Homework 5

Solution Key

This homework must be typed in LaTeX and handed in via Gradescope.

Please ensure that your solutions are complete, concise, and communicated clearly. Use full sentences and plan your presentation before you write. Except in the rare cases where it is indicated otherwise, consider every problem as asking you to prove your result.

Problem 1

Given two $n \times n$ square matrices A and B, we want to compute their product $C = A \times B$ efficiently. This is a fundamental problem in linear algebra, and the interest it has generated both in the Computer Science and Math research community cannot be overstated. For a long time, the best-known algorithms to compute this product had runtime $O(n^3)$. However, in 1971, Volk Strassen presented what is now known as the first Fast Matrix Multiplication algorithm.

We begin by partitioning A, B, C into equally sized block matrices:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

where $A_{i,j}, B_{i,j}, C_{i,j} \in \mathbb{R}^{2^{n-1} \times 2^{n-1}}$.

The naive $O(n^3)$ approach would compute the product through its block matrices as such:

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

Strassen instead noticed that it was possible to compute C as follows:

$$
\begin{aligned}
M_1 &:= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
M_2 &:= (A_{2,1} + A_{2,2})B_{1,1} \\
M_3 &:= A_{1,1}(B_{1,2} - B_{2,2}) \\
M_4 &:= A_{2,2}(B_{2,1} - B_{1,1}) \\
M_5 &:= (A_{1,1} + A_{1,2})B_{2,2} \\
M_6 &:= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
M_7 &:= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
\end{aligned}
$$

$$
\begin{aligned}
C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
C_{1,2} &= M_3 + M_5 \\
C_{2,1} &= M_2 + M_4 \\
C_{2,2} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

(a) Analyze the running time of the Naïve algorithm. Rather than using asymptotic notation, characterize the running time (especially the combine step) with an exact constant multiplicative term multiplying the dominating term.

(b) Analyze the running time of Strassen's approach using the Master Theorem. Rather than using asymptotic notation, analyze the running time (especially the combine step) with an exact constant multiplicative term multiplying the dominating term.

(c) Compare the two algorithms. Which one is asymptotically faster? Can we conclude that one is better than the other for any input size?

(a) The naive approach is given below:

Algorithm 1 Naive Matrix Multiplication

```
1: procedure NaiveApproach(A, B)
2:     if A and B are 2x3 matrices then
3:         return [[A11B11 + A12B21, A11B12 + A12B22],
                   [A21B11 + A22B21, A21B12 + A22B22]]
4:     C11 = NaiveApproach(A11, B11) + NaiveApproach(A12, B21)
5:     C12 = NaiveApproach(A11, B12) + NaiveApproach(A12, B22)
6:     C21 = NaiveApproach(A21, B11) + NaiveApproach(A22, B21)
7:     C22 = NaiveApproach(A21, B12) + NaiveApproach(A22, B22)
8:     return [[C11, C12], [C21, C22]]
```

The divide step reduces matrix dimensions by half and must be executed 8 times, once for each sub-matrix multiplication. The combine step requires 4 additions, each of which

is an $(n/2)^2$ operation. Therefore the recurrence relation is $F(n) = 8F(n/2) + n^2$. By the Master Theorem, $F(n) \in \Theta(n^3)$.

(b) Strassen's approach is given below:

Algorithm 2 Strassen's Matrix Multiplication

```
1: procedure Strassen(A, B)
2:     if A and B are 2x2 matrices then
3:         M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})
4:         M_2 = (A_{2,1} + A_{2,2})B_{1,1}
5:         M_3 = A_{1,1}(B_{1,2} - B_{2,2}
6:         M_4 = A_{2,2}(B_{2,1} - B_{1,1})
7:         M_5 = (A_{1,1} + A_{1,2})B_{2,2}
8:         M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})
9:         M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
10:        return ( M_1 + M_4 - M_5 + M_7    M_3 + M_5
                    M_2 + M_4                M_1 - M_2 + M_3 + M_6 )
11:    M_1 = Strassen(A_{1,1} + A_{2,2}, B_{1,1} + B_{2,2})
12:    M_2 = Strassen(A_{2,1} + A_{2,2}, B_{1,1})
13:    M_3 = Strassen(A_{1,1}, B_{1,2} - B_{2,2})
14:    M_4 = Strassen(A_{2,2}, B_{2,1} - B_{1,1})
15:    M_5 = Strassen(A_{1,1} + A_{1,2}, B_{2,2})
16:    M_6 = Strassen(A_{2,1} - A_{1,1}, B_{1,1} + B_{1,2})
17:    M_7 = Strassen(A_{1,2} - A_{2,2}, B_{2,1} + B_{2,2})
18:    return ( M_1 + M_4 - M_5 + M_7    M_3 + M_5
                M_2 + M_4                M_1 - M_2 + M_3 + M_6 )
```

The divide step reduces matrix dimensions by half and must be executed 7 times, once for each sub-matrix multiplication. The combine step requires 18 additions, each of which is an $(n/2)^2$ operation. Therefore the recurrence relation is $G(n) = 7F(n/2) + (9/2)n^2$. By the Master Theorem, $G(n) \in \Theta(n^{\log_2 7})$.

(c) Strassen's approach is asymptotically faster than the naive approach, as $n^3 > n^{2.8074}$ for large values of n. However, as there is a larger cost associated with the combining step in Strassen's approach (18 sub-matrix additions / subtractions vs. 4), the naive approach is faster for smaller values of n.

Problem 2

The recurrence $T(n) = 7T(n/2) + n^2$ describes the running time of an algorithm A. A competing algorithm A' has a running time of $T'(n) = aT'(n/4) + n^2$. What is the largest integer value for a such that A' is asymptotically faster than A?

Solution

By the Master Theorem, $T(n) \in \Theta(n^{\log_2 7})$ and $T'(n) \in \Theta(n^{\log_4 a})$. When $n^{\log_2 7} = n^{\log_4 a}$, algorithm A is asymptotically as fast as A'. Therefore, we want to find a such that $n^{\log_2 7} > n^{\log_4 a}$. Solving, $a < 49$.

Problem 3

An $m \times n$ array A of real numbers is a *Gnome array* if for all i, j, k, l such that $1 \leq i \leq k \leq m$ and $1 \leq j \leq \ell \leq n$, we have

$$A[i, j] + A[k, l] \leq A[i, l] + A[k, j].$$

That is, whenever we pick two rows and two columns of a Gnome array and consider the four elements at the intersections of rows and columns, the sum of the upper-left and lower-right elements is less or equal than to the sum of the lower-left and upper-right elements.

(a) Prove that an array is Gnome iff for all $i = 1, 2, \ldots, m-1$ and $j = 1, 2 \ldots, n-1$, we have:

$$A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j].$$

Hint: For the if part, use induction separately on rows and column

(b) Let $f(i)$ be the index of the column containing the leftmost minimum element of row i. Prove that $f(1) \leq f(2) \leq \ldots, f(m)$ for an $m \times n$ Gnome array.

(c) Here is a description of a divide-and-conquer algorithm that computes the leftmost minimum element in each row of a $m \times n$ Gnome array

- Construct a sub-array A' of A consisting of the even-numbered rows of A.
- Recursively invoke the algorithm to find the leftmost minimum for each row of A'.
- Using the information on the leftmost minimum of the even-numbered rows of A you just computed, computed the leftmost minimum in the odd-rows of A.

Explain how to compute the leftmost minimum of the odd-numbered rows of A given that you have already computed the leftmost minimum for even-numbered rows of A. Prove that this can be done in $O(n + m)$ time.

(d) Write the recurrence describing the running time of the algorithm, and obtain a explicit closed-form solution for it.

