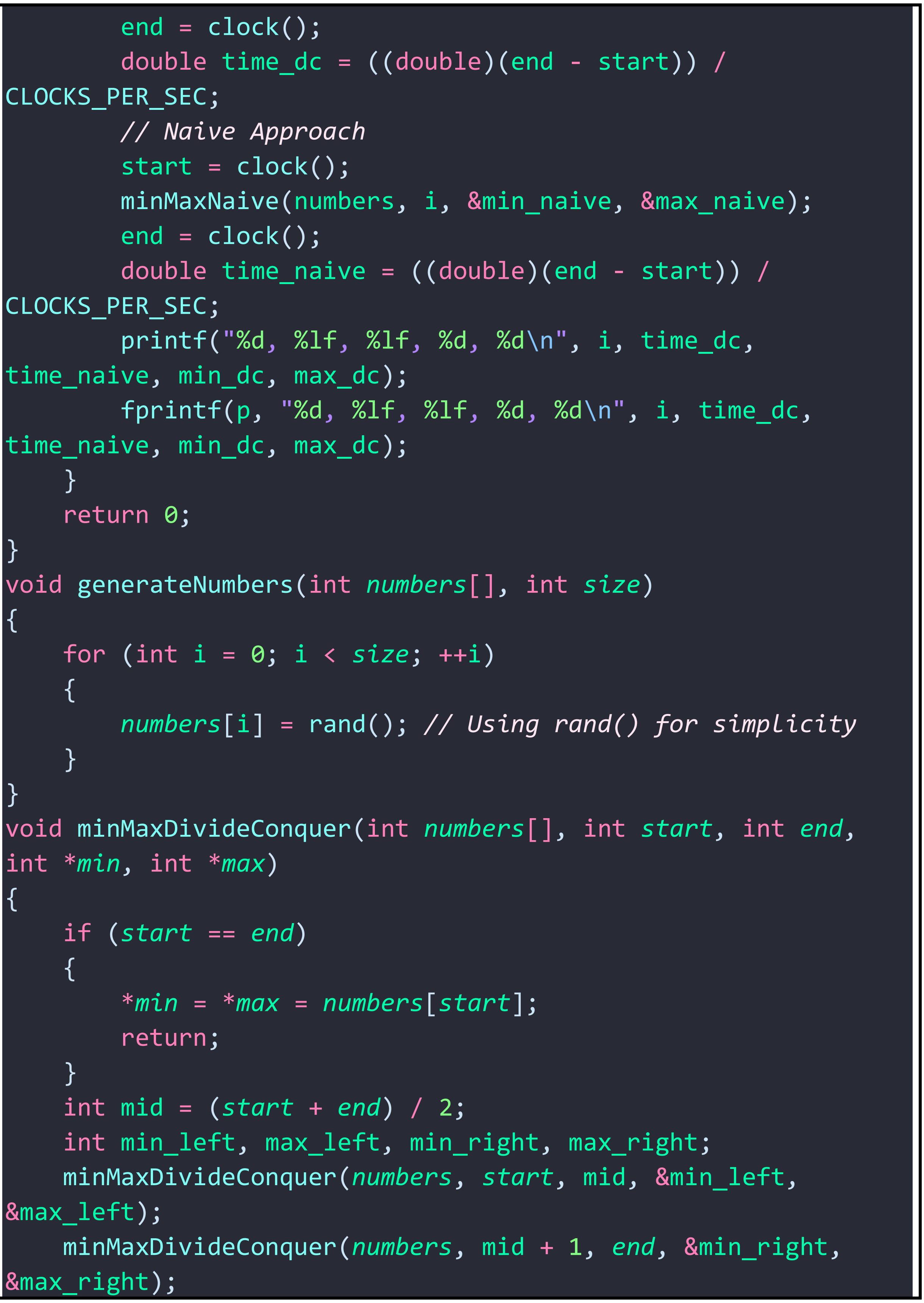
BHARATIYA VIDYA BHAVAN’S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

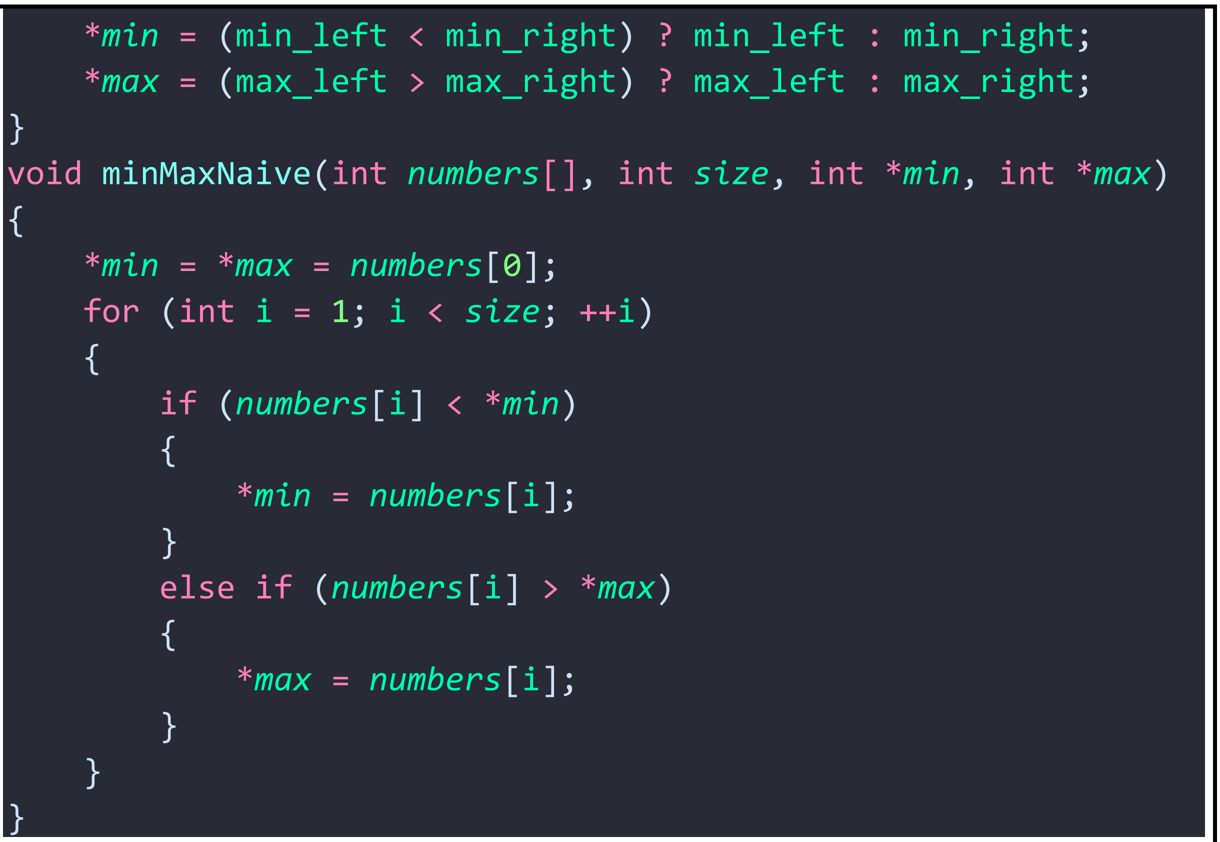
Name	Manish Shashikant Jadhav
UID	2023301005
Subject	Design and Analysis of Algorithms (DAA)
Experiment No.	3
Aim	Experiment based on divide and conquer (MIN-MAX and Strassen’s Multiplication.
Min-Max	

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ARRAY_SIZE 100000
// Function prototypes
void generateNumbers(int numbers[], int size);
void minMaxDivideConquer(int numbers[], int start, int end,
int *min, int *max);
void minMaxNaive(int numbers[], int size, int *min, int *max);
int main()
{
    FILE *p = fopen("minmax.csv", "w");
    fprintf(p, "Number, Time (Divide & Conquer), Time (Naive),
Min, Max\n");
