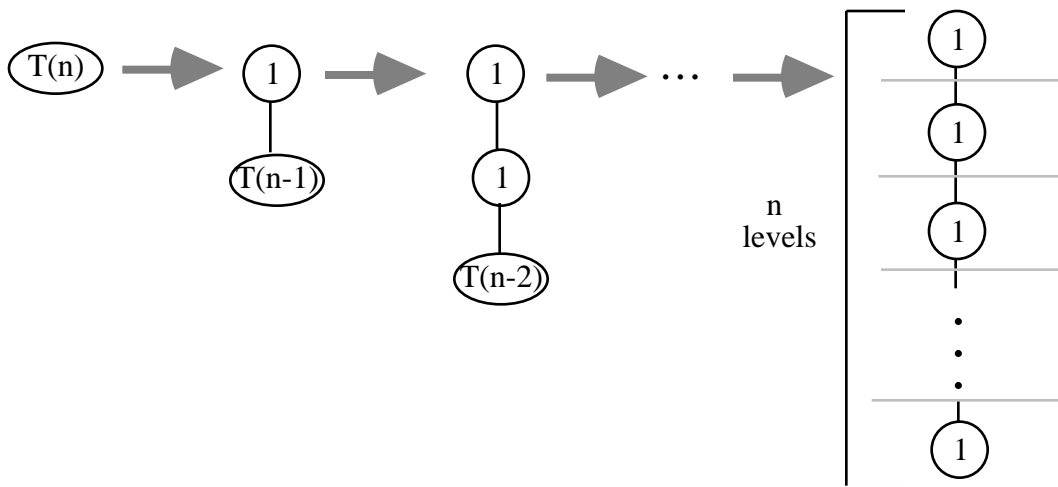
Solving Recurrence Equations: The Recursion-Tree Method

There are several methods for solving recurrence equations (see *CLRS* for details). We will mainly use the **recursion-tree method** (*CLR* also calls it the **iteration method**). This is a two-step strategy for solving recurrence equations:

1. Construct a recursion tree by unwinding the recurrence equation.
2. Determine the cost of the entire tree by summing the costs of the nodes.

Example 1: $T(n) = T(n - 1) + 1$

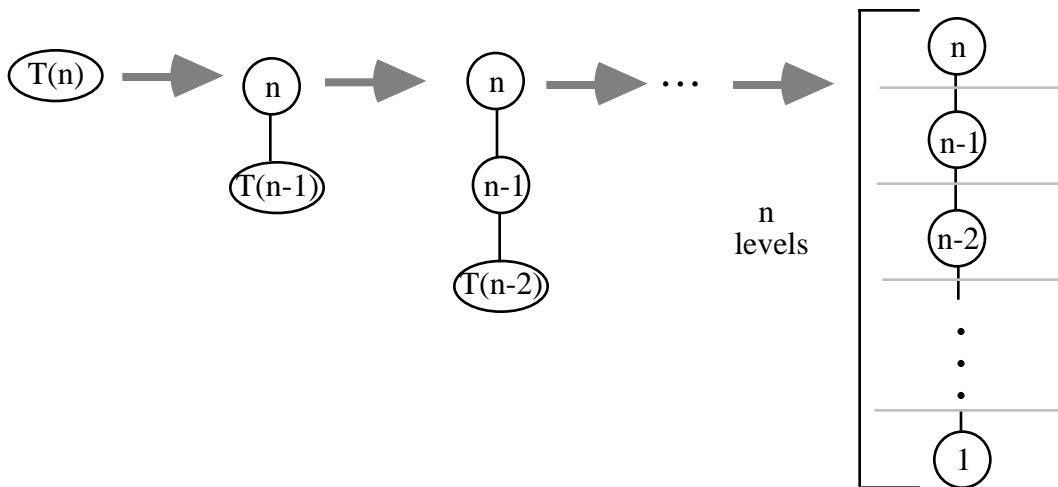
Also derivable: $T(n - 1) = T(n - 2) + 1$
 $T(n - 2) = T(n - 3) + 1$
 etc.