Hint: if you get stuck on any of the points, you can work on the remaining assuming the results you were asked to prove in the previous ones hold.

(a) We want to prove:

(1) A is Gnome implies $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$.

(2) $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$ implies A is Gnome.

Trivially, by definition of Gnome array, (1) is true. To prove the converse, we first induct on rows and use the result to induct on columns.

Rows: We would like to show that $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$ implies $A[i, j] + A[k, j+1] \leq A[i, j+1] + A[k, j]$ for $i \leq k \leq m$. The base case $k = i+1, l = j+1$

is given by the problem statement. Let us assume that this implication holds for some $k = i + n$; we want to show that it must therefore also hold for $k = i + n + 1$. We have the following:

$$A[i+n, j] + A[i+n+1, j+1] \leq A[i+n, j+1] + A[i+n+1, j] \text{ (base case)}$$

$$A[i, j] + A[i+n, j+1] \leq A[i, j+1] + A[i+n, j] \text{ (inductive hypothesis)}$$

Summing the inequalities (i.e. $a \leq b, c \leq d \implies a + b \leq c + d$), we can conclude that $A[i, j] + A[i+n+1, j+1] \leq A[i, j+1] + A[i+n+1, j]$.

Columns: We have shown that $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j] \implies A[i, j] + A[k, j+1] \leq A[i, j+1] + A[k, j]$ for $i \leq k \leq m$. We would like to use this result to prove $A[i, j] + A[k, \ell] \leq A[i, \ell] + A[k, j]$ for $j \leq \ell \leq n$. The proof to do so is analogous to our previous induction over rows.

Therefore, $A[i, j] + A[i+1, j+1] \leq A[i, j+1] + A[i+1, j]$ implies A is Gnome.

(b) Define a_i as the index of the leftmost minimal element in row a, and b_j as the index of the leftmost minimal element in row b, where $i < j$. Assume $a_i > b_j$. By Gnome array property:

$$A[i, b_j] + A[j, a_i] \leq A[i, a_i] + A[j, b_j]$$

We have reached a contradiction because $A[i, a_i]$ and $A[j, b_j]$ are the leftmost minimal elements in their respective rows. Therefore, $a_i \leq b_j$ for $i < j$.

(c) If μ_i is the index of the i-th row's leftmost minimum, then we know $\mu_{i-1} \leq \mu_i \leq \mu_{i+1}$. For $i = 2k + 1, k \geq 0$, finding μ_i takes $\mu_{i+1} - \mu_{i-1} + 1$ steps at most, since we only need to compare those elements. Thus:

$$\begin{aligned}
T(m, n) &= \sum_{i=0}^{m/2-1} (\mu_{2i+2} - \mu_{2i} + 1) \\
&= \sum_{i=0}^{m/2-1} \mu_{2i+2} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2 \\
&= \sum_{i=1}^{m/2} \mu_{2i} - \sum_{i=0}^{m/2-1} \mu_{2i} + m/2 \\
&= \mu_m - \mu_0 + m/2 \\
&= n + m/2 \\
&\implies O(n + m)
\end{aligned}$$

(d) The divide time is $O(1)$, the conquer part is $T(m/2)$, and the merge part is $O(n + m)$.

Using substitution:

$$\begin{aligned} T(m) &= T(m/2) + cn + dm \\ &= cn + dm + cn + dm/2 + cn + dm/4 + \dots \\ &= \sum_{i=0}^{\log m - 1} cn + \sum_{i=0}^{\log m - 1} \frac{dm}{2^i} \\ &= cn \log m + dm \sum_{i=0}^{\log m - 1} \frac{1}{2^i} \\ &\leq \boxed{cn \log m + 2dm} \end{aligned}$$

Since

$$\sum_{i=0}^{\log m - 1} \frac{1}{2^i} \leq \sum_{i=0}^{\infty} \frac{1}{2^i} = 2 \tag{1}$$