We can verify that the worst-case data for Select3 are $a[i] = n + 1 - i$, for $1 \le i \le n$, and $k = \frac{n}{2}$. The computing time for Select3 is relatively insensitive to the input permutation. This permutation affects only the number of times the second **if** statement of Algorithm 3.20 is executed. On the average, this will be done about half the time. This can be achieved by using $a[i] = n + 1 - i$, $1 \le i \le n/2$, and $a[i] = n + 1$, $n/2 < i \le n$. The k value needed to obtain the average computing time is readily seen to be $n/4$.

(a) What test data would you use to determine worst-case and average times for Select4?

(b) Use the ideas above to obtain a table of worst-case and average times for Select1, Select2, Select3, and Select4.

16. Program Select1 and Select3. Determine when algorithm Select1 becomes better than Select3 on the average and also when Select2 better than Select3 for worst-case performance.

17. [Project] Program the algorithms of Exercise 4 as well as Select3 and Select4. Carry out a complete test along the lines discussed in Exercise 15. Write a detailed report together with graphs explaining the data sets, test strategies, and determination of $c_1, \ldots, c_4$. Write the final composite algorithms and give tables of computing times for these algorithms.

3.7 STRASSEN'S MATRIX MULTIPLICATION

Let A and B be two $n \times n$ matrices. The product matrix $C = AB$ is also an $n \times n$ matrix whose i, jth element is formed by taking the elements in the ith row of A and the jth column of B and multiplying them to get

$$C(i,j) = \sum_{1 \le k \le n} A(i,k)B(k,j) \qquad (3.10)$$

for all i and j between 1 and n. To compute $C(i,j)$ using this formula, we need n multiplications. As the matrix C has n^2 elements, the time for the resulting matrix multiplication algorithm, which we refer to as the conventional method is $\Theta(n^3)$.

The divide-and-conquer strategy suggests another way to compute the product of two $n \times n$ matrices. For simplicity we assume that n is a power of 2, that is, that there exists a nonnegative integer k such that $n = 2^k$. In case n is not a power of two, then enough rows and columns of zeros can be added to both A and B so that the resulting dimensions are a power of two (see the exercises for more on this subject). Imagine that A and B are each partitioned into four square submatrices, each submatrix having dimensions $\frac{n}{2} \times \frac{n}{2}$. Then the product AB can be computed by using the above formula for the product of 2×2 matrices: if AB is

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \tag{3.11}$$

then

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned} \tag{3.12}$$

If $n = 2$, then formulas (3.11) and (3.12) are computed using a multiplication operation for the elements of A and B. These elements are typically floating point numbers. For $n > 2$, the elements of C can be computed using *matrix* multiplication and addition operations applied to matrices of size $n/2 \times n/2$. Since n is a power of 2, these matrix products can be recursively computed by the same algorithm we are using for the $n \times n$ case. This algorithm will continue applying itself to smaller-sized submatrices until n becomes suitably small ($n = 2$) so that the product is computed directly.

To compute AB using (3.12), we need to perform eight multiplications of $n/2 \times n/2$ matrices and four additions of $n/2 \times n/2$ matrices. Since two $n/2 \times n/2$ matrices can be added in time cn^2 for some constant c, the overall computing time $T(n)$ of the resulting divide-and-conquer algorithm is given by the recurrence

$$T(n) = \begin{cases} b & n \le 2 \\ 8T(n/2) + cn^2 & n > 2 \end{cases}$$

where b and c are constants.

This recurrence can be solved in the same way as earlier recurrences to obtain $T(n) = O(n^3)$. Hence no improvement over the conventional method has been made. Since matrix multiplications are more expensive than matrix additions ($O(n^3)$ versus $O(n^2)$), we can attempt to reformulate the equations for C_{ij} so as to have fewer multiplications and possibly more additions. Volker Strassen has discovered a way to compute the C_{ij}'s of (3.12) using only 7 multiplications and 18 additions or subtractions. His method involves first computing the seven $n/2 \times n/2$ matrices P, Q, R, S, T, U, and V as in (3.13). Then the C_{ij}'s are computed using the formulas in (3.14). As can be seen, P, Q, R, S, T, U, and V can be computed using 7 matrix multiplications and 10 matrix additions or subtractions. The C_{ij}'s require an additional 8 additions or subtractions.

$$\begin{aligned}
P &= (A_{11}+A_{22})(B_{11}+B_{22}) \\
Q &= (A_{21}+A_{22})B_{11} \\
R &= A_{11}(B_{12}-B_{22}) \\
S &= A_{22}(B_{21}-B_{11}) \\
T &= (A_{11}+A_{12})B_{22} \\
U &= (A_{21}-A_{11})(B_{11}+B_{12}) \\
V &= (A_{12}-A_{22})(B_{21}+B_{22})
\end{aligned} \tag{3.13}$$

$$\begin{aligned}
C_{11} &= P+S-T+V \\
C_{12} &= R+T \\
C_{21} &= Q+S \\
C_{22} &= P+R-Q+U
\end{aligned} \tag{3.14}$$

The resulting recurrence relation for $T(n)$ is

$$T(n) = \begin{cases} b & n \le 2 \\ 7T(n/2)+an^2 & n > 2 \end{cases} \tag{3.15}$$

where a and b are constants. Working with this formula, we get

$$\begin{aligned}
T(n) &= an^2[1+7/4+(7/4)^2+\cdots+(7/4)^{k-1}]+7^kT(1) \\
&\le cn^2(7/4)^{\log_2 n}+7^{\log_2 n}, \ c \text{ a constant} \\
&= cn^{\log_2 4+\log_2 7-\log_2 4}+n^{\log_2 7} \\
&= O(n^{\log_2 7}) \approx O(n^{2.81})
\end{aligned}$$

EXERCISES

1. Verify by hand that Equations 3.13 and 3.14 yield the correct values for C_{11}, C_{12}, C_{21}, and C_{22}.
2. Write an algorithm that multiplies two $n \times n$ matrices using $O(n^3)$ operations. Determine the precise number of multiplications, additions, and array element accesses.
3. If k is a nonnegative constant, then prove that the recurrence

$$T(n) = \begin{cases} k & n = 1 \\ 3T(n/2)+kn & n > 1 \end{cases} \tag{3.16}$$

   has the following solution (for n a power of 2):

$$T(n) = 3kn^{\log_2 3} - 2kn \tag{3.17}$$