ADS Tutorial 2 - Solutions*

Instructor: Heng Guo Tutors: Georgios Kalantzis and Dhairya Mangla

October 11, 2026

Problem 1

Use Strassen's algorithm to compute the matrix product

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 8 & 4 \\ 6 & 2 \end{pmatrix}.$$

(Taken from Cormen, Leiserson, Rivest, and Stein (CLRS), 2nd edition, exercise 28.2-1.)

Solution

This question is simply a matter of brute computation. **Note: There was a typo in the lecture slides shown in class. We should have $P_5 = (A_{11} + A_{12}) \cdot B_{22}$, not $P_5 = (A_{11} + A_{22}) \cdot B_{22}$. The correct version of P_5 is used below. The slides on the website is fixed now.** With the blocks named as in Lecture 4, we can set up the equations for $P_1, \cdots, P_7$ from the lecture:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) = (1+7)(8+2) = 80 \\
P_2 &= (A_{21} + A_{22})B_{11} = (5+7)8 = 96 \\
P_3 &= A_{11}(B_{12} - B_{22}) = 1(4-2) = 2 \\
P_4 &= A_{22}(B_{21} - B_{11}) = 7(6-8) = -14 \\
P_5 &= (A_{11} + A_{12})B_{22} = (1+3)2 = 8 \\
P_6 &= (-A_{11} + A_{21})(B_{11} + B_{12}) = (-1+5)(8+4) = 48 \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) = (3-7)(6+2) = -32
\end{aligned}$$

Thus, using the definitions of matrix entries C_{ij} from lecture, we have

$$\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7 = 80 + (-14) - 8 + (-32) = 26 \\
C_{12} &= P_3 + P_5 = 2 + 8 = 10 \\
C_{21} &= P_2 + P_4 = 96 + (-14) = 82 \\
C_{22} &= P_1 + P_3 - P_2 + P_6 = 80 + 2 - 96 + 48 = 34,
\end{aligned}$$

giving us an overall matrix product

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} 26 & 10 \\ 82 & 34 \end{pmatrix}.$$

As a quick sanity check, we can multiply the two matrices we're given the "normal" way to confirm that our solution is correct.

*The solutions contain additional explanations that are not necessary, if you were to answer such a question in an exam.

Problem 2

Describe an algorithm for efficiently multiplying a $(p \times q)$ matrix with a $(q \times r)$ matrix, where p, q, r are arbitrary positive integers. The running time should be $\Theta(n^{\log_2 7})$, where $n = \max\{p, q, r\}$.

Solution

We already have an $O(n^{\log_2 7})$ algorithm for multiplying two square matrices of size $N \times N$, where $N = 2^k$ for some integer $k \geq 1$. (This is just the STRASSEN algorithm.) For matrix multiplication of arbitrary dimension, we can still use the STRASSEN algorithm just as before; we simply need to add a pre- and post-processing step to get the matrices into the desired dimensions.

In particular, suppose our two matrices are of respective sizes $p \times q$ and $q \times r$. Let $n := \max\{p, q, r\}$ and define $N := 2^{\lceil \log_2 n \rceil}$; i.e., N is the first power of 2 greater than or equal to n. Now we can simply pad our original matrices with zeroes to be $N \times N$ matrices (for instance, by adding rows of 0's to the bottom and columns of 0's to the right of the each original matrix). Now we can directly apply the STRASSEN algorithm as before. Finally, to finish the multiplication, we remove the padded zeros from the final product to recover the desired $p \times r$ matrix product.

What's the runtime of this new algorithm? The pre-and post-processing stages can be accomplished in time $O(N^2)$, and we know from lecture that the runtime of the STRASSEN algorithm is $\Theta(N^{\log_2 7})$. To convert our runtimes from functions of N to functions of n, we simply note that since N is defined as the lowest power-of-2 upper bound on n, it must be true that $N < 2n$. This means that our runtimes above only change by a constant factor; thus, their asymptotic behaviour is the same. Therefore, the overall runtime is $\Theta(N^{\log_2 7}) = \Theta(n^{\log_2 7})$.

Observation: A tangential issue with respect to this algorithm is that for this general "rectangular" case it is NOT clear that this "reduce to STRASSEN" algorithm is often a good strategy. Suppose without loss of generality that $p = \max\{p, q, r\}$. Then the naïve matrix multiplication algorithm is $\Theta(pqr)$. Our asymptotic running-time from "reduce to STRASSEN" is only better if $qr \geq p^{\log_2 7 - 1} \sim p^{1.8}$, which is not necessarily the case in the "rectangular" setting.

