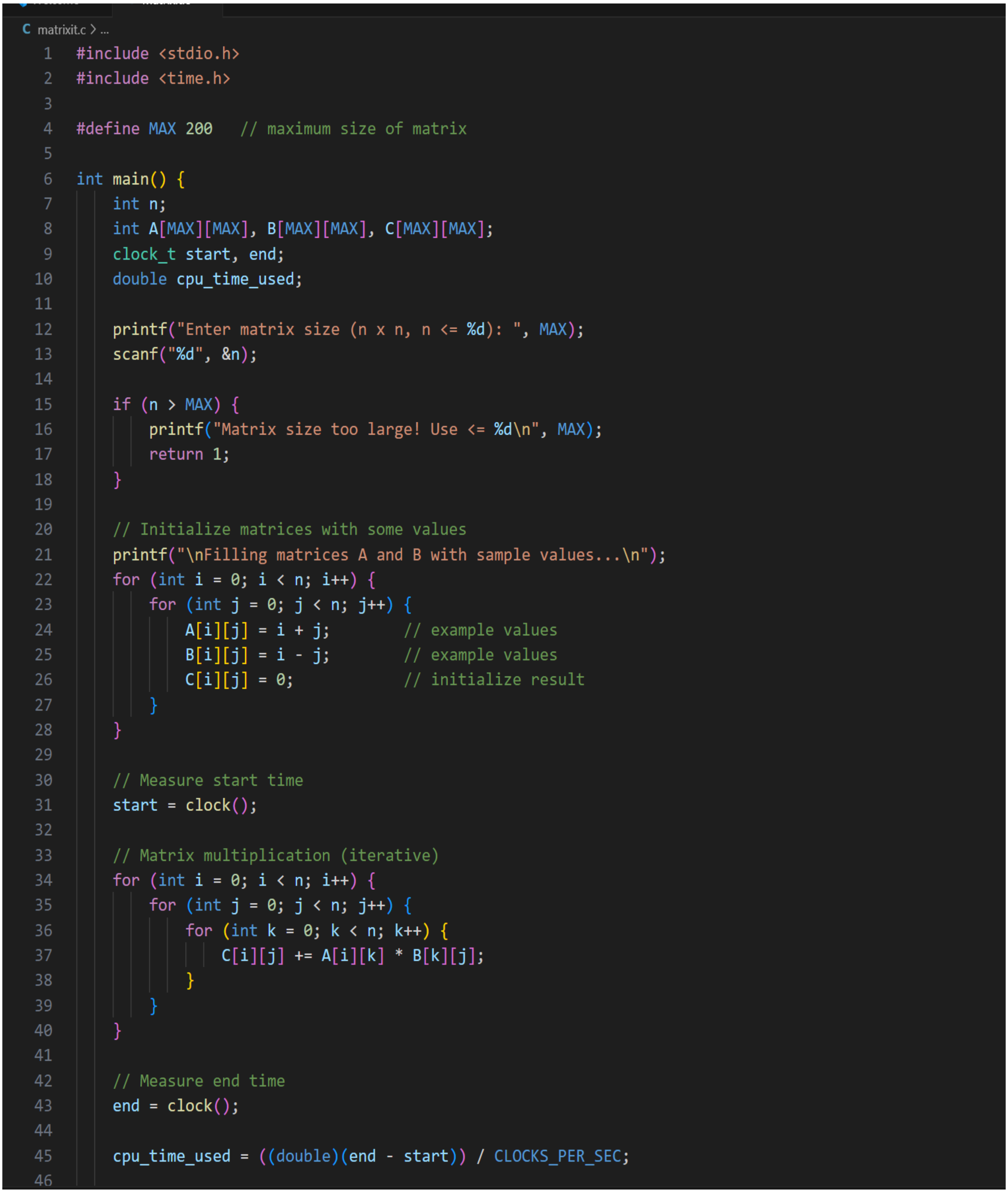
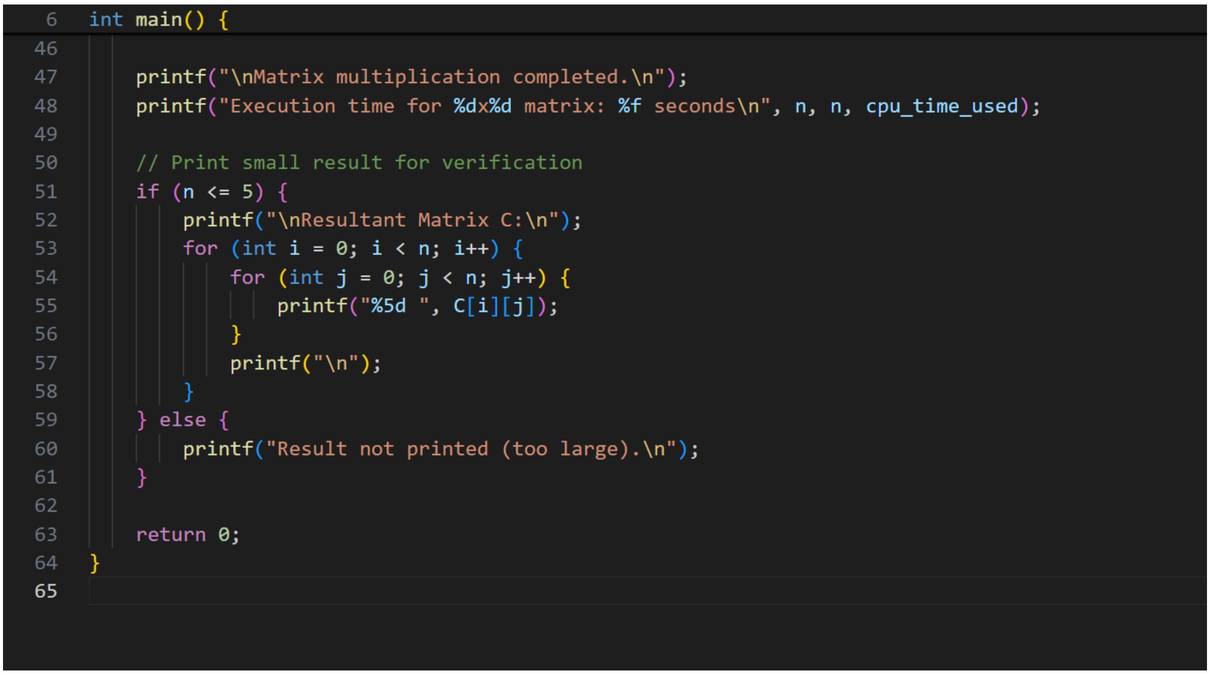
3(a). Write a program in C language to multiply two square matrices using the **iterative approach**. Compare the execution time for different matrix sizes.

Code:

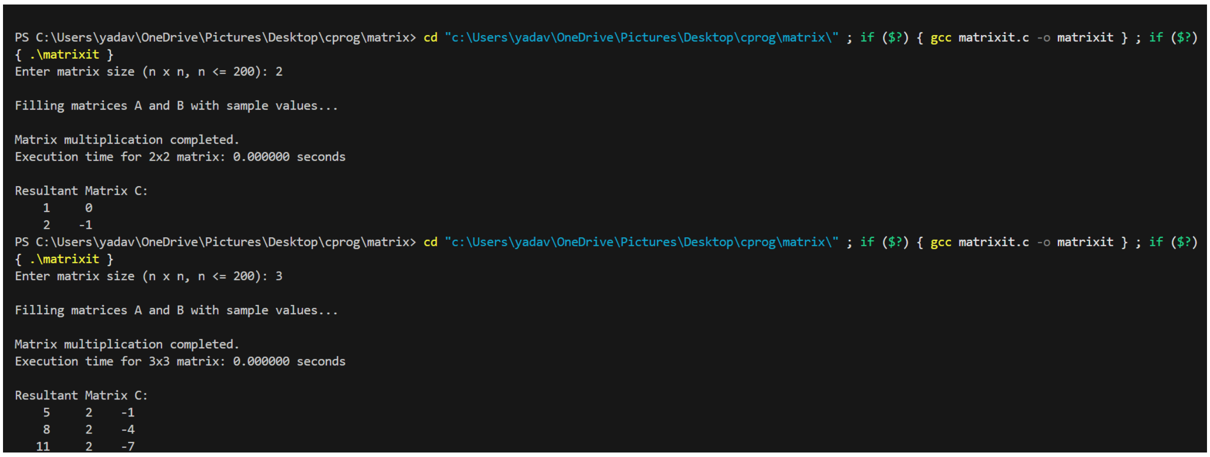


```c
#include <stdio.h>
#include <time.h>

#define MAX 200   // maximum size of matrix

int main() {
    int n;
    int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];
    clock_t start, end;
    double cpu_time_used;

    printf("Enter matrix size (n x n, n <= %d): ", MAX);
    scanf("%d", &n);

    if (n > MAX) {
        printf("Matrix size too large! Use <= %d\n", MAX);
        return 1;
    }

    // Initialize matrices with some values
    printf("\nFilling matrices A and B with sample values...\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = i + j;        // example values
            B[i][j] = i - j;        // example values
            C[i][j] = 0;            // initialize result
        }
    }

    // Measure start time
    start = clock();

    // Matrix multiplication (iterative)
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    // Measure end time
    end = clock();

    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

