DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 3

Name: Janhavi R Deshmukh

Branch: CSE(DS) D1

UID: 2021700017

Aim: – Experiment based on Strassen's matrix multiplication.

Algorithm:

```
STRASSEN(A, B)
n = A.rows
if n == 1
return a[1, 1] * b[1, 1]
let C be a new n × n matrix
A[1, 1] = A[1..n / 2][1..n / 2]
A[1, 2] = A[1..n / 2][n / 2 + 1..n]
A[2, 1] = A[n / 2 + 1..n][1..n / 2]
A[2, 2] = A[n / 2 + 1..n][n / 2 + 1..n]
B[1, 1] = B[1..n / 2][1..n / 2]
B[1, 2] = B[1..n / 2][n / 2 + 1..n]
B[2, 1] = B[n / 2 + 1..n][1..n / 2]
B[2, 2] = B[n / 2 + 1..n][n / 2 + 1..n]
S[1] = B[1, 2] - B[2, 2]
S[2] = A[1, 1] + A[1, 2]
S[3] = A[2, 1] + A[2, 2]
S[4] = B[2, 1] - B[1, 1]
S[5] = A[1, 1] + A[2, 2]
S[6] = B[1, 1] + B[2, 2]
S[7] = A[1, 2] - A[2, 2]
S[8] = B[2, 1] + B[2, 2]
S[9] = A[1, 1] - A[2, 1]
S[10] = B[1, 1] + B[1, 2]
P[1] = STRASSEN(A[1, 1], S[1])
P[2] = STRASSEN(S[2], B[2, 2])
P[3] = STRASSEN(S[3], B[1, 1])
P[4] = STRASSEN(A[2, 2], S[4])
P[5] = STRASSEN(S[5], S[6])
P[6] = STRASSEN(S[7], S[8])
P[7] = STRASSEN(S[9], S[10])
C[1..n / 2][1..n / 2] = P[5] + P[4] - P[2] + P[6]
C[1..n / 2][n / 2 + 1..n] = P[1] + P[2]
C[n / 2 + 1..n][1..n / 2] = P[3] + P[4]
C[n / 2 + 1..n][n / 2 + 1..n] = P[5] + P[1] - P[3] - P[7]
return C
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 8

int **allocatespace(int n)
{
    int **a=malloc(n*sizeof(int*));
     for (int i = 0; i < n; i++)
   {
     a[i] = malloc(n * sizeof(int));
   }

   return a;
}

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];
      }
    }
}



void strassenMultiply(int **A,int **B,int size,int size2,int **new )
{
     if(size==1)
     {
```

```
        //if only one element in Matrix
        new[0][0]=A[0][0]*B[0][0];
    }

    else
    {

        int i,j;
        int m=size/2; //dividing matrix into 4 parts


        int **A11=allocatespace(m);
        int **A12=allocatespace(m);
        int **A21=allocatespace(m);
        int **A22=allocatespace(m);

        int **B11=allocatespace(m);
        int **B12=allocatespace(m);
        int **B21=allocatespace(m);
        int **B22=allocatespace(m);

        int **P=allocatespace(m);
        int **Q=allocatespace(m);
        int **R=allocatespace(m);
        int **S=allocatespace(m);
        int **T=allocatespace(m);
        int **U=allocatespace(m);
        int **V=allocatespace(m);

        for (i = 0; i < m; i++)
      {
          for (j = 0; j < m; j++)
          {
            //top-left half of the matrix
            A11[i][j] = A[i][j];
            //top-right half of the matrix
            A12[i][j] = A[i][j + m];
           //bottom-left half of the matrix
           A21[i][j] = A[i + m][j];
           //bottom-right half of the matrix
           A22[i][j] = A[i + m][j + m];

           B11[i][j] = B[i][j];
           B12[i][j] = B[i][j + m];
           B21[i][j] = B[i + m][j];
           B22[i][j] = B[i + m][j + m];
           }
       }
```

```
//creating 2 matrix for temporary storage
int **temp1 = allocatespace(m);
int **temp2 = allocatespace(m);

//first formula
//P = (A11 + A22)*(B11 + B22)

add(A11,A22,m,temp1);
add(B11,B22,m,temp2);
strassenMultiply(temp1,temp2,m,size,P);
free(temp1);
free(temp2);

int **temp3=allocatespace(m);
//Q = (A21 + A22) * B11
add(A21, A22,m, temp3);
strassenMultiply(temp3, B11, m, size, Q);
free(temp3);

int **temp4=allocatespace(m);
//R = A11 * (B12 - B22)
sub(B12, B22, m, temp4);
strassenMultiply(A11, temp4, m, size, R);
free(temp4);

int **temp5=allocatespace(m);
//S = A22 * (B21 - B11)
sub(B21, B11, m, temp5);
strassenMultiply(A22, temp5, m, size, S);
free(temp5);

 int **temp6=allocatespace(m);
