DAA EXP 3

Name	Atharva Bilonikar
UID no.	2021700011
Experiment No.	3

AIM:	Implement Strassen's Matrix Multiplication
Program 1	

PROGRAM:	#include<stdio.h>
	#include<math.h>
	#include<stdlib.h>
	#define MAX_SIZE 32
	void add(int **a, int **b, int size,int **c);
	void sub(int **a, int **b, int size,int **c);
	void multiply(int **c,int **d,int size,int size2,int **new){
	if(size == 1){
	new[0][0] = c[0][0] *d[0][0];
	}
	else {
	int i,j;
	int nsize =size/2;
	int **c11 = malloc(nsize * sizeof(int *));
	for(i=0;i<nsize;i++){
	c11[i]= malloc(nsize*sizeof(int));
	}
	int **c12 = malloc(nsize * sizeof(int *));
	for(i=0;i<nsize;i++){
	c12[i]= malloc(nsize * sizeof(int));
	}

```
int **c21 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   c21[i]= malloc(nsize * sizeof(int));
}
int **c22 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   c22[i]= malloc(nsize*sizeof(int));
}
int **d11 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   d11[i]= malloc(nsize*sizeof(int));
}
int **d12 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   d12[i]= malloc(nsize*sizeof(int));
}
int **d21 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   d21[i]= malloc(nsize*sizeof(int));
}
int **d22 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   d22[i]= malloc(nsize*sizeof(int));
}
int **m1 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   m1[i]= malloc(nsize*sizeof(int));
}
int **m2 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   m2[i]= malloc(nsize*sizeof(int));
}
int **m3 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
```

```
    m3[i]= malloc(nsize*sizeof(int));
  }
  int **m4 = malloc(nsize * sizeof(int *));
  for(i=0;i<nsize;i++){
    m4[i]= malloc(nsize*sizeof(int));
  }
  int **m5 = malloc(nsize * sizeof(int *));
  for(i=0;i<nsize;i++){
    m5[i]= malloc(nsize*sizeof(int));
  }
  int **m6 = malloc(nsize * sizeof(int *));
  for(i=0;i<nsize;i++){
    m6[i]= malloc(nsize*sizeof(int));
  }
  int **m7 = malloc(nsize * sizeof(int *));
  for(i=0;i<nsize;i++){
    m7[i]= malloc(nsize * sizeof(int));
  }
  for(i=0;i<nsize;i++){
    for(j=0;j<nsize;j++){
      c11[i][j]=c[i][j];
      c12[i][j]=c[i][j+nsize];
      c21[i][j]=c[i+nsize][j];
      c22[i][j]=c[i+nsize][j+nsize];
      d11[i][j]=d[i][j];
      d12[i][j]=d[i][j+nsize];
      d21[i][j]=d[i+nsize][j];
      d22[i][j]=d[i+nsize][j+nsize];
    }
  }
  int **temp1 = malloc(nsize * sizeof(int *));
  for(i=0;i<nsize;i++){
    temp1[i]= malloc(nsize*sizeof(int));
  }
```

```
int **temp2 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp2[i]= malloc(nsize*sizeof(int));
}

add(c11,c22,nsize,temp1);
add(d11,d22,nsize,temp2);
multiply(temp1,temp2,nsize,size,m1);
free(temp1);
free(temp2);

int **temp3 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp3[i]= malloc(nsize*sizeof(int));
}
add(c21,c22,nsize,temp3);
multiply(temp3,d11,nsize,size,m2);
free(temp3);


int **temp4 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp4[i]= malloc(nsize*sizeof(int));
}
sub(d12,d22,nsize,temp4);
multiply(c11,temp4,nsize,size,m3);
free(temp4);


int **temp5 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp5[i]= malloc(nsize*sizeof(int));
}
sub(d21,d11,nsize,temp5);
```

```
multiply(c22,temp5,nsize,size,m4);
free(temp5);


int **temp6 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp6[i]= malloc(nsize*sizeof(int));
}
add(c11,c12,nsize,temp6);
multiply(temp6,d22,nsize,size,m5);
free(temp6);

int **temp7 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp7[i]= malloc(nsize*sizeof(int));
}
int **temp8 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp8[i]= malloc(nsize*sizeof(int));
}
sub(c21,c11,nsize,temp7);
add(d11,d12,nsize,temp8);
multiply(temp7,temp8,nsize,size,m6);
free(temp7);
free(temp8);

int **temp9 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp9[i]= malloc(nsize*sizeof(int));
}
int **temp10 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   temp10[i]= malloc(nsize*sizeof(int));
}
```

```
sub(c12,c22,nsize,temp9);
add(d21,d22,nsize,temp10);
multiply(temp9,temp10,nsize,size,m7);
free(temp9);
free(temp10);


int **te1 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te1[i]= malloc(nsize*sizeof(int));
}
int **te2 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te2[i]= malloc(nsize*sizeof(int));
}
int **te3 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te3[i]= malloc(nsize*sizeof(int));
}
int **te4 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te4[i]= malloc(nsize*sizeof(int));
}
int **te5 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te5[i]= malloc(nsize*sizeof(int));
}
int **te6 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te6[i]= malloc(nsize*sizeof(int));
}
int **te7 = malloc(nsize * sizeof(int *));
for(i=0;i<nsize;i++){
   te7[i]= malloc(nsize*sizeof(int));
```

```
    }
    int **te8 = malloc(nsize * sizeof(int *));
    for(i=0;i<nsize;i++){
       te8[i]= malloc(nsize*sizeof(int));
    }

    add(m1,m7,nsize,te1);
    sub(m4,m5,nsize,te2);
    add(te1,te2,nsize,te3);    //c11

    add(m3,m5,nsize,te4);//c12
    add(m2,m4,nsize,te5);//c21

    add(m3,m6,nsize,te6);
    sub(m1,m2,nsize,te7);

    add(te6,te7,nsize,te8);//c22

    int a=0;
    int b=0;
    int c=0;
    int d=0;
    int e=0;
    int nsize2= 2*nsize;
    for(i=0;i<nsize2;i++){
       for(j=0;j<nsize2;j++){
          if(j>=0 && j<nsize && i>=0 && i<nsize){
             new[i][j] = te3[i][j];
          }
          if(j>=nsize && j<nsize2 && i>=0 && i<nsize){
             a=j-nsize;
             new[i][j] = te4[i][a];
          }
          if(j>=0 && j<nsize && i>= nsize && i < nsize2){
```

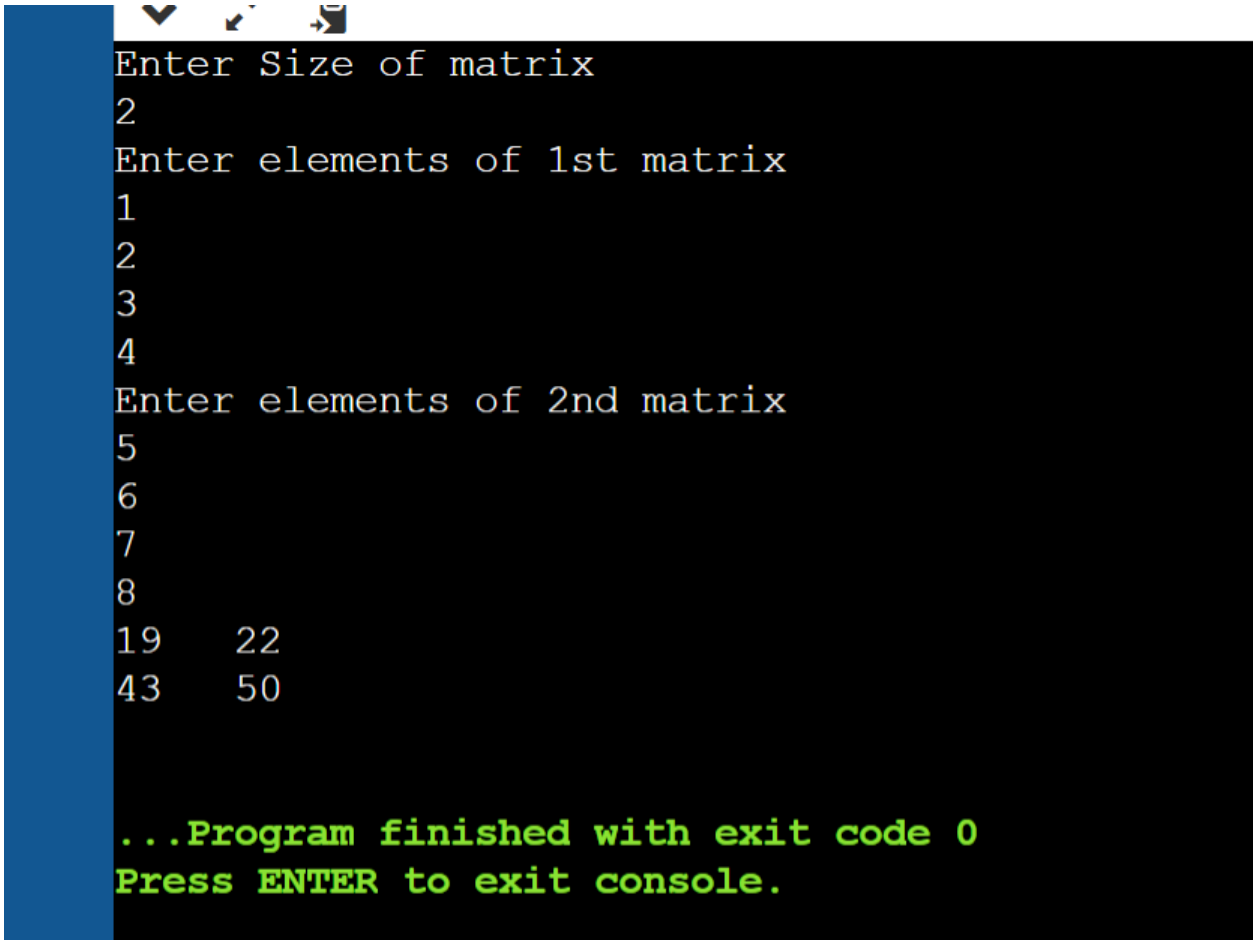
```
                c=i-nsize;
                new[i][j] = te5[c][j];
            }
            if(j>=nsize && j< nsize2 && i>= nsize && i< nsize2
){
                d=i-nsize;
                e=j-nsize;
                new[i][j] =te8[d][e];
            }
        }
    }
  free(m1);
  free(m2);
  free(m3);
  free(m4);
  free(m5);
  free(m6);
  free(m7);
  free(te1);
  free(te2);
  free(te3);
  free(te4);
  free(te5);
  free(te6);
  free(te7);
  free(te8);
  free(c11);
  free(c12);
  free(c21);
  free(c22);
  free(d11);
  free(d12);
  free(d21);
  free(d22);
```

```
    }
}

void main(){
    int size,p,itr,itr1,i,j,nsize;
    printf("Enter Size of matrix\n");
    scanf("%d",&size);
    int tempS = size;
    if(size & size-1 != 0){
        p = log(size)/log(2);
        size = pow(2,p+1);
    }
    int **a = malloc(size * sizeof(int *));
    for(i=0;i<size;i++){
        a[i] = malloc(size*sizeof(int));
    }
    int **b = malloc(size * sizeof(int *));
    for(i=0;i<size;i++){
        b[i] = malloc(size*sizeof(int));
    }
    printf("Enter elements of 1st matrix\n");
    for(itr=0;itr<size;itr++){
    for(itr1=0;itr1<size;itr1++){
        if(itr>=tempS || itr1>=tempS )
            a[itr][itr1]=0;
        else
            scanf("%d",&a[itr][itr1]);
        }
    }
    printf("Enter elements of 2nd matrix\n");
    for(itr=0;itr<size;itr++){
    for(itr1=0;itr1<size;itr1++){
        if(itr>=tempS || itr1>=tempS)
            a[itr][itr1]=0;
```

```
        else
            scanf("%d",&b[itr][itr1]);
        }
    }
    int **new = malloc(size * sizeof(int *));
    for(i=0;i<size;i++){
        new[i] = malloc(size*sizeof(int));
    }
    multiply(a,b,size,size,new);

    if(tempS<size)
        size =tempS;
    for(i=0;i<size;i++){
        for(j=0;j<size;j++){
            printf("%d   ",new[i][j]);
        }
        printf("\n");
    }
}

void add(int **a, int **b, int size,int **c){
    int i,j;
    for(i=0;i<size;i++){
        for(j=0;j<size;j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }
}

void sub(int **a,int **b,int size,int **c){
    int i,j;
    for(i=0;i<size;i++){
                for(j=0;j<size;j++){
                        c[i][j]= a[i][j] - b[i][j];
```

```
        }
    }
}
```

RESULT:



CONCLUSION:	In this experiment , I understood the implementation of divide and conquer Strategy through strassen's matrix multiplication and its advantages over

	Brute force matrix multiplication