```



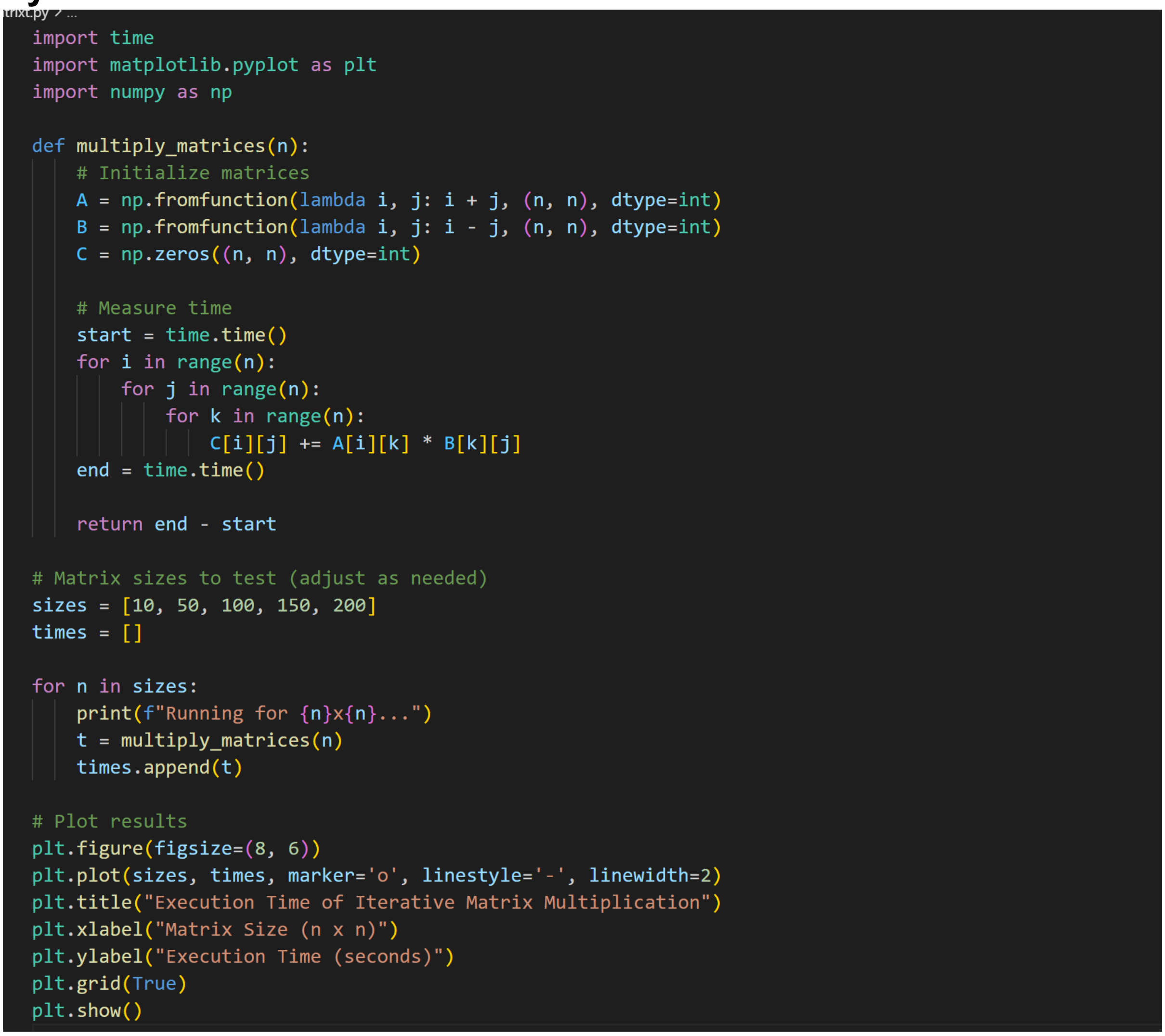
```c
int main() {

    printf("\nMatrix multiplication completed.\n");
    printf("Execution time for %dx%d matrix: %f seconds\n", n, n, cpu_time_used);

    // Print small result for verification
    if (n <= 5) {
        printf("\nResultant Matrix C:\n");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%5d ", C[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("Result not printed (too large).\n");
    }

    return 0;
}
```

Output:



```
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixit.c -o matrixit } ; if ($?)
{ .\matrixit }
Enter matrix size (n x n, n <= 200): 2

Filling matrices A and B with sample values...

Matrix multiplication completed.
Execution time for 2x2 matrix: 0.000000 seconds

Resultant Matrix C:
    1     0
    2    -1
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixit.c -o matrixit } ; if ($?)
{ .\matrixit }
Enter matrix size (n x n, n <= 200): 3

Filling matrices A and B with sample values...

Matrix multiplication completed.
Execution time for 3x3 matrix: 0.000000 seconds

Resultant Matrix C:
    5     2    -1
    8     2    -4
   11     2    -7
```

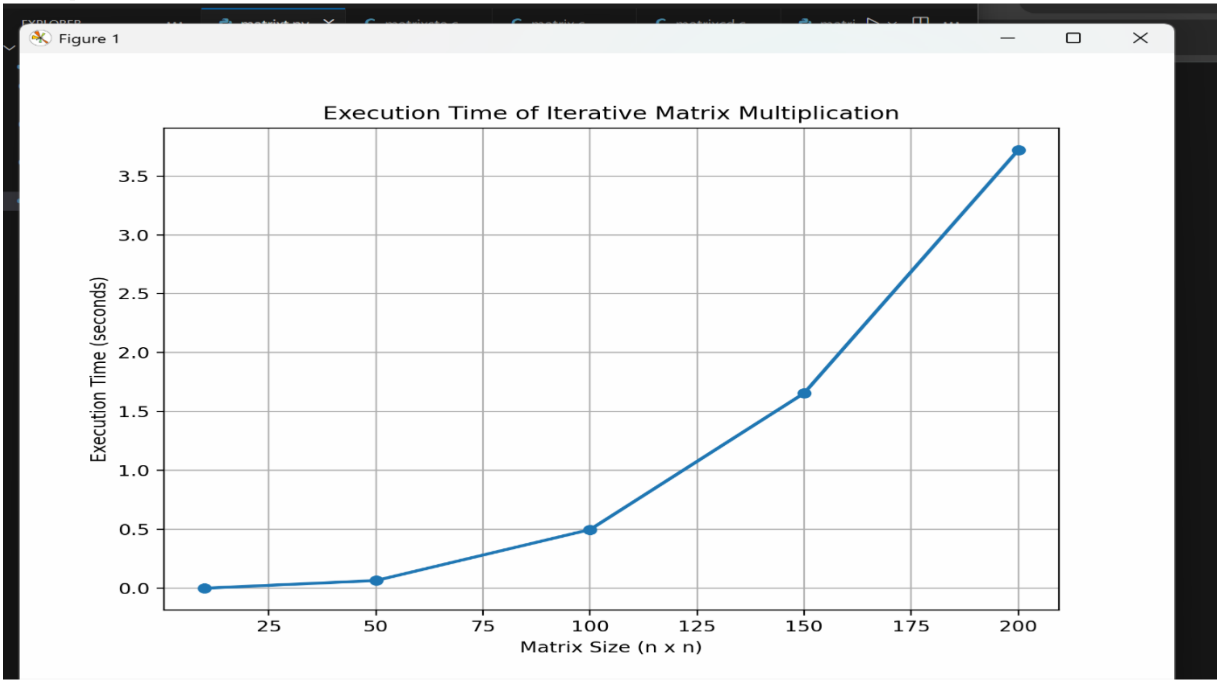
Python code:



trixt.py > ...

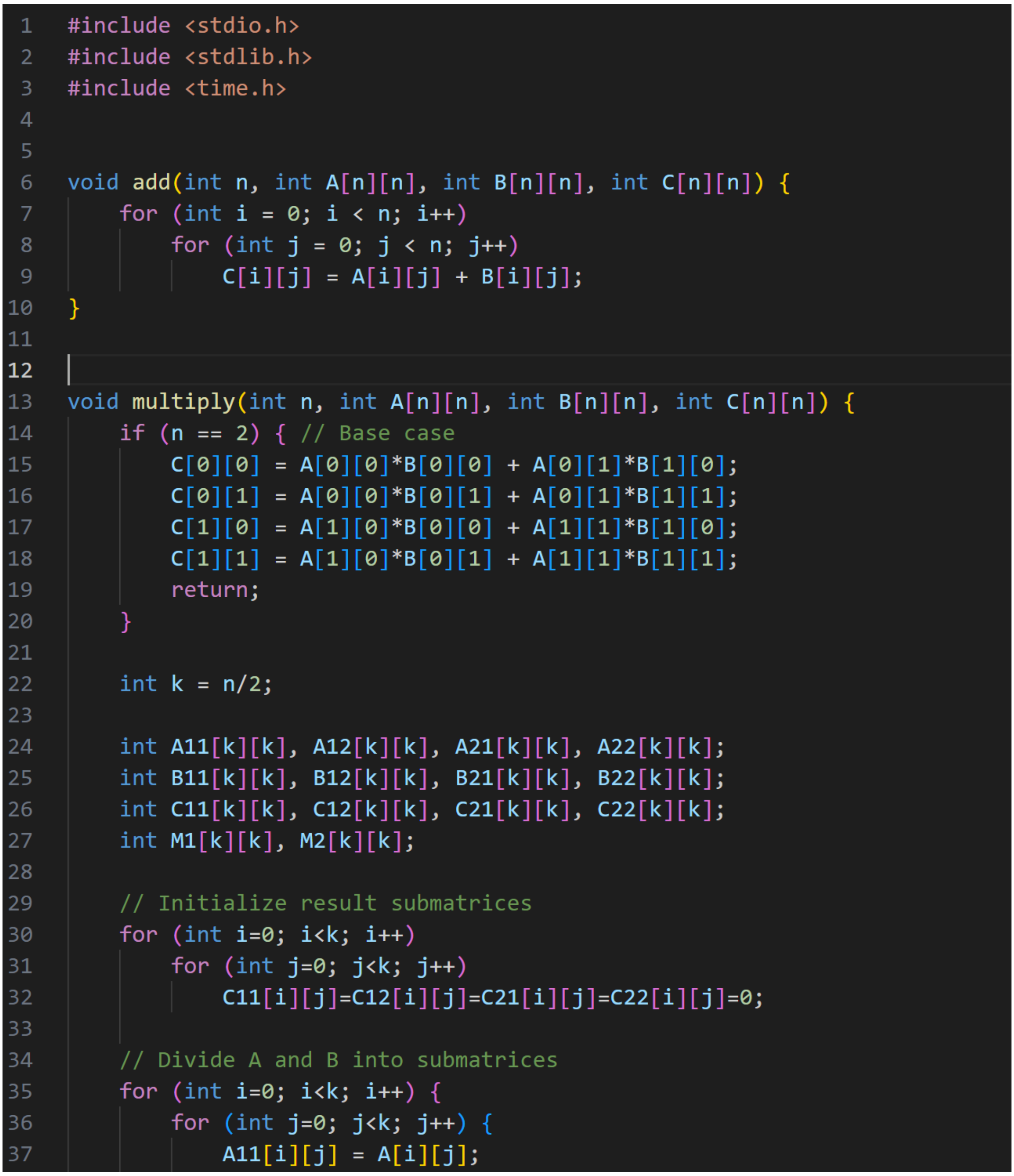
```python
import time
import matplotlib.pyplot as plt
import numpy as np

def multiply_matrices(n):
    # Initialize matrices
    A = np.fromfunction(lambda i, j: i + j, (n, n), dtype=int)
    B = np.fromfunction(lambda i, j: i - j, (n, n), dtype=int)
    C = np.zeros((n, n), dtype=int)

    # Measure time
    start = time.time()
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    end = time.time()

    return end - start

# Matrix sizes to test (adjust as needed)
sizes = [10, 50, 100, 150, 200]
times = []

for n in sizes:
    print(f"Running for {n}x{n}...")
    t = multiply_matrices(n)
    times.append(t)

# Plot results
plt.figure(figsize=(8, 6))
plt.plot(sizes, times, marker='o', linestyle='-', linewidth=2)
plt.title("Execution Time of Iterative Matrix Multiplication")
plt.xlabel("Matrix Size (n x n)")
plt.ylabel("Execution Time (seconds)")
plt.grid(True)
plt.show()
```