    int numbers[ARRAY_SIZE];
    int min_dc, max_dc, min_naive, max_naive;
    // Generate 100,000 random integer numbers using rand()
    generateNumbers(numbers, ARRAY_SIZE);
    printf("Number, Time (Divide & Conquer), Time (Naive),
Min, Max\n");
    for (int i = 100; i <= ARRAY_SIZE; i += 100)
    {
        clock_t start, end;

        // Divide and Conquer
        start = clock();
        minMaxDivideConquer(numbers, 0, i - 1, &min_dc,
&max_dc);
```

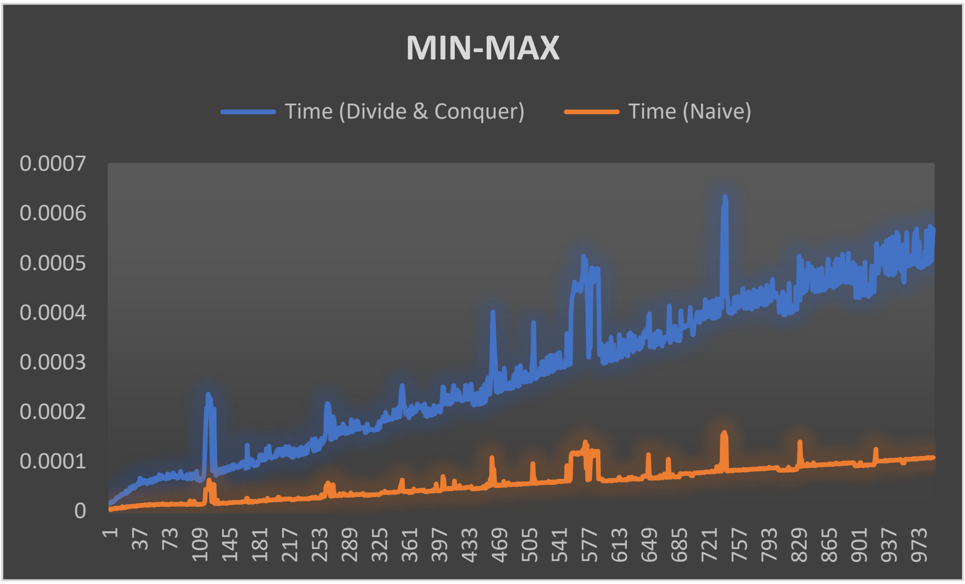


```
        end = clock();
        double time_dc = ((double)(end - start)) /
CLOCKS_PER_SEC;
        // Naive Approach
        start = clock();
        minMaxNaive(numbers, i, &min_naive, &max_naive);
        end = clock();
        double time_naive = ((double)(end - start)) /
