| Name | Ronak Thakar |
|----------------|--------------|
| UID no. | 2021700066 |
| Experiment No. | 3 |

| AIM: | Implement Divide and Conquer technique. |
|-----------------------|--|
| Program | |
| PROBLEM STATEMENT: | Implement Strassen's Matrix Multiplication algorithm and compare it with standard matrix multiplication. |
| ALGORITHM/ THEORY: | Let us assume two matrices X and Y. We want to calculate the resultant matrix Z by multiplying X and Y. Naïve Method: |
| | First, we will discuss naïve method and its complexity. Here, we are calculating $Z = X \times Y$. Using Naïve method, two matrices (X and Y) can be multiplied if the order of these matrices are $p \times q$ and $q \times r$. Following is the algorithm. |
| | For $I \leftarrow 1$ to p do For $j \leftarrow 1$ to r do $Z[i,j] \leftarrow 0$ For $k = 1$ to q do $Z[i,j] \leftarrow Z[i,j] + X[i,k] * Y[k,j]$ |
| | There are three for loops in this algorithm and one is nested in other. Hence, the algorithm takes $O(n^3)$ time to execute. Strassen's Matrix Multiplication Algorithm: |
| | Strassen's Matrix multiplication can be performed only on square matrices where n is a power of 2. Order of both of the matrices are n \times n. Divide X, Y into four (n/2) \times (n/2) matrices and Using Strassen's Algorithm compute the following – |
| | m1 = (a[0][0] + a[1][1])*(b[0][0] + b[1][1]); $m2 = (a[1][0] + a[1][1])*b[0][0];$ $m3 = a[0][0]*(b[0][1] - b[1][1]);$ $m4 = a[1][1]*(-b[0][0] + b[1][0]);$ $m5 = (a[0][0] + a[0][1])*b[1][1];$ $m6 = (-a[0][0] + a[1][0])*(b[0][0] + b[0][1]);$ $m7 = (a[0][1] - a[1][1])*(b[1][0] + b[1][1]);$ Then, $c[0][0] = m1 + m4 - m5 + m7;$ |

```
c[1][0] = m2 + m4; \\ c[0][1] = m3 + m5; \\ c[1][1] = m1 + m3 - m2 + m6; \\ Using the recurrence relation, we get <math>T(n) = O(n^{\log 7}) And T(n) = 7T(n/2) + n^2 Hence, the complexity of Strassen's matrix multiplication algorithm is O(n^{\log 7}). In the naïve method we use to do 8 multiplications using Strassen's we have reduced that to 7 multiplications.
```

```
#include<stdio.h>
#define row 2
#define col 2
void mat_in(int a[row][col]){
   for(int i=0;i < row;i++){
     for(int j=0;j<col;j++){
      printf("\nEnter elemnet %d%d: ",i,j);
      scanf("%d",&a[i][j]);
void mat print(int a[row][col]){
  printf("\n|\t\t|\n");
   for(int i=0;i< row;i++)
    printf("|");
    printf("\t");
    for(int j=0;j<col;j++){}
       printf("%d\t",a[i][j]);
    printf("|");
    printf("\n|\t\t\t\n");
int main(){
  int a[row][col],b[row][col],m1,m2,m3,m4,m5,m6,m7,c[row][col];
  printf("\nEnter matrix A values: ");
  mat in(a);
  printf("\nEnter matrix B values: ");
  mat in(b);
  //Strassen's matrix multiplication;
  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[0][0] + b[1][0]);
  m5 = (a[0][0] + a[0][1])*b[1][1];
```

```
m6 = (-a[0][0] + a[1][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[1][0] = m2 + m4;

c[0][1] = m3 + m5;

c[1][1] = m1 + m3 - m2 + m6;

printf("\n\nMatrix A::\n");

mat_print(a);

printf("\n\nMatrix B::\n");

mat_print(b);

printf("\n\nThe answer is:\n");

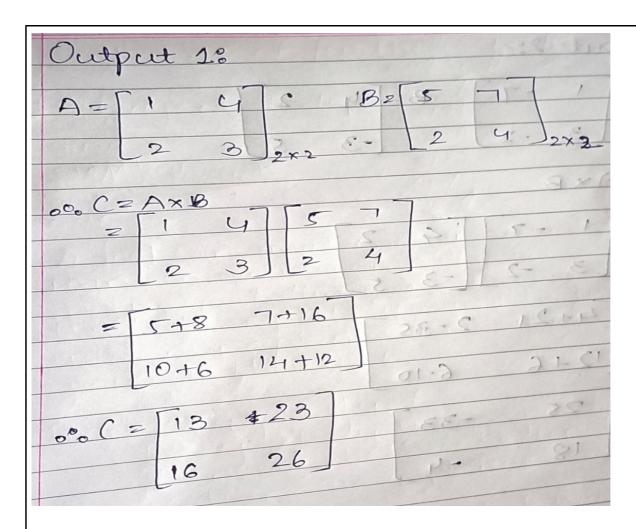
mat_print(c);

}
```

Result:

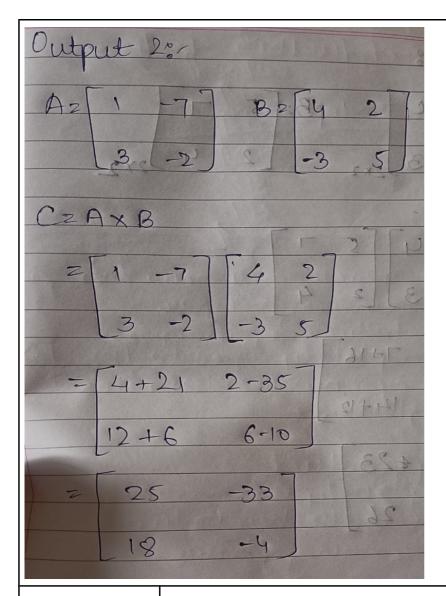
OUTPUT 1:

Verification of algorithm:



OUTPUT2:

Verification of algorithm:



CONCLUSION

We used Strassen's Matrix Multiplication but without recursive calls and then we have compared the logic with the Standard Matrix Multiplication logic. On comparison we found out that Strassen's is better than standard method for multiplication of square matrices.