Total cost of nodes = number of nodes = n

Example 2: $T(n) = T(n - 1) + n$

Also derivable: $T(n - 1) = T(n - 2) + (n - 1)$
 $T(n - 2) = T(n - 3) + (n - 2)$
 etc.



Total cost of nodes = $1 + 2 + 3 + \dots + (n - 2) + (n - 1) + n = \sum_{k=1}^n k = \{\text{Arithmetic series; see below}\}$

Arithmetic Series

A series is arithmetic if $a_k = c + a_{(k-1)}$.

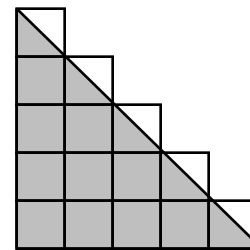
Trick: Sum corresponding pairs of numbers:



$$\sum_{k=1}^n a_k = \frac{n}{2}(a_1 + a_n)$$

Examples:

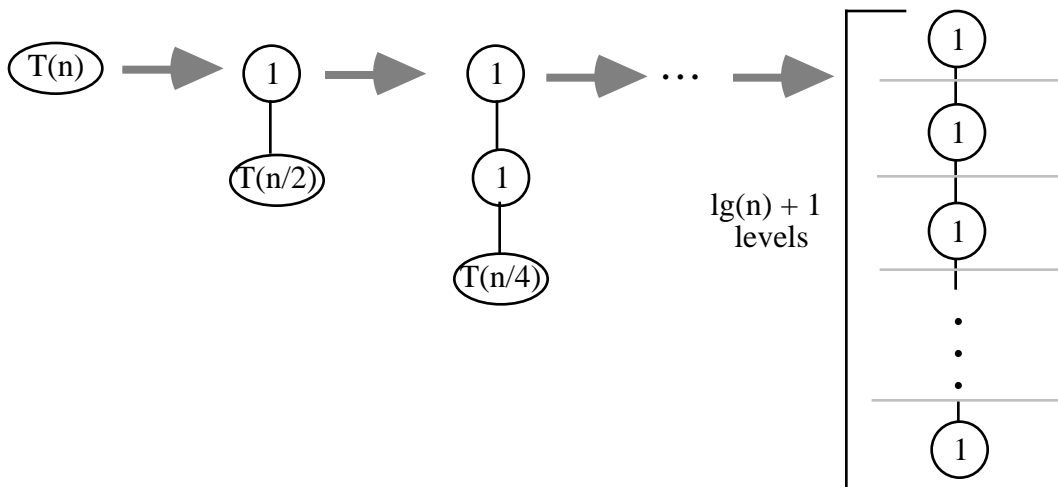
- $1 + 2 + 3 + \dots + n =$



- Sum of first n elements of series $7 + 10 + 13 + 16 + \dots$

Example 3: $T(n) = T(n/2) + 1$

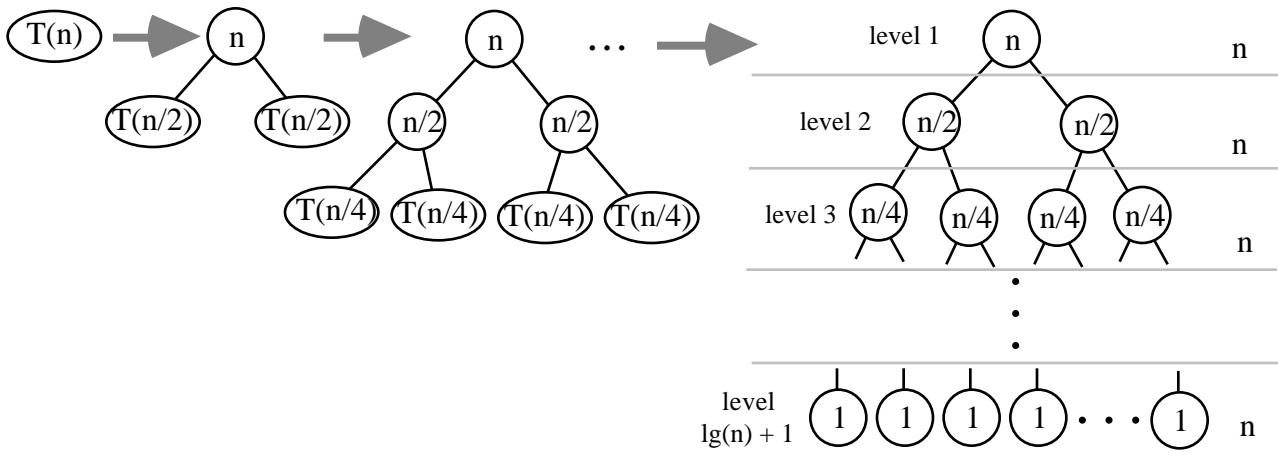
Also derivable: $T(n/2) = T(n/4) + 1$
 $T(n/4) = T(n/8) + 1$
 etc.



