## Meet Jain, 23BCP093

**DAA Lab 6: Matrix Multiplication (DnC)**

**Theory:**

Matrix multiplication using the divide and conquer approach involves splitting the matrices into smaller submatrices and recursively multiplying them. The matrix is divided into four smaller submatrices, and the multiplication is performed by recursively solving smaller instances of the problem. This method reduces the problem size at each step, leading to a time complexity of O(n3) for standard matrix multiplication, but with the potential to optimize further through more advanced techniques.

**Code:**

#include <bits/stdc++.h>

using namespace std;

typedef vector<vector<int>> Matrix;

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

    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];

        return C;

    }

    int half = n / 2;

    Matrix A11(half, vector<int>(half)), A12(half, vector<int>(half)),

           A21(half, vector<int>(half)), A22(half, vector<int>(half));

    Matrix B11(half, vector<int>(half)), B12(half, vector<int>(half)),

           B21(half, vector<int>(half)), B22(half, vector<int>(half));

    for (int i = 0; i < half; i++) {

        for (int j = 0; j < half; j++) {

            A11[i][j] = A[i][j];

            A12[i][j] = A[i][j + half];

            A21[i][j] = A[i + half][j];

            A22[i][j] = A[i + half][j + half];

            B11[i][j] = B[i][j];

            B12[i][j] = B[i][j + half];

            B21[i][j] = B[i + half][j];

            B22[i][j] = B[i + half][j + half];

        }

    }

    Matrix C11 = add(multiply(A11, B11), multiply(A12, B21), half);

    Matrix C12 = add(multiply(A11, B12), multiply(A12, B22), half);

    Matrix C21 = add(multiply(A21, B11), multiply(A22, B21), half);

    Matrix C22 = add(multiply(A21, B12), multiply(A22, B22), half);

    for (int i = 0; i < half; i++) {

        for (int j = 0; j < half; j++) {

            C[i][j] = C11[i][j];

            C[i][j + half] = C12[i][j];

            C[i + half][j] = C21[i][j];

            C[i + half][j + half] = C22[i][j];

        }

    }

    return C;

}

int main() {

    Matrix A = {{2, 3}, {1, 4}};

    Matrix B = {{5, 2}, {3, 1}};

    Matrix result = multiply(A, B);

    cout << "Resultant Matrix:\n";

    for (const auto &row : result) {

        for (int val : row) {

            cout << val << " ";

        }

        cout << endl;

    }

    return 0;

}

**Output:**

**A black screen with white text

AI-generated content may be incorrect.**

**Time Complexity Analysis:**

The time complexity of matrix multiplication using the standard divide and conquer approach can be analysed as follows:

1. Divide Step: The matrix is divided into four smaller submatrices, which takes constant time.
2. Conquer Step: The algorithm recursively multiplies submatrices. Each multiplication involves solving 8 smaller matrix multiplications (since there are 8 submatrix multiplications in total).
3. Combine Step: After multiplying the submatrices, combining the results takes linear time O(n2).

Let T(n) represent the time complexity of multiplying two n×n matrices. The recurrence relation for divide and conquer is:

T(n)=8T(n2)+O(n2)

Solving this recurrence using the master theorem gives:

T(n)=O(n3)

Thus, the time complexity of matrix multiplication using the standard divide and conquer approach is O(n3).