Ch. 4: Divide-and-Conquer

● in divide-and-conquer, we solve a problem recursively, applying three steps at each level of recursion:

− Divide the problem into a number of subproblems that are smaller instances of the same problem

− Conquer the subproblems by solving them recursively; if the subproblems are small enough, however, just solve the subproblems in a straightforward manner

− Combine the solutions to the subproblems into the solution for the original problem

●

**Recurrences**

● a recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs

● T(n) = T(n – 1) + θ(1)

the above recurrence means, each recursive call would take constant time plus the time for the recursive calls it makes

● in this Ch there are three methods for solving recurrences – that is, for obtaining asymptotic “θ” or “O” bounds on the solution:

1. substitution method

♦ we 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; we use techniques for bonding summations to solve the recurrence

3. master method

♦ provides bonds for recurrences of the form T(n) = a T(n / b) + f(n) where , and f(n) is a given function

♦ the equation characterizes a divide-and-conquer algorithm that creates a subproblems, each of which is 1/b the size of the original problem, and in which the divide and combine steps together take f(n) time

● recurrences that are represented by inequalities, such as T(n) 2 T(n / 2) + θ(n), it only has an upper bound so its solution is O-notation (big-oh notation)

**The maximum-subarray problem**

**Strassen’s algorithm for matrix multiplication**

● Pseudocode for square-matrix-multiply-recursive



● the running time of SQUARE-MATRIX-MULTIPLY\_RECURSIVE:

If n=1, has a runtime of (as shown in the pseudocde)

b/c each recursive call multiples two n/2 x n/2 matrices, T(n/2)

b/c each of the four matrix additions (lines 6-9) takes and the number of additions is constant

b/c there are 8 of these

So



And from the master method, we can tell the above equation has the solution T(n) =

●

**Strassen’s method**