CLOCKS_PER_SEC;
        printf("%d, %lf, %lf, %d, %d\n", i, time_dc,
time_naive, min_dc, max_dc);
        fprintf(p, "%d, %lf, %lf, %d, %d\n", i, time_dc,
time_naive, min_dc, max_dc);
    }
    return 0;
}
void generateNumbers(int numbers[], int size)
{
    for (int i = 0; i < size; ++i)
    {
        numbers[i] = rand(); // Using rand() for simplicity
    }
}
void minMaxDivideConquer(int numbers[], int start, int end,
int *min, int *max)
{
    if (start == end)
    {
        *min = *max = numbers[start];
        return;
    }
    int mid = (start + end) / 2;
    int min_left, max_left, min_right, max_right;
    minMaxDivideConquer(numbers, start, mid, &min_left,
&max_left);
    minMaxDivideConquer(numbers, mid + 1, end, &min_right,
&max_right);
```

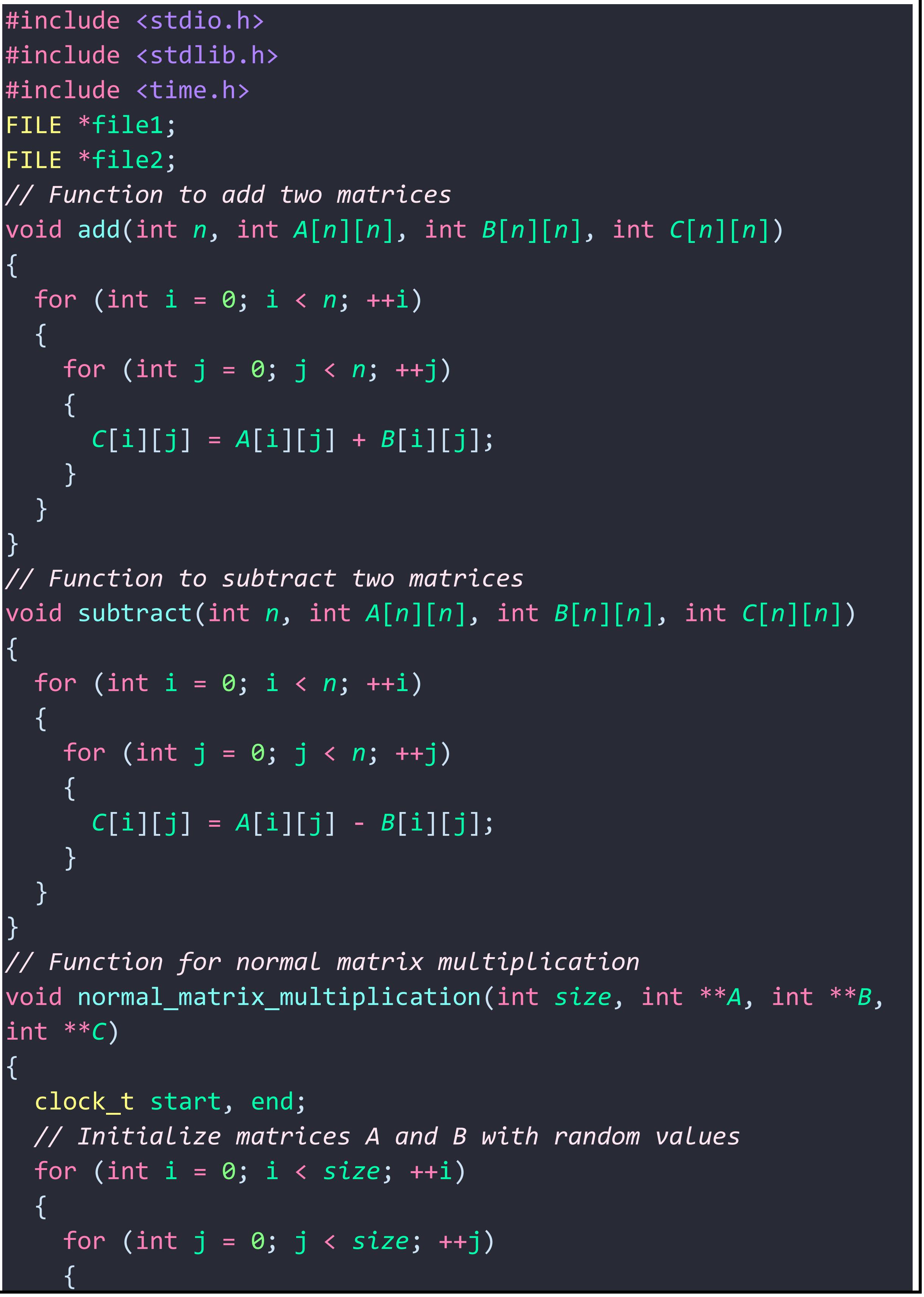


```
    *min = (min_left < min_right) ? min_left : min_right;
    *max = (max_left > max_right) ? max_left : max_right;
}
void minMaxNaive(int numbers[], int size, int *min, int *max)
{
    *min = *max = numbers[0];
    for (int i = 1; i < size; ++i)
    {
        if (numbers[i] < *min)
        {
            *min = numbers[i];
        }
        else if (numbers[i] > *max)
        {
            *max = numbers[i];
        }
    }
}
```

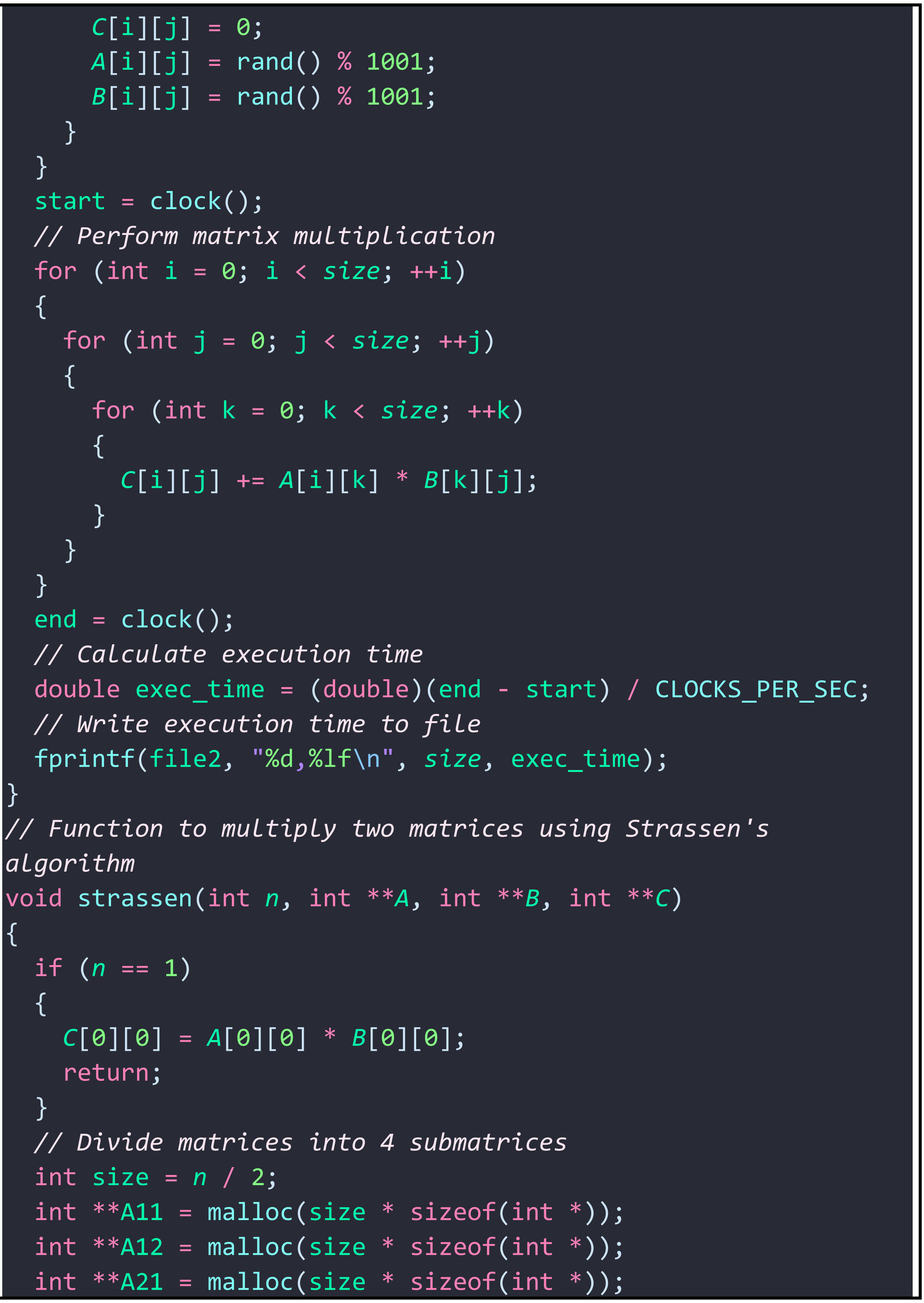
Graphs



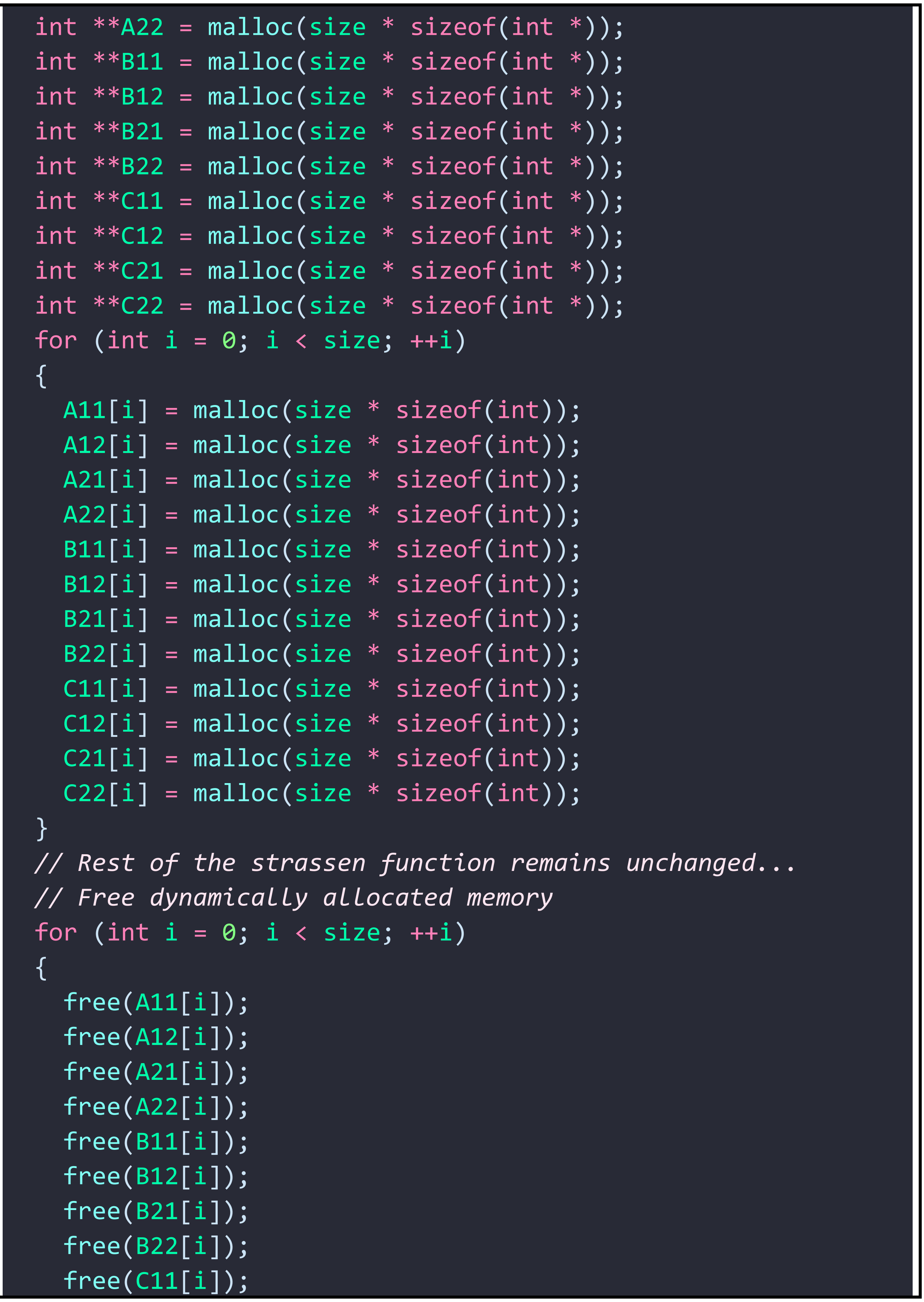
Strassens:



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
FILE *file1;
FILE *file2;
// Function to add two matrices
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];
    }
  }
}
// Function to subtract two matrices