Problem 3

Consider the Selection algorithm that we saw in the lectures, but with one of the following two modifications:

(a) Instead of splitting into groups of size 5, we split into groups of size 3.

(b) Instead of splitting into groups of size 5, we split into groups of size 7.

For each of the cases (a) and (b), prove an asymptotic bound on the running time of the algorithm using the analysis that was presented in the lectures. What do you observe?

Solution

(a) Groups of size 3. Just as we did in lecture, we can write a lower bound on the total number of elements in the array larger than the median of medians x. The only differences between the lower bound formula used in the lecture slides for partitions of size 5 and partitions of size 3:

- The total number of groups is $\lceil n/3 \rceil$ rather than $\lceil n/5 \rceil$.
- The number of elements $> x$ in each of the groups whose "baby median" exceeds the median of medians x is 2 when group size is 3. (The number 2 corresponds to the 1 element greater than that group's median, along with the median itself.)

Thus, our updated lower bound on the number of elements in the array larger than x is as follows:

$$2\left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{3} \right\rceil \right\rceil - 2\right) \geq \frac{n}{3} - 4.$$

Thus, the size of the lower subarray is at most $n - \left(\frac{n}{3} - 4\right) = \frac{2n}{3} + 4$. By symmetry, it should be the case that the upper subarray also has size at most $\frac{2n}{3} + 4$. Thus the subarray on which we recurse has size at most $|S_{\max}| \leq \frac{2n}{3} + 4$. This is something we can use in our recurrence relation.

Recall from lecture that the runtime of SELECTION is upper-bounded by the following recurrence relation:

$$T(n) \leq T(n/k) + T(|S_{\max}|) + bn,$$

where k is the partition size and b is some positive constant. Substituting our upper bound on $|S_{\max}|$ when $k = 3$ gives the recurrence relation

$$T(n) \leq T(n/3) + T(2n/3 + 4) + bn.$$

We could try "unrolling" this recurrence relation as we've done in the past; but the expansion quickly becomes very complicated. So instead, we use a bit of problem-specific knowledge. The goal of SELECTION is to find the kth smallest element of the input array. One way of approaching this problem is to simply sort then entire array and return its kth element, which would take time $O(n \log n)$. However, we're hopeful that, in developing a method that avoids sorting the entire array, we might be able to achieve a better runtime. We start with the optimistic guess that the runtime of our algorithm is $T(n) = O(n)$; i.e., that there exist constants c, $n_0 > 0$ such that $T(n) \leq cn$ for all $n \geq n_0$. Do such positive constants c and n_0 exist? Our next step is to try and find out.

Let's suppose for a moment that our guess is correct; i.e., that $T(n) \leq cn$ for some positive constant c. Then, substituting into our original recurrence relation, we have

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3} + 4\right) + bn \\ &\leq c \cdot \frac{n}{3} + c\left(\frac{2}{3}n + 4\right) + bn \\ &= cn + (4c + bn) \end{aligned}$$

Let's pause for a moment to take stock. Recall that our original guess is that $T(n) = O(n)$; i.e., there exist positive constants c and n_0 such that $T(n) \leq cn$ for all $n \geq n_0$. Looking at the last line in the expansion of $T(n)$ above, we find that this is true if and only if there exist positive constants c and n_0 such that the term in parenthesis, $4c + bn$, is at most zero. But for $c, b, n > 0$, the term $4c + bn$ is *always* positive. Thus, it would seem that $T(n)$ is *not* $O(n)$.

What should we do now? Let's step back for a moment. The reason we started substituting $T(n) \leq cn$ into our original recurrence relation was because we suspected (or hoped) that $T(n)$ was $O(n)$. We now know that this guess cannot be verified in this way. But since we know that the median of a list can be found by sorting the list in time $O(n \log n)$, we know that our algorithm's runtime shouldn't be *more* than $O(n \log n)$. Thus, we repeat the process above, but this time guessing that $T(n) = O(n \log n)$ (here $\log = \log_2$); i.e., that there exist positive constants c, n_0 such that $T(n) \leq cn \log n$ for all $n \geq n_0$.