Graph:

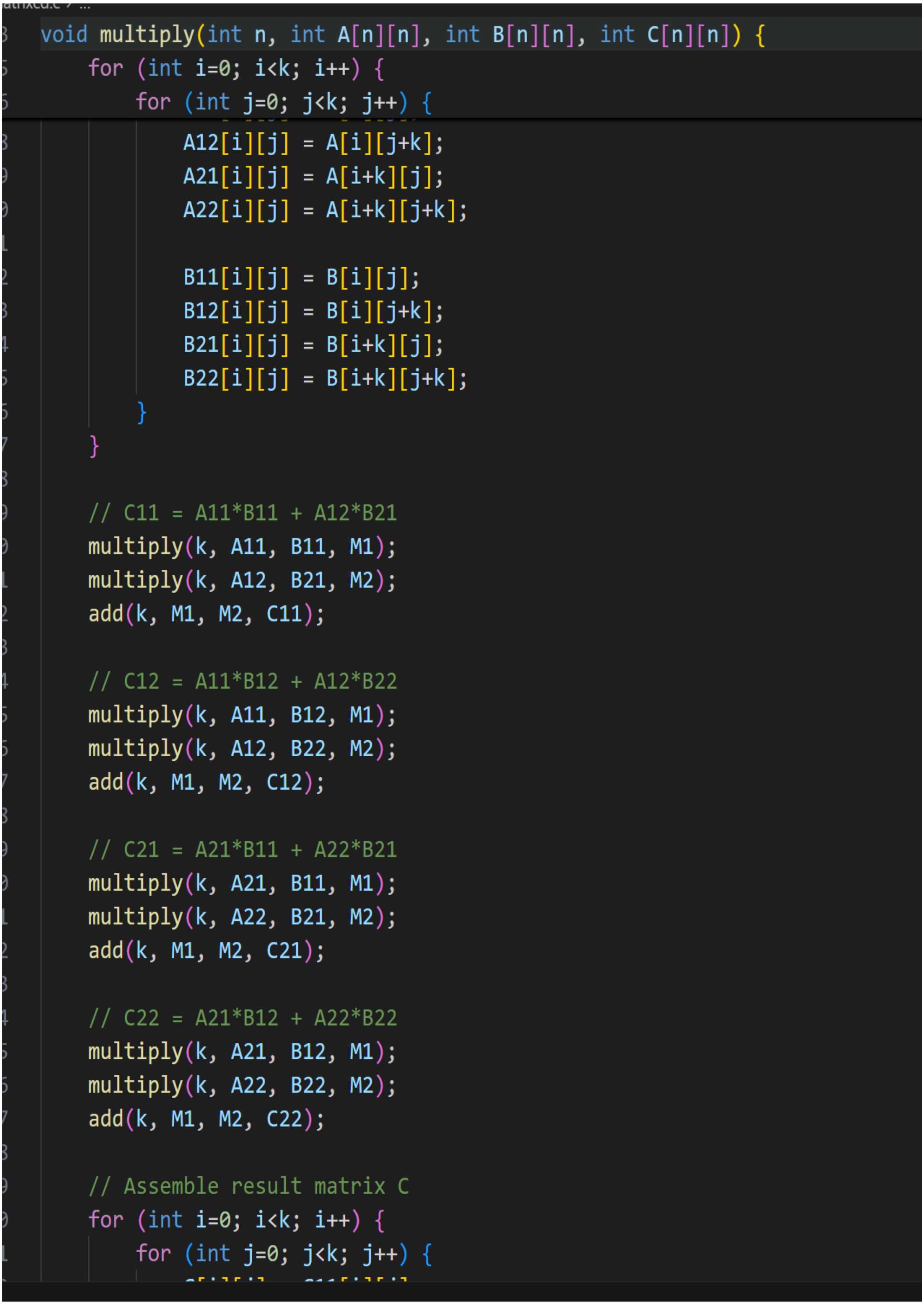


3(b) Write a program in C language to multiply two square matrices using the divide and the conquer. Compare the execution time for different matrix sizes.

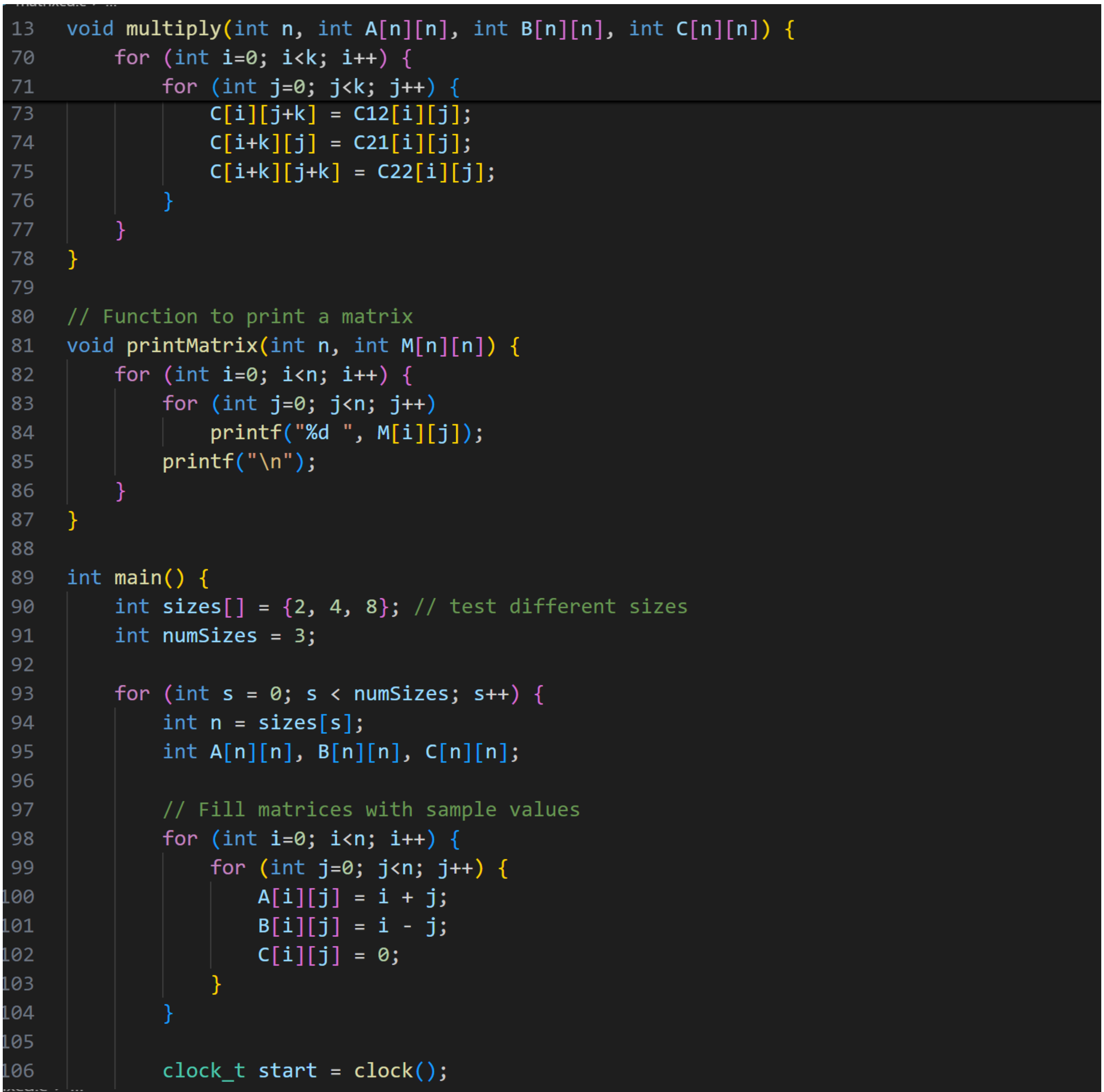
Code:



