for the rod-cutting problem, and as we shall see in Section 14.3, this recursive algorithm takes exponential time. That's no better than the brute-force method of checking each way of parenthesizing the product.

Fortunately, there aren't all that many distinct subproblems: just one subproblem for each choice of i and j satisfying $1 \le i \le j \le n$, or $\binom{n}{2} + n = \Theta(n^2)$ in all.[4] A recursive algorithm may encounter each subproblem many times in different branches of its recursion tree. This property of overlapping subproblems is the second hallmark of when dynamic programming applies (the first hallmark being optimal substructure).

Instead of computing the solution to recurrence (14.7) recursively, let's compute the optimal cost by using a tabular, bottom-up approach, as in the procedure MATRIX-CHAIN-ORDER. (The corresponding top-down approach using memoization appears in Section 14.3.) The input is a sequence $p = \langle p_0, p_1, \ldots, p_n \rangle$ of matrix dimensions, along with n, so that for $i = 1, 2, \ldots, n$, matrix A_i has dimensions $p_{i-1} \times p_i$. The procedure uses an auxiliary table $m[1:n, 1:n]$ to store the $m[i, j]$ costs and another auxiliary table $s[1:n-1, 2:n]$ that records which index k achieved the optimal cost in computing $m[i, j]$. The table s will help in constructing an optimal solution.

```
MATRIX-CHAIN-ORDER(p, n)
 1  let m[1:n, 1:n] and s[1:n − 1, 2:n] be new tables
 2  for i = 1 to n                          // chain length 1
 3      m[i, i] = 0
 4  for l = 2 to n                          // l is the chain length
 5      for i = 1 to n − l + 1              // chain begins at A_i
 6          j = i + l − 1                   // chain ends at A_j
 7          m[i, j] = ∞
 8          for k = i to j − 1              // try A_{i:k} A_{k+1:j}
 9              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10              if q < m[i, j]
11                  m[i, j] = q             // remember this cost
12                  s[i, j] = k             // remember this index
13  return m and s
```

In what order should the algorithm fill in the table entries? To answer this question, let's see which entries of the table need to be accessed when computing the

[4] The $\binom{n}{2}$ term counts all pairs in which $i < j$. Because i and j may be equal, we need to add in the n term.