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3} + 4\right) + bn \\ &\leq c\left(\frac{n}{3}\right)\log\left(\frac{n}{3}\right) + c\left(\frac{2n}{3} + 4\right)\log\left(\frac{2n}{3} + 4\right) + bn \\ &= cn\log n + \left(- c\left[n\log(3) + \frac{2n}{3}\log\left(\frac{n}{3}\right) - \left(\frac{2n}{3} + 4\right)\log\left(\frac{2n}{3} + 4\right)\right] + bn\right), \end{aligned}$$

where, while the last step looks complicated, it's simply an expansion of the term $c(n/3)\log(n/3)$ according to log rules in order to be able to write $cn \log n$ as the first term of the last line. To simplify notation, let's call the bracketed expression in the last line $g(n)$.

From the above, we can see that $T(n) \leq cn \log n$ if and only if $-c\,g(n) + bn \leq 0$. But due to the complexity of $g(n)$, it's hard to get a sense of this function's behaviour just from looking at this expression. Instead, we can simply plug it into a graphing calculator. We see graphically that $g(n) > 0$ for all $n \geq 25$. Thus, for $n \geq 25$ and $c \geq \frac{bn}{g(n)}$, we have that $T(n) \leq cn \log n$. In other words, $T(n) = O(n \log n)$. To fix a value of c in terms of b, we note (again, by graphing) that $\frac{n}{g(n)} \leq 2$ for all $n \geq 70$. So for all $n \geq 70$, $c \geq 2b$ suffices.

Now we need to prove our claim using mathematical induction. To prove that $T(n) = O(n \log n)$, we simply need to show that there exists some $c > 0$ such that $T(n) \leq cn \log n$ for all $n \geq 70$. But as it turns out, we can actually demonstrate this claim for *all* $n > 0$ without much extra work. So we do that below, for all $n \geq 2$ (for $n = 1$ we have $n \log n = 0$).

Claim. Let $T(n) \leq T(n/3) + T(2n/3 + 4) + bn$. Then there exists some constant $c > 0$ such that $T(n) \leq cn \log n$ for all $n \geq 2$.

Proof. First, we define

$$a := \max_{n \in \{2,3,\cdots,70\}} \left\{ T(n)/n \log n \right\},$$

and $c := \max\{a, 2b\}$.

Base case. Suppose $n \in \{2, 3, \cdots, 70\}$. Then by the definition of a, we know that $T(n) \leq an \log n$. And by the definition of c, it must be true that $a \leq c$. Thus, we have

$$T(n) \leq an \log n \leq cn \log n$$

for all $n \in \{2, 3, \cdots, 70\}$.

Induction step. Suppose the claim holds for all $n \leq 70 = k$. Then for $n = k + 1$, we have the following:

$$\begin{aligned}
T(n) &\leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3} + 4\right) + bn && \text{Our original recurrence relation} \\
&\leq c\left(\frac{n}{3}\right) \log \left(\frac{n}{3}\right) + c\left(\frac{2n}{3} + 4\right) \log \left(\frac{2n}{3} + 4\right) + bn && \text{Applying the induction hypothesis} \\
&= cn \log n + (-c\,g(n) + bn) && \text{Algebraic simplification (see above)} \\
&\leq cn \log n + \left(- c\,g(n) + \tfrac{1}{2}cn \right) && \text{By the definition of } c \\
&= cn \log n + c\left(\tfrac{1}{2}n - g(n)\right) && \text{Algebraic manipulation} \\
&\leq cn \log n,
\end{aligned}$$

where the last step comes from the fact that for all $n \geq 70$, we showed earlier that $\frac{n}{g(n)} \leq 2$; or, equivalently, $\frac{1}{2}n \leq g(n)$. □

Remark. The complexity of this variant, namely 3 in a group, is still open. The issue is that the recurrence may not be tight. A lower bound example is not known and apparently not easy to construct.

(b) Groups of size 7.

Following the same reasoning as in part (a), we have the following lower bound on the number of elements in the array larger than the "median of medians" x when partitions have size 7:

$$4\left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right) \geq \frac{2n}{7} - 8.$$

Thus, the size of the lower subarray is at most $n - \left(\frac{2n}{7} - 8\right) = \frac{5n}{7} + 8$. By symmetry, it should be the case that the upper subarray also has size at most $\frac{5n}{7} + 8$. Thus the subarray on which we recurse has size at most $|S_{\max}| \leq \frac{5n}{7} + 8$. This is something we can use in our recurrence relation.

