Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058, India
(Autonomous College Affiliated to University of Mumbai)

Experiment No.	5
Aim	To implement Strassen's Multiplication Matrix
Name	Harsh Mukesh Jain
UID No.	2021300048
Class & Division	SE Comps A (C – Batch)
Date of Performance	27-02-2023
Date of Submission	6-03-2023

Theory/Experiment:

In linear algebra, the Strassen algorithm, named after Volker Strassen, is an algorithm for matrix multiplication. It is faster than the standard matrix multiplication algorithm for large matrices, with a better asymptotic complexity, although the naive algorithm is often better for smaller matrices.

Naïve Method :

First, we will discuss naïve method and its complexity. Here, we are calculating Z = X × Y. Using Naïve method, two matrices (X and Y) can be multiplied if the order of these matrices are p × q and q × r. Following is the algorithm.

Algorithm: Matrix-Multiplication (X, Y, Z)

```
for i = 1 to p do
  for j = 1 to r do
    Z[i,j] := 0
    for k = 1 to q do
      Z[i,j] := Z[i,j] + X[i,k] × Y[k,j]
```

Complexity

Here, we assume that integer operations take O(1) time. There are three for loops in this algorithm and one is nested in other. Hence, the algorithm takes O(n3) time to execute.

Strassen's Matrix Multiplication Algorithm :

In this context, using Strassen's Matrix multiplication algorithm, the time consumption can be improved a little bit.

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 × n.

Divide X, Y and Z into four (n/2)×(n/2) matrices as represented below –

$$Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix} \qquad X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Using Strassen's Algorithm compute the following −

M1 := (A + C) × (E + F)
M2 := (B + D) × (G + H)
M3 := (A − D) × (E + H)
M4 := A × (F − H)
M5 := (C + D) × (E)
M6 := (A + B) × (H)
M7 := D × (G − E)

Then,
I:=M2+M3−M6−M7
J:=M4+M6
K:=M5+M7
L:=M1−M3−M4−M5

Analysis
T(n) = c
= 7 x T(n/2) + d x n^2 if n = 1otherwise
where c and d are constants

Using this recurrence relation, we get T(n) = O(n^{log7})
Hence, the complexity of Strassen's matrix multiplication algorithm is O(n^{log7})

CODE :

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

int ** createZeroMatrix(int n){
	int ** array = (int**)malloc(n*sizeof(int *));
	int i,j;
	for(i = 0;i < n; i++) {
		array[i] = (int*)malloc(n*sizeof(int));
		for(j = 0; j < n; j++) {
		array[i][j] = 0;
		}
	}
	return array;
}

int ** create(int n,int pow){
	int ** array = createZeroMatrix(pow);
	int i,j,no;
	for(i = 0;i < n; i++) {
		for(j = 0; j < n; j++) {
			printf("Enter an element at (%d, %d) position:",i+1,j+1);
			scanf("%d", &no);
			array[i][j] = no;
			}
	}
	return array;
}

void printMatrix(int ** matrix,int n) {
	int i,j;
	for(i=0;i<n;i++){
		for(j=0;j<n;j++){
			printf("%d\t\t",matrix[i][j]);
```

```
            }
            printf("\n");
        }
}


void compose(int** matrix,int** result,int row,int col,int n){
        int i,j,r=row,c=col;
        for(i=0;i<n;i++){
                c=col;
                for(j=0;j<n;j++){
                        result[r][c]=matrix[i][j];
                        c++;
                }
                r++;
        }
}


int** divide(int ** matrix,int n, int row,int col) {
        int n_new=n/2;

