Walchand College of Engineering, Sangli
Computer Science & Engineering
Third Year
Course: Design and analysis of algorithm Lab (3CS351)
Lab course coordinator:
Dr. B. F. Momin- Batch: - T6, T7, T8
Mr. Kiran P. Kamble- Batch: - T1, T2, T3, T4, T5

Week 4 Assignment

Part: 2

Divide and conquer strategy

Name: Trupti Rajendra Patil

Prn no: 2020BTECS00051

Strassen's Matrix Multiplication

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

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

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

$p1 = a(f - h)$ $\quad$ $p2 = (a + b)h$
$p3 = (c + d)e$ $\quad$ $p4 = d(g - e)$
$p5 = (a + d)(e + h)$ $\quad$ $p6 = (b - d)(g + h)$
$p7 = (a - c)(e + f)$

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

Naive Method:

Algorithm:

Using three nested for loops.

Traverse the matrix using two loops and use one more for loop nested with the two for loops for multiplication.

Code:

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

int main(){
    vector<vector<int>> a={{2, 2, 3, 1},{1, 4, 1, 2},{2, 3, 1, 1},
{1, 3, 1, 2}};
    vector<vector<int>> b={{2, 1, 2, 1},{3, 1, 2, 1},{3, 2, 1, 1},
{1, 4, 3, 2}};
    int n=a.size();

    vector<vector<int>> c(n, vector<int>(n));;

    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            c[i][j]=0;
            for(int k=0;k<n;k++){
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }

    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<c[i][j]<<" \t";
        }cout<<endl;
    }
```

```
}
```

Complexity Analysis:

Time complexity: O(n^3).

Auxiliary Space: O(n^2)

Output:

```
PROBLEMS    OUTPUT    TERMINAL    JUPYTER    DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ> cd "c:\Users\t
Q\" ; if ($?) { g++ A4Q1a.cpp -o A4Q1a } ; if ($?) { .\A4Q1a }
20      14      14      9
19      15      17      10
17      11      14      8
16      14      15      9
PS C:\Users\trupti patil\OneDrive\Desktop\ACADEMICS\SEM5\DAA\ExpQ>
```

Divide and Conquer:

Algorithm:

Divide matrices A and B in 4 sub-matrices of size N/2 x N/2

Calculate following values recursively. ae + bg, af + bh, ce + dg and cf +dh.

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(5);
            cout << matrix[i][j];
        }
        cout << endl;
    }
    cout << endl;
    return;
}

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

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 << "\nmultiplication not possible\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>> result_matrix_00(split_index,
                                             row_vector);
        vector<vector<int>> result_matrix_01(split_index,
                                             row_vector);
        vector<vector<int>> result_matrix_10(split_index,
                                             row_vector);
        vector<vector<int>> result_matrix_11(split_index,
                                             row_vector);

        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];
            }

        add_matrix(multiply_matrix(a00, b00),
                   multiply_matrix(a01, b10),
                   result_matrix_00, split_index);
        add_matrix(multiply_matrix(a00, b01),
                   multiply_matrix(a01, b11),
                   result_matrix_01, split_index);
        add_matrix(multiply_matrix(a10, b00),
                   multiply_matrix(a11, b10),
                   result_matrix_10, split_index);
        add_matrix(multiply_matrix(a10, b01),
                   multiply_matrix(a11, b11),
                   result_matrix_11, split_index);

        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];
            }

        result_matrix_00.clear();
        result_matrix_01.clear();
        result_matrix_10.clear();
        result_matrix_11.clear();
        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
```

```
    }
    return result_matrix;
}
```

Complexity Analysis:

Time Complexity:

T(n) = 8T(n/2) + O(n^2)

T(n)=aT(n/b)+O(n^k(log n)^p)

a=8,b=2,k=2,p=0

b^k=4 which is <a

hence Time complexity is O(n^(log a base b))

O(n^3)

From Master's Theorem, time complexity of above method is O(n^3)

Output:

```
Array A =>
    2    5    1    3
    1    4    2    3
    4    1    3    1
    2    2    1    1


Array B =>
    1    1    4    2
    1    3    5    2
    2    3    1    1
    3    2    1    4


Result Array =>
   18   26   37   27
   18   25   29   24
   14   18   25   17
    9   13   20   13
```

Strassen's Matrix Multiplication:

Algorithm:

STRESSEN_MAT_MUL (int *A, int *B, int *C, int n)

if n == 1 then

    *C = *C + (*A) * (*B)

else

    STRESSEN_MAT_MUL (A, B, C, n/4)

    STRESSEN_MAT_MUL (A, B + (n/4), C + (n/4), n/4)

    STRESSEN_MAT_MUL (A + 2 * (n/4), B, C + 2 * (n/4), n/4)

    STRESSEN_MAT_MUL (A + 2 * (n/4), B + (n/4), C + 3 * (n/4), n/4)

    STRESSEN_MAT_MUL (A + (n/4), B + 2 * (n/4), C, n/4)

    STRESSEN_MAT_MUL (A + (n/4), B + 3 * (n/4), C + (n/4), n/4)

    STRESSEN_MAT_MUL (A + 3 * (n/4), B + 2 * (n/4), C + 2 * (n/4), n/4)

    STRESSEN_MAT_MUL (A + 3 * (n/4), B + 3 * (n/4), C + 3 * (n/4), n/4)