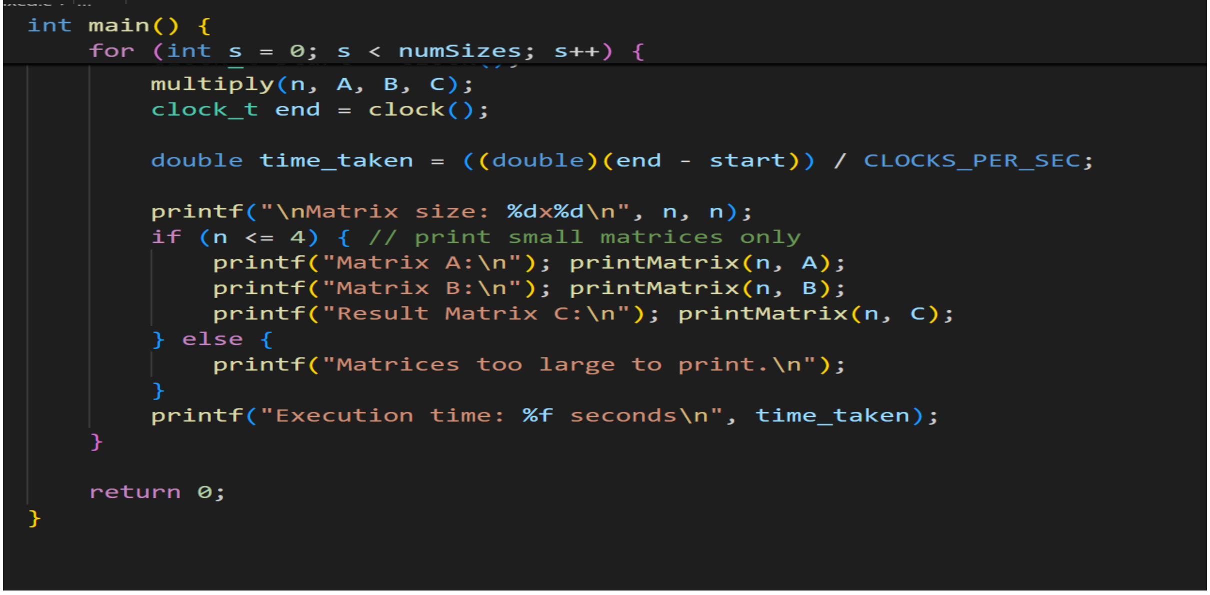
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


void add(int n, int A[n][n], int B[n][n], int C[n][n]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
}


void multiply(int n, int A[n][n], int B[n][n], int C[n][n]) {
    if (n == 2) { // Base case
        C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0];
        C[0][1] = A[0][0]*B[0][1] + A[0][1]*B[1][1];
        C[1][0] = A[1][0]*B[0][0] + A[1][1]*B[1][0];
        C[1][1] = A[1][0]*B[0][1] + A[1][1]*B[1][1];
        return;
    }

    int k = n/2;

    int A11[k][k], A12[k][k], A21[k][k], A22[k][k];
    int B11[k][k], B12[k][k], B21[k][k], B22[k][k];
    int C11[k][k], C12[k][k], C21[k][k], C22[k][k];
    int M1[k][k], M2[k][k];

    // Initialize result submatrices
    for (int i=0; i<k; i++)
        for (int j=0; j<k; j++)
            C11[i][j]=C12[i][j]=C21[i][j]=C22[i][j]=0;

    // Divide A and B into submatrices
    for (int i=0; i<k; i++) {
        for (int j=0; j<k; j++) {
            A11[i][j] = A[i][j];
```



```c
void multiply(int n, int A[n][n], int B[n][n], int C[n][n]) {
    for (int i=0; i<k; i++) {
        for (int j=0; j<k; 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];
        }
    }

    // C11 = A11*B11 + A12*B21
    multiply(k, A11, B11, M1);
    multiply(k, A12, B21, M2);
    add(k, M1, M2, C11);

    // C12 = A11*B12 + A12*B22
    multiply(k, A11, B12, M1);
    multiply(k, A12, B22, M2);
    add(k, M1, M2, C12);

    // C21 = A21*B11 + A22*B21
    multiply(k, A21, B11, M1);
    multiply(k, A22, B21, M2);
    add(k, M1, M2, C21);

    // C22 = A21*B12 + A22*B22
    multiply(k, A21, B12, M1);
    multiply(k, A22, B22, M2);
    add(k, M1, M2, C22);

    // Assemble result matrix C
    for (int i=0; i<k; i++) {
        for (int j=0; j<k; j++) {
```

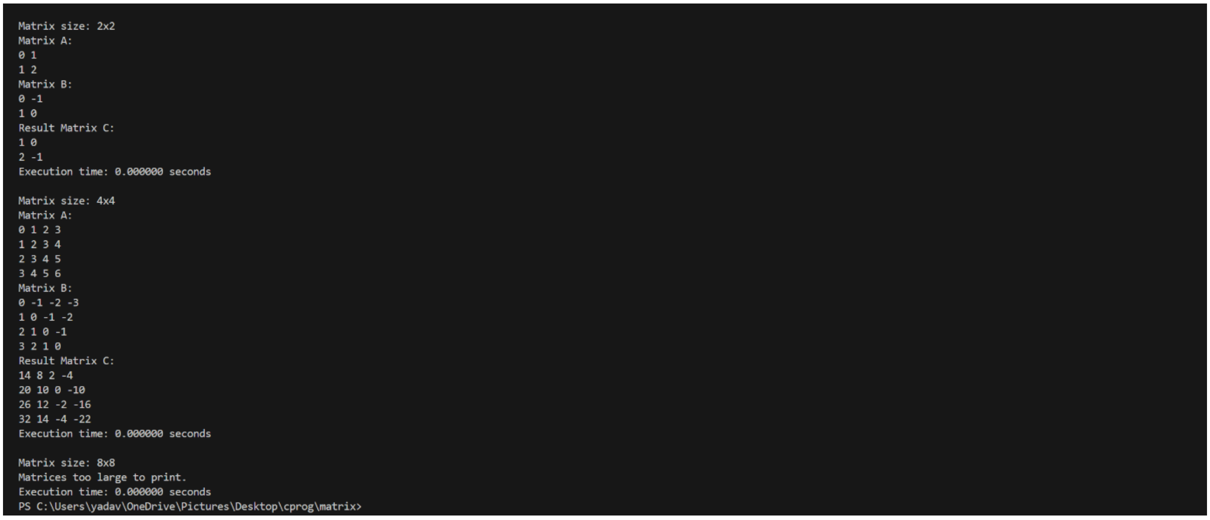


```c
void multiply(int n, int A[n][n], int B[n][n], int C[n][n]) {
    for (int i=0; i<k; i++) {
        for (int j=0; j<k; j++) {
            C[i][j+k] = C12[i][j];
            C[i+k][j] = C21[i][j];
            C[i+k][j+k] = C22[i][j];
        }
    }
}

// Function to print a matrix
void printMatrix(int n, int M[n][n]) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++)
            printf("%d ", M[i][j]);
        printf("\n");
    }
}

int main() {
    int sizes[] = {2, 4, 8}; // test different sizes
    int numSizes = 3;

    for (int s = 0; s < numSizes; s++) {
        int n = sizes[s];
        int A[n][n], B[n][n], C[n][n];

        // Fill matrices with sample values
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                A[i][j] = i + j;
                B[i][j] = i - j;
                C[i][j] = 0;
            }
        }

        clock_t start = clock();
```



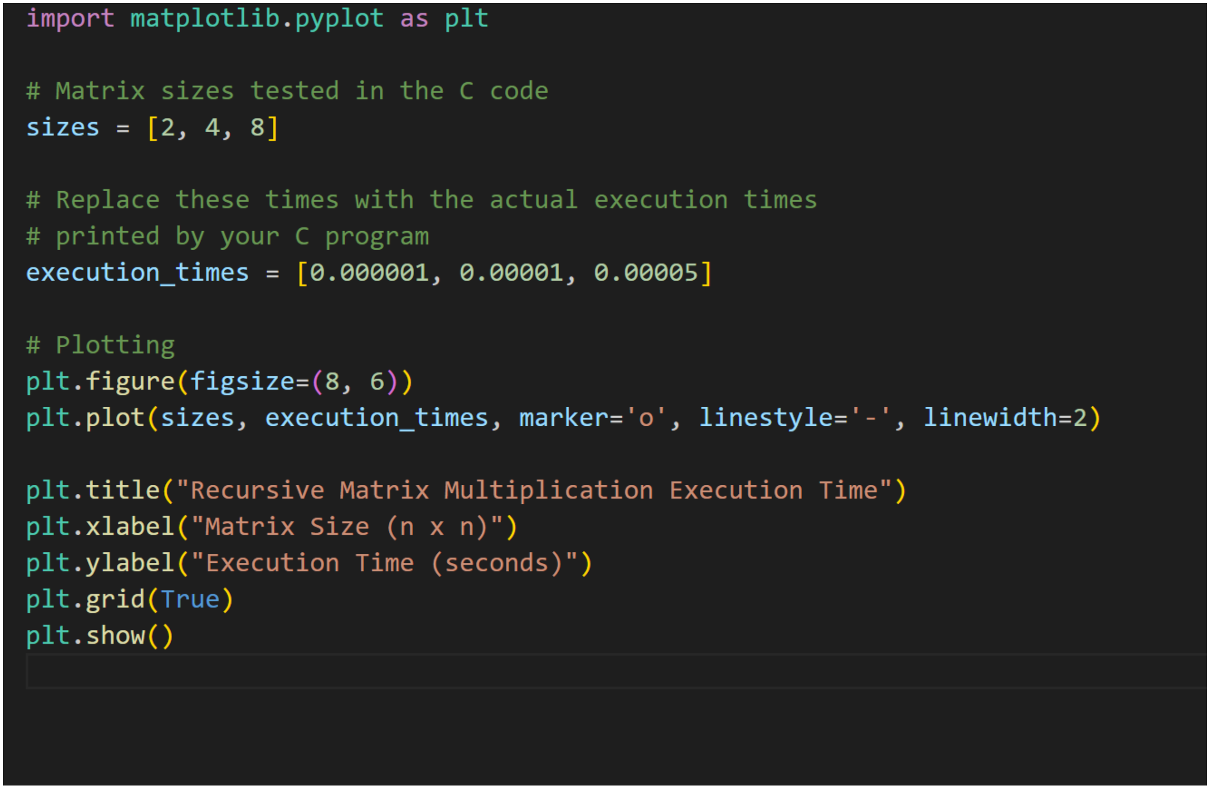
```c
int main() {
    for (int s = 0; s < numSizes; s++) {
        multiply(n, A, B, C);
        clock_t end = clock();