void subtract(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];
    }
  }
}
// Function for normal matrix multiplication
void normal_matrix_multiplication(int size, int **A, int **B,
int **C)
{
  clock_t start, end;
  // Initialize matrices A and B with random values
  for (int i = 0; i < size; ++i)
  {
    for (int j = 0; j < size; ++j)
    {
```

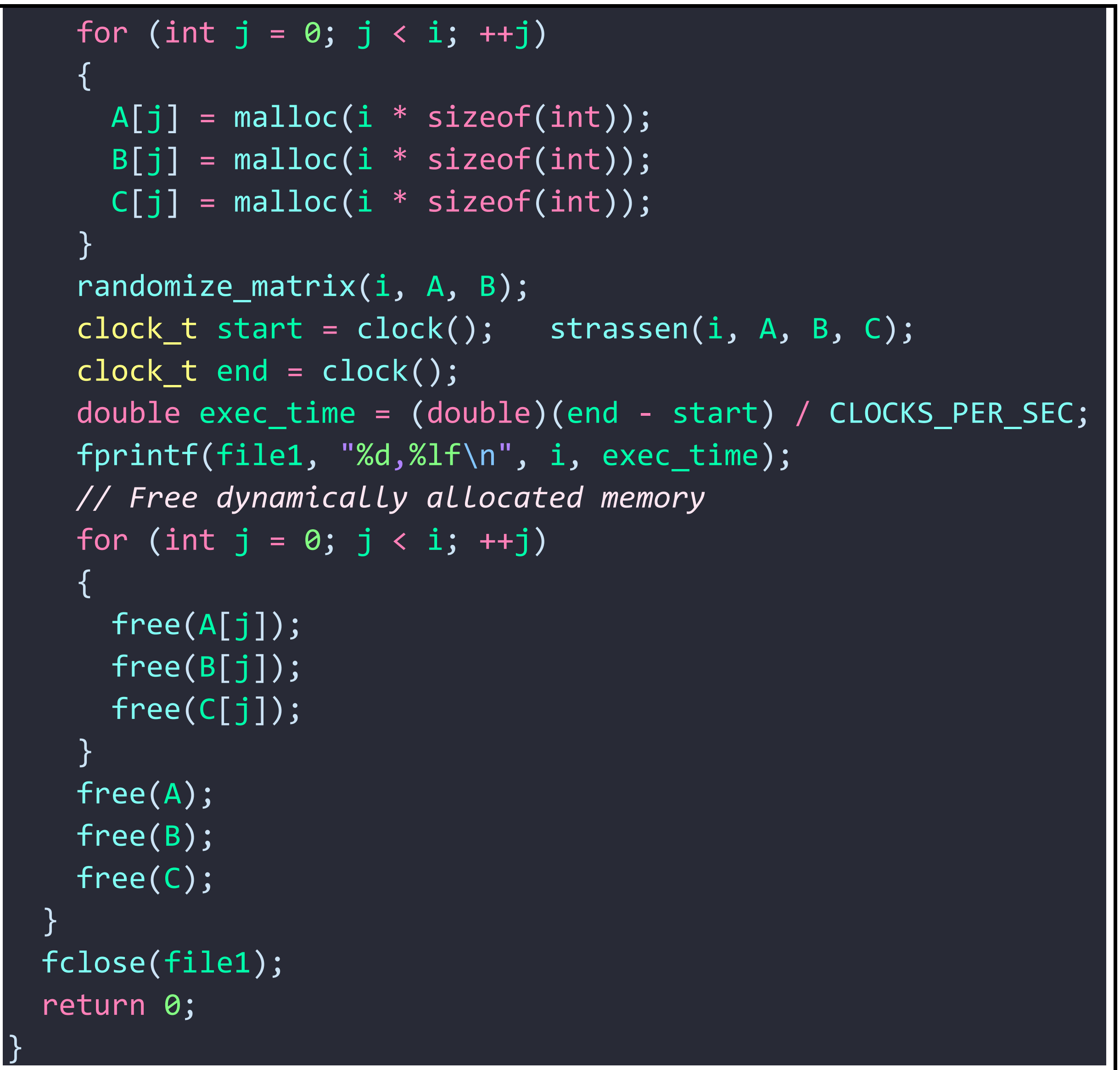


```
      C[i][j] = 0;
      A[i][j] = rand() % 1001;
      B[i][j] = rand() % 1001;
    }
  }
  start = clock();
  // Perform matrix multiplication
  for (int i = 0; i < size; ++i)
  {
    for (int j = 0; j < size; ++j)
    {
      for (int k = 0; k < size; ++k)
      {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
  end = clock();
  // Calculate execution time
