Lab 5

Radhika Raghu
1410110317

Time Taken	
Using Recursion	0.011211
Using Strassen's Algorithm	0.006051

Code for Matrix Multiplication using Strassen's Algorithm

```
/*
Lab 5
STRASSEN'S ALGORITHM
Radhika Raghu
1410110317
*/

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

const int N = 128;

typedef struct _m {
int rs;		//start of row
int re;		//end of roww
int cs;		//start of column
int ce;		//end of column
int a[N][N];//matrix
}m;

m A;
m B;

void display(m matrix){

	int i, j;
	for (i = 0 ; i <= 128 ; i++){
		for (j = 0 ; j <= 128 ; j++)
			printf("%d ", matrix.a[i][j]);
		printf("\n");
   }
	printf("\n");
}

m plus(m m1, m m2){
```

```
        m result;
        int m1_i, m1_j;
        int m2_i, m2_j;
        int i, j;
        int n = m1.re - m1.rs;

        result.rs = result.cs = 0;
        result.re = result.ce = n;

        for (m1_i=m1.rs, m2_i=m2.rs, i=0 ; m1_i<=m1.re ; m1_i++, m2_i++, i++)
                for (m1_j=m1.cs, m2_j=m2.cs, j=0 ; m1_j<=m1.ce ; m1_j++, m2_j++, j++)
                        result.a[i][j] = m1.a[m1_i][m1_j] + m2.a[m2_i][m2_j];

        return result;
}
m minus(m m1, m m2){

        m result;
        int m1_i, m1_j;
        int m2_i, m2_j;
        int i, j;
        int n = m1.re - m1.rs;

        result.rs = result.cs = 0;
        result.re = result.ce = n;

        for (m1_i=m1.rs, m2_i=m2.rs, i=0 ; m1_i<=m1.re ; m1_i++, m2_i++, i++)
                for (m1_j=m1.cs, m2_j=m2.cs, j=0 ; m1_j<=m1.ce ; m1_j++, m2_j++, j++)
                        result.a[i][j] = m1.a[m1_i][m1_j] - m2.a[m2_i][m2_j];

        return result;

}

m multiply(m m1, m m2){

    m A, B, C, D, E, F, G, H;
    m P1, P2, P3, P4, P5, P6, P7;
    m Q1, Q2, Q3, Q4;
    m result;
    int m1_i, m1_j;
    int i, j;
    int n = m1.re - m1.rs + 1;

    /*If N == 2*/
    if (n <= 2) {

        int a, b, c, d, e, f, g, h;

        /*Applying Strassen's Formulae*/
        m m3 = m1;
        a = m1.a[m1.rs][m1.cs];
        b = m1.a[m1.rs][m1.cs+1];
        c = m1.a[m1.rs+1][m1.cs];
        d = m1.a[m1.rs+1][m1.cs+1];
```

```
        e = m2.a[m2.rs][m2.cs];
        f = m2.a[m2.rs][m2.cs+1];
        g = m2.a[m2.rs+1][m2.cs];
        h = m2.a[m2.rs+1][m2.cs+1];
        m3.a[m3.rs][m3.cs] = a*e + b*g;
    m3.a[m3.rs][m3.cs+1] = a*f + b*h;
    m3.a[m3.rs+1][m3.cs] = c*e + d*g;
    m3.a[m3.rs+1][m3.cs+1] = c*f + d*h;

        return m3;
}
/*When N > 2*/
result.rs = result.cs = 0;
result.ce = result.re = n-1;

A = B = C = D = m1;
E = F = G = H = m2;

/*Dividing the matrices*/
A.rs = m1.rs;
A.re = m1.re/2;
A.cs = m1.cs;
A.ce = m1.ce/2;

B.rs = m1.rs;
B.re = m1.re/2;
B.cs = m1.ce/2 + 1;
B.ce = m1.ce;

C.rs = m1.re/2 + 1;
C.re = m1.re;
C.cs = m1.cs;
C.ce = m1.ce/2;

D.rs = m1.re/2 + 1;
D.re = m1.re;
D.cs = m1.ce/2 + 1;
D.ce = m1.ce;

E.rs = m2.rs;
E.re = m2.re/2;
E.cs = m2.cs;
E.ce = m2.ce/2;

F.rs = m2.rs;
F.re = m2.re/2;
F.cs = m2.ce/2 + 1;
F.ce = m2.ce;

G.rs = m2.re/2 + 1;
G.re = m2.re;
G.cs = m2.cs;
G.ce = m2.ce/2;

H.rs = m2.re/2 + 1;
H.re = m2.re;
```

```
    H.cs = m2.ce/2 + 1;
    H.ce = m2.ce;

    /*Strassen's Formulae*/

    P1 = multiply(A, minus(F, H));
    P2 = multiply(plus(A, B), H);
    P3 = multiply(plus(C, D), E);
    P4 = multiply(D, minus(G, E));
    P5 = multiply(plus(A, D), plus(E, H));
    P6 = multiply(minus(B, D), plus(G, H));
    P7 = multiply(minus(A, C), plus(E, F));

    Q1 = plus(minus(plus(P5, P4), P2), P6);
    Q2 = plus(P1, P2);
    Q3 = plus(P3, P4);
    Q4 = minus(minus(plus(P1, P5), P3), P7);

    for (m1_i=Q1.rs, i=0 ; m1_i<=Q1.re ; m1_i++, i++)
            for (m1_j=Q1.cs, j=0 ; m1_j<=Q1.ce ; m1_j++, j++)
                    result.a[i][j] = Q1.a[m1_i][m1_j];

    for (m1_i=Q2.rs, i=0 ; m1_i<=Q2.re ; m1_i++, i++)
            for (m1_j=Q2.cs, j=n/2 ; m1_j<=Q2.ce ; m1_j++, j++)
                    result.a[i][j] = Q2.a[m1_i][m1_j];

    for (m1_i=Q3.rs, i=n/2 ; m1_i<=Q3.re ; m1_i++, i++)
            for (m1_j=Q3.cs, j=0 ; m1_j<=Q3.ce ; m1_j++, j++)
                    result.a[i][j] = Q3.a[m1_i][m1_j];

    for (m1_i=Q4.rs, i=n/2 ; m1_i<=Q4.re ; m1_i++, i++)
            for (m1_j=Q4.cs, j=n/2 ; m1_j<=Q4.ce ; m1_j++, j++)
                    result.a[i][j] = Q4.a[m1_i][m1_j];
    return result;
}

int main(void){

            clock_t end_t, start_t;
            double total_t;

            /*Initialising matrix A*/
            int i, j;
            for(i = 0; i < N; i++) {
                    for(j = 0; j < N; j++)
                            A.a[i][j] = rand() % 10;
            }

            /*Initialising matrix B*/
            for(i = 0; i < N; i++){
                    for(j = 0; j < N; j++)
                            B.a[i][j] = rand() % 10;
            }

            printf("----------Matrix A---------- \n ");
            display(A);
```

```
        printf("----------Matrix B---------- \n ");
        display(B);

        start_t = clock();

        printf("----------RESULT---------- \n");
        display(multiply(A, B));

        end_t = clock();
        total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
        printf("Time Taken for performing Strassen's Algorithm : %lf\n", total_t);

        return 0;
}
```

