8.3 Divide-and-Conquer Algorithms and Recurrence Relations

Divide and Conquer Algorithm: A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. - Wikipedia

Theorem 1: Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + c$$

whenever n is divisible by b, where $a \geq 1$, b is an integer greater than 1, and c is a positive real number. Then

$$f(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log n) & \text{if } a = 1 \end{cases}$$

Furthermore, when $n = b^k$ and $a \neq 1$, where k is a positive integer,

$$f(n) = C_1 n^{\log_b a} + C_2$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

Master Theorem: Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

8.3 pg. 535 # 9

Suppose that $f(n) = f(n/5) + 3n^2$ when n is a positive integer divisible by 5, and $f(1) = 4$. Find

a $f(5)$.

Just have to work up to the desired number.

$f(5) = f(5/5) + 3(5)^2$
$= f(1) + 3(25)$
$= 4 + 75$
$= 79$

b $f(125)$.

$f(25) = f(25/5) + 3(25)^2$
$= f(5) + 3(625)$

$= 79 + 1875$
$= 1954$

$f(125) = f(125/5) + 3(125)^2$
$= f(25) + 3(15625)$
$= 1954 + 46875$
$= 48829$

c $f(3125)$.

$f(625) = f(625/5) + 3(625)^2$
$= f(125) + 3(390625)$
$= 48829 + 1171875$
$= 1220704$

$f(3125) = f(3125/5) + 3(3125)^2$
$= f(625) + 3(9765625)$
$= 1220704 + 29296875$
$= 30517579$

8.3 pg. 535 # 11

Give a big-O estimate for the function $f(n) = f(n/2)+1$ if f is an increasing function and $n = 2^k$.

Use Master Theorem with $a = 1, b = 2, c = 1, d = 0$. Since $a = b^d$, we know that $f(n)$ is $O(n^d \log n) = O(\log n)$.

8.3 pg. 535 # 13

Give a big-O estimate for the function $f(n) = 2f(n/3) + 4$ if f is an increasing function and $n = 3^k$.

Use Master Theorem with $a = 2, b = 3, c = 4, d = 0$. Since $a > b^d$, we know that $f(n)$ is $O(n^{\log_b a}) = O(n^{\log_3 2})$.

8.3 pg. 535 # 17

Suppose that the votes of n people for different candidates (where there can be more than two candidates) for a particular office are the elements of a sequence. A person wins the election if this person receives a majority of the votes.

a Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and, if so, determine who this candidate is. [*Hint:* Assume that n is even and split the sequence of votes into two sequences, each with $n/2$ elements. Note that a candidate could not received a majority of votes without receiving a majority of votes in at least one of the two halves.]

We will use the hint to devise our algorithm. We first note that our base case is that a sequence with one element means that the one person on the list is the winner. For our recursive step,

we will divide the list into two equal parts and count which name occurs the most in the two parts. We know that the winner requires a majority of votes and that would mean that the winner will need at least half+1 of a part to be his/her name. Keep applying this recursive step to each half and we will eventually have only at most two names in the list. Count the number of occurrences in the whole list of the two remaining names and this will decide the winner. This will only require at most $2n$ additional comparisons for a list of length n.

b Use the master theorem to give a big-O estimate for the number of comparisons needed by the algorithm you devised in part (a).

Our function is $f(n) = 2f(n/2) + 2n$. So $a = 2, b = 2, c = 2, d = 1$. By the Master theorem, $a = b^d$ so, the big-O is $O(n^d \log n) = O(n \log n)$.