Recall from lecture that the runtime of SELECTION is upper-bounded by the following recurrence relation:

$$T(n) \leq T(n/k) + T(|S_{\max}|) + bn,$$

where k is the partition size and b is some positive constant. Substituting our upper bound on $|S_{\max}|$ when $k = 7$ gives the recurrence relation

$$T(n) \leq T(n/7) + T(5n/7 + 8) + bn.$$

As in part (a), we start with the optimistic guess that the runtime of $T(n) = O(n)$; i.e., that there exist constants c, $n_0 > 0$ such that $T(n) \leq cn$ for all $n \geq n_0$. Do such positive constants c and n_0 exist? Our next step is to try and find out.

Let's suppose for a moment that our guess is correct; i.e., that $T(n) \leq cn$ for some positive constant c. Then, substituting into our original recurrence relation, we have

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7} + 8\right) + bn \\ &\leq c \cdot \frac{n}{7} + c\left(\frac{5}{7}n + 8\right) + bn \\ &= \frac{6}{7}cn + 8c + bn \\ &= cn + \left(-\frac{1}{7}cn + 8c + bn\right) \end{aligned}$$

Our original guess is that $T(n) = O(n)$; i.e., there exist positive constants c and n_0 such that $T(n) \leq cn$ for all $n \geq n_0$. Looking at the last line in the expansion of $T(n)$ above, we find that this is true if and only if there exist positive constants c and n_0 such that the term in parenthesis, $-\frac{1}{7}cn + 8c + bn$, is at most zero. Let's see if we can find such constants. Through a bit of algebraic manipulation, we find that

$$-\frac{1}{7}cn + 8c + bn \leq 0 \iff c \geq 7b\frac{n}{n - 56}.$$

If $n \geq 112$, then $\frac{n}{n-56} \leq 2$, in which case it suffices to have $c \geq 14b$. Thus, it would seem that $T(n)$ is indeed $O(n)$. Now, we must prove this formally via mathematical induction. To prove that $T(n) = O(n)$, we simply need to show that for there exists some $c > 0$ such that $T(n) \leq cn$ for all $n \geq 112$. But as it turns out, we can actually demonstrate this claim for *all* $n > 0$ without much extra work. So we do that below.

Claim. Let $T(n) \leq T(n/7) + T(5n/7 + 8) + bn$. Then there exists some constant $c > 0$ such that $T(n) \leq cn$ for all $n > 0$.

Proof. First, we define

$$a := \max_{n \in \{1,2,\cdots,112\}} \left\{T(n)/n\right\},$$

and $c := \max\{a, 14b\}$.

Base case. Suppose $n \in \{1, 2, \cdots, 112\}$. Then by the definition of a, we know that $T(n) \leq an$. And by the definition of c, it must be true that $a \leq c$. Thus, we have

$$T(n) \leq an \leq cn$$

for all $n \in \{1, 2, \cdots, 112\}$.

Induction step. Suppose the claim holds for all $n \leq 112 = k$. Then for $n = k + 1$, we have the following:

$$
\begin{aligned}
T(n) &\leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7} + 8\right) + bn && \text{Our original recurrence relation} \\
&\leq c \cdot \frac{n}{7} + c\left(\frac{5}{7}n + 8\right) + bn && \text{Applying the induction hypothesis} \\
&= cn + \left(-\frac{1}{7}cn + 8c + bn\right) && \text{Algebraic simplification} \\
&\leq cn + \left(-\frac{1}{7}cn + 8c + \frac{1}{14}cn\right) && \text{By the definition of } c \\
&= cn + \left(8c - \frac{1}{14}cn\right) && \text{Algebraic simplification} \\
&\leq cn + \left(8c - \frac{112}{14}c\right) && \text{Using assumption that } n \geq 112 \\
&= cn + (8c - 8c) \\
&= cn.
\end{aligned}
$$

□

Observation. With groups of 7, as with the groups of 5 in the lecture, the analysis gives linear time: the two recursive calls have total size about $\frac{n}{7} + \frac{5n}{7} = \frac{6n}{7} < n$ (for groups of 5, $\frac{n}{5} + \frac{7n}{10} = \frac{9n}{10}$), so the work shrinks geometrically from level to level. With groups of 3 the two calls have total size about $\frac{n}{3} + \frac{2n}{3} = n$, every level of the recursion costs about bn, and the analysis only gives $O(n \log n)$. So 5 is the smallest odd group size for which this analysis gives a linear-time algorithm.