Question 1:

You're tasked with optimizing matrix multiplication for a real-time graphics rendering engine. The current implementation relies on Strassen's algorithm to efficiently multiply matrices, which is crucial for rendering complex scenes with multiple objects and effects.

Input Format:

- The first line contains an integer n representing the size of the square matrices A and B.
- The next n lines contain the elements of Matrix A, each line containing n space-separated integers.
- The next n lines contain the elements of Matrix B, each line containing n space-separated integers.

Output Format:

- The output displays the original matrices A and B, followed by the resultant Matrix C after performing Strassen's Multiplication.
- Each matrix is formatted with elements enclosed in square brackets and separated by spaces.

Title for Question 1: Basic Strassen's Matrix Multiplication

Solution:

```
#include <iostream>
#include <vector>


// Function to add two matrices
std::vector<std::vector<int>> addMatrices(const std::vector<std::vector<i
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] + B[i][j];
    return result;
}


// Function to subtract two matrices
std::vector<std::vector<int>> subtractMatrices(const std::vector<std::vec
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] - B[i][j];
    return result;
}


// Function to multiply two matrices using Strassen's algorithm
std::vector<std::vector<int>> strassenMultiply(const std::vector<std::vec
```

```
    if (n == 1) {
        std::vector<std::vector<int>> result(1, std::vector<int>(1, A[0][
        return result;
    }


    // Matrix dimensions for the submatrices
    int newSize = n / 2;


    // Creating submatrices
    std::vector<std::vector<int>> A11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A22(newSize, std::vector<int>(newSize))


    std::vector<std::vector<int>> B11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B22(newSize, std::vector<int>(newSize))


    // Initializing submatrices
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];


            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }


    // Computing products needed for Strassen's algorithm
    std::vector<std::vector<int>> P1 = strassenMultiply(A11, subtractMatr
    std::vector<std::vector<int>> P2 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P3 = strassenMultiply(addMatrices(A21,
    std::vector<std::vector<int>> P4 = strassenMultiply(A22, subtractMatr
    std::vector<std::vector<int>> P5 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P6 = strassenMultiply(subtractMatrices(
    std::vector<std::vector<int>> P7 = strassenMultiply(subtractMatrices(


    // Computing submatrices of C
    std::vector<std::vector<int>> C11 = addMatrices(subtractMatrices(addM
    std::vector<std::vector<int>> C12 = addMatrices(P1, P2, newSize);
    std::vector<std::vector<int>> C21 = addMatrices(P3, P4, newSize);
    std::vector<std::vector<int>> C22 = subtractMatrices(subtractMatrices


    // Combining submatrices to get the result
    std::vector<std::vector<int>> result(n, std::vector<int>(n, 0));
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
```

```
            result[i][j] = C11[i][j];
            result[i][j + newSize] = C12[i][j];
            result[i + newSize][j] = C21[i][j];
            result[i + newSize][j + newSize] = C22[i][j];
        }


    return result;
}


// Function to display a matrix
void displayMatrix(const std::vector<std::vector<int>>& matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cout << "[" << matrix[i][j] << "] ";
        }
        std::cout << '\n';
    }
}


int main() {
    int n;
    std::cin >> n;


    // Create matrices A, B, C
    std::vector<std::vector<int>> A(n, std::vector<int>(n));
    std::vector<std::vector<int>> B(n, std::vector<int>(n));


    // Input elements of Matrix A
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> A[i][j];


    // Input elements of Matrix B
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> B[i][j];


    // Perform Strassen's Multiplication
    std::vector<std::vector<int>> C = strassenMultiply(A, B, n);


    // Output Original Matrices and Result
    std::cout << "\nOriginal Matrix A:\n";
    displayMatrix(A, n);


    std::cout << "\nOriginal Matrix B:\n";
    displayMatrix(B, n);


    std::cout << "\nResultant Matrix C (Strassen's Multiplication):\n";
    displayMatrix(C, n);
```

```
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2 1 2 3 4 1 0 0 1	Original Matrix A: [1] [2] [3] [4] Original Matrix B: [1] [0] [0] [1] Resultant Matrix C (Strassen's Multiplication): [1] [2] [3] [4]
2	3 2 1 3 4 2 1 3 0 2 2 0 1 3 1 2 1 2 0	Original Matrix A: [2] [1] [3] [4] [2] [1] [3] [0] [2] Original Matrix B: [2] [0] [1] [3] [1] [2] [1] [2] [0] Resultant Matrix C (Strassen's Multiplication): [7] [1] [0] [14] [2] [0] [0] [0] [0]
3	3 -1 2 0 3 4 2 1 0 -2 2 1 3 4 2 1 3 0 2	Original Matrix A: [-1] [2] [0] [3] [4] [2] [1] [0] [-2] Original Matrix B: [2] [1] [3] [4] [2] [1] [3] [0] [2] Resultant Matrix C (Strassen's Multiplication): [6] [3] [0] [22] [11] [0] [0] [0] [0]
4	3 2 -1 3 4 2 1 3 0 2 -1 2 0 3 4 2 1 0 -2	Original Matrix A: [2] [-1] [3] [4] [2] [1] [3] [0] [2] Original Matrix B: [-1] [2] [0] [3] [4] [2] [1] [0] [-2] Resultant Matrix C (Strassen's Multiplication): [-5] [0] [0] [2] [16] [0] [0] [0] [0]
5	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 -1 0 0 0 0 -1 0 0 0 0 -1 0 0 0 0 -1	Original Matrix A: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] Original Matrix B: [-1] [0] [0] [0] [0] [-1] [0] [0] [0] [0] [-1] [0] [0] [0] [0] [-1] Resultant Matrix C (Strassen's Multiplication): [-1] [-2] [-3] [-4] [-5] [-6] [-7] [-8] [-9] [-10] [-11] [-12] [-13] [-14] [-15] [-16]
6	5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1	Original Matrix A: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] Original Matrix B: [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [1] Resultant Matrix C (Strassen's Multiplication): [1] [2] [3] [4] [0] [6] [7] [8] [9] [0] [11] [12] [13] [14] [0] [16] [17] [18] [19] [0] [0] [0] [0] [0] [0]

White List:

Black List:

Question 2:

Implement a program in that utilizes Strassen's Matrix Multiplication algorithm to efficiently multiply two matrices. including its ability to handle matrices of size that are powers of 2. functions involved in adding, subtracting, and multiplying matrices, particularly focusing on the recursive nature of Strassen's algorithm. Additionally, elaborate on how the program prompts the user to input matrices A and B and displays the resulting matrix C. Analyze the program's significance in scientific computations, emphasizing its potential for improving computational efficiency, especially when dealing with large matrices. Finally, explore the practical applications of Strassen's algorithm in various numerical simulations and scientific simulations.

Input Format:

- The first line contains an integer n representing the size of the square matrices A and B. This n should be a power of 2.

- The next n lines contain the elements of Matrix A, each line containing n space-separated integers.
- The next n lines contain the elements of Matrix B, each line containing n space-separated integers.

Output Format:

- The output displays the original matrices A and B, followed by the resultant Matrix C after performing Strassen's Multiplication.
- Each matrix is formatted with elements enclosed in square brackets and separated by spaces.

Title for Question 2: Matrix Size Error Handling

Solution:

```
#include <iostream>
#include <cmath>
#include <vector>


// Function to check if a number is a power of 2
bool isPowerOfTwo(int n) {
    return (n > 0) && ((n & (n - 1)) == 0);
}


// Function to add two matrices
std::vector<std::vector<int>> addMatrices(const std::vector<std::vector<i
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] + B[i][j];
    return result;
}


// Function to subtract two matrices
std::vector<std::vector<int>> subtractMatrices(const std::vector<std::vec
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] - B[i][j];
    return result;
}


// Function to multiply two matrices using Strassen's algorithm
std::vector<std::vector<int>> strassenMultiply(const std::vector<std::vec
    if (n == 1) {
        std::vector<std::vector<int>> result(1, std::vector<int>(1, A[0][
        return result;
    }
```

```
    // Matrix dimensions for the submatrices
    int newSize = n / 2;


    // Creating submatrices
    std::vector<std::vector<int>> A11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A22(newSize, std::vector<int>(newSize))


    std::vector<std::vector<int>> B11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B22(newSize, std::vector<int>(newSize))


    // Initializing submatrices
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];


            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }


    // Computing products needed for Strassen's algorithm
    std::vector<std::vector<int>> P1 = strassenMultiply(A11, subtractMatr
    std::vector<std::vector<int>> P2 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P3 = strassenMultiply(addMatrices(A21,
    std::vector<std::vector<int>> P4 = strassenMultiply(A22, subtractMatr
    std::vector<std::vector<int>> P5 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P6 = strassenMultiply(subtractMatrices(
    std::vector<std::vector<int>> P7 = strassenMultiply(subtractMatrices(


