Name	Mohammed Muzammil Ansari
UID no.	2022701001
Experiment No.	3

AIM:	Experiment on implementing strassen's multiplication.
Program 1	
Algorithm:	STRASSENS-MULTIPLICATION (A, B): 1. n = A.rows 2. Let C be a new n X n matrix 3. if n == 2: a. P1 A11 x (B12 – B22) b. P2 (A11 + A12) x B22 c. P3 (A21 + A22) x B21 d. P4 A22 x (B21 – B11) e. P5 (A11 + A22) x (B11 + B22) f. P6 (A12 – A22) x (B21 + B22) g. P7 (A11 – A21) x (B11 + B12) h. C11 P5 + P4 – P2 + P6 i. C12 P1 + P2 j. C21 P3 + P4 k. C22 P5 + P1 – P3 – P7 l. return C 4. Divide input matrices A and B and output matrix C into 4 submatrices of size n/2 X n/2 each as follows: 5. P1 STRASSENS-MULTIPLICATION(A11, (B12 – B22)) 6. P2 STRASSENS-MULTIPLICATION(A11 + A12, B22) 7. P3 STRASSENS-MULTIPLICATION(A21 + A22, B21) 8. P4 STRASSENS-MULTIPLICATION(A22, B21 – B11) 9. P5 STRASSENS-MULTIPLICATION(A11 + A22, B11 + B22) 10. P6 STRASSENS-MULTIPLICATION(A12 – A22, B21 + B22) 11. P7 STRASSENS-MULTIPLICATION(A11 – A21, B11 + B12) 12. C11 P5 + P4 – P2 + P6 13. C12 P1 + P2 14. C21 P3 + P4