End

Code:

```
#include <bits/stdc++.h>
#include <cmath>
#define vi vector<int>
#define vii vector<vi>
using namespace std;

int nextPowerOf2(int k)
{
    return pow(2, int(ceil(log2(k))));
}

void display(vii C, int m, int n)
{
    for (int i = 0; i < m; i++)
    {
        cout << ""
```

```
             << " ";
        for (int j = 0; j < n; j++)
        {
            cout << C[i][j] << " ";
        }
        cout << "" << endl;
    }
}

void add(vii &A, vii &B, vii &C, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}
void sub(vii &A, vii &B, vii &C, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

void Strassen_algorithm(vii &A, vii &B, vii &C, int size)
{
    if (size == 1)
    {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }
    else
    {
        int newSize = size / 2;
        vi z(newSize);
```

```
        vii a(newSize, z), b(newSize, z), c(newSize, z), d(newSize,
z),
            e(newSize, z), f(newSize, z), g(newSize, z), h(newSize,
z),
            c11(newSize, z), c12(newSize, z), c21(newSize, z),
c22(newSize, z),
            p1(newSize, z), p2(newSize, z), p3(newSize, z),
p4(newSize, z),
            p5(newSize, z), p6(newSize, z), p7(newSize, z),
fResult(newSize, z),
            sResult(newSize, z);
        int i, j;

        for (i = 0; i < newSize; i++)
        {
            for (j = 0; j < newSize; j++)
            {
                a[i][j] = A[i][j];
                b[i][j] = A[i][j + newSize];
                c[i][j] = A[i + newSize][j];
                d[i][j] = A[i + newSize][j + newSize];

                e[i][j] = B[i][j];
                f[i][j] = B[i][j + newSize];
                g[i][j] = B[i + newSize][j];
                h[i][j] = B[i + newSize][j + newSize];
            }
        }

        sub(f, h, sResult, newSize);
        Strassen_algorithm(a, sResult, p1, newSize);

        add(a, b, fResult, newSize);
        Strassen_algorithm(fResult, h, p2, newSize);

        add(c, d, fResult, newSize);
        Strassen_algorithm(fResult, e, p3, newSize);

        sub(g, e, sResult, newSize);
        Strassen_algorithm(d, sResult, p4, newSize);

        add(a, d, fResult, newSize);
```

```
        add(e, h, sResult, newSize);
        Strassen_algorithm(fResult, sResult, p5, newSize);

        sub(b, d, fResult, newSize);
        add(g, h, sResult, newSize);
        Strassen_algorithm(fResult, sResult, p6, newSize);

        sub(a, c, fResult, newSize);
        add(e, f, sResult, newSize);
        Strassen_algorithm(fResult, sResult, p7, newSize);

        add(p1, p2, c12, newSize);
        add(p3, p4, c21, newSize);

        add(p4, p5, fResult, newSize);
        add(fResult, p6, sResult, newSize);
        sub(sResult, p2, c11, newSize);

        sub(p1, p3, fResult, newSize);
        add(fResult, p5, sResult, newSize);
        sub(sResult, p7, c22, newSize);

        for (i = 0; i < newSize; i++)
        {
            for (j = 0; j < newSize; j++)
            {
                C[i][j] = c11[i][j];
                C[i][j + newSize] = c12[i][j];
                C[i + newSize][j] = c21[i][j];
                C[i + newSize][j + newSize] = c22[i][j];
            }
        }
    }
}

void ConvertToSquareMat(vii &A, vii &B, int r1, int c1, int r2, int
c2)
{
    int maxSize = max({r1, c1, r2, c2});
    int size = nextPowerOf2(maxSize);

    vi z(size);
```

```
    vii Aa(size, z), Bb(size, z), Cc(size, z);

    for (unsigned int i = 0; i < r1; i++)
    {
        for (unsigned int j = 0; j < c1; j++)
        {
            Aa[i][j] = A[i][j];
        }
    }
    for (unsigned int i = 0; i < r2; i++)
    {
        for (unsigned int j = 0; j < c2; j++)
        {
            Bb[i][j] = B[i][j];
        }
    }
    Strassen_algorithm(Aa, Bb, Cc, size);
    vi temp1(c2);
    vii C(r1, temp1);
    for (unsigned int i = 0; i < r1; i++)
    {
        for (unsigned int j = 0; j < c2; j++)
        {
            C[i][j] = Cc[i][j];
        }
    }
    display(C, r1, c1);
}
int main()
{
    vii a = {{2, 5, 1, 3},
             {1, 4, 2, 3},
             {4, 1, 3, 1},
             {2, 2, 1, 1}};

    vii b = {{1, 1, 4, 2},
             {1, 3, 5, 2},
             {2, 3, 1, 1},
             {3, 2, 1, 4}};

    ConvertToSquareMat(a, b, 4, 4, 4, 4);
    return 0;
```

```
}
```

Complexity Analysis:

Strassen's approach performs seven multiplications on the problem of size 1 x 1, which in turn finds the multiplication of 2 x 2 matrices using addition. To solve the problem of size n, Strassen's approach creates seven problems of size (n – 2). Recurrence equation for Strassen's approach is given as,

$T(n) = 7.T(n/2)$

$T(n/2) = 7.T(n/4)$

$\Rightarrow T(n) = 7^2.T(n/2^2)$

.

$T(n) = 7^k.T(2^k)$

Let's assume $n = 2^k \Rightarrow k = \log_2 n$

$T (n) = 7^k .T(2^k/2^k)$

$= 7^k .T (1)$

$= 7^k$

$= 7^{\log_2 n}$

$= n^{\log_2 7}$

$= n^{2.81}$

Output:

```
18 26 37 27
18 25 29 24
14 18 25 17
9 13 20 13
```