Algorithms

Matrix Multiplication – Programming Project

Objective: Assume $n = 2^k$ is a power of 2 for some $k \geq 0$. Let A and B be two $n \times n$ matrices with positive integer entries. Write a program that computes and outputs the product $C = A \times B$.

Method: Implement both the direct matrix multiplication algorithm, which uses $\Theta(n^3)$ arithmetic operations, and Strassen's recursive algorithm, which uses $\Theta(n^{\log_2 7})$ arithmetic operations.

Programming language and AI: You may use any programming language. You may get help from any AI model. However, you **must** understand everything you submit down to the last detail. If you use AI, your final report must accurately specify all of its contributions.

Input: A valid input consists of a sequence of $2n^2+1$ positive integers formatted across $2n+1$ lines. The first line contains one positive integer n (the size of the matrices). The next n lines contain the n rows of A, each with n integers, and the last n lines contain the n rows of B, each with n integers.

Example: Below is the valid input for multiplying the following matrices:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{pmatrix} \times \begin{pmatrix} 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 \\ 19 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \end{pmatrix}$$

```
4
1  2  3  4
5  6  7  8
9  1  2  3
4  5  6  7
11 12 13 14
15 16 17 18
19 11 12 13
14 15 16 17
```

Input methods: Two ways to enter the input should be implemented.

- An interactive mode that repeatedly prompts the user to enter complete valid inputs.
- A data file containing a single valid input formatted across $2n + 1$ lines.

Remark: Assume that n is a power of 2 and that the input is valid.

Output:

- For small enough n, display all three matrices as $C = A \times B$.

$$\begin{pmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

- Otherwise, generate an output file containing n lines, where each line represents one row of the product matrix.

```
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
```

Performance evaluation I: Count the exact number of operations (additions/subtractions, multiplications, and the combined total) executed by both implementations for multiplying two all-ones matrices A and B for the largest possible $n = 2^k$.

- For each method, summarize your results in a table with the following four columns: (i) n, (ii) number of additions/subtractions, (iii) number of multiplications, and (iv) total number of operations. Each table must have $k + 1$ data rows, corresponding to $n = 2^i$ for $0 \leq i \leq k$. The first two rows of the table for Strassen's method are shown below.

n	#additions/subtractions	#multiplications	#operations
1	0	1	1
2	18	7	25

- In addition, plot three charts, using logarithmic or log-log scales, showing the number of operations performed by both methods as a function of n: (i) a chart for the number of additions/subtractions, (ii) a chart for the number of multiplications, and (iii) a chart for the total number of operations. Each chart should plot the corresponding column from both tables, with one curve for each algorithm.
- Your results **must** match the theoretical formulas.

Performance evaluation II: Measure the wall-clock running time it takes both methods to multiply two all-ones matrices A and B for the largest possible $n = 2^k$.

- If n is too small to yield a measurable duration, repeat the computation m times (for a sufficiently large m), and calculate the average running time by dividing the total execution time by m.
- Summarize the wall-clock running times of both implementations in tables and charts, as you did for the total operation count.
- Note that here there is no distinction between additions/subtractions and multiplications.

Extension I: For both evaluation methods, estimate a crossover threshold h such that the direct algorithm is faster for $n \leq 2^h$ and Strassen's algorithm is faster for $n > 2^h$, under your test conditions.

- Implement a "hybrid" algorithm that stops the recursion once the matrices have size $2^h \times 2^h$ and then applies the direct method.
- For both evaluation methods, add the hybrid algorithm to the operation-count and running-time tables and comparison charts.

Extension II: Implement the direct algorithm and Strassen's algorithm for any integer $n \geq 1$.

- Demonstrate the correctness of your implementation of Strassen's algorithm by showing, for $n = 3, 5, 7, 9$, the outputs generated by both methods for some examples of your choice.
- Compare the performance of both algorithms using the same evaluation methods applied to the power-of-2 case. Do this for interesting families of values for n. For $k \geq 1$, examples include: (i) $n = 2^k + 1$ $(3, 5, 9, 17, \ldots)$, (ii) $n = 2^{k+1} - 1$ $(3, 7, 15, 31, \ldots)$, and (iii) $n = 3 \cdot 2^{k-1}$ $(3, 6, 12, 24, \ldots)$.
- Additional extension: implement the following two methods.
  - Choose the smallest k such that $n \leq 2^k$, pad both matrices with zeros to size $2^k \times 2^k$, apply Strassen's algorithm, and return the leading $n \times n$ submatrix of the result.
  - At each recursive step, split the matrices normally when n is even. When n is odd, first add a zero row and column to each matrix, recurse on the resulting $(n + 1) \times (n + 1)$ matrices, and discard the added row and column from the result.

  Then compare both methods.

Testing the code: Once you are ready, send an email to amotz.CUNY+70010F26@gmail.com. This email should contain the following:

- Instructions for testing your code using both input methods.
- All the tables and charts generated by your project.

Meeting: In addition to the email, we will have an approximately 30-minute in-person or Zoom meeting during which you will demonstrate your project and explain your implementation.

Main grading factors:

- Correctness.
- Accuracy in implementing the algorithms.
- A user-friendly input interface and clear output.
- Completion of optional extensions.
- "Surprising initiatives" shown by the students!

Grading:

- The project itself will receive a score of 100, regardless of its quality, provided that you make a genuine effort to complete it. However, its weight toward your final course grade will depend on the grading factors listed above. For excellent work, the project may account for up to 15% of your final grade; for work that satisfies only the basic requirements, its weight will be closer to 0%.
- If you do not submit the project, or if it does not satisfy the minimum requirements, it will account for 3% of your final grade and will receive a score of 0.