Total cost of nodes = number of nodes = $\lg(n) + 1 = (\lg(n))$

Example 4: $T(n) = 2T(n/2) + n$

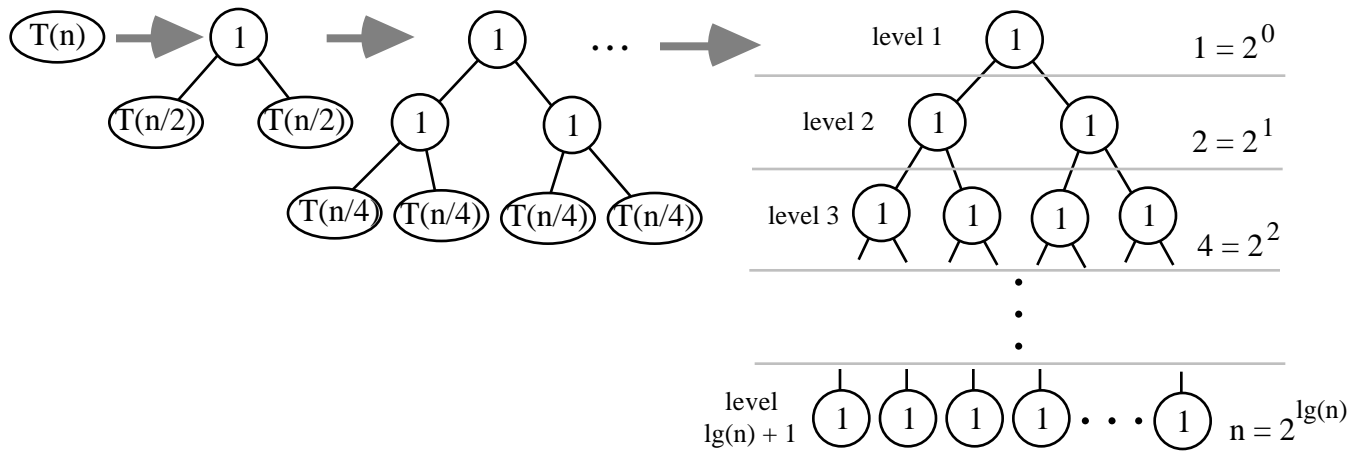
Also derivable: $T(n/2) = 2T(n/4) + n/2$
 $T(n/4) = 2T(n/8) + n/4$



Total cost = $(n)(\text{number of levels}) = n(\lg(n) + 1) = (n \lg(n))$

Example 5: $T(n) = 2T(n/2) + 1$

Also derivable: $T(n/2) = 2T(n/4) + 1$
 $T(n/4) = 2T(n/8) + 1$
 etc.



Total cost = number of nodes
 = sum of nodes at each level
 = $1 + 2 + 4 + 8 + \dots + 2^{\lg(n)}$ {Geometric series!}

$$= \sum_{k=0}^{\lg(n)} 2^k =$$

Geometric Series

A series is geometric if $a_k = c a_{(k-1)}$.

