NAME :- SAHIL PRAVIN THAKUR

PRN : - 2020BTECS00042

BATCH : - T7

SUB :-   DAA LAB

NAME :- SAHIL PRAVIN THAKUR PRN :- 2020BTECS00042 BATCH :- T7

Assignment: 4

Q1)

A) Implement **Naive Method** multiply two matrices. and justify Complexity is $O(n^3)$

Algorithm:-

matrixMultiply(A, B):

Assume dimension of A is (m x n), dimension of B is (p x q)

Begin

  if n is not same as p, then exit

  otherwise define C matrix as (m x q)

  for i in range 0 to m - 1, do

    for j in range 0 to q – 1, do

      for k in range 0 to p, do

        C[i, j] = C[i, j] + (A[i, k] * A[k, j])

      done

    done

  done

End

Code

```
#include<iostream>
using namespace std;
const int n1 = 3, m1 = 2, n2 = 2, m2 = 3;
void  multiply(int mat1[][m1],int mat2[][m2])
{
    int result[n1][m2];

    for (int i = 0; i < n1;i++)
    {
        for (int j = 0;j<m2;j++)
        {   result[i][j] = 0;
            for (int k = 0; k < m1;k++)
            {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }

    for (int i = 0; i < n1;i++)
    {
```

```
        for (int j = 0; j < m2;j++)
        {
            cout<<result[i][j]<<" ";
        }
        cout<<endl;
    }
}
int main()
{
    int mat1[3][2] = {
        {1, 1},
        {2, 2},
        {3, 3}};
    int mat2[2][3] = {
        {1, 1, 1},
        {2, 2, 2}};

        cout<<endl<<"Mat1 "<<endl<<endl;
        for (int i = 0; i < n1; i++)
        {
            for (int j = 0; j < m1; j++)
            {
                cout << mat1[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl
             << "Mat2 " << endl
             << endl;

        for (int i = 0; i < n2; i++)
        {
            for (int j = 0; j < m2; j++)
            {
                cout << mat2[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl<<"Multiplication" << endl;
        multiply(mat1, mat2);

        return 0;
}
```

Time Complexity :-

Here we are running three nested loop from 1 – n so here time complexity

` is **O(N^3)**

Output:-

```
Mat1

1 1
2 2
3 3

Mat2

1 1 1
2 2 2

Multiplication
3 3 3
6 6 6
9 9 9
PS D:\sem5\DAA Lab\Assignment4>
```

B) Implement **Divide and Conquer** multiply tow matrices . and justify Complexity is $O(n^3)$

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

Step1 :- Divide matrix in the group in 4 parts .

Step2 :- when size of each matrix is 2 thane multiply then and return result

Step3 :- combine the result of multiplication of individual parts and return result

```
#include <iostream>
#include <vector>
using namespace std;

void add(vector<vector<int>> matrix_A,
                vector<vector<int>> matrix_B,
                vector<vector<int>> &matrix_C,
                int mid)
{
    for (auto i = 0; i < mid; i++)
        for (auto j = 0; j < mid; j++)
            matrix_C[i][j] = matrix_A[i][j] + matrix_B[i][j];
}

vector<vector<int>> DACM(vector<vector<int>> mat1, vector<vector<int>> mat2)
{
    int n1 = mat1.size();
    int m1 = mat1[0].size();
    int n2 = mat2.size();
    int m2 = mat2.size();
    vector<vector<int>> result(n1, vector<int>(m1, 0));
    if (n1 == 1)
    {
        result[0][0] = mat1[0][0] * mat2[0][0];
    }
    else{
        int mid = m1 / 2;
        vector<vector<int>> a1(mid, vector<int>(mid, 0));
        vector<vector<int>> a2(mid, vector<int>(mid, 0));
        vector<vector<int>> a3(mid, vector<int>(mid, 0));
        vector<vector<int>> a4(mid, vector<int>(mid, 0));
        vector<vector<int>> b1(mid, vector<int>(mid, 0));
        vector<vector<int>> b2(mid, vector<int>(mid, 0));
        vector<vector<int>> b3(mid, vector<int>(mid, 0));
        vector<vector<int>> b4(mid, vector<int>(mid, 0));
```

```
        vector<vector<int>> r1(mid, vector<int>(mid, 0));
        vector<vector<int>> r2(mid, vector<int>(mid, 0));
        vector<vector<int>> r3(mid, vector<int>(mid, 0));
        vector<vector<int>> r4(mid, vector<int>(mid, 0));

        for (int i = 0; i < mid; i++)
        {
            for (int j = 0; j < mid; j++)
            {
                a1[i][j] = mat1[i][j];
                a2[i][j] = mat1[i + mid][j];
                a3[i][j] = mat1[i][j + mid];
                a4[i][j] = mat1[i + mid][j + mid];
                b1[i][j] = mat2[i][j];
                b2[i][j] = mat2[i + mid][j];
                b3[i][j] = mat2[i][j + mid];
                b4[i][j] = mat2[i + mid][j + mid];
            }
        }
        add(DACM(a1, b1), DACM(a2, b3), r1, mid);
        add(DACM(a1, b2), DACM(a2, b4), r2, mid);
        add(DACM(a3, b1), DACM(a4, b3), r3, mid);
        add(DACM(a1, b1), DACM(a2, b3), r4, mid);

        for (auto i = 0; i < mid; i++)
            for (auto j = 0; j < mid; j++)
            {
                result[i][j] = r1[i][j];
                result[i][j + mid] = r2[i][j];
                result[mid + i][j] = r3[i][j];
                result[i + mid][j + mid] = r4[i][j];
            }

        r1.clear();
        r2.clear();
        r3.clear();
        r4.clear();
        a1.clear();
        a2.clear();
        a3.clear();
        a4.clear();
        b1.clear();
        b2.clear();
        b3.clear();
        b4.clear();
    }

    return result;
}
int main()
{
```

```
    vector<vector<int>> mat1 = {{1, 1, 1, 1},
                                {2, 2, 2, 2},
                                {3, 3, 3, 3},
                                {2, 2, 2, 2}};
    vector<vector<int>> mat2 = {{1, 1, 1, 1},
                                {2, 2, 2, 2},
                                {3, 3, 3, 3},
                                {2, 2, 2, 2}};
    vector<vector<int>>result = DACM(mat1, mat2);
    for (int i = 0; i < result.size();i++)
    {
        for (int j = 0; j < result[0].size();j++)
        {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }

        return 0;
}
```