        double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

        printf("\nMatrix size: %dx%d\n", n, n);
        if (n <= 4) { // print small matrices only
            printf("Matrix A:\n"); printMatrix(n, A);
            printf("Matrix B:\n"); printMatrix(n, B);
            printf("Result Matrix C:\n"); printMatrix(n, C);
        } else {
            printf("Matrices too large to print.\n");
        }
        printf("Execution time: %f seconds\n", time_taken);
    }

    return 0;
}
```

Output:



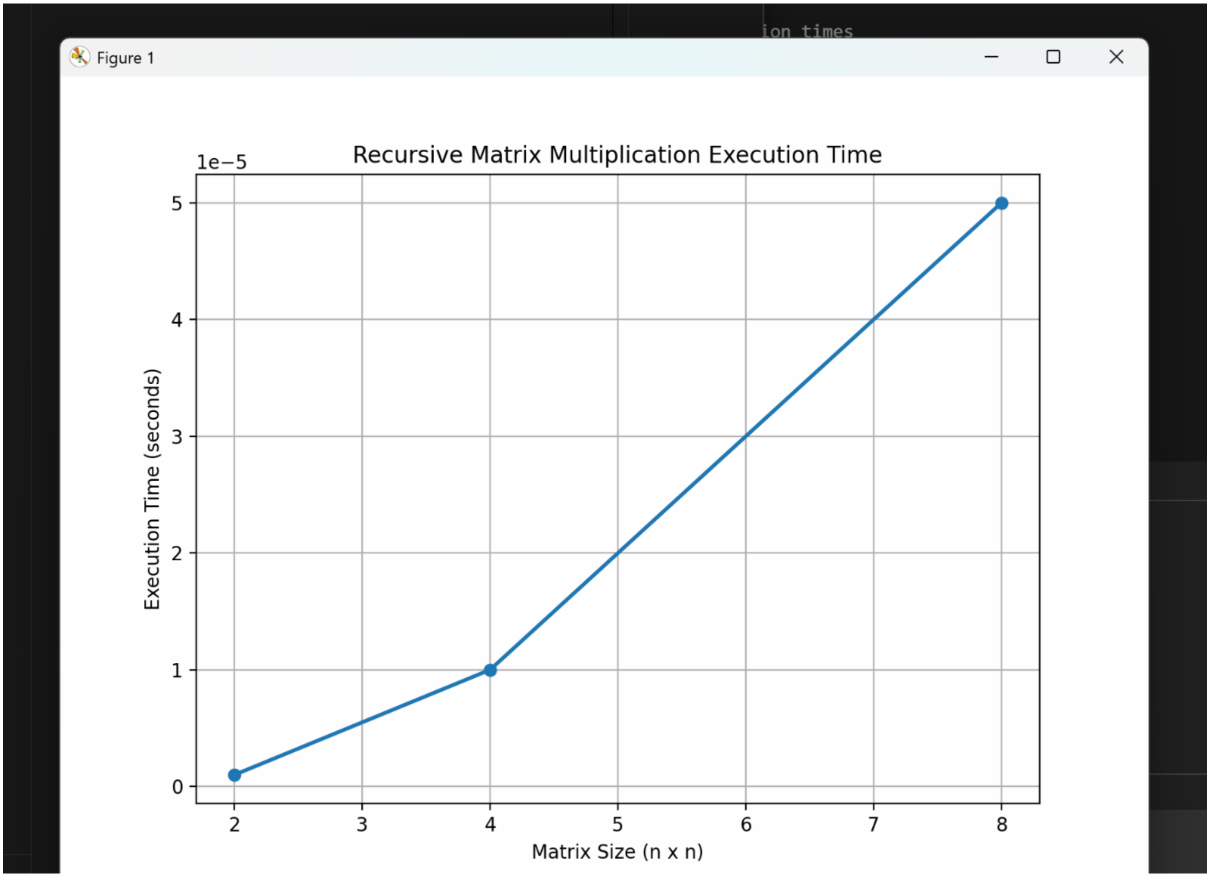
```
Matrix size: 2x2
Matrix A:
0 1
1 2
Matrix B:
0 -1
1 0
Result Matrix C:
1 0
2 -1
Execution time: 0.000000 seconds

Matrix size: 4x4
Matrix A:
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
Matrix B:
0 -1 -2 -3
1 0 -1 -2
2 1 0 -1
3 2 1 0
Result Matrix C:
14 8 2 -4
20 10 0 -10
26 12 -2 -16
32 14 -4 -22
Execution time: 0.000000 seconds

Matrix size: 8x8
Matrices too large to print.
Execution time: 0.000000 seconds
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix>
```

Python code:



```python
import matplotlib.pyplot as plt

# Matrix sizes tested in the C code
sizes = [2, 4, 8]

# Replace these times with the actual execution times
# printed by your C program
execution_times = [0.000001, 0.00001, 0.00005]

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(sizes, execution_times, marker='o', linestyle='-', linewidth=2)