Let $S(n)$ stand for $\sum_{k=0}^n a_0 c^k = a_0 + a_0 c + a_0 c^2 + a_0 c^3 + \dots + a_0 c^n$

(Note carefully: $S(n)$ has $n+1$ terms, *not* n terms!)

Notice the following:

$$\begin{aligned} cS(n) &= a_0 c + a_0 c^2 + a_0 c^3 + \dots + a_0 c^n + a_0 c^{n+1} \\ - S(n) &= a_0 + a_0 c + a_0 c^2 + a_0 c^3 + \dots + a_0 c^n \end{aligned}$$

$$(c - 1) S(n) = a_0 c^{n+1} - a_0 = a_0(c^{n+1} - 1)$$

$$S(n) = \frac{a_0(c^{n+1} - 1)}{(c - 1)}$$

If $0 < c < 1$ and $n \rightarrow \infty$, the above formula can be rewritten as:

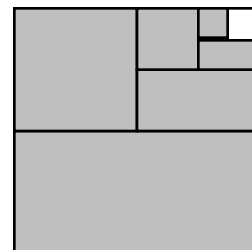
$$\lim_{n \rightarrow \infty} S(n) = \frac{a_0}{(1 - c)} \quad \text{if } 0 < c < 1$$

Examples:

$$1 + 2 + 4 + 8 + \dots + 2^n =$$

$$1 + 2 + 4 + 8 + \dots + 2^{\lg(n)} =$$

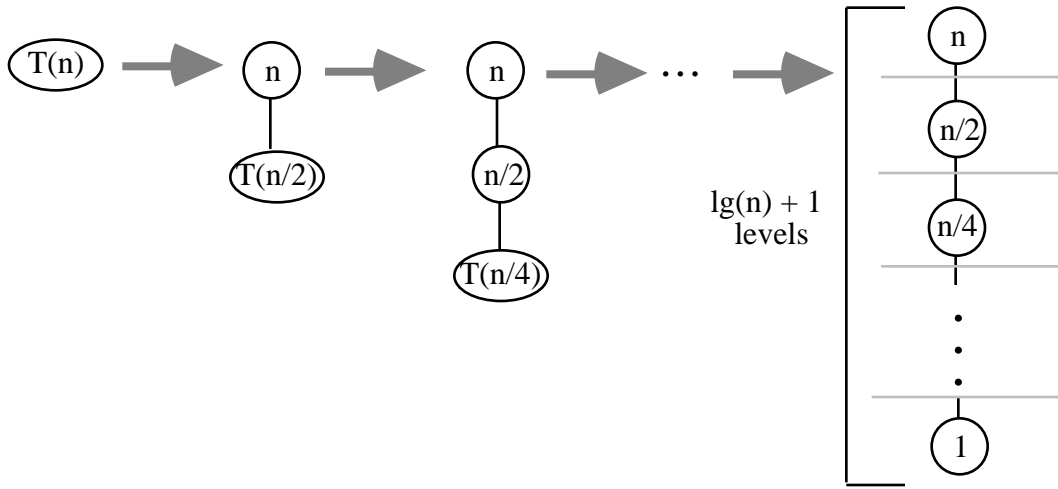
$$1/2 + 1/4 + 1/8 + \dots =$$



$$1/3 + 1/9 + 1/27 + \dots = \quad \text{{Paper tearing example}}$$

Example 6: $T(n) = T(n/2) + n$

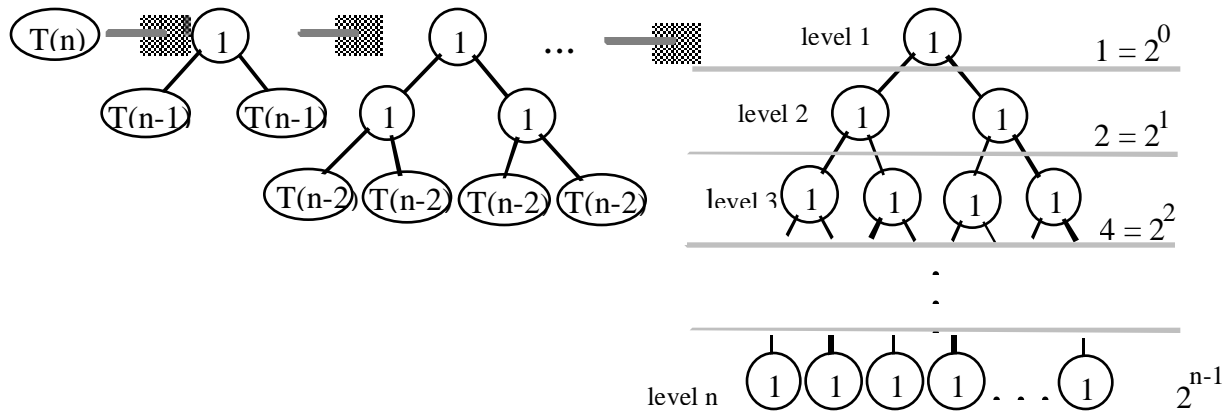
Also derivable: $T(n/2) = T(n/4) + n/2$
 $T(n/4) = T(n/8) + n/4$



$$\text{Total cost} = n + n/2 + n/4 + \dots + n/2^{\lg(n)} = \sum_{k=0}^{\lg(n)} \frac{n}{2^k} < \sum_{k=0}^{\lg(n)} \frac{n}{2^k} =$$

Example 7: $T(n) = 2T(n - 1) + 1$

Also derivable: $T(n-1) = 2T(n-2) + 1$
 $T(n-2) = 2T(n-3) + 1$



Total cost = $1 + 2 + 4 + \dots + 2^{n-1} =$

Solving Recurrence Equations: The Substitution Method

This is an alternative method for solving recurrence equations we shall see several times in this course.

In the substitution method, we guess a solution involving various coefficients, and determine the coefficients that lead to a solution (if any).