        int ** array = createZeroMatrix(n_new);
        int i,j,r=row,c=col;
        for(i = 0;i < n_new; i++) {
                c=col;
                for(j = 0; j < n_new; j++) {
                array[i][j] = matrix[r][c];
                        c++;
                }
                r++;
        }
        return array;
}
```

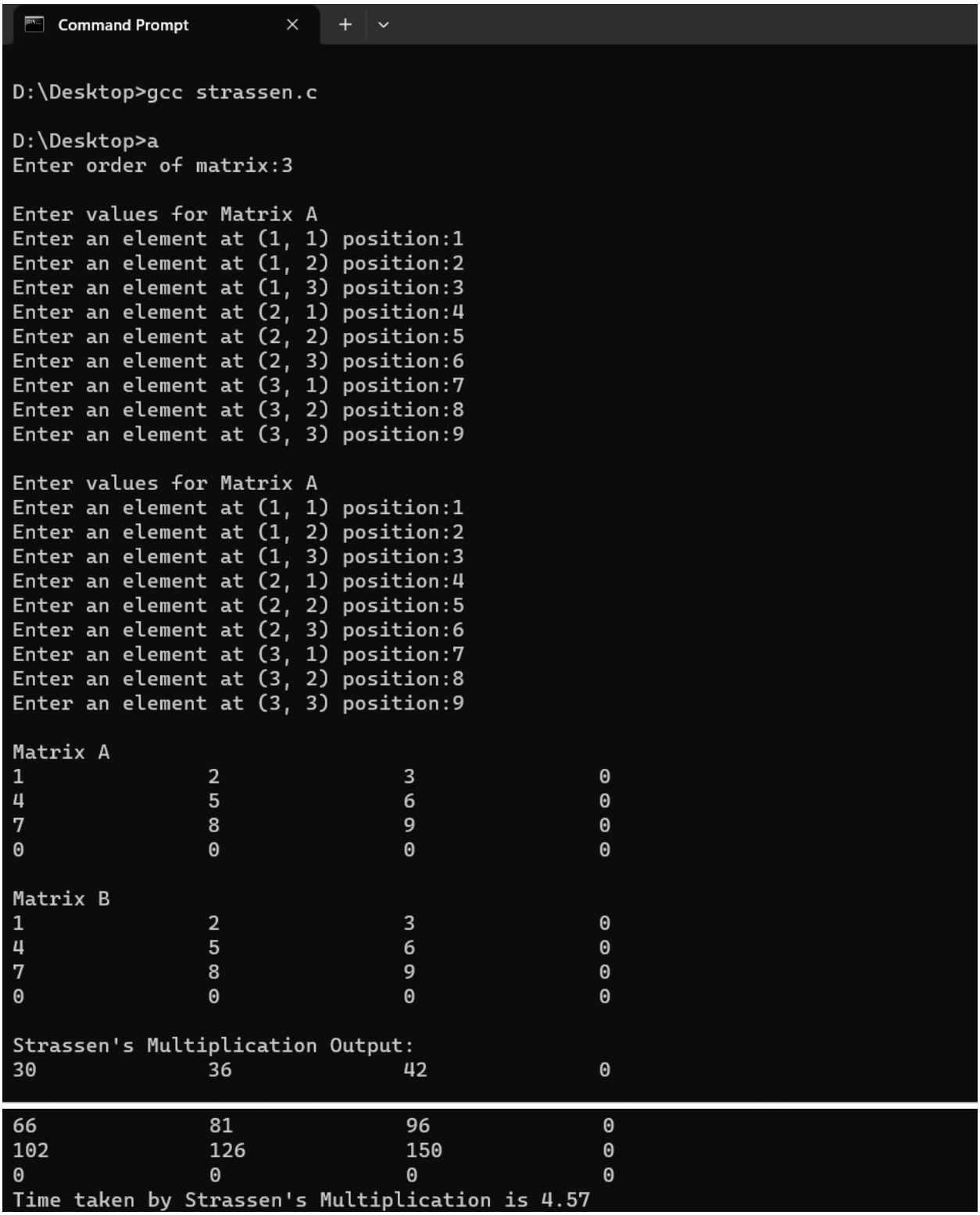
```
int** addMatrix(int** matrixA,int** matrixB,int n){
	int ** res = createZeroMatrix(n);
	int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			res[i][j]=matrixA[i][j]+matrixB[i][j];

	return res;
}

int** subMatrix(int** matrixA,int** matrixB,int n){
	int ** res = createZeroMatrix(n);
	int i,j;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			res[i][j]=matrixA[i][j]-matrixB[i][j];

	return res;
}


int** strassensMultRec(int ** matrixA, int** matrixB,int n){
	int ** result = createZeroMatrix(n);
	if(n>1) {
		//Divide the matrix
		int ** a11 = divide(matrixA, n, 0, 0);
		int ** a12 = divide(matrixA, n, 0, (n/2));
		int ** a21 = divide(matrixA, n, (n/2), 0);
		int ** a22 = divide(matrixA, n, (n/2), (n/2));
		int ** b11 = divide(matrixB, n, 0, 0);
		int ** b12 = divide(matrixB, n, 0, n/2);
		int ** b21 = divide(matrixB, n, n/2, 0);
		int ** b22 = divide(matrixB, n, n/2, n/2);
```