Code for Matrix Multiplication using Recursion

```
/*
Lab 5
RECURSION
Radhika Raghu
1410110317
*/


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

#define SIZE 128

void multiply(int A[SIZE][SIZE], int B[SIZE][SIZE], int C[SIZE][SIZE]){

  static int i = 0, j = 0, k = 0;

  if (i >= SIZE)
    return;
  else if (i < SIZE){
    if (j < SIZE){
      if (k < SIZE){

        C[i][j] += A[i][k] * B[k][j];
        k++;
        multiply(A, B, C);
      }
      k = 0;
      j++;
      multiply(A, B, C);
    }
    j = 0;
    i++;
    multiply(A, B, C);
  }
}

void display(int C[SIZE][SIZE]){

  int i, j;

  for (i = 0 ; i <= SIZE ; i++){
    for (j = 0 ; j <= SIZE ; j++)
      printf("%d ", C[i][j]);
    printf("\n");
  }
  printf("\n");
}

int main(){

  int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE] = {0};
```

```
  clock_t end_t, start_t;
  double total_t;

  /*Initialising matrix A*/
  int i, j;
  for(i = 0; i < SIZE; i++) {
    for(j = 0; j < SIZE; j++)
      A[i][j] = rand() % 10;
  }

  /*Initialising matrix B*/
  for(i = 0; i < SIZE; i++){
    for(j = 0; j < SIZE; j++)
      B[i][j] = rand() % 10;
  }

  printf("----------Matrix A---------- \n ");
  display(A);

  printf("----------Matrix B---------- \n ");
  display(B);

  start_t = clock();

  printf("----------RESULT---------- \n");
  multiply(A, B, C);
  display(C);

  end_t = clock();
  total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
  printf("Time Taken for performing Matrix Multipicaiton using Recursion : %lf\n", total_t);

  return 0;


}
```