plt.title("Recursive Matrix Multiplication Execution Time")
plt.xlabel("Matrix Size (n x n)")
plt.ylabel("Execution Time (seconds)")
plt.grid(True)
plt.show()
```

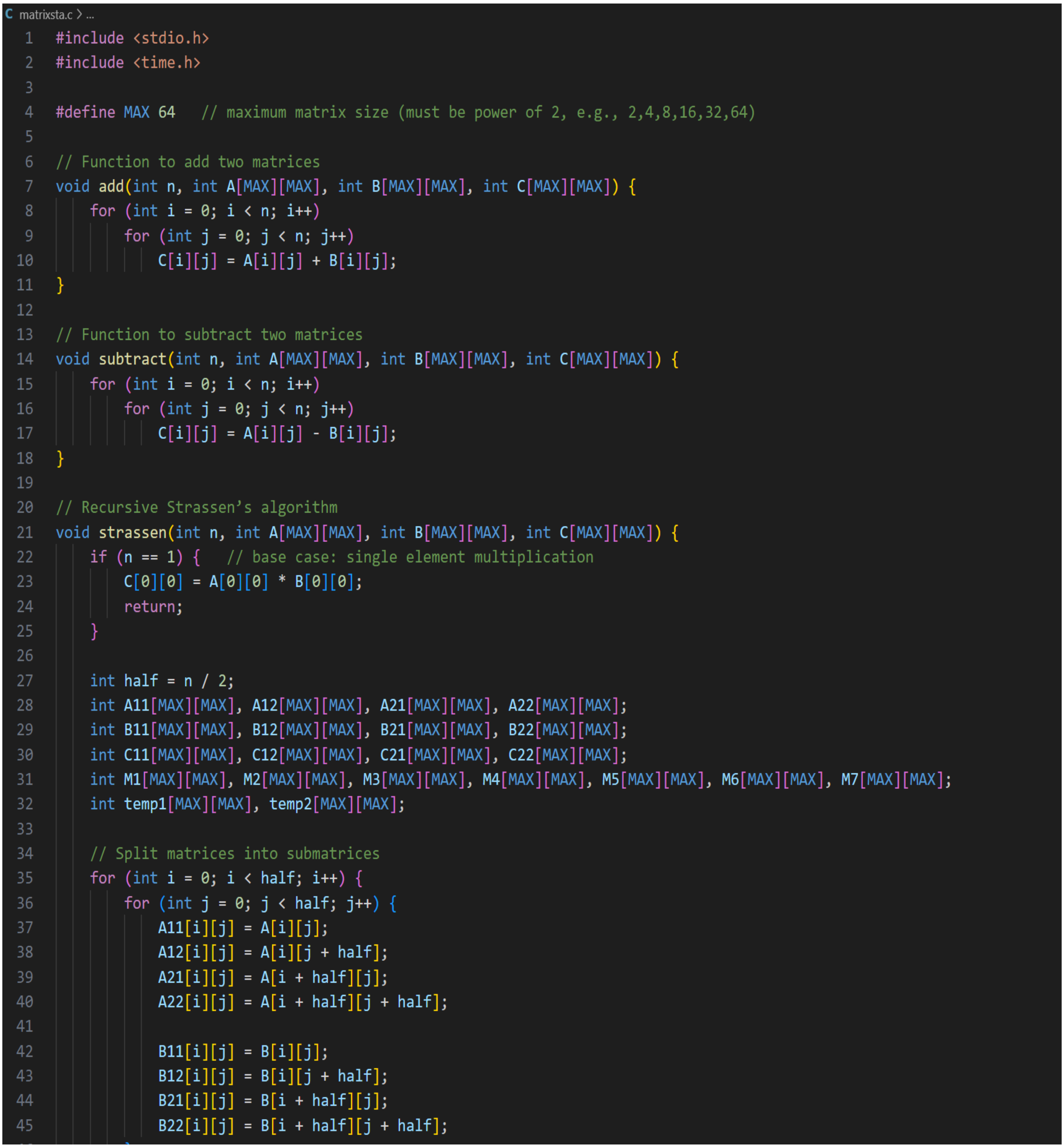
Graph:


ion times
Figure 1
Recursive Matrix Multiplication Execution Time
1e−5
0
1
2
3
4
5
Execution Time (seconds)
2
3
4
5
6
7
8
Matrix Size (n x n)

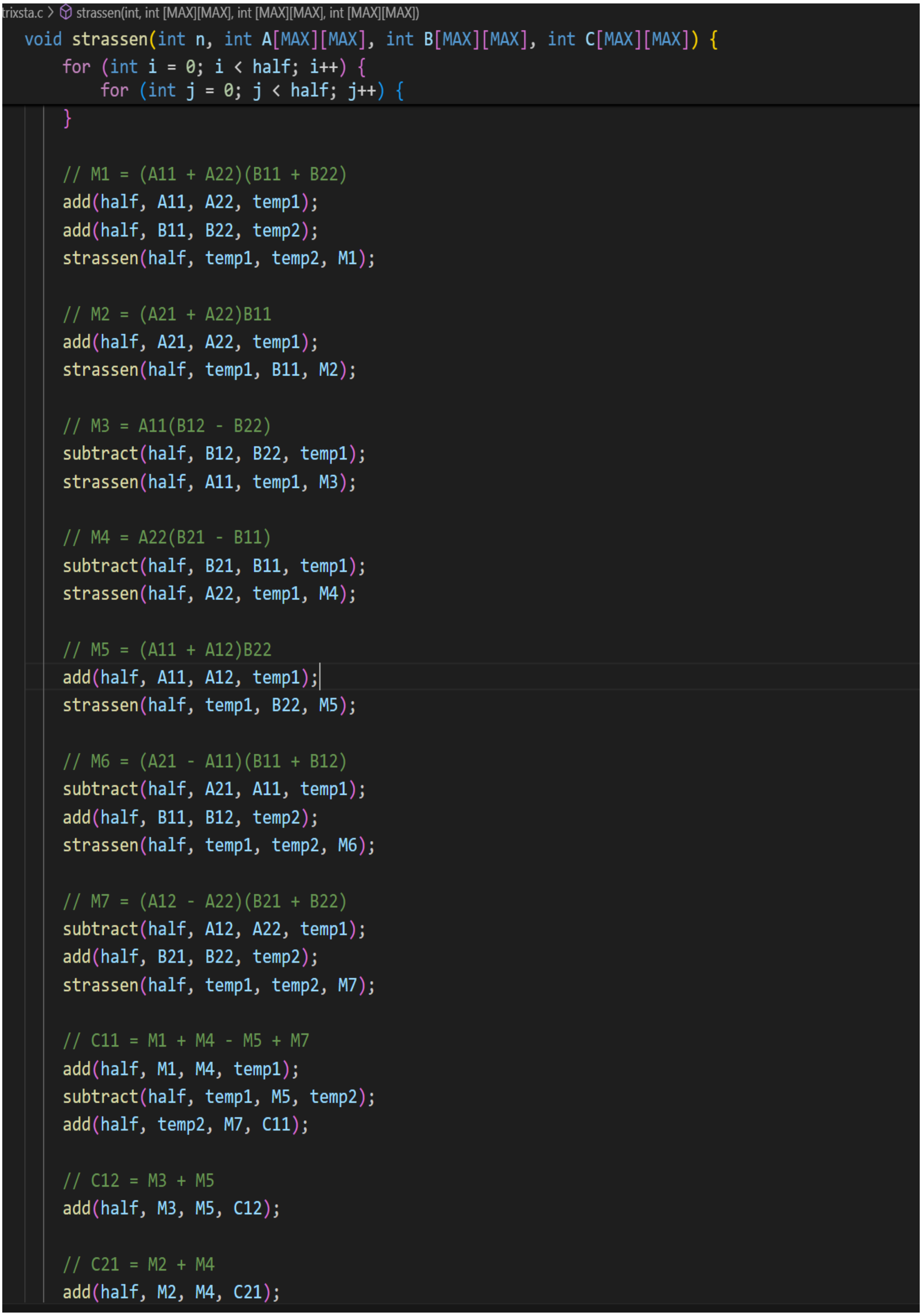
3(c) . Given two square matrices A and B of size n × n (n is a power of 2), write a C code to multiply them using , which reduces the number of recursive multiplications from 8 to 7 by introducing additional addition/subtraction operations. Compare the execution time for different matrix sizes.

Code

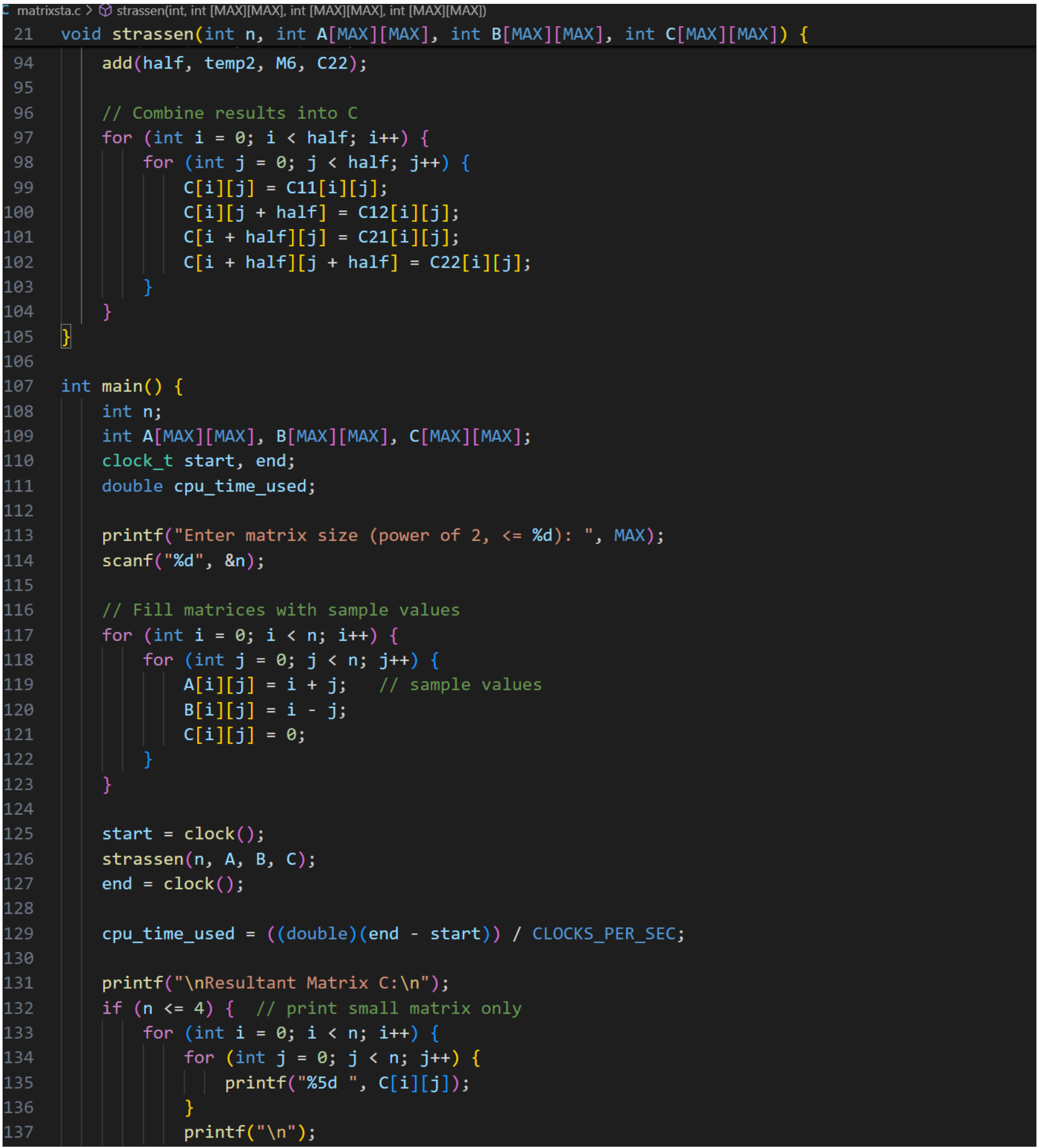
:



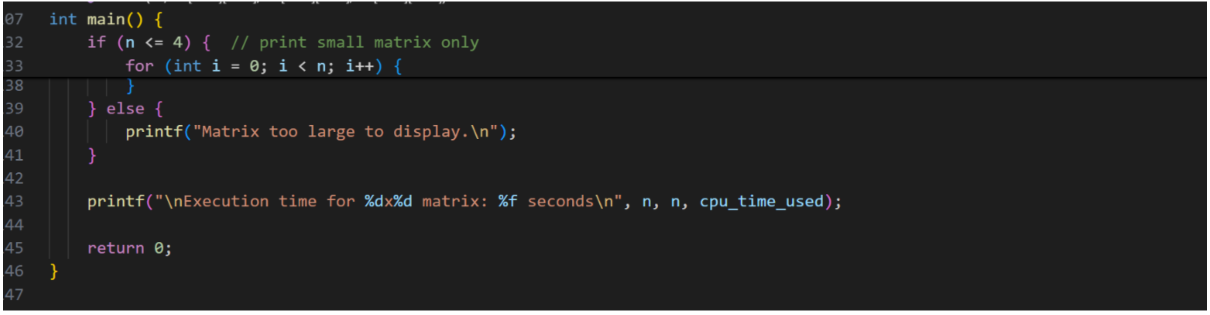
```c
#include <stdio.h>
#include <time.h>