```
		//Recursive call for Divide and Conquer
		int** m1=
strassensMultRec(addMatrix(a11,a22,n/2),addMatrix(b11,b22,n/2),n/2);
		int** m2= strassensMultRec(addMatrix(a21,a22,n/2),b11,n/2);
		int** m3= strassensMultRec(a11,subMatrix(b12,b22,n/2),n/2);
		int** m4= strassensMultRec(a22,subMatrix(b21,b11,n/2),n/2);
		int** m5= strassensMultRec(addMatrix(a11,a12,n/2),b22,n/2);
		int** m6=
strassensMultRec(subMatrix(a21,a11,n/2),addMatrix(b11,b12,n/2),n/2);
		int** m7=
strassensMultRec(subMatrix(a12,a22,n/2),addMatrix(b21,b22,n/2),n/2);

		int** c11 =
addMatrix(subMatrix(addMatrix(m1,m4,n/2),m5,n/2),m7,n/2);
		int** c12 = addMatrix(m3,m5,n/2);
		int** c21 = addMatrix(m2,m4,n/2);
		int** c22 =
addMatrix(subMatrix(addMatrix(m1,m3,n/2),m2,n/2),m6,n/2);
		//Compose the matrix
		compose(c11,result,0,0,n/2);
		compose(c12,result,0,n/2,n/2);
		compose(c21,result,n/2,0,n/2);
		compose(c22,result,n/2,n/2,n/2);
	}
	else {
		//This is the terminating condition for recurssion.
		result[0][0]=matrixA[0][0]*matrixB[0][0];
	}
	return result;
}



int ** strassensMultiplication(int ** matrixA, int** matrixB,int n){
	int ** result = strassensMultRec(matrixA,matrixB,n);
```

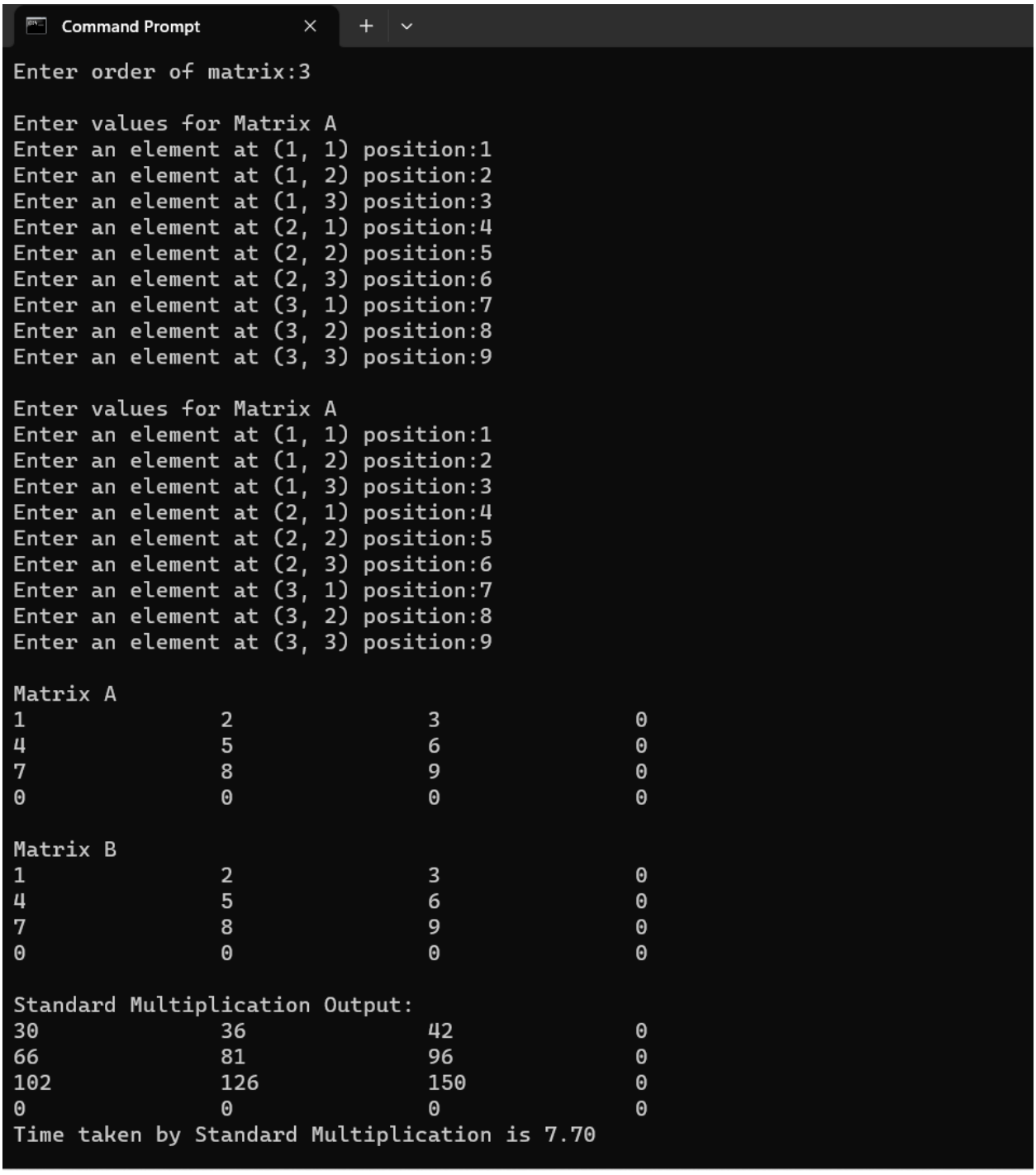
```
	return result;
}



