# COMP 460 Lecture Notes

R. I. Greenberg

## 1 Divide-and-Conquer Approach to Algorithm Design

Insertion sort used an incremental approach. Now we look at another approach in which an algorithm solves a problem by recursively calling itself to solve related subproblems.

**Ex.: Merge Sort**:

- Divide $n$-element sequence into two subsequences of $n/2$ elements each.
- Sort the two subsequences recursively using merge sort.
- Merge the two sorted subsequences to produce the answer.

The recursion "bottoms out" when we encounter a sequence of length 1; then, just return it.

We can do the merging procedure in $\Theta(n)$ time.

We can write a *recurrence* for the running time $T(n)$ of merge sort:

$$T(n) = \begin{cases} \Theta(1) & \text{for } n = 1 \\ \underbrace{2T(n/2)}_{\substack{\text{sorting} \\ \text{2 subarrays}}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{for } n > 1 \end{cases}$$

(Assume for now that $n$ is a power of 2, so we don't have to worry about uneven divisions.)

We will see $T(n)$ is $\Theta(n \lg n)$. So merge sort is better than insertion sort in the worst case.

## 2 Asymptotic Notation

- Functions that are $O(g(n))$ are defined by:

$$O(g(n)) = \{\, f(n) : \exists c, n_0 > 0 \text{ s.t. } 0 \le f(n) \le cg(n) \forall n \ge n_0 \,\}$$

- Functions that are $\Omega(g(n))$ are defined by:

$$\Omega(g(n)) = \{\, f(n) : \exists c, n_0 > 0 \text{ s.t. } 0 \le cg(n) \le f(n) \forall n \ge n_0 \,\}$$

- $f(n)$ is $\Theta(g(n))$ if and only if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

- Exs. relating to insertion sort: It is easy to see that insertion sort has $O(n^2)$ worst-case running time:

  - At most $n$ values of $j$, and, for each one, at most $n$ values of $i$.
  - Constant amt. of work for each of the $n^2$ pairs of values for $i$ and $j$.

  $O(n^2)$ worst-case implies $O(n^2)$ in general.

  $\Theta(n^2)$ worst-case does not imply $\Theta(n^2)$ always. (We saw $\Theta(n)$ best case.)

  Best case is $\Omega(n)$ implies always $\Omega(n)$.

  Our book will not say insertion-sort is $\Omega(n^2)$, because $\exists$ inputs for which time is $\Theta(n)$. But can still say *worst-case* running time is $\Omega(n^2)$, and many writers make such statements with "worst-case" being implicit.

- Other notations:

  $f(n)$ is $o(g(n))$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (and $f$, $g$ asymp. nonneg.)

  $f(n)$ is $\omega(g(n))$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ (and $f$, $g$ asymp. nonneg.)

  We can also define these two notations in the style used above for $O$ and $\Omega$ by just changing "$\exists c$" to "$\forall c$" (still with $\exists n_0$).

# 3 Recurrences

Three solution techniques:

- Substitution: Guess a soln., and prove it by induction.

- Recursion-tree method (referred to as "iteration" in CLR 1st ed.)

  An example appears in the text.

- Master method for recurrences of the form

$$T(n) = aT(n/b) + f(n) \qquad a \ge 1, b > 1\ .$$

  (The method also works with "$T(\lfloor n/b \rfloor)$" or "$T(\lceil n/b \rceil)$" instead of "$T(n/b)$".)

  Three cases (incorporating result of a textbook exercise into case 2):

1. If $\exists\varepsilon > 0$ such that $f(n) = O(n^{\log_b a-\varepsilon})$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$, with $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
3. If $\exists\varepsilon > 0$ such that $f(n) = \Omega(n^{\log_b a+\varepsilon})$, then $T(n) = \Theta(f(n))$.

(In case 3, we also need a regularity condition, but as a practical matter, it will not be an issue for the types of functions we'll look at in this course.)

Note: Sometimes none of the 3 cases apply; then another solution method must be used.