    // Computing submatrices of C
    std::vector<std::vector<int>> C11 = addMatrices(subtractMatrices(addM
    std::vector<std::vector<int>> C12 = addMatrices(P1, P2, newSize);
    std::vector<std::vector<int>> C21 = addMatrices(P3, P4, newSize);
    std::vector<std::vector<int>> C22 = subtractMatrices(subtractMatrices


    // Combining submatrices to get the result
    std::vector<std::vector<int>> result(n, std::vector<int>(n, 0));
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
            result[i][j] = C11[i][j];
            result[i][j + newSize] = C12[i][j];
            result[i + newSize][j] = C21[i][j];
            result[i + newSize][j + newSize] = C22[i][j];
        }
```

```
    return result;
}


// Function to display a matrix
void displayMatrix(const std::vector<std::vector<int>>& matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cout << "[" << matrix[i][j] << "] ";
        }
        std::cout << '\n';
    }
}


int main() {
    int n;


    // Input matrix size with error handling
    do {
        std::cin >> n;


        if (n <= 0 || !isPowerOfTwo(n)) {
            std::cout << "Invalid input. Please enter a positive integer
        }
    } while (n <= 0 || !isPowerOfTwo(n));


    // Create matrices A, B, C
    std::vector<std::vector<int>> A(n, std::vector<int>(n));
    std::vector<std::vector<int>> B(n, std::vector<int>(n));


    // Input elements of Matrix A
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> A[i][j];


    // Input elements of Matrix B
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> B[i][j];


    // Perform Strassen's Multiplication
    std::vector<std::vector<int>> C = strassenMultiply(A, B, n);


    // Output Original Matrices and Result
    std::cout << "\nOriginal Matrix A:\n";
    displayMatrix(A, n);


    std::cout << "\nOriginal Matrix B:\n";
    displayMatrix(B, n);
```

```
    std::cout << "\nResultant Matrix C (Strassen's Multiplication):\n";
    displayMatrix(C, n);


    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2 1 2 3 4 1 0 0 1	Original Matrix A: [1] [2] [3] [4] Original Matrix B: [1] [0] [0] [1] Resultant Matrix C (Strassen's Multiplication): [1] [2] [3] [4]
2	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 2 0 0 2 0 2 2 0 0 1 1 0 1 0 0 1	Original Matrix A: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] Original Matrix B: [2] [0] [0] [2] [0] [2] [2] [0] [0] [1] [1] [0] [1] [0] [0] [1] Resultant Matrix C (Strassen's Multiplication): [6] [7] [7] [6] [18] [19] [19] [18] [30] [31] [31] [30] [42] [43] [43] [42]
3	2 1 2 3 4 2 0 0 2	Original Matrix A: [1] [2] [3] [4] Original Matrix B: [2] [0] [0] [2] Resultant Matrix C (Strassen's Multiplication): [2] [4] [6] [8]
4	1 5 6	Original Matrix A: [5] Original Matrix B: [6] Resultant Matrix C (Strassen's Multiplication): [30]
5	8 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 2 0 0 2 1 0 0 1 1 1 0 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1 0	Original Matrix A: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [2] [0] [0] [2] [1] [0] [0] [1] [1] [1] [0] [0] [1] [0] [0] [1] [0] [1] [1] [0] [1] [0] [1] [1] [1] [0] [1] [1] [0] [1] [0] [0] Original Matrix B: [1] [0] [0] [1] [0] [1] [1] [0] [1] [1] [0] [1] [1] [0] [0] [0] [1] [1] [0] [1] [0] [0] [1] [0] [0] [0] [1] [1] [0] [1] [0] [1] [1] [1] [0] [0] [0] [1] [0] [1] [0] [0] [1] [0] [0] [1] [1] [1] [0] [0] [1] [1] [1] [0] [1] [0] [0] [0] [0] [0] [0] [0] [0] [0] Resultant Matrix C (Strassen's Multiplication): [11] [10] [17] [17] [9] [16] [17] [15] [43] [34] [41] [57] [25] [48] [49] [39] [75] [58] [65] [97] [41] [80] [81] [63] [107] [82] [89] [137] [57] [112] [113] [87] [3] [1] [2] [4] [0] [5] [2] [3] [3] [2] [0] [2] [1] [2] [1] [1] [3] [3] [1] [3] [2] [1] [2] [1] [2] [1] [2] [3] [0] [3] [3] [2]
6	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 0	Original Matrix A: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] Original Matrix B: [1] [0] [0] [1] [1] [1] [0] [0] [0] [1] [1] [1] [1] [0] [1] [0] Resultant Matrix C (Strassen's Multiplication): [7] [5] [7] [4] [19] [13] [15] [12] [31] [21] [23] [20] [43] [29] [31] [28]