Output:-

```
PS D:\sem5\DAA Lab\Assignment4> cd "d:\sem5
8 16 24 16
8 8 24 24
8 16 8 16
8 8 8 8
PS D:\sem5\DAA Lab\Assignment4>
```

Time Complexity:-

Time complexity is O(N^3)

Recurrence equation for this is.

$$\boxed{T(n) = 8T\left(\frac{n}{2}\right) + bn^2}$$

put $n = n/2$

$$T(n/2) = 8T\left(\frac{n}{4}\right) + \frac{bn^2}{4}$$

$$T(n) = 64T\left(\frac{n}{4}\right) + 3bn^2$$

$$T(n) = k^3 T\left(\frac{n}{k}\right) + 3bn^2$$

$$\frac{n}{k} = 1$$

$$\boxed{n = k}$$

$$T(n) = n^3 T(1) + 3bn^2 \qquad T(1) = c$$

$$= n^3 c + 3bn^2$$

$$\boxed{T(n) = O(n^3)}$$

C) Implement **Strassen's Matrix Multiplication** and justify Complexity is $O(n^{2.8})$

p1 = a(f - h)
p3 = (c + d)e
p5 = (a + d)(e + h)
p7 = (a - c)(e + f)

p2 = (a + b)h
p4 = d(g - e)
p6 = (b - d)(g + h)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Algorithm:-

Step1:-

Divide matrix in 4 parts recursively .

Step2:-

When size of input is 1 base case hit and then multiply 2 matrix by starsen multiplication 7 formula and return the answer