#define MAX 64   // maximum matrix size (must be power of 2, e.g., 2,4,8,16,32,64)

// Function to add two matrices
void add(int n, int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
}

// Function to subtract two matrices
void subtract(int n, int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
}

// Recursive Strassen’s algorithm
void strassen(int n, int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX]) {
    if (n == 1) {   // base case: single element multiplication
        C[0][0] = A[0][0] * B[0][0];
        return;
    }

    int half = n / 2;
    int A11[MAX][MAX], A12[MAX][MAX], A21[MAX][MAX], A22[MAX][MAX];
    int B11[MAX][MAX], B12[MAX][MAX], B21[MAX][MAX], B22[MAX][MAX];
    int C11[MAX][MAX], C12[MAX][MAX], C21[MAX][MAX], C22[MAX][MAX];
    int M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX], M4[MAX][MAX], M5[MAX][MAX], M6[MAX][MAX], M7[MAX][MAX];
    int temp1[MAX][MAX], temp2[MAX][MAX];

    // Split matrices into submatrices
    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];
```


trixsta.c > strassen(int, int [MAX][MAX], int [MAX][MAX], int [MAX][MAX])

```c
void strassen(int n, int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX]) {
    for (int i = 0; i < half; i++) {
        for (int j = 0; j < half; j++) {
    }

    // M1 = (A11 + A22)(B11 + B22)
    add(half, A11, A22, temp1);
    add(half, B11, B22, temp2);
    strassen(half, temp1, temp2, M1);

    // M2 = (A21 + A22)B11
    add(half, A21, A22, temp1);
    strassen(half, temp1, B11, M2);

    // M3 = A11(B12 - B22)
    subtract(half, B12, B22, temp1);
    strassen(half, A11, temp1, M3);

    // M4 = A22(B21 - B11)
    subtract(half, B21, B11, temp1);
    strassen(half, A22, temp1, M4);

    // M5 = (A11 + A12)B22
    add(half, A11, A12, temp1);
    strassen(half, temp1, B22, M5);

    // M6 = (A21 - A11)(B11 + B12)
    subtract(half, A21, A11, temp1);
    add(half, B11, B12, temp2);
    strassen(half, temp1, temp2, M6);

    // M7 = (A12 - A22)(B21 + B22)
    subtract(half, A12, A22, temp1);
    add(half, B21, B22, temp2);
    strassen(half, temp1, temp2, M7);

    // C11 = M1 + M4 - M5 + M7
    add(half, M1, M4, temp1);
    subtract(half, temp1, M5, temp2);
    add(half, temp2, M7, C11);

    // C12 = M3 + M5
    add(half, M3, M5, C12);

    // C21 = M2 + M4
    add(half, M2, M4, C21);
```

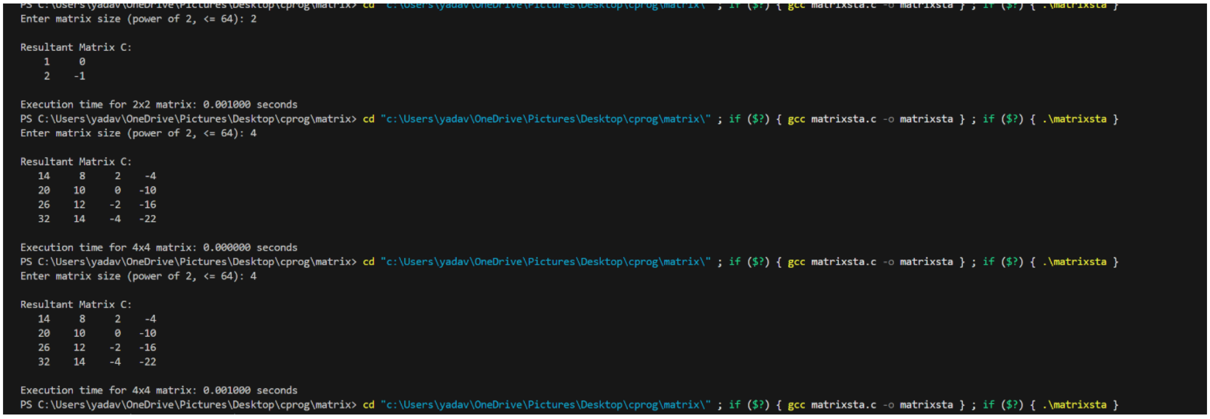

matrixsta.c > strassen(int, int [MAX][MAX], int [MAX][MAX], int [MAX][MAX])

```c
void strassen(int n, int A[MAX][MAX], int B[MAX][MAX], int C[MAX][MAX]) {
    add(half, temp2, M6, C22);

    // Combine results into C
    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];
        }
    }
}

int main() {
    int n;
    int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];
    clock_t start, end;
    double cpu_time_used;

    printf("Enter matrix size (power of 2, <= %d): ", MAX);
    scanf("%d", &n);

    // Fill matrices with sample values
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = i + j;   // sample values
            B[i][j] = i - j;
            C[i][j] = 0;
        }
    }

    start = clock();
    strassen(n, A, B, C);
    end = clock();

    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("\nResultant Matrix C:\n");
    if (n <= 4) {  // print small matrix only
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%5d ", C[i][j]);
            }
            printf("\n");
```



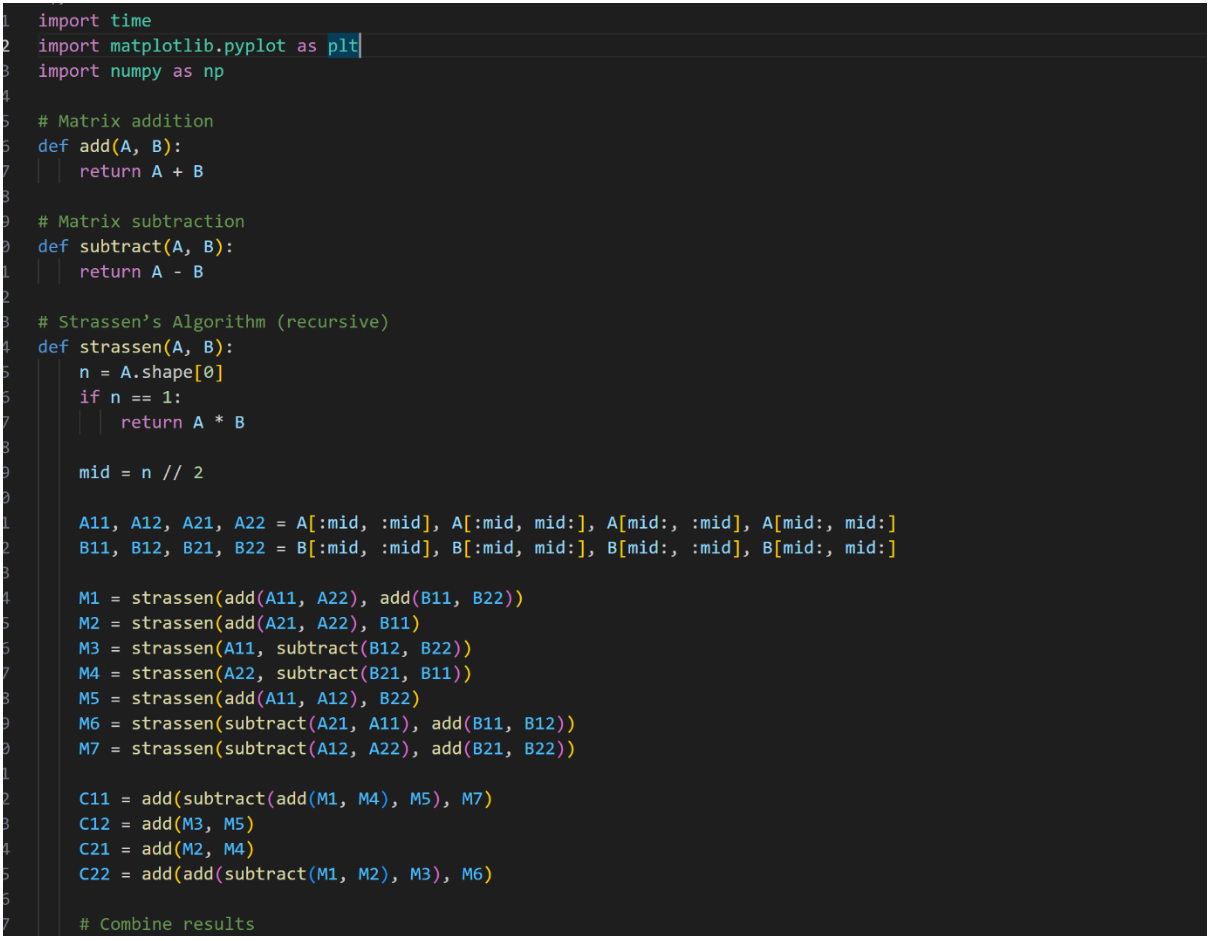
```c
int main() {
    if (n <= 4) {  // print small matrix only
        for (int i = 0; i < n; i++) {
        }
    } else {
        printf("Matrix too large to display.\n");
    }

