RECURRANCES AND SUMMATIONS

This handout summarizes highlights of 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 ≥ 1)
T(n) = [# of subproblems] T (size of each subproblem) + [cost of divide & glue]

Base Case (n < 1)
T(n) = 0

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

Deriving Recurrence Equations

<table>
<thead>
<tr>
<th>Insert an element into a sorted list of length n</th>
<th>T(n) = _____T(_______) + ________</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion-sort a list of length n.</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
<tr>
<td>Binary search on an array of n elements.</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
<tr>
<td>Find the maximum of the leaves of a balanced binary tree with n leaves.</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
<tr>
<td>Merge-sort a list of n elements</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
<tr>
<td>Find the maximum element of array A[0..n-1] by iteratively setting A[i] to max(A[2i], A[2i+1]) for lg(n) iterations</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
<tr>
<td>Towers of Hanoi</td>
<td>T(n) = _____T(_______) + ________</td>
</tr>
</tbody>
</table>
Solving Recurrence Equations: The Substitution Method

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 form \( an + b \)

**Example 2:** \( T(n) = T(n - 1) + n \)

Assume solution has form \( an^2 + bn + c \):

What if we assumed solution had form \( an + b \)?
Solving Recurrence Equations: The Iteration Method

The **iteration method** is a two-step strategy for solving recurrence equations:

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

There are other methods for solving recurrence equations; see CLR for details. However, we will use the iteration method almost exclusively.

---

**Example 3: T(n) = T(n - 1) + 1**

Also derivable:

\[
T(n - 1) = T(n - 2) + 1 \\
T(n - 2) = T(n - 3) + 1 \\
etc.
\]

![Recursion Tree Diagram](image)

Total cost of nodes = number of nodes = \( n \)
Example 4: \( T(n) = T(n - 1) + n \)

Also derivable: 
\[
T(n - 1) = T(n - 2) + (n - 1) \\
T(n - 2) = T(n - 3) + (n - 2) \\
e tc.
\]

Total cost of nodes = \( 1 + 2 + 3 + \ldots + (n - 2) + (n - 1) + n \) \{Arithmetic series; see below\}

\[
= \sum_{k=1}^{n} k = \frac{n(n+1)}{2}
\]

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

Also derivable: 
\[
T(n/2) = T(n/4) + 1 \\
T(n/4) = T(n/8) + 1 \\
e tc.
\]

Total cost of nodes = number of nodes = \( \lg(n) + 1 = \Theta(\lg(n)) \)
(We start with one \( T(n) \) node and generate a number of subnodes equal to the number of times \( n \) can be successively divided by 2. The latter quantity is \( \lg(n) \); including the 1 generated by the original \( T(n) \) node gives \( \lg(n) + 1 \) nodes.)
Example 6: \( T(n) = 2T(n/2) + 1 \)

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
= \( 1 + 2 + 4 + 8 + \ldots + 2^{\lg(n)} \) \{Geometric series!\}

\[
\sum_{k=0}^{\lg(n)} 2^k = 2^{\lg(n)} - 1
\]

Example 7: \( T(n) = 2T(n/2) + n \)

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

Total cost = \( (n)(\text{number of levels}) = n(\lg(n) + 1) = \Theta(n \cdot \lg(n)) \)
Example 8: $T(n) = T(n/2) + n$

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

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

Example 9: $T(n) = 2T(n - 1) + 1$

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

What is the picture?

What is the sum?
Summations

Sigma notation:
\[ \sum_{j=1}^{n} a_j = a_1 + a_2 + \ldots + a_n = \sum_{k=1}^{n} a_k \]

Linearity Laws:
\[ \sum_{k=1}^{n} c \cdot a_k = c \cdot \sum_{k=1}^{n} a_k \]
\[ \sum_{k=1}^{n} (a_k + b_k) = \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \]

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:
- \( 1 + 2 + 3 + \ldots + n = \)
- Sum of first \( n \) elements of series \( 7 + 10 + 13 + 16 + \ldots \)
Geometric Series

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

Let \( S(n) \) stand for \( \sum_{k=0}^{n} a_1 \cdot c^k = a_0 + a_0 c + a_0 c^2 + a_0 c^3 + ... + a_0 c^n \)

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

Notice the following:

\[
cS(n) = a_0 c + a_0 c^2 + a_0 c^3 + ... + a_0 c^{n+1}
\]

\[
- S(n) = a_0 + a_0 c + a_0 c^2 + a_0 c^3 + ... + a_0 c^n
\]

\[
(c - 1) S(n) = a_0 c^{n+1} - a_0 = a_0(c^{n+1} - 1)
\]

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

If \( 0 < c < 1 \) and \( n \to \infty \), the above formula can be rewritten as:

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

Examples:

\[
1 + 2 + 4 + 8 + ... + 2^n =
\]

\[
1 + 2 + 4 + 8 + ... + 2^\log(n) =
\]

\[
1/2 + 1/4 + 1/8 + .... =
\]

\[
1/3 + 1/9 + 1/27 + ... =
\]