Step3:-

After that combine result of 4 parts and return the result

Code:-

```
     #include <bits/stdc++.h>
using namespace std;

#define ROW_1 4
#define COL_1 4

#define ROW_2 4
#define COL_2 4

void print(string display, vector<vector<int>> matrix,
           int start_row, int start_column, int end_row,
           int end_column)
{
    cout << endl
         << display << " =>" << endl;
```

```
    for (int i = start_row; i <= end_row; i++)
    {
        for (int j = start_column; j <= end_column; j++)
        {
            cout << setw(10);
            cout << matrix[i][j];
        }
        cout << endl;
    }
    cout << endl;
    return;
}

vector<vector<int>>
add_matrix(vector<vector<int>> matrix_A,
           vector<vector<int>> matrix_B, int split_index,
           int multiplier = 1)
{
    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++)
            matrix_A[i][j] = matrix_A[i][j] + (multiplier * matrix_B[i][j]);
    return matrix_A;
}

vector<vector<int>>
multiply_matrix(vector<vector<int>> matrix_A,
                vector<vector<int>> matrix_B)
{
    int col_1 = matrix_A[0].size();
    int row_1 = matrix_A.size();
    int col_2 = matrix_B[0].size();
    int row_2 = matrix_B.size();

    if (col_1 != row_2)
    {
        cout << "\nError: The number of columns in Matrix "
                "A must be equal to the number of rows in "
                "Matrix B\n";
        return {};
    }

    vector<int> result_matrix_row(col_2, 0);
    vector<vector<int>> result_matrix(row_1,
                                      result_matrix_row);

    if (col_1 == 1)
        result_matrix[0][0] = matrix_A[0][0] * matrix_B[0][0];
    else
    {
        int split_index = col_1 / 2;
```

```
        vector<int> row_vector(split_index, 0);

        vector<vector<int>> a00(split_index, row_vector);
        vector<vector<int>> a01(split_index, row_vector);
        vector<vector<int>> a10(split_index, row_vector);
        vector<vector<int>> a11(split_index, row_vector);
        vector<vector<int>> b00(split_index, row_vector);
        vector<vector<int>> b01(split_index, row_vector);
        vector<vector<int>> b10(split_index, row_vector);
        vector<vector<int>> b11(split_index, row_vector);

        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++)
            {
                a00[i][j] = matrix_A[i][j];
                a01[i][j] = matrix_A[i][j + split_index];
                a10[i][j] = matrix_A[split_index + i][j];
                a11[i][j] = matrix_A[i + split_index]
                                    [j + split_index];
                b00[i][j] = matrix_B[i][j];
                b01[i][j] = matrix_B[i][j + split_index];
                b10[i][j] = matrix_B[split_index + i][j];
                b11[i][j] = matrix_B[i + split_index]
                                    [j + split_index];
            }

        vector<vector<int>> p(multiply_matrix(
            a00, add_matrix(b01, b11, split_index, -1)));
        vector<vector<int>> q(multiply_matrix(
            add_matrix(a00, a01, split_index), b11));
        vector<vector<int>> r(multiply_matrix(
            add_matrix(a10, a11, split_index), b00));
        vector<vector<int>> s(multiply_matrix(
            a11, add_matrix(b10, b00, split_index, -1)));
        vector<vector<int>> t(multiply_matrix(
            add_matrix(a00, a11, split_index),
            add_matrix(b00, b11, split_index)));
        vector<vector<int>> u(multiply_matrix(
            add_matrix(a01, a11, split_index, -1),
            add_matrix(b10, b11, split_index)));
        vector<vector<int>> v(multiply_matrix(
            add_matrix(a00, a10, split_index, -1),
            add_matrix(b00, b01, split_index)));

        vector<vector<int>> result_matrix_00(add_matrix(
            add_matrix(add_matrix(t, s, split_index), u,
                       split_index),
            q, split_index, -1));
        vector<vector<int>> result_matrix_01(
            add_matrix(p, q, split_index));
        vector<vector<int>> result_matrix_10(
```