    printf("\nExecution time for %dx%d matrix: %f seconds\n", n, n, cpu_time_used);

    return 0;
}
```

Output:

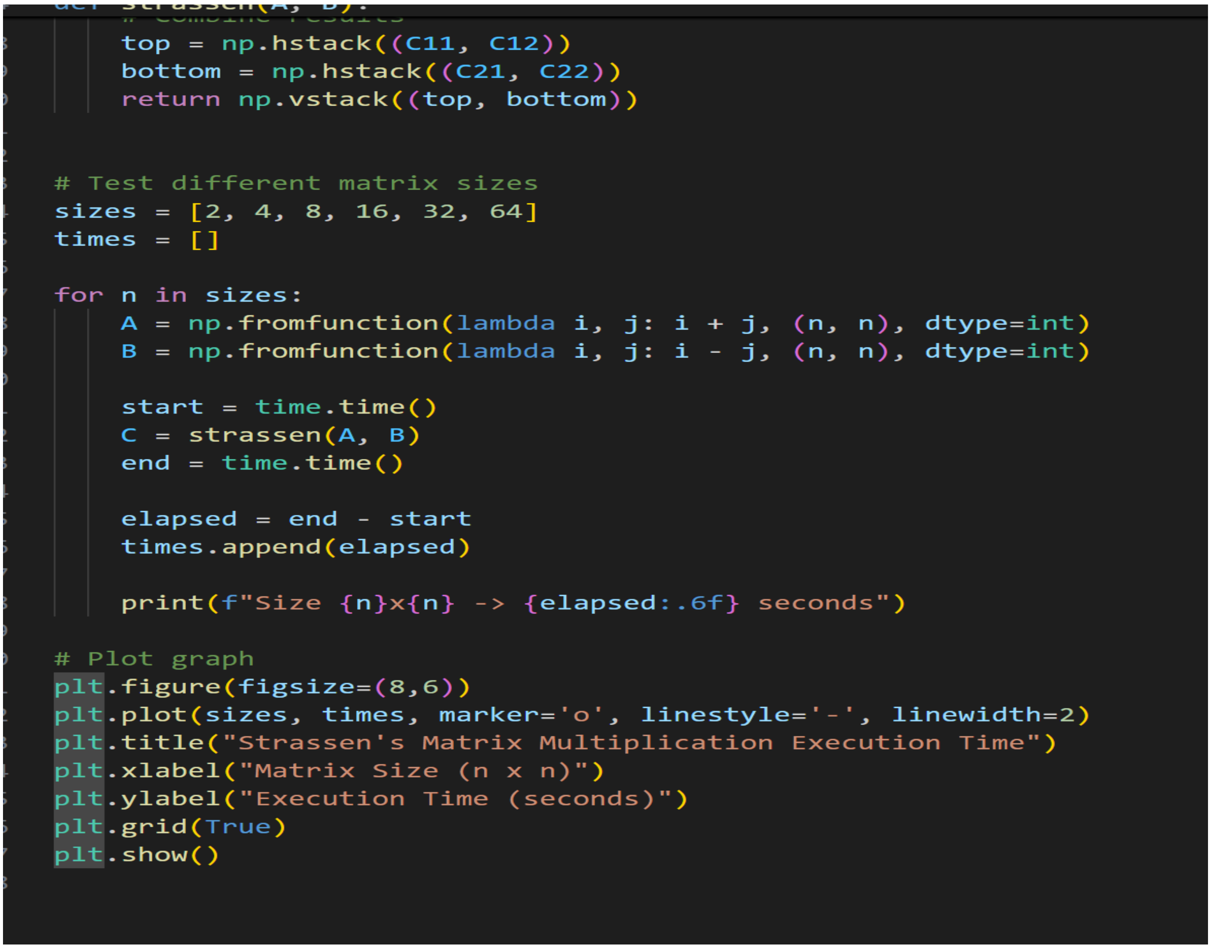


```
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixsta.c -o matrixsta } ; if ($?) { .\matrixsta }
Enter matrix size (power of 2, <= 64): 2

Resultant Matrix C:
    1     0
    2    -1

Execution time for 2x2 matrix: 0.001000 seconds
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixsta.c -o matrixsta } ; if ($?) { .\matrixsta }
Enter matrix size (power of 2, <= 64): 4

Resultant Matrix C:
   14     8     2    -4
   20    10     0   -10
   26    12    -2   -16
   32    14    -4   -22

Execution time for 4x4 matrix: 0.000000 seconds
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixsta.c -o matrixsta } ; if ($?) { .\matrixsta }
Enter matrix size (power of 2, <= 64): 4

Resultant Matrix C:
   14     8     2    -4
   20    10     0   -10
   26    12    -2   -16
   32    14    -4   -22

Execution time for 4x4 matrix: 0.001000 seconds
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc matrixsta.c -o matrixsta } ; if ($?) { .\matrixsta }
```

Python code:

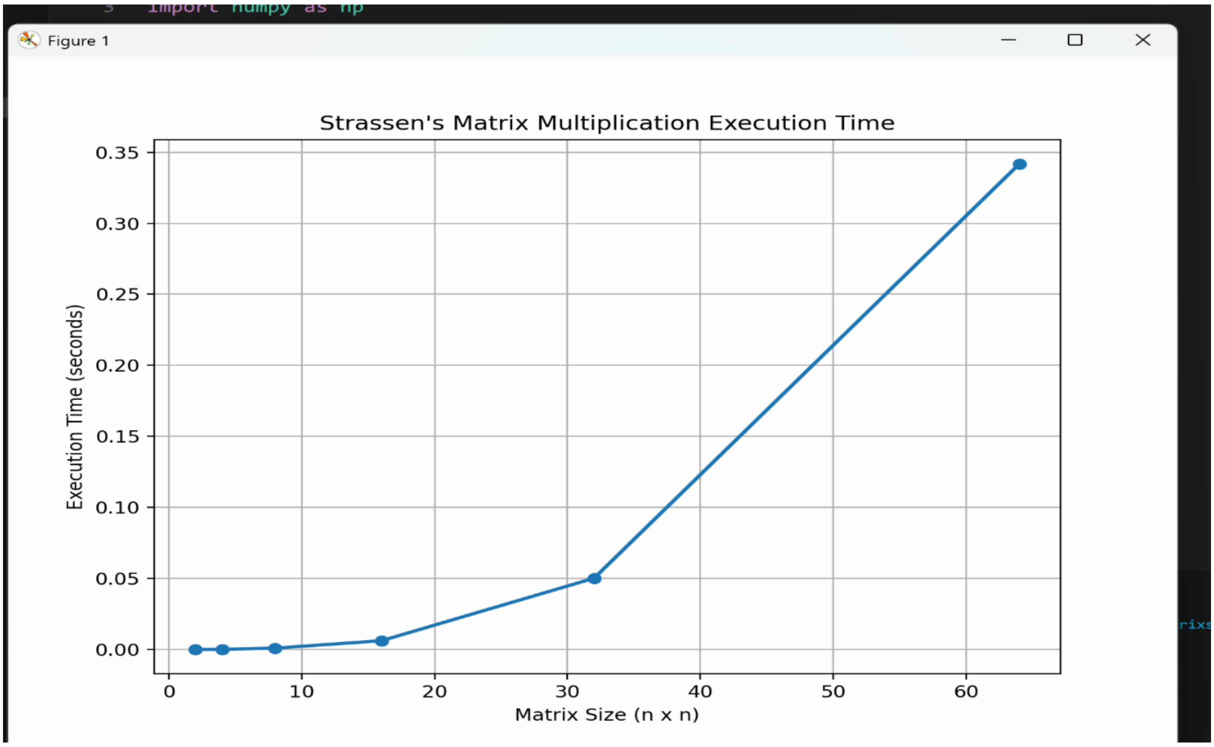


```python
import time
import matplotlib.pyplot as plt
import numpy as np

# Matrix addition
def add(A, B):
    return A + B

# Matrix subtraction
def subtract(A, B):
    return A - B

# Strassen’s Algorithm (recursive)
def strassen(A, B):
    n = A.shape[0]
    if n == 1:
        return A * B

    mid = n // 2

    A11, A12, A21, A22 = A[:mid, :mid], A[:mid, mid:], A[mid:, :mid], A[mid:, mid:]
    B11, B12, B21, B22 = B[:mid, :mid], B[:mid, mid:], B[mid:, :mid], B[mid:, mid:]

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

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

    # Combine results
```



```
def strassen(A, B):
        # Combine results
        top = np.hstack((C11, C12))
        bottom = np.hstack((C21, C22))
        return np.vstack((top, bottom))


# Test different matrix sizes
sizes = [2, 4, 8, 16, 32, 64]
times = []

for n in sizes:
    A = np.fromfunction(lambda i, j: i + j, (n, n), dtype=int)
    B = np.fromfunction(lambda i, j: i - j, (n, n), dtype=int)

    start = time.time()
    C = strassen(A, B)
    end = time.time()

    elapsed = end - start
    times.append(elapsed)

    print(f"Size {n}x{n} -> {elapsed:.6f} seconds")

# Plot graph
plt.figure(figsize=(8,6))
plt.plot(sizes, times, marker='o', linestyle='-', linewidth=2)
plt.title("Strassen's Matrix Multiplication Execution Time")
plt.xlabel("Matrix Size (n x n)")
plt.ylabel("Execution Time (seconds)")
plt.grid(True)
plt.show()
```

Graph:


import numpy as np
Figure 1
Strassen's Matrix Multiplication Execution Time
Execution Time (seconds)
Matrix Size (n x n)