White List:

Black List:

Question 3:

Implement a program in that utilizes Strassen's algorithm to compute the inverse of a square matrix. Begin by describing the importance of matrix inversion in various mathematical and engineering applications. Provide insights into the recursive nature of Strassen's algorithm and its impact on the efficiency of matrix inversion. consider the input process for obtaining the matrix size and elements from the user, ensuring error handling for invalid inputs. Analyze the output, showcasing both the original matrix and its inverse computed using Strassen's algorithm.

Input Format:

- The program prompts the user to enter the dimension of the square matrix n.
- Then, the user is asked to input the elements of the square matrix A row by row.

Output Format:

- The program outputs the original matrix A.
- Finally, it displays the inverted matrix obtained using Strassen's algorithm.

Title for Question 3: Matrix Inversion

Solution:

```
#include <iostream>
#include <vector>
// Function to add two matrices
std::vector<std::vector<int>> addMatrices(const std::vector<std::vector<i
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] + B[i][j];
    return result;
}
// Function to subtract two matrices
std::vector<std::vector<int>> subtractMatrices(const std::vector<std::vec
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] - B[i][j];
    return result;
}
// Function to multiply two matrices using Strassen's algorithm
std::vector<std::vector<int>> strassenMultiply(const std::vector<std::vec
    if (n == 1) {
        std::vector<std::vector<int>> result(1, std::vector<int>(1, A[0][
        return result;
    }
    // Matrix dimensions for the submatrices
    int newSize = n / 2;
    // Creating submatrices
    std::vector<std::vector<int>> A11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> A22(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B11(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B12(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B21(newSize, std::vector<int>(newSize))
    std::vector<std::vector<int>> B22(newSize, std::vector<int>(newSize))
    // Initializing submatrices
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
```

```
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];
            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }
    // Computing products needed for Strassen's algorithm
    std::vector<std::vector<int>> P1 = strassenMultiply(A11, subtractMatr
    std::vector<std::vector<int>> P2 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P3 = strassenMultiply(addMatrices(A21,
    std::vector<std::vector<int>> P4 = strassenMultiply(A22, subtractMatr
    std::vector<std::vector<int>> P5 = strassenMultiply(addMatrices(A11,
    std::vector<std::vector<int>> P6 = strassenMultiply(subtractMatrices(
    std::vector<std::vector<int>> P7 = strassenMultiply(subtractMatrices(
    // Computing submatrices of C
    std::vector<std::vector<int>> C11 = addMatrices(subtractMatrices(addM
    std::vector<std::vector<int>> C12 = addMatrices(P1, P2, newSize);
    std::vector<std::vector<int>> C21 = addMatrices(P3, P4, newSize);
    std::vector<std::vector<int>> C22 = subtractMatrices(subtractMatrices
    // Combining submatrices to get the result
    std::vector<std::vector<int>> result(n, std::vector<int>(n, 0));
    for (int i = 0; i < newSize; ++i)
        for (int j = 0; j < newSize; ++j) {
            result[i][j] = C11[i][j];
            result[i][j + newSize] = C12[i][j];
            result[i + newSize][j] = C21[i][j];
            result[i + newSize][j + newSize] = C22[i][j];
        }
    return result;
}
// Function to compute the matrix inverse using Strassen's algorithm
std::vector<std::vector<int>> strassenInvert(const std::vector<std::vecto
    // Create an identity matrix
    std::vector<std::vector<int>> identity(n, std::vector<int>(n, 0));
    for (int i = 0; i < n; ++i)
        identity[i][i] = 1;
    // Use Strassen's algorithm to find the inverse
    return strassenMultiply(A, identity, n);
}
// Function to display a matrix
void displayMatrix(const std::vector<std::vector<int>>& matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cout << "[" << matrix[i][j] << "] ";
        }
        std::cout << '\n';
    }
}
int main() {
    int n;
    // Input matrix size with error handling
    do {
    //    std::cout << "Enter the dimension of the square matrix (n): ";
        std::cin >> n;
        if (n <= 0) {
            std::cout << "Invalid input. Please enter a positive integer.
        }
```

```
    } while (n <= 0);
    // Create matrix A
    std::vector<std::vector<int>> A(n, std::vector<int>(n));
    // Input elements of Matrix A
 //   std::cout << "Enter the elements of the square matrix A:\n";
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> A[i][j];
    // Perform Strassen's Matrix Inversion
    std::vector<std::vector<int>> invertedMatrix = strassenInvert(A, n);
    // Output Original Matrix and Inverted Matrix
    std::cout << "\nOriginal Matrix A:\n";
    displayMatrix(A, n);
    std::cout << "\nInverted Matrix using Strassen's Algorithm:\n";
    displayMatrix(invertedMatrix, n);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	3 2 0 1 1 1 2 0 1 1	Original Matrix A: [2] [0] [1] [1] [1] [2] [0] [1] [1] Inverted Matrix using Strassen's Algorithm: [2] [0] [0] [1] [1] [0] [0] [0] [0]
2	2 1 2 3 4	Original Matrix A: [1] [2] [3] [4] Inverted Matrix using Strassen's Algorithm: [1] [2] [3] [4]
3	4 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1	Original Matrix A: [1] [0] [0] [0] [0] [1] [0] [0] [0] [0] [1] [0] [0] [0] [0] [1] Inverted Matrix using Strassen's Algorithm: [1] [0] [0] [0] [0] [1] [0] [0] [0] [0] [1] [0] [0] [0] [0] [1]
4	3 1 2 3 0 1 4 5 6 0	Original Matrix A: [1] [2] [3] [0] [1] [4] [5] [6] [0] Inverted Matrix using Strassen's Algorithm: [1] [2] [0] [0] [1] [0] [0] [0] [0]
5	2 2 1 1 2	Original Matrix A: [2] [1] [1] [2] Inverted Matrix using Strassen's Algorithm: [2] [1] [1] [2]
6	5 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1	Original Matrix A: [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] Inverted Matrix using Strassen's Algorithm: [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [1] [0] [0] [0] [0] [0] [0]

White List:

Black List:

Question 4:

Implement an iterative version of the Strassen's Matrix Multiplication algorithm and compare its performance with the recursive version. Discuss the advantages and disadvantages of each approach.

Input Format:

- The program expects an integer n as input, representing the size of the square matrices A and B.

- Input the elements of matrix A in row-major order.
- Input the elements of matrix B in row-major order.

Output Format:

- The program outputs the resultant matrix C obtained using Iterative Strassen's Multiplication.

Title for Question 4: Recursive vs Iterative Implementation

Solution:

```
#include <iostream>
#include <vector>
// Function to add two matrices
std::vector<std::vector<int>> addMatrices(const std::vector<std::vector<i
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] + B[i][j];
    return result;
}


// Function to subtract two matrices
std::vector<std::vector<int>> subtractMatrices(const std::vector<std::vec
    std::vector<std::vector<int>> result(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            result[i][j] = A[i][j] - B[i][j];
    return result;
}


// Function to multiply two matrices using Strassen's algorithm iterative
std::vector<std::vector<int>> strassenMultiplyIterative(const std::vector
    // Implement the iterative version of Strassen's algorithm here
    // ...
    return A; // Placeholder, replace with actual result
}
// Function to display a matrix
void displayMatrix(const std::vector<std::vector<int>>& matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cout << "[" << matrix[i][j] << "] ";
        }
        std::cout << '\n';
    }
}
int main() {
    int n;
    // Input matrix size
    std::cin >> n;
    // Create matrices A, B
    std::vector<std::vector<int>> A(n, std::vector<int>(n));
    std::vector<std::vector<int>> B(n, std::vector<int>(n));
```

```
    // Input elements of Matrix A
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> A[i][j];
    // Input elements of Matrix B
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            std::cin >> B[i][j];
    // Perform Iterative Strassen's Multiplication
    std::vector<std::vector<int>> CIterative = strassenMultiplyIterative(
    // Output Result
    std::cout << "\nResultant Matrix C (Iterative Strassen's Multiplicati
    displayMatrix(CIterative, n);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	3 2 0 1 1 1 0 0 1 2 1 0 1 0 1 0 1 0 1	Resultant Matrix C (Iterative Strassen's Multiplication): [2] [0] [1] [1] [1] [0] [0] [1] [2]
2	5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1	Resultant Matrix C (Iterative Strassen's Multiplication): [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25]
3	3 1 2 3 4 5 6 7 8 9 1 0 0 0 1 0 0 0 1	Resultant Matrix C (Iterative Strassen's Multiplication): [1] [2] [3] [4] [5] [6] [7] [8] [9]
4	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Resultant Matrix C (Iterative Strassen's Multiplication): [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]
5	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 2 0 0 2 1 0 0 1 1 1 0 0 0 1 1 1	Resultant Matrix C (Iterative Strassen's Multiplication): [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]
6	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1	Resultant Matrix C (Iterative Strassen's Multiplication): [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16]

White List:

Black List:

Question 5:

You are working on an image processing application. You have two grayscale images represented as matrices, and you want to perform matrix multiplication using the Strassen algorithm to apply a certain image processing operation efficiently.

Input Format:

- The program expects the size 'n' of the square matrices A and B as input.
- The next 2n lines should contain n integers each, representing the elements of matrix A.
- The following 2n lines should contain n integers each, representing the elements of matrix B.

Output Format:

- Matrix A: Displaying the input matrix A.
- Matrix B: Displaying the input matrix B.
- Result (Strassen's Multiplication): Displaying the result of Strassen's Matrix Multiplication.