  double exec_time = (double)(end - start) / CLOCKS_PER_SEC;
  // Write execution time to file
  fprintf(file2, "%d,%lf\n", size, exec_time);
}
// Function to multiply two matrices using Strassen's
algorithm
void strassen(int n, int **A, int **B, int **C)
{
  if (n == 1)
  {
    C[0][0] = A[0][0] * B[0][0];
    return;
  }
  // Divide matrices into 4 submatrices
  int size = n / 2;
  int **A11 = malloc(size * sizeof(int *));
  int **A12 = malloc(size * sizeof(int *));
  int **A21 = malloc(size * sizeof(int *));
```

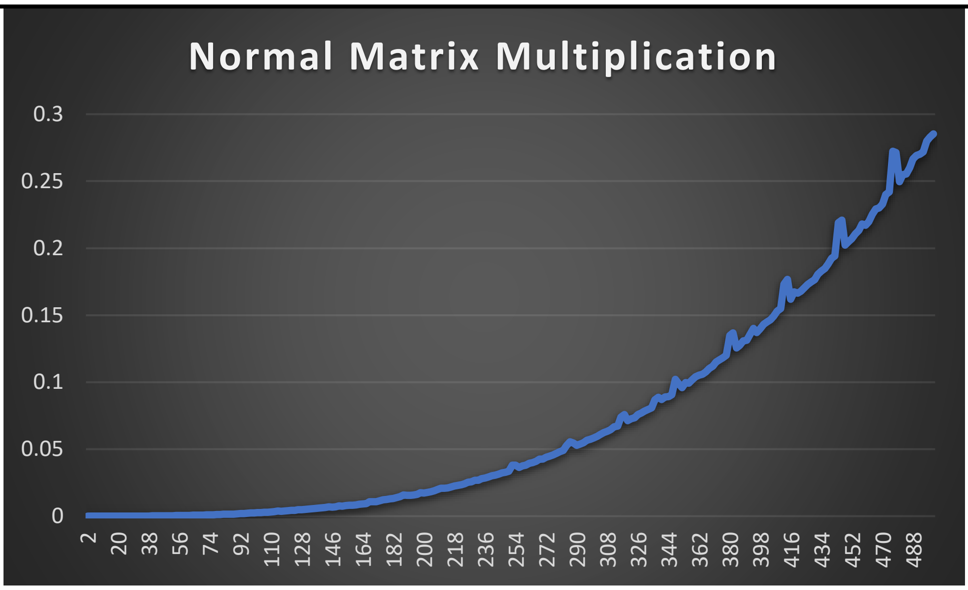


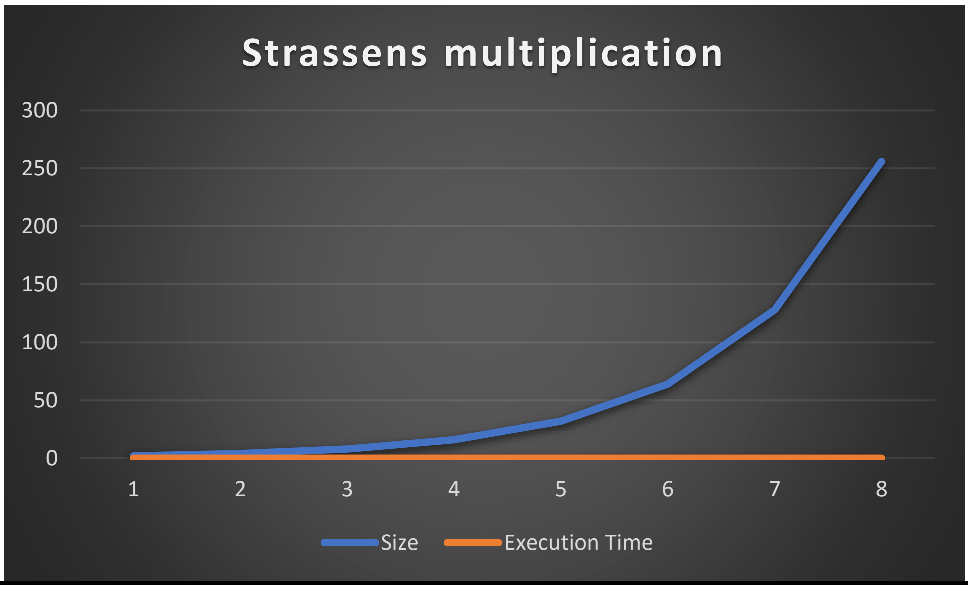
```
int **A22 = malloc(size * sizeof(int *));
int **B11 = malloc(size * sizeof(int *));
int **B12 = malloc(size * sizeof(int *));
int **B21 = malloc(size * sizeof(int *));
int **B22 = malloc(size * sizeof(int *));
int **C11 = malloc(size * sizeof(int *));
int **C12 = malloc(size * sizeof(int *));
int **C21 = malloc(size * sizeof(int *));
int **C22 = malloc(size * sizeof(int *));
for (int i = 0; i < size; ++i)
{
  A11[i] = malloc(size * sizeof(int));
  A12[i] = malloc(size * sizeof(int));
  A21[i] = malloc(size * sizeof(int));
  A22[i] = malloc(size * sizeof(int));
  B11[i] = malloc(size * sizeof(int));
  B12[i] = malloc(size * sizeof(int));
  B21[i] = malloc(size * sizeof(int));
  B22[i] = malloc(size * sizeof(int));
  C11[i] = malloc(size * sizeof(int));
  C12[i] = malloc(size * sizeof(int));
  C21[i] = malloc(size * sizeof(int));
  C22[i] = malloc(size * sizeof(int));
}
// Rest of the strassen function remains unchanged...
// Free dynamically allocated memory
for (int i = 0; i < size; ++i)
{
  free(A11[i]);
  free(A12[i]);
  free(A21[i]);
  free(A22[i]);
  free(B11[i]);
  free(B12[i]);
  free(B21[i]);
  free(B22[i]);
  free(C11[i]);
```

```c
    free(C12[i]);
    free(C21[i]);
    free(C22[i]);
  }
  free(A11);
  free(A12);
  free(A21);
  free(A22);
  free(B11);
  free(B12);
  free(B21);
  free(B22);
  free(C11);
  free(C12);
  free(C21);
  free(C22);
}
// Function to randomly initialize matrices A and B
void randomize_matrix(int n, int **A, int **B)
{
  for (int i = 0; i < n; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
      A[i][j] = rand() % 1025;
      B[i][j] = rand() % 1025;
    }
  }
}
int main()
{
  // Seed for random number generation
  srand(time(NULL));
  // File to store Normal Matrix Multiplication results
  file2 = fopen("Normal_Matrix_Multiplication_File.csv", "w");
  fprintf(file2, "Size,Execution Time\n");
  // Perform Normal Matrix Multiplication for various matrix
sizes
  for (int i = 2; i <= 500; i += 2)
  {
    int **A = malloc(i * sizeof(int *));
    int **B = malloc(i * sizeof(int *));
    int **C = malloc(i * sizeof(int *));
    for (int j = 0; j < i; ++j)
    {
      A[j] = malloc(i * sizeof(int));
      B[j] = malloc(i * sizeof(int));
      C[j] = malloc(i * sizeof(int));
    }
    normal_matrix_multiplication(i, A, B, C);
    // Free dynamically allocated memory
    for (int j = 0; j < i; ++j)
    {
      free(A[j]);
      free(B[j]);
      free(C[j]);
    }
    free(A);
    free(B);
    free(C);
  }
  fclose(file2);
  // File to store Strassen's Matrix Multiplication results
  file1 = fopen("Strassens_Matrix_Multiplication_File.csv",
"w");
  fprintf(file1, "Size,Execution Time\n");
  // Perform Strassen's Matrix Multiplication for various
matrix sizes
  for (int i = 2; i <= 256; i *= 2)
  {
    int **A = malloc(i * sizeof(int *));
    int **B = malloc(i * sizeof(int *));
    int **C = malloc(i * sizeof(int *));
```



```
    for (int j = 0; j < i; ++j)
    {
      A[j] = malloc(i * sizeof(int));
      B[j] = malloc(i * sizeof(int));
      C[j] = malloc(i * sizeof(int));
    }
    randomize_matrix(i, A, B);
    clock_t start = clock();   strassen(i, A, B, C);
    clock_t end = clock();
    double exec_time = (double)(end - start) / CLOCKS_PER_SEC;
    fprintf(file1, "%d,%lf\n", i, exec_time);
    // Free dynamically allocated memory
    for (int j = 0; j < i; ++j)
    {
      free(A[j]);
      free(B[j]);
      free(C[j]);
    }
    free(A);
    free(B);
    free(C);
  }
  fclose(file1);
  return 0;
}
```

<table>
<tr><td><b>Graphs</b></td><td>




</td></tr>
<tr><td><b>Pseudo Code & Example</b></td><td></td></tr>
<tr><td><b>Conclusion</b></td><td>Hence, by completing this experiment I came to know about divide and conquer approach.</td></tr>
</table>