Recurrences

Introduction

- A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.
  Eg: For example, the worst-case running time T (n) of the MERGE-SORT procedure could be described by the recurrence:

$$T(n) = \begin{cases} \Theta(1) & if \;\; n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & if \;\; n > 1 \end{cases} \tag{1}$$

- Here, we will discuss 3 methods for solving recurrences:
  (Further, it should be noted that "solving" means obtaining asymptotic "Θ" or "O" bounds on the solution.)
  1) substitution method:
     (guess a bound and then use mathematical induction to prove our guess correct)
  2) recursion-tree method:
     (converts the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion; use techniques for bounding summations to solve the recurrence)
  3) master method:
     (It provides bounds for recurrences of the form T (n) = aT (n/b) + f (n), where a ≥ 1, b > 1, and f (n) is a given function; it requires memorization of three cases.

A few Assumptions:

In practice, certain technical details are neglected when we state and solve recurrences:

- integer arguments to functions ⇒ Normally, the running time T (n) of an algorithm is only defined when n is an integer, since for most algorithms, the size of the input is always an integer.
  The actual recurrence for the worst-case running time T (n) of the MERGE-SORT procedure will be:

$$T(n) = \begin{cases} \Theta(1) & if \;\; n = 1 \\ T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + \Theta(n) & if \;\; n > 1 \end{cases} \tag{2}$$

- we will ignore statements of the boundary conditions of recurrences and assume that T (n) is constant for small n.
  Eg: For example, we normally write recurrence (1) as follows:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

- we often omit floors, ceilings, and boundary conditions.

<u>The master method:</u>

- It provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

  The above recurrence describes the running time of an algorithm that divides a problem of size *n* into *a* subproblems, each of size *n/b*, where *a* and *b* are positive constants, i.e. where $a \geq 1$ and $b > 1$ are constants and *f (n)* is an asymptotically positive function.

- The a subproblems are solved recursively, each in time T (n/b). The cost of dividing the problem and combining the results of the subproblems is described by the function f (n).
- If we interpret n/b to mean either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$, then T (n) can be bounded asymptotically as follows:

  1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- Discussion:
  - In each of the three cases, we are comparing the function *f (n)* with the function $n^{\log_b a}$
  - the solution to the recurrence is determined by the larger of the two functions.
  - **<u>Case 1</u>** the function $n^{\log_b a}$ is the larger, then the solution is T (n) = $\Theta\ (n^{\log_b a})$

    <u>Case 3</u> the function f (n) is the larger, then the solution is T (n) = Θ (f (n)).

    <u>Case 2</u> the two functions are the same size, we multiply by a logarithmic factor, and the solution is:

$$T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(f(n) lgn\right)$$

- In the first case, not only must f (n) be smaller than $n^{\log_b a}$, it must be polynomially smaller. That is, f (n) must be asymptotically smaller than $n^{\log_b a}$ a by a factor of n^{ϵ} for some constant $\epsilon > 0$.
- In the third case, not only must f (n) be larger than $n^{\log_b a}$, it must be polynomially larger. Further, it must satisfy the "regularity" condition that a f (n/b) ≤ cf (n). (This condition is satisfied by most of the polynomially bounded functions that we shall encounter)

- It is important to realize that the three cases do not cover all the possibilities for f (n). There is a gap between cases 1 and 2 when f (n) is smaller than $n^{\log_b a}$ but not polynomially smaller. Similarly, there is a gap between cases 2 and 3 when f (n) is larger than $n^{\log_b a}$ but not polynomially larger. If the function f (n) falls into one of these gaps, or if the regularity condition in case 3 fails to hold, the master method cannot be used to solve the recurrence.

Example 1: $T(n) = 9T\left(\frac{n}{3}\right) + n$

Example 2: $T(n) = T\left(\frac{2n}{3}\right) + 1$

Example 3: $T(n) = 3T\left(\frac{n}{4}\right) + n\,lgn$

Example 4: $T(n) = 2T\left(\frac{n}{2}\right) + n\,lgn$