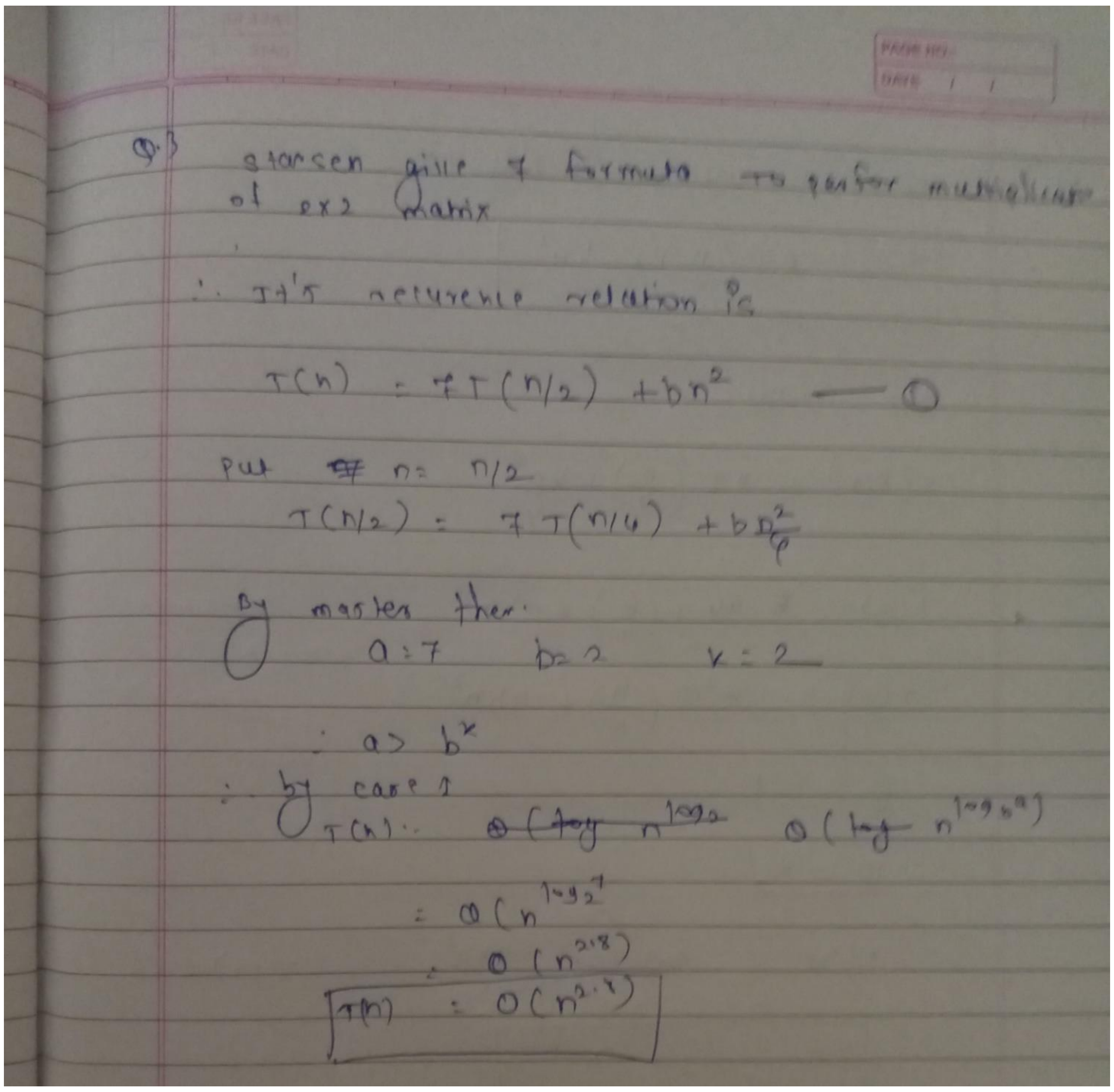
```
            add_matrix(r, s, split_index));
        vector<vector<int>> result_matrix_11(add_matrix(
            add_matrix(add_matrix(t, p, split_index), r,
                       split_index, -1),
            v, split_index, -1));

        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++)
            {
                result_matrix[i][j] = result_matrix_00[i][j];
                result_matrix[i][j + split_index] = result_matrix_01[i][j];
                result_matrix[split_index + i][j] = result_matrix_10[i][j];
                result_matrix[i + split_index]
                             [j + split_index] = result_matrix_11[i][j];
            }

        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
        p.clear();
        q.clear();
        r.clear();
        s.clear();
        t.clear();
        u.clear();
        v.clear();
        result_matrix_00.clear();
        result_matrix_01.clear();
        result_matrix_10.clear();
        result_matrix_11.clear();
    }
    return result_matrix;
}

int main()
{
    vector<vector<int>> matrix_A = {{1, 1, 1, 1},
                                    {2, 2, 2, 2},
                                    {3, 3, 3, 3},
                                    {2, 2, 2, 2}};


    vector<vector<int>> matrix_B = {{1, 1, 1, 1},
                                    {2, 2, 2, 2},
                                    {3, 3, 3, 3},
                                    {2, 2, 2, 2}};
```

```
    vector<vector<int>> result_matrix(
        multiply_matrix(matrix_A, matrix_B));
    cout << "Result of multiplication of two matrix<" << endl
         << endl;
    for (int i = 0; i < ROW_1;i++)
    {
        for (int j = 0; j < COL_1;j++)
        {
            cout << result_matrix[i][j] << " ";
        }
        cout << endl;
    }
        return 0;
    }
```

Explanation :-

Strassen give 7 formula for multiplication of two 2X2 which slightly reduced complexity of multiplication of two matrix

Time complexity :-



Q.3 strassen give 7 formula to perform multiplication of 2x2 matrix

∴ It's recurrence relation is

$T(n) = 7T(n/2) + bn^2$ —— ①

put n = n/2

$T(n/2) = 7T(n/4) + b\frac{n^2}{4}$

By master theorem

$a = 7 \quad b = 2 \quad k = 2$

∴ $a > b^k$

∴ by case 1

$T(n) = \cancel{\Theta(\log n^{\log_2})} \quad \Theta(\log n^{\log_b a})$

$= \Theta(n^{\log_2 7})$

$= \Theta(n^{2.8})$

$\boxed{T(n) = O(n^{2.7})}$