Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

Name	Tejas Jadhav
UID No.	2022301006
Class	COMPS A (B batch)
Experiment No.	3

Aim: Experiment on Recurrence Relation

- Perform Matrix Multiplication using normal multiplication.
- Perform Matrix Multiplication using Strassen's Matrix Multiplication.
- Compare the results.

Theory:

- **Recurrence**

A *recurrence relation* or *recurrence* is an equation that describes a function in terms of its values on smaller inputs.

For example:

$F(n) = F(n-1) + F(n-2)$ $\qquad$ $F(0) = 0$ and $F(1) = 1$

Above recurrence is the definition of Fibonacci series.

Recurrence is also relevant to functions in programming. A recursive call to a function will also behave like how above equation calculates but the value of the function would be the number of times it is executed.

For Example:

```
MergeSort(A, p, r)
    if p < r
        mid = (p + r) / 2
        MergeSort(A, p, mid)
        MergeSort(A, mid + 1, r)
        Merge(A, p, mid, r)
```

The above recursive function can be written in function of Time T(n) as follows:

$$T(n) = T(n/2) + T(n/2) + n$$
$$T(n) = 2T(n/2) + n$$

We can use the following methods to solve recurrences:
1) Substitution Method
2) Recurrence Tree Method
3) Master's Method

- **Strassen's Matrix Multiplication**

Usually, to multiply 2x2 matrix we need 8 multiplications. In Strassen's Matrix Multiplication this can be done in 7 multiplications. When recursively applied, Strassen's Matrix multiplication performs better than normal matrix multiplication.

To perform Strassen's Matrix Multiplication:
1) Divide the input matrices A and B and output matrix C into n/2 x n/2 submatrices.
2) Compute 7 Matrices P1 to P7 using the equations given by Strassen.
3) Compute the desired submatrices C11; C12; C21; C22 of the result matrix C by adding and subtracting various combinations of the Pi matrices.

Equation for Strassen's Matrix Multiplication:

$S1 = B12 - B21$
$S2 = A11 - A12$
$S3 = A21 - A22$
$S4 = B21 - B11$
$S5 = A11 + A22$
$S6 = B11 + B22$
$S7 = A12 - A22$
$S8 = B21 + B22$
$S9 = A11 - A21$
$S10 = B11 + B12$

P1 = A11 . S1
P2 = S2 . B22
P3 = S3 . B11
P4 = A22 . S4
P5 = S5 . S6
P6 = S7 . S8
P7 = S9 . S10

C11 = P5 + P4 – P2 + P6
C12 = P1 + P2
C21 = P3 + P4
C22 = P5 + P1 – P3 – P7

Algorithm:

1) Normal Matrix Multiplication

```
NormalMatrixMultiplication(A, B, n)
    Let c be resultant matrix of size n x n
    For i = 1 to n
        for j = 1 to n
            c[i, j] = 0
            for k = 1 to n
                c[i, j] = c[i, j] + a[i, k] * b[k, j]
    return c
```

2) Strassen's Matrix Multiplication

Assuming that + and – with matrices refer to MatrixAddition and MatrixSubtraction,

```
SMM(A, B, n)
    Let c be new n x n matrix
    if (n == 1)
        c[0, 0] = A[0, 0] * B[0, 0]
        return c
    else
        k = n / 2
        Let A11, A12, A21, A22 and B11, B12, B21, B22 be k x k matrices

        P1 = SMM(A11, B12 – B22, k)
        P2 = SMM(A11 + A12, B22, k)
        P3 = SMM(A21 + A22, B11, k)
        P4 = SMM(A22, B21 – B11, k)
        P5 = SMM(A11 – A22, B11 – B22, k)
        P6 = SMM(A12 – A22, B21 + B22, k)
        P7 = SMM(A11 – A21, B11 + B12, k)

        c[1,1] = P5 + P4 – P2 + P6
        c[1,2] = P1 + P2
        c[2,1] = P3 + P4
        c[2,2] = P5 + P1 – P3 – P7

        return c
```

Code:

1) Strassen's Matrix Multiplication

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

long normal_mul_count = 0;
long strass_mul_count = 0;

bool is_power_of_two(int n) {
    if (n == 0) return true;
    return ceil(log2(n)) == floor(log2(n));
```

```
}

int** new_matrix(int n) {
    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
    }
    return m;
}

int** get_random_matrix(int n) {
    int** m = new_matrix(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            m[i][j] = rand() % 10;
        }
    }

    return m;
}

void print_matrix(int** m, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << left << setw(4) << m[i][j] << "  ";
        }
        cout << endl;
    }
}

int** normal_mm(int** a, int** b, int n) {
    int** c = new_matrix(n);

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

```
            c[i][j] = sum;
        }
    }

    return c;
}

// basic operation required for strassens

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

int** mat_sub(int** a, int** b, int n) {
    int** c = new_matrix(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