	15. C22 P5 + P1 – P3 – P7 16. return C

PROGRAM:	#include <stdio.h>
	#include <stdlib.h>
	// prototypes
	int **addSquareMatrices(int **a, int **b, int n, int a_p, int a_q, int b_p, int b_q);
	int **subtractSquareMatrices(int **a, int **b, int n, int a_p, int a_q,int b_p, int b_q);
	int **strassensMultiplication(int **a, int **b, int n);
	int **actualStrassensMultiplication(int **a, int **b, int n, int a_p, int a_q, int b_p, int b_q);
	int **mallocSqaureMatrix(int n);
	void freeSquareMatrix(int **mat, int n);
	int **mallocSqaureMatrix(int n)
	{
	int **new = malloc(n * sizeof(int *));
	for (int i = 0; i < n; i++)
	new[i] = malloc(n * sizeof(int));
	return new;
	}
	void freeSquareMatrix(int **mat, int n)
	{
	for (int i = 0; i < n; i++)
	free(mat[i]);
	free(mat);
	}
	int **addSquareMatrices(int **a, int **b, int n, int a_p, int a_q, int b_p, int b_q)
	{
	int **sum = mallocSqaureMatrix(n);
	for (int i = 0; i < n; i++)
	{
	for (int j = 0; j < n; j++)
	sum[i][j] = a[a_p + i][a_q + j] + b[b_p + i][b_q + j];
	}
	return sum;
	}
	int **subtractSquareMatrices(int **a, int **b, int n, int a_p, int a_q,
	int b_p, int b_q)
	{
	int **diff = mallocSqaureMatrix(n);
	for (int i = 0; i < n; i++)

```
{
for (int j = 0; j < n; j++)
diff[i][j] = a[a_p + i][a_q + j] - b[b_p + i][b_q + j];

}
return diff;
}
int **strassensMultiplication(int **a, int **b, int n)
{
return actualStrassensMultiplication(a, b, n, 0, 0, 0, 0);
}
int **actualStrassensMultiplication(int **a, int **b, int n, int a_p, int a_q, int b_p, int b_q)
{
int **prod = mallocSqaureMatrix(n);
if (n == 2)
{
int p1 = a[a_p][a_q] * (b[b_p][b_q + 1] - b[b_p + 1][b_q + 1]);
int p2 = (a[a_p][a_q] + a[a_p][a_q + 1]) * b[b_p + 1][b_q + 1];
int p3 = (a[a_p + 1][a_q] + a[a_p + 1][a_q + 1]) * b[b_p][b_q];
int p4 = a[a_p + 1][a_q + 1] * (b[b_p + 1][b_q] - b[b_p][b_q]);
int p5 = (a[a_p][a_q] + a[a_p + 1][a_q + 1]) * (b[b_p][b_q] + b[b_p + 1][b_q + 1]);
int p6 = (a[a_p][a_q + 1] - a[a_p + 1][a_q + 1]) * (b[b_p + 1][b_q] + b[b_p + 1][b_q + 1]);
int p7 = (a[a_p][a_q] - a[a_p + 1][a_q]) * (b[b_p][b_q] + b[b_p][b_q + 1]);
prod[0][0] = p5 + p4 - p2 + p6;
prod[0][1] = p1 + p2;
prod[1][0] = p3 + p4;
prod[1][1] = p5 + p1 - p3 - p7;
}
else
{
int x = n / 2;
int **temp = subtractSquareMatrices(b, b, x, b_p, b_q + x, b_p + x, b_q + x);
int **p1 = actualStrassensMultiplication(a, temp, x, a_p, a_q, 0, 0);
freeSquareMatrix(temp, x);
temp = addSquareMatrices(a, a, x, a_p, a_q, a_p, a_q + x);
int **p2 = actualStrassensMultiplication(temp, b, x, 0, 0, b_p + x, b_q + x);
freeSquareMatrix(temp, x);
temp = addSquareMatrices(a, a, x, a_p + x, a_q, a_p + x, a_q + x);
int **p3 = actualStrassensMultiplication(temp, b, x, 0, 0, b_p, b_q);
freeSquareMatrix(temp, x);
```

```
temp = subtractSquareMatrices(b, b, x, b_p + x, b_q, b_p, b_q);
int **p4 = actualStrassensMultiplication(a, temp, x, a_p + x, a_q + x, 0, 0);
freeSquareMatrix(temp, x);
temp = addSquareMatrices(a, a, x, a_p, a_q, a_p + x, a_q + x);
int **temp2 = addSquareMatrices(b, b, x, b_p, b_q, b_p + x, b_q + x);

int **p5 = actualStrassensMultiplication(temp, temp2, x, 0, 0, 0, 0);
freeSquareMatrix(temp, x);
freeSquareMatrix(temp2, x);
temp = subtractSquareMatrices(a, a, x, a_p, a_q + x, a_p + x, a_q + x);
temp2 = addSquareMatrices(b, b, x, b_p + x, b_q, b_p + x, b_q + x);
int **p6 = actualStrassensMultiplication(temp, temp2, x, 0, 0, 0, 0);
freeSquareMatrix(temp, x);
freeSquareMatrix(temp2, x);
temp = subtractSquareMatrices(a, a, x, a_p, a_q, a_p + x, a_q);
temp2 = addSquareMatrices(b, b, x, b_p, b_q, b_p, b_q + x);
int **p7 = actualStrassensMultiplication(temp, temp2, x, 0, 0, 0, 0);
freeSquareMatrix(temp, x);
freeSquareMatrix(temp2, x);
temp = addSquareMatrices(p5, p4, x, 0, 0, 0, 0);
temp2 = addSquareMatrices(temp, p6, x, 0, 0, 0, 0);
freeSquareMatrix(temp, x);
temp = subtractSquareMatrices(temp2, p2, x, 0, 0, 0, 0);
freeSquareMatrix(temp2, x);
for (int i = 0; i < x; i++)
{
for (int j = 0; j < x; j++)
prod[i][j] = temp[i][j];
}
freeSquareMatrix(temp, x);
temp = addSquareMatrices(p1, p2, x, 0, 0, 0, 0);
for (int i = 0; i < x; i++)
{
for (int j = x; j < n; j++)
prod[i][j] = temp[i][j - x];
}
freeSquareMatrix(temp, x);
temp = addSquareMatrices(p3, p4, x, 0, 0, 0, 0);
for (int i = x; i < n; i++)
{
```

```
for (int j = 0; j < x; j++)
prod[i][j] = temp[i - x][j];
}
freeSquareMatrix(temp, x);
temp = addSquareMatrices(p5, p1, x, 0, 0, 0, 0);
temp2 = subtractSquareMatrices(temp, p3, x, 0, 0, 0, 0);
freeSquareMatrix(temp, x);
temp = subtractSquareMatrices(temp2, p7, x, 0, 0, 0, 0);
for (int i = x; i < n; i++)
{
for (int j = x; j < n; j++)
prod[i][j] = temp[i - x][j - x];
}
freeSquareMatrix(temp, x);

freeSquareMatrix(temp2, x);
}
return prod;
}
int main()
{
printf("Enter matrix dimension(must be in power of 2): ");
int n;
scanf("%d", &n);
int **a = mallocSqaureMatrix(n);
int **b = mallocSqaureMatrix(n);
printf("Enter first matrix elements: ");
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
scanf("%d", &a[i][j]);
}
printf("Enter second matrix elements: ");
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
scanf("%d", &b[i][j]);
}
printf("Product of first and second matrices:\n");
int **prod = strassensMultiplication(a, b, n);
```

```
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
printf("%15d", prod[i][j]);
printf("\n");
}
}
```

RESULT:

```
input
Enter matrix dimension(must be in power of 2): 4 4
Enter first matrix elements: 1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Enter second matrix elements: 1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Product of first and second matrices:
            120             66             76             86
            216            174            196            218
            376            286            324            362
            536            398            452            506


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION: Thus, we have implemented stressens multiplication for the matrix having dimension in the power of 2.