K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A1	**Roll No.: 16010123012**
Experiment No.: 2	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Comparative Analysis of Strassen's matrix multiplication.

Objective: To learn the divide and conquer strategy of solving the problems of different types

---

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T. H. Cormen, C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Binary_search_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary_search_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**

Pre Lab/ Prior Concepts:

Data structures

Historical Profile:

Strassen's Algorithm is a groundbreaking algorithm in computer science and mathematics that introduced a faster method for matrix multiplication compared to the traditional method. It has a rich history, being one of the first major breakthroughs in computational complexity for matrix operations.Matrix multiplication is a fundamental operation in linear algebra with applications in computer graphics, scientific computing, machine learning, and more.Strassen's algorithm is an advanced technique for matrix multiplication, introduced by

Volker Strassen in 1969, which significantly improves the time complexity of traditional matrix multiplication algorithms.

Traditional Matrix Multiplication:

Complexity: $O(n3)$ for multiplying two n×n matrices using the standard algorithm. Strassen's Matrix Multiplication:Reduces the number of multiplications required in the divide-and-conquer approach from 8 to 7.Complexity: Approximately $O(n^{2.81})$.

New Concepts to be learned:

Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

Algorithm :

Input : Two n×n matrices A and B, where n is a power of 2 (if not, pad the matrices with zeros).

Step 1: Divide the Matrices

A=[A11 A12

A21 A22] ,

B=[B11 B12

B21 B22]

Step 2: Compute Seven Intermediate Products

Define seven products based on specific combinations of additions and subtractions of submatrices:

1. M1=(A11+A22)*(B11+B22)
2. M2=(A21+A22)*B11
3. M3=A11*(B12−B22)
4. M4=A22*(B21−B11)
5. M5=(A11+A12)*B22
6. M6=(A21−A11)*(B11+B12)
7. M7=(A12−A22)*(B21+B22)

Step 3: Combine Results. (Use the seven intermediate products to compute the resulting matrix C)

C= [C11 C21

C12 C22] where:

- C11=M1+M4-M5+M7
- C12=M3+M5
- C21=M2+M4
- C22=M1−M2+M3+M6

Code:

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;
void addMatrix(vector<vector<int>> &A, vector<vector<int>> &B,
vector<vector<int>> &C, int size)
{
  for (int i = 0; i < size; i++)
  {
    for (int j = 0; j < size; j++)
    {
      C[i][j] = A[i][j] + B[i][j];
    }
  }
}
void subtractMatrix(vector<vector<int>> &A, vector<vector<int>> &B,
vector<vector<int>> &C, int size)
{
  for (int i = 0; i < size; i++)
  {
    for (int j = 0; j < size; j++)
    {
      C[i][j] = A[i][j] - B[i][j];
    }
  }
}
void strassenMatrixMultiplication(vector<vector<int>> &A, vector<vector<int>>
&B, vector<vector<int>> &C, int size)
{
  if (size == 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]);
    C[0][0] = m1 + m4 - m5 + m7;
    C[0][1] = m3 + m5;
    C[1][0] = m2 + m4;
    C[1][1] = m1 - m2 + m3 + m6;
  }
  else
  {
```

```
    int newSize = size / 2;
    vector<int> inner(newSize);
    vector<vector<int>>
        A11(newSize, inner), A12(newSize, inner), A21(newSize, inner),
A22(newSize, inner),
        B11(newSize, inner), B12(newSize, inner), B21(newSize, inner),
B22(newSize, inner),
        C11(newSize, inner), C12(newSize, inner), C21(newSize, inner),
C22(newSize, inner),
        M1(newSize, inner), M2(newSize, inner), M3(newSize, inner),
M4(newSize, inner),
        M5(newSize, inner), M6(newSize, inner), M7(newSize, inner),
        AResult(newSize, inner), BResult(newSize, inner);
    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];
      }
    }
    addMatrix(A11, A22, AResult, newSize);
    addMatrix(B11, B22, BResult, newSize);
    strassenMatrixMultiplication(AResult, BResult, M1, newSize);
    addMatrix(A21, A22, AResult, newSize);
    strassenMatrixMultiplication(AResult, B11, M2, newSize);
    subtractMatrix(B12, B22, BResult, newSize);
    strassenMatrixMultiplication(A11, BResult, M3, newSize);
    subtractMatrix(B21, B11, BResult, newSize);
    strassenMatrixMultiplication(A22, BResult, M4, newSize);
    addMatrix(A11, A12, AResult, newSize);
    strassenMatrixMultiplication(AResult, B22, M5, newSize);
    subtractMatrix(A21, A11, AResult, newSize);
    addMatrix(B11, B12, BResult, newSize);
    strassenMatrixMultiplication(AResult, BResult, M6, newSize);
    subtractMatrix(A12, A22, AResult, newSize);
    addMatrix(B21, B22, BResult, newSize);
    strassenMatrixMultiplication(AResult, BResult, M7, newSize);
    addMatrix(M1, M4, AResult, newSize);
    subtractMatrix(AResult, M5, BResult, newSize);
```