Title for Question 5: Efficient Matrix Multiplication using Strassen's Algorithm

Solution:

```
#include <iostream>
#include <vector>
#include <chrono>


using namespace std;


// Function to add two matrices
vector<vector<int>> addMatrices(const vector<vector<int>>& A, const vecto
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));


    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }


    return result;
}


// Function to subtract two matrices
vector<vector<int>> subtractMatrices(const vector<vector<int>>& A, const
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));


    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }


    return result;
}
```

```
// Function to perform standard matrix multiplication (for base case)
vector<vector<int>> standardMultiplication(const vector<vector<int>>& A,
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));


    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }


    return result;
}


// Function to perform Strassen's Matrix Multiplication
void strassen(const vector<vector<int>>& A, const vector<vector<int>>& B,
    int n = A.size();


    // Base case: If the matrices are smaller than 2x2, perform standard
    if (n == 2) {
        int m1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
        int m2 = (A[1][0] + A[1][1]) * B[0][0];
        int m3 = A[0][0] * (B[0][1] - B[1][1]);
        int m4 = A[1][1] * (B[1][0] - B[0][0]);
        int m5 = (A[0][0] + A[0][1]) * B[1][1];
        int m6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
        int m7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);


        result[0][0] = m1 + m4 - m5 + m7;
        result[0][1] = m3 + m5;
        result[1][0] = m2 + m4;
        result[1][1] = m1 - m2 + m3 + m6;
    } else {
        // Divide matrices into quadrants
        int size = n / 2;


        vector<vector<int>> A11(size, vector<int>(size));
        vector<vector<int>> A12(size, vector<int>(size));
        vector<vector<int>> A21(size, vector<int>(size));
        vector<vector<int>> A22(size, vector<int>(size));


        vector<vector<int>> B11(size, vector<int>(size));
        vector<vector<int>> B12(size, vector<int>(size));
        vector<vector<int>> B21(size, vector<int>(size));
        vector<vector<int>> B22(size, vector<int>(size));


        // Populate the quadrants
        for (int i = 0; i < size; ++i) {
```

```
            for (int j = 0; j < size; ++j) {
                A11[i][j] = A[i][j];
                A12[i][j] = A[i][j + size];
                A21[i][j] = A[i + size][j];
                A22[i][j] = A[i + size][j + size];


                B11[i][j] = B[i][j];
                B12[i][j] = B[i][j + size];
                B21[i][j] = B[i + size][j];
                B22[i][j] = B[i + size][j + size];
            }
        }


        // Create matrices for intermediate results
        vector<vector<int>> P1(size, vector<int>(size));
        vector<vector<int>> P2(size, vector<int>(size));
        vector<vector<int>> P3(size, vector<int>(size));
        vector<vector<int>> P4(size, vector<int>(size));
        vector<vector<int>> P5(size, vector<int>(size));
        vector<vector<int>> P6(size, vector<int>(size));
        vector<vector<int>> P7(size, vector<int>(size));


        // Recursive steps
        strassen(A11, subtractMatrices(B12, B22), P1);
        strassen(addMatrices(A11, A12), B22, P2);
        strassen(addMatrices(A21, A22), B11, P3);
        strassen(A22, subtractMatrices(B21, B11), P4);
        strassen(addMatrices(A11, A22), addMatrices(B11, B22), P5);
        strassen(subtractMatrices(A12, A22), addMatrices(B21, B22), P6);
        strassen(subtractMatrices(A11, A21), addMatrices(B11, B12), P7);


        // Calculate result matrices
        vector<vector<int>> C11 = subtractMatrices(addMatrices(P5, P4), s
        vector<vector<int>> C12 = addMatrices(P1, P2);
        vector<vector<int>> C21 = addMatrices(P3, P4);
        vector<vector<int>> C22 = subtractMatrices(addMatrices(P5, P1), a


        // Combine result matrices
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                result[i][j] = C11[i][j];
                result[i][j + size] = C12[i][j];
