CISC320 Algorithms, Spring 2010
Recurrence Relations, Master Theorem

Big-O upper bounds on functions defined by a recurrence may be determined from a big-O bounds on their parts. Here is a key theorem, particularly useful when estimating the costs of divide and conquer algorithms.

Master Theorem (for divide and conquer recurrences):
Let T(n) be a function defined on positive n, and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n/b) + f(n), & n > 1, \end{cases}$$

for some constants $c, a > 0, b > 1, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } d > \log_b(a), \\ O(n^d \log(n)), & \text{if } d = \log_b(a), \\ O(n^{\log_b(a)}), & \text{if } d < \log_b(a). \end{cases}$$

We will discuss many applications of the Master theorem. First for the sake of comparison, here is a theorem with similar structure, useful for the analysis of some algorithms. Jokingly, call it the

Muster Theorem (for subtract and conquer recurrences):
Let $T(n)$ be a function defined on positive n, and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases},$$

for some constants $c, a > 0, b > 0, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } a < 1, \\ O(n^{d+1}), & \text{if } a = 1, \\ O(n^d a^{n/b}), & \text{if } a > 1. \end{cases}$$

□

Proof: Expanding the recursion by one step, we have

$$\begin{aligned} T(n) &\leq a(\ T(n-2b) + f(n-b)\ ) + f(n) \\ &= a^2 T(n-2b) + af(n-b) + f(n). \end{aligned}$$

Taking another step, we get

$$T(n) \leq a^3 T(n-3b) + a^2 f(n-2b) + af(n-b) + f(n),$$

etc. This leads to $T(n) \leq \sum_{i=0}^{n/b} a^i f(n-ib)$ plus a constant. Since $f(n-ib)$ is in $\mathrm{O}(n-ib)$ which is in $\mathrm{O}(n)$, we have that $T(n)$ is in $O(n^d \sum_{i=0}^{n/b} a^i)$. The homework problem 0.2 finishes the job.

Remark: This theorem is written to reveal a similarity to the Master theorem. The first case is there for the sake of similarity. It doesn't occur in algorithm analysis, since if a is the number of recursive calls, $a < 1$ implies no recursive calls and no need for the theorem. The second case arises often. Insertion sort and `modexp()` are two examples at hand. The third case is not so common, but applies for instance to the iconic Towers of Hanoi problem. We give it a quick sketch.

function HanoiPuzzle(n)
Input: n = height of stack of disks to be moved from one peg to another.
Rules of puzzle: There are three pegs and a stack of n disks on one of the pegs. In the stack each disk has smaller radius than the one below it. The goal of the game is to move the stack to another peg. Each move consists in moving a disk from one peg to another. A disk may never be put on top of a smaller disk.
Output: puzzle solved - stack is moved.
if $n = 0$, return
HanoiPuzzle($n - 1$) *[Move n-1 disks to another peg following rules of the game.]*
Move one disk *[Move the largest disk to the open peg (a legal move).]*
HanoiPuzzle($n - 1$) *[Move n-1 disks so as to end up on top of the largest disk.]*

For T(n) being the number of moves to solver HanoiPuzzle(n), the recurrence is

$$T(n) = 2T(n-1) + 1.$$

In this case $a = 2, b = 1, d = 0$, and the theorem tells us we have 2^n disk moves necessary to solve the Towers of Hanoi puzzle.