Example 1: $T(n) = T(n - 1) + 1$

Assume solution has the form $T(n) = an + b$:

What if we assumed the solution had form $T(n) = an^2 + bn + c$?

Example 2: $T(n) = T(n - 1) + n$

Assume solution has the form $T(n) = an^2 + bn + c$:

What if we assumed the solution had form $T(n) = an + b$?

Example 3: $T(n) = 2T((n - 1)/2) + n$

Here an exact solution is tricky. Instead assume $T(n) = a \lg(n)$, where $n \geq 1$.
What must be true of a ?

Summary

The following table summarizes and generalizes the above examples. Equivalence classes of functions are arranged in the table from "biggest" to "smallest". That is, if function f appears above function g in the table, then g is $O(f)$ but f is not $O(g)$ (or, equivalently, f is $\Omega(g)$ but g is not $\Omega(f)$). It is worth noting that there are many more equivalence classes than are listed in the table, but the ones in the table are the ones most commonly encountered in this course.

Equivalence class of functions	Name	Typical Recurrence Equations
(3^n)	exponential (base = 3)	$T(n) = 3 \cdot T(n - a) + b, a > 0, b > 0$
(2^n)	exponential (base = 2)	$T(n) = 2 \cdot T(n - a) + b, a > 0, b > 0$
(n^2)	quadratic	$T(n) = T(n - a) + b \cdot n, a > 0, b > 0$
$(n \log(n))$	$n \log(n)$	$T(n) = k \cdot T((n-a)/k) + b \cdot n, k > 1, a \geq 0, b > 1$
(n)	linear	$T(n) = T(n - a) + b, n > 0, a > 0, b > 0$ or $T(n) = k \cdot T((n-a)/k) + b, k > 1, a \geq 0, b > 0$ or $T(n) = T((n-a)/k) + b \cdot n, k > 1, a \geq 0, b > 1$
$(\log(n))$	logarithmic (base is irrelevant)	$T(n) = T(n/k) + a, k > 1, a > 0$
(1)	constant	$T(n) = a$