```
    addMatrix(BResult, M7, C11, newSize);
    addMatrix(M3, M5, C12, newSize);
    addMatrix(M2, M4, C21, newSize);
    addMatrix(M1, M3, AResult, newSize);
    subtractMatrix(AResult, M2, BResult, newSize);
    addMatrix(BResult, M6, C22, newSize);
    for (int i = 0; i < newSize; i++)
    {
      for (int j = 0; j < newSize; j++)
      {
        C[i][j] = C11[i][j];
        C[i][j + newSize] = C12[i][j];
        C[i + newSize][j] = C21[i][j];
        C[i + newSize][j + newSize] = C22[i][j];
      }
    }
  }
}
int main()
{
  int n;
  cout << "Enter the size of the matrices (must be a power of 2): ";
  cin >> n;
  vector<vector<int>> A(n, vector<int>(n));
  vector<vector<int>> B(n, vector<int>(n));
  vector<vector<int>> C(n, vector<int>(n, 0));
  cout << "Enter the elements of the first matrix: ";
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> A[i][j];
    }
  }
  cout << "Enter the elements of the second matrix: ";
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      cin >> B[i][j];
    }
  }
  strassenMatrixMultiplication(A, B, C, n);
  cout << "Resultant matrix: " << endl;
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
```

```
    {
      cout << C[i][j] << " ";
    }
    cout << endl;
  }
}
```

Output:

```
Enter the size of the matrices (must be a power of 2): 4
Enter the elements of the first matrix: 1 3 1 3 2 4 2 4 1 3 1 3 2 4 2 4
Enter the elements of the second matrix: 1 3 1 3 2 4 2 4 1 3 1 3 2 4 2 4
Resultant matrix:
14 30 14 30
20 44 20 44
14 30 14 30
20 44 20 44
```

The space complexity: $O(n^2)$

The Time complexity: $O(n^{\log 7})$

Lab Work:

Q. $A = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 4 \\ 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 4 \end{bmatrix}$ $\quad B = \begin{bmatrix} 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 4 \\ 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 4 \end{bmatrix}$

$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ $\quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$A_{11} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = A_{12} = A_{21} = A_{22} = B_{11} = B_{12} = B_{21} = B_{22}$

$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$

$\hookrightarrow \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 4 & 8 \end{bmatrix}$

$\therefore M_1 = \begin{bmatrix} 2 & 6 \\ 4 & 8 \end{bmatrix} \cdot \begin{bmatrix} 2 & 6 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 28 & 60 \\ 40 & 88 \end{bmatrix}$

$M_2 = (A_{21} + A_{22}) \times B_{11}$

$= \begin{bmatrix} 2 & 6 \\ 4 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

$= \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$

$M_3 = A_{11} \times (B_{12} - B_{22})$

$= 0 \quad [\because B_{12} - B_{22} = 0]$

$M_4 = A_{22} \times (B_{21} - B_{11})$

$= 0 \quad [\because B_{21} = B_{11} = 0]$

$M_5 = (A_{11} + A_{12}) \cdot B_{22}$

$= \begin{bmatrix} 2 & 6 \\ 4 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

$2 \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$

$$M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$
$$= 0$$

$$M_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$
$$= 0$$

$$C = \begin{bmatrix} C_{11} & C_{21} \\ C_{12} & C_{22} \end{bmatrix}$$

where,

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$= \begin{bmatrix} 28 & 60 \\ 40 & 88 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$$

$$C_{12} = M_3 + M_5$$
$$= 0 + \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix} = \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$$

$$C_{21} = M_2 + M_4$$
$$= \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix} + 0 = \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$$

$$C_{22} = M_1 - M_2 + M_3 + M_6$$
$$= \begin{bmatrix} 28 & 60 \\ 40 & 88 \end{bmatrix} - \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix} + [0] + [0] = \begin{bmatrix} 14 & 30 \\ 20 & 44 \end{bmatrix}$$

$$\therefore C = \begin{bmatrix} 14 & 30 & 14 & 30 \\ 20 & 44 & 20 & 44 \\ 14 & 30 & 14 & 30 \\ 20 & 44 & 20 & 44 \end{bmatrix}$$

Space : $O(n^2)$

Time : $O(n^{\log 7})$
$\approx O(n^{2.807})$

CONCLUSION:

I implemented Strassen's Matrix Multiplication using the Divide and Conquer approach. By applying Strassen's method to 4×4 matrices, I saw a significant performance improvement, reducing time complexity from $O(n^3)$ to $O(n^{2.81})$. While the method adds some overhead for additions and subtractions, it remains efficient for larger matrices. The resulting matrix confirmed the algorithm's correctness.