 //T = (A11 + A12) * B22
 add(A11, A12, m, temp6);
 strassenMultiply(temp6,B22,m, size, T);
 free(temp6);

 int **temp7=allocatespace(m);
 int **temp8=allocatespace(m);

 //U = (A21 - A11) * (B11 + B12)
 sub(A21, A11, m, temp7);
 add(B11, B12, m, temp8);
 strassenMultiply(temp7, temp8, m, size, U);
 free(temp7);
 free(temp8);
```

```
        int **temp9=allocatespace(m);
        int **temp10=allocatespace(m);

        //V = (A12 - A22)*(B21 + B22)
        sub(A12, A22, m, temp9);
        add(B21, B22, m, temp10);
        strassenMultiply(temp9, temp10, m, size, V);
        free(temp9);
        free(temp10);


        //matrices for temporary storage
        int **t1=allocatespace(m);
        int **t2=allocatespace(m);
        int **t3=allocatespace(m);
        int **t4=allocatespace(m);
        int **t5=allocatespace(m);
        int **t6=allocatespace(m);
        int **t7=allocatespace(m);
        int **t8=allocatespace(m);

        add(P, V, m, t1);
        sub(S, T, m, t2);
        add(t1, t2, m, t3); // C11

        add(R, T, m, t4); // C12
        add(Q, S, m, t5); // C21

        add(R, U, m, t6);
        sub(P, Q, m, t7);
        add(t6, t7, m, t8); // C22

        int a = 0;
        int b = 0;
        int c = 0;
        int d = 0;
        int e = 0;
        int nsize = 2 * m;

        for (i = 0; i < nsize; i++)
       {
          for (j = 0; j < nsize; j++)
        {
          //C11
          if (j >= 0 && j < m && i >= 0 && i <m)
          {
           new[i][j] = t3[i][j];
          }
```

```
      //C12
     if (j >= m && j < nsize && i >= 0 && i<m)
     {
      a = j - m;
      new[i][j] = t4[i][a];
     }

     //C21
      if (j >= 0 && j < m && i >= m && i <nsize)
     {
       c = i - m;
       new[i][j] = t5[c][j];
     }


    //C22
    if (j >= m && j < nsize && i >= m && i <nsize)
    {
    d = i - m;
    e = j - m;
    new[i][j] = t8[d][e];
    }


    }
   }

  //deallocating all the spaces
  free(P);
  free(Q);
  free(R);
  free(S);
  free(T);
  free(U);
  free(V);
  free(t1);
  free(t2);
  free(t3);
  free(t4);
  free(t5);
  free(t6);
  free(t7);
  free(t8);
  free(A11);
  free(A12);
  free(A21);
  free(A22);
  free(B11);
```

```
        free(B12);
        free(B21);
        free(B22);


    }
}

void main()
{
  int size, nsize;
  printf("Enter Size of matrix\n");
  scanf("%d", &size);


  //space for matrix A
  int **a=allocatespace(size);

  //space for matrix B
  int **b=allocatespace(size);


  //allocating space to store result of multiplication
  int **new =allocatespace(size);



  //Taking inputs for matrix A
  printf("Enter elements of matrix A:\n");

  for (int i = 0; i < size; i++)
  {
    for (int j = 0; j < size; j++)
   {
      scanf("%d", &a[i][j]);
   }

  }

  //Taking inputs for matrix B
  printf("Enter elements of matrix B:\n");

  for (int i = 0; i < size; i++)
  {
    for (int j = 0; j < size; j++)
   {
      scanf("%d", &b[i][j]);
   }
```

```
  }


  strassenMultiply(a, b,size,size,new);
  printf("Matrix C:\n");
  for (int i = 0; i < size; i++)
  {
    for (int j = 0; j < size; j++)
   {
     printf("%d ", new[i][j]);
   }
    printf("\n");
  }

}
```

Output:

```
Enter Size of matrix
4
Enter elements of matrix A:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
Enter elements of matrix B:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
Matrix C:
10 10 10 10
20 20 20 20
30 30 30 30
40 40 40 40
PS C:\Users\janha\OneDrive\Documents\C programs\arrays>
```

Conclusion:

Implemented Strassen's Matrix multiplication which uses divide and conquer approach.
Time Complexity for all cases: O(n^2.81)