begin
1. for i ← 1 until n do m_{ii} ← 0;
2. for l ← 1 until n - 1 do
3. for i ← 1 until n - l do
   begin
   j ← i + l;
5. m_{ij} ← \text{MIN}_{i \leq k < j} (m_{ik} + m_{k+1,i} + r_{l-1} \ast r_k \ast r_j)
   end;
6. write m_{1n}
end

Fig. 2.15. Dynamic programming algorithm for ordering matrix multiplications.

\begin{tabular}{|c|c|c|c|}
\hline
 & 1 & 2 & 3 \\
\hline
1 & m_{11} = 0 & m_{22} = 0 & m_{33} = 0 & m_{44} = 0 \\
\hline
2 & m_{12} = 10,000 & m_{23} = 1000 & m_{34} = 5000 & \\
\hline
3 & m_{13} = 1200 & m_{24} = 3000 & \\
\hline
4 & m_{14} = 2200 & \\
\hline
\end{tabular}

Fig. 2.16. Costs of computing products $M_i \times M_{i+1} \times \cdots \times M_j$.

required to evaluate the product is 2200. An order in which the multiplica-
tions may be done can be determined by recording, for each table entry, a
value of $k$ which gives rise to the minimum seen in (2.9). □

2.9 EPILOGUE

This chapter has touched upon a number of fundamental techniques used in
efficient algorithm design. We have seen how high-level data structures such
as lists, queues, and stacks allow the algorithm designer to remove himself
from such mundane chores as manipulating pointers and permit him to focus
on the overall structure of the algorithm itself. We have also seen how the
powerful techniques of recursion and dynamic programming often lead to
elegant and natural algorithms. We also presented certain general principles
such as divide-and-conquer and balancing.

These techniques are certainly not the only tools available but they are
among the more important. As we progress through the remainder of this
book, we shall encounter a number of other techniques. These will range