Experiment 3: Matrix Multiplication (Traditional vs Strassen)

Problem Statement

Implement **Traditional Matrix Multiplication** and **Strassen's Algorithm** using the divide-and-conquer technique.
Compare their **time complexity, space complexity, and efficiency**.

Theory

Traditional Matrix Multiplication

- Follows the definition of matrix multiplication.
- For each element `(i, j)` in result matrix `C`, compute:

$$ C[i][j] = \sum_{k=1}^{n} A[i][k] \times B[k][j] $$

- Requires **n^3 multiplications**.
- Works for all `n × n` matrices.
- Simple but slow for large `n`.

Strassen's Algorithm

- A **Divide and Conquer** algorithm.
- Splits each matrix into 4 sub-matrices (of size `n/2 × n/2`).
- Instead of 8 multiplications, it cleverly reduces them to **7 multiplications + extra additions/subtractions**.
- Reduces the time complexity from **$O(n^3)$** → **O(n^2.81)**.
- Works only for **square matrices of size 2^k × 2^k**.

Code Files

- `traditional.cpp` → Implementation of Traditional Matrix Multiplication.
- `strassen.cpp` → Implementation of Strassen's Algorithm.

Time & Space Complexity

Algorithm	Time Complexity	Space Complexity	Notes
Traditional	$O(n^3)$	O(1)	Works for all `n × n` matrices

Algorithm	Time Complexity	Space Complexity	Notes
Strassen	O(n^2.81)	O(n^2) (extra space for sub-matrices)	Works best for large matrices, requires `n = 2^k`

📊 Pros and Cons

☑ Traditional Method

Pros

- Simple and easy to implement.
- Works for all matrix sizes (no restrictions).
- Low memory usage.

Cons

- Slow for large `n` (cubic time complexity).

☑ Strassen's Method

Pros

- Faster asymptotically (`O(n^2.81)` vs `O(n³)`).
- Demonstrates the power of divide-and-conquer.
- Useful for **very large matrices**.

Cons

- Only works efficiently for **2^k × 2^k** matrices (needs padding otherwise).
- More complex to implement.
- Requires extra memory for sub-matrices.
- For small matrices, overhead may make it **slower than traditional**.

🏁 Conclusion

- For **small matrices**, the **Traditional method** is more practical.
- For **large matrices**, **Strassen's Algorithm** is asymptotically faster but requires extra space.
- Strassen's algorithm is a classic example of how **divide-and-conquer reduces complexity** at the cost of implementation difficulty and memory usage.