Name	Naman Badlani
UID No.	2021300008
Subject	Design And Analysis Of Algorithm
Class	Comps A
Experiment No.	4
AIM	Experiment to perform Strassens Multiplication for a 2 x 2 Matrix

Theory –

Strassen's multiplication is a fast algorithm for multiplying large matrices. The algorithm uses a divide-and-conquer approach to reduce the number of multiplications required to compute the product of two matrices.

The key insight of Strassen's algorithm is that it is possible to compute the product of two matrices using only seven multiplications instead of the usual eight. This is achieved by cleverly combining the intermediate matrix operations in a way that eliminates one of the multiplication steps. The resulting algorithm has a lower computational complexity than the standard algorithm for matrix multiplication, making it faster for large matrices.

Algorithm –

1. Divide both input matrices A and B into four equal-sized submatrices: A11, A12, A21, A22 and B11, B12, B21, B22.

2. Compute seven intermediate matrices using the submatrices:
   M1 = (A11 + A22) x (B11 + B22)
   M2 = (A21 + A22) x B11
   M3 = A11 x (B12 - B22)
   M4 = A22 x (B21 - B11)
   M5 = (A11 + A12) x B22

M6 = (A21 - A11) x (B11 + B12)
M7 = (A12 - A22) x (B21 + B22)

3. Compute the output submatrices using the intermediate matrices:
   C11 = M1 + M4 - M5 + M7
   C12 = M3 + M5
   C21 = M2 + M4
   C22 = M1 - M2 + M3 + M6

4. Combine the output submatrices to form the final product matrix C.

Program –

```c
#include<stdio.h>
int main(){
    int a[2][2], b[2][2], c[2][2], i, j;
    int m1, m2, m3, m4 , m5, m6, m7;

    printf("Enter the 4 elements of first matrix: ");
    for(i = 0;i < 2; i++){
        for(j = 0;j < 2; j++){
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter the 4 elements of second matrix: ");
    for(i = 0; i < 2; i++){
        for(j = 0;j < 2; j++){
            scanf("%d", &b[i][j]);
        }
    }

    printf("\nThe first matrix is\n");
    for(i = 0; i < 2; i++){
        printf("\n");
        for(j = 0; j < 2; j++){
            printf("%d\t", a[i][j]);
        }
    }
```

```
    printf("\nThe second matrix is\n");
    for(i = 0;i < 2; i++){
        printf("\n");
        for(j = 0;j < 2; j++){
            printf("%d\t", b[i][j]);
        }
    }

    m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
    m2= (a[1][0] + a[1][1]) * b[0][0];
    m3= a[0][0] * (b[0][1] - b[1][1]);
    m4= a[1][1] * (b[1][0] - b[0][0]);
    m5= (a[0][0] + a[0][1]) * b[1][1];
    m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
    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;

    printf("\nAfter multiplication using Strassen's algorithm \n");
    for(i = 0; i < 2 ; i++){
        printf("\n");
        for(j = 0;j < 2; j++){
            printf("%d\t", c[i][j]);
        }
    }

    return 0;
    }
```

Result Analysis –

```
PS C:\Users\ashok\Desktop\Sem IV> cd "c:\Users\ash

Enter the 4 elements of first matrix: 1 2 3 4
Enter the 4 elements of second matrix: 5 6 7 20

The first matrix is

1       2
3       4
The second matrix is

5       6
7       20
After multiplication using Strassen's algorithm

19      46
43      98
PS C:\Users\ashok\Desktop\Sem IV> 
```

- As the algorithm suggested, the time complexity of this method is – $T(n) = 7T(n/2) + O(n^2)$ which evaluates to $O(n^{2.807})$ approximately. This is less than the conventional method which had a time complexity = $O(n^3)$.
- Hence, this method is comparatively faster and the difference is very much visible when the matrix order is very high.