int main() {
	int i = 0, j = 0,n = 0;
	printf("Enter order of matrix:");
	scanf("%d",&n);
	//To handle when n is not power of k we do the padding with zero
	int pow = 1;
	while(pow<n){
		pow=pow*2;
	}
	n--;
	printf("\nEnter values for Matrix A\n");
	int ** matrixA = create(n,pow);
	printf("\nEnter values for Matrix A\n");
	int  ** matrixB = create(n,pow);
	n = pow;
	int ** standardRes,**strassenRes;

	printf("\nMatrix A\n");
	printMatrix(matrixA,n);

	printf("\nMatrix B\n");
	printMatrix(matrixB,n);

	clock_t start, end;
	start = clock();
	printf("\nStrassen's Multiplication Output:\n");
	int ** strassensRes = strassensMultiplication(matrixA,matrixB,n);
	printMatrix(strassensRes,n);
	end = clock();
	printf("Time taken by Strassen's Multiplication is %.2f", ((double)(end -
```

```
start)/CLOCKS_PER_SEC));

        return 0;
}
```

OUTPUT :



```
Command Prompt

D:\Desktop>gcc strassen.c

D:\Desktop>a
Enter order of matrix:3

Enter values for Matrix A
Enter an element at (1, 1) position:1
Enter an element at (1, 2) position:2
Enter an element at (1, 3) position:3
Enter an element at (2, 1) position:4
Enter an element at (2, 2) position:5
Enter an element at (2, 3) position:6
Enter an element at (3, 1) position:7
Enter an element at (3, 2) position:8
Enter an element at (3, 3) position:9

Enter values for Matrix A
Enter an element at (1, 1) position:1
Enter an element at (1, 2) position:2
Enter an element at (1, 3) position:3
Enter an element at (2, 1) position:4
Enter an element at (2, 2) position:5
Enter an element at (2, 3) position:6
Enter an element at (3, 1) position:7
Enter an element at (3, 2) position:8
Enter an element at (3, 3) position:9

Matrix A
1               2               3               0
4               5               6               0
7               8               9               0
0               0               0               0

Matrix B
1               2               3               0
4               5               6               0
7               8               9               0
0               0               0               0

Strassen's Multiplication Output:
30              36              42              0
66              81              96              0
102             126             150             0
0               0               0               0
Time taken by Strassen's Multiplication is 4.57
```


Command Prompt
Enter order of matrix:3
Enter values for Matrix A
Enter an element at (1, 1) position:1
Enter an element at (1, 2) position:2
Enter an element at (1, 3) position:3
Enter an element at (2, 1) position:4
Enter an element at (2, 2) position:5
Enter an element at (2, 3) position:6
Enter an element at (3, 1) position:7
Enter an element at (3, 2) position:8
Enter an element at (3, 3) position:9
Enter values for Matrix A
Enter an element at (1, 1) position:1
Enter an element at (1, 2) position:2
Enter an element at (1, 3) position:3
Enter an element at (2, 1) position:4
Enter an element at (2, 2) position:5
Enter an element at (2, 3) position:6
Enter an element at (3, 1) position:7
Enter an element at (3, 2) position:8
Enter an element at (3, 3) position:9
Matrix A
1 2 3 0
4 5 6 0
7 8 9 0
0 0 0 0
Matrix B
1 2 3 0
4 5 6 0
7 8 9 0
0 0 0 0
Standard Multiplication Output:
30 36 42 0
66 81 96 0
102 126 150 0
0 0 0 0
Time taken by Standard Multiplication is 7.70

CONCLUSION:-

By performing this experiment, I was able to write recursive code for Strassen’s Matrix Multiplication. In conclusion, Strassen's algorithm is a clever approach to matrix multiplication that reduces the number of operations required to compute the product of two matrices. By dividing the matrices into submatrices and using recursive calls, Strassen's algorithm achieves a lower computational complexity than the traditional algorithm.