Ahsania Mission University of Science & Technology

Department of Computer Science and Engineering

1st Batch, 2nd Year 2nd Semester, Spring 2025

Lab Report

Course Code: CSE 2202

Course Title: Computer Algorithms Sessional

Experiment No. : 02

Experiment Date : 05-05-2025

Submission Date : 21-05-2025

Experiment Title : Matrix Multiplication

Submitted To:

Md. Fahim Faisal
Lecturer,
Department of Computer Science and Engineering (CSE)
Faculty of Engineering, Ahsania Mission University of Science & Technology

Submitted By-

Name : Md. Aktaruzzaman Aktar

ID No. : 1012320005101015

1st Batch, 2nd Year 2nd Semester, Spring 2025
Department of Computer Science and Engineering, AMUST.

Task No.: 01

Problem Statement: Write a program in C++ to perform matrix multiplication using the naive method.

Theory: Matrix multiplication is a binary operation that produces a matrix from two matrices. If A is a matrix of size m x n and B is a matrix of size n x p, then their product C = A × B is a matrix of size m x p. The naive algorithm uses three nested loops to compute the dot product between rows of A and columns of B.

Formula:
C[i][j] = Σ (A[i][k] * B[k][j]) for k from 0 to n-1

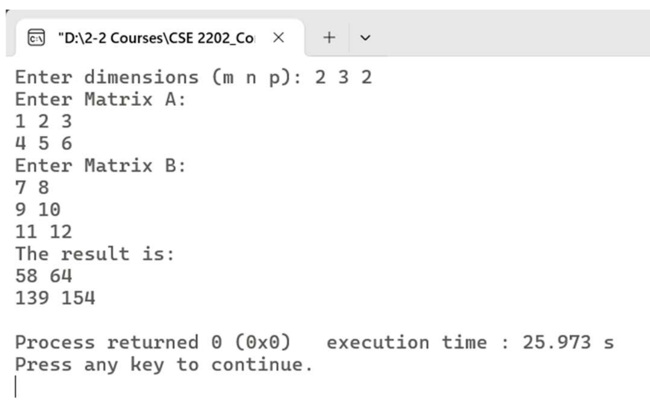
Time Complexity: O(m × n × p)

Source Code:

```
#include <iostream>

using namespace std;

int main() {

    int m, n, p;

    cout << "Enter dimensions (m n p): ";

    cin >> m >> n >> p;

    int A[m][n], B[n][p], C[m][p];

    cout << "Enter Matrix A:" << endl;

    for (int i = 0; i < m; i++)

        for (int j = 0; j < n; j++)

            cin >> A[i][j];

cout << "Enter Matrix B:" << endl;

    for (int i = 0; i < n; i++)

        for (int j = 0; j < p; j++)

            cin >> B[i][j];

for (int i = 0; i < m; i++)

        for (int j = 0; j < p; j++)

            C[i][j] = 0;

 for (int i = 0; i < m; i++)

        for (int j = 0; j < p; j++)
```

```
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
 cout << "The result is:" << endl;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }
 return 0;
}
```

Output:



Conclusion: The naive approach is straightforward and efficient for small matrices but becomes computationally expensive for large matrices.

Task No.: 02

Problem Statement: Write a program in C++ to multiply two square matrices using divide and conquer.

Theory: This method recursively splits square matrices into four equal parts and combines their results. Each product matrix is calculated using 8 multiplications and additions of submatrices.

Time Complexity: O(n^3)

Source Code:

```
#include <iostream>

#include <vector>

using namespace std;


typedef vector<vector<int>> Matrix;


Matrix add(const Matrix &A, const Matrix &B) {

    int n = A.size();

    Matrix C(n, vector<int>(n));

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            C[i][j] = A[i][j] + B[i][j];

    return C;

}


Matrix subtract(const Matrix &A, const Matrix &B) {

    int n = A.size();

    Matrix C(n, vector<int>(n));

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            C[i][j] = A[i][j] - B[i][j];

    return C;

}


Matrix multiply(const Matrix &A, const Matrix &B) {

    int n = A.size();

    Matrix C(n, vector<int>(n, 0));
```

```
if (n == 1) {
    C[0][0] = A[0][0] * B[0][0];
} else {
    int k = n / 2;
    Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k)),
           A21(k, vector<int>(k)), A22(k, vector<int>(k));
    Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k)),
           B21(k, vector<int>(k)), B22(k, vector<int>(k));
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];
            A22[i][j] = A[i + k][j + k];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];
            B22[i][j] = B[i + k][j + k];
        }

    Matrix C11 = add(multiply(A11, B11), multiply(A12, B21));
    Matrix C12 = add(multiply(A11, B12), multiply(A12, B22));
    Matrix C21 = add(multiply(A21, B11), multiply(A22, B21));
    Matrix C22 = add(multiply(A21, B12), multiply(A22, B22));

    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
```

```
            C[i][j + k] = C12[i][j];
            C[i + k][j] = C21[i][j];
            C[i + k][j + k] = C22[i][j];
        }
    }
    return C;
}

int main() {
    int n;
    cout << "Enter matrix size (power of 2): ";
    cin >> n;

    Matrix A(n, vector<int>(n)), B(n, vector<int>(n));

    cout << "Enter Matrix A:" << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> A[i][j];

    cout << "Enter Matrix B:" << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> B[i][j];

    Matrix C = multiply(A, B);

    cout << "Resultant Matrix C:" << endl;
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output:

```
"D:\2-2 Courses\CSE 2202_Co

Enter matrix size (power of 2): 2
Enter Matrix A:
1 2
3 4
Enter Matrix B:
5 6
7 8
Resultant Matrix C:
19 22
43 50

Process returned 0 (0x0)   execution time : 22.605 s
Press any key to continue.
```

Conclusion: Divide and conquer method is recursively structured and works well for problems that benefit from parallelization.

Task No.: 03

Problem Statement: Write a program in C++ to multiply two matrices using Strassen's algorithm.

Theory: Strassen's algorithm is a divide-and-conquer algorithm that improves matrix multiplication complexity from O(n^3) to approximately O(n^2.81) by reducing the number of multiplications from 8 to 7.

Time Complexity: O(n^2.81)

Source Code:

```
#include <iostream>
#include <vector>
using namespace std;
```

```
typedef vector<vector<int>> Matrix;

Matrix add(const Matrix &A, const Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}

Matrix subtract(const Matrix &A, const Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}

Matrix strassen(const Matrix &A, const Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<int>(n, 0));

    if (n == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return C;
```

```
}

int k = n / 2;
Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k)),
       A21(k, vector<int>(k)), A22(k, vector<int>(k));
Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k)),
       B21(k, vector<int>(k)), B22(k, vector<int>(k));

for (int i = 0; i < k; i++) {
    for (int j = 0; j < k; j++) {
        A11[i][j] = A[i][j];
        A12[i][j] = A[i][j + k];
        A21[i][j] = A[i + k][j];
        A22[i][j] = A[i + k][j + k];
        B11[i][j] = B[i][j];
        B12[i][j] = B[i][j + k];
        B21[i][j] = B[i + k][j];
        B22[i][j] = B[i + k][j + k];
    }
}

Matrix M1 = strassen(add(A11, A22), add(B11, B22));
Matrix M2 = strassen(add(A21, A22), B11);
Matrix M3 = strassen(A11, subtract(B12, B22));
Matrix M4 = strassen(A22, subtract(B21, B11));
Matrix M5 = strassen(add(A11, A12), B22);
Matrix M6 = strassen(subtract(A21, A11), add(B11, B12));
Matrix M7 = strassen(subtract(A12, A22), add(B21, B22));
```

```
    Matrix C11 = add(subtract(add(M1, M4), M5), M7);
    Matrix C12 = add(M3, M5);
    Matrix C21 = add(M2, M4);
    Matrix C22 = add(subtract(add(M1, M3), M2), M6);

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[i + k][j] = C21[i][j];
            C[i + k][j + k] = C22[i][j];
        }
    }

    return C;
}

int main() {
    int n;
    cout << "Enter matrix size (power of 2): ";
    cin >> n;

    Matrix A(n, vector<int>(n)), B(n, vector<int>(n));

    cout << "Enter Matrix A:" << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
```

```
            cin >> A[i][j];

    cout << "Enter Matrix B:" << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> B[i][j];

    Matrix C = strassen(A, B);

    cout << "Resultant Matrix C:" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

Output:

```
"D:\2-2 Courses\CSE 2202_Co

Enter matrix size (power of 2): 2
Enter Matrix A:
1 2
3 4
Enter Matrix B:
5 6
7 8
Resultant Matrix C:
19 22
43 50

Process returned 0 (0x0)   execution time : 12.929 s
Press any key to continue.
```

Conclusion: Strassen's algorithm is efficient for large matrices and demonstrates the power of mathematical optimization.