                result[i + size][j] = C21[i][j];
                result[i + size][j + size] = C22[i][j];
            }
        }
    }
}


// Function to measure the execution time of the Strassen algorithm
double measureExecutionTime(const vector<vector<int>>& A, const vector<ve
    auto start = chrono::high_resolution_clock::now();

    int n = A.size();
```

```
    vector<vector<int>> result(n, vector<int>(n, 0));

    strassen(A, B, result);


    auto end = chrono::high_resolution_clock::now();


    chrono::duration<double> duration = end - start;
    return duration.count();
}


int main() {
    // Get matrix size from the user
    int n;
    cin >> n;


    // Generate random matrices
    vector<vector<int>> A(n, vector<int>(n));
    vector<vector<int>> B(n, vector<int>(n));


    // Fill matrices with random values
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> A[i][j];
        }
    }


    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> B[i][j];
        }
    }


    // Measure execution time for Strassen's multiplication
    double executionTime = measureExecutionTime(A, B);


    // Print result and execution time
    cout << "\nMatrix A:" << endl;
    for (const auto& row : A) {
        for (int elem : row) {
            cout << elem << " ";
        }
        cout << endl;
    }


    cout << "\nMatrix B:" << endl;
    for (const auto& row : B) {
        for (int elem : row) {
            cout << elem << " ";
        }
        cout << endl;
    }
```

```
    cout << "\nResult (Strassen's Multiplication):" << endl;
    vector<vector<int>> result(n, vector<int>(n, 0));
    strassen(A, B, result);
    for (const auto& row : result) {
        for (int elem : row) {
            cout << elem << " ";
        }
        cout << endl;
    }
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2 1 2 3 4 5 6 7 8	Matrix A: 1 2 3 4 Matrix B: 5 6 7 8 Result (Strassen's Multiplication): 19 22 43 50
2	4 1 7 8 7 6 8 7 8 9 7 11 12 3 4 15 6 2 1 4 3 7 4 8 5 1 9 8 0 12 1 16 10	Matrix A: 1 7 8 7 6 8 7 8 9 7 11 12 3 4 15 6 Matrix B: 2 1 4 3 7 4 8 5 1 9 8 0 12 1 16 10 Result (Strassen's Multiplication): 143 108 236 108 171 109 272 138 222 148 372 182 121 160 260 89
3	4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 2 1 4 3 7 6 8 5 11 9 13 10 12 14 16 15	Matrix A: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Matrix B: 2 1 4 3 7 6 8 5 11 9 13 10 12 14 16 15 Result (Strassen's Multiplication): 97 96 123 103 225 216 287 235 353 336 451 367 481 456 615 499
4	4 2 1 4 3 7 6 8 5 11 9 13 10 12 14 16 15 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	Matrix A: 2 1 4 3 7 6 8 5 11 9 13 10 12 14 16 15 Matrix B: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Result (Strassen's Multiplication): 82 92 102 112 174 200 226 252 303 346 389 432 421 478 535 592
5	4 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 -3 -2 -1 0 -7 -6 -5 -4 -11 -10 -9 -8 -15 -14 -13 -12	Matrix A: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Matrix B: -3 -2 -1 0 -7 -6 -5 -4 -11 -10 -9 -8 -15 -14 -13 -12 Result (Strassen's Multiplication): -74 -68 -62 -56 -218 -196 -174 -152 -362 -324 -286 -248 -506 -452 -398 -344
6	2 -1 0 3 2 0 -2 -3 1	Matrix A: -1 0 3 2 Matrix B: 0 -2 -3 1 Result (Strassen's Multiplication): 0 2 -6 -4

White List:

Black List:

Question 6:

Consider that you are working for a company that sells multiple products. The sales data for each product is stored in matrices, and you need to calculate the total revenue generated for each product by multiplying the quantity sold (Matrix A) with the price per unit (Matrix B).

Input Format:

- The first line contains two integers, row_A and col_A, representing the number of rows and columns of Matrix A.
- The next row_A lines each contain col_A integers, representing the elements of Matrix A.
- The following line contains two integers, row_B and col_B, representing the number of rows and columns of Matrix B.
- The next row_B lines each contain col_B integers, representing the elements of Matrix B.

Output Format:

- Display Matrix A, Matrix B, and the resulting matrix in the specified format.

Title for Question 6: Matrix Multiplication

Solution:

```
#include <iostream>
#include<iomanip>
#include <vector>


using namespace std;


void print(const string& display, const vector<vector<int>>& matrix) {
    cout << endl << display << " =>" << endl;
    for (const auto& row : matrix) {
        for (int element : row) {
            cout << element<<" ";
        }
        cout << endl;
    }
    cout << endl;
}


vector<vector<int>> multiply_matrix(const vector<vector<int>>& matrix_A,
    int col_1 = matrix_A[0].size();
    int row_1 = matrix_A.size();
    int col_2 = matrix_B[0].size();
    int row_2 = matrix_B.size();


    if (col_1 != row_2) {
        cout << "\nError: The number of columns in Matrix A must be equal
        return {};
    }


    vector<vector<int>> result_matrix(row_1, vector<int>(col_2, 0));


    for (int i = 0; i < row_1; ++i) {
        for (int j = 0; j < col_2; ++j) {
            for (int k = 0; k < col_1; ++k) {
```

```
                result_matrix[i][j] += matrix_A[i][k] * matrix_B[k][j];
            }
        }
    }


    return result_matrix;
}
int main() {
    int row_A, col_A, row_B, col_B;
    cin >> row_A >> col_A;
    vector<vector<int>> matrix_A(row_A, vector<int>(col_A));
    for (int i = 0; i < row_A; ++i) {
        for (int j = 0; j < col_A; ++j) {
            cin >> matrix_A[i][j];
        }
    }
    cin >> row_B >> col_B;
    vector<vector<int>> matrix_B(row_B, vector<int>(col_B));
    for (int i = 0; i < row_B; ++i) {
        for (int j = 0; j < col_B; ++j) {
            cin >> matrix_B[i][j];
        }
    }
    if (col_A != row_B) {
        cout << "\nError: The number of columns in Matrix A must be equal
        return 1;
    }
    vector<vector<int>> result_matrix = multiply_matrix(matrix_A, matrix_
    print("Array A", matrix_A);
    print("Array B", matrix_B);
    print("Result Array", result_matrix);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2 2 1 2 3 4 2 2 5 6 7 8	Array A => 1 2 3 4 Array B => 5 6 7 8 Result Array => 19 22 43 50
2	3 3 2 0 1 3 2 1 1 2 3 3 3 1 0 1 1 2 3 3 1 2	Array A => 2 0 1 3 2 1 1 2 3 Array B => 1 0 1 1 2 3 3 1 2 Result Array => 5 1 4 8 5 11 12 7 13
3	2 2 2 3 4 5 2 2 1 0 0 1	Array A => 2 3 4 5 Array B => 1 0 0 1 Result Array => 2 3 4 5
4	3 1 1 2 3 1 3 4 5 6	Array A => 1 2 3 Array B => 4 5 6 Result Array => 4 5 6 8 10 12 12 15 18
5	2 3 1 2 3 4 5 6 3 2 7 8 9 10 11 12	Array A => 1 2 3 4 5 6 Array B => 7 8 9 10 11 12 Result Array => 58 64 139 154
6	2 2 1 0 0 1 2 2 1 2 3 4	Array A => 1 0 0 1 Array B => 1 2 3 4 Result Array => 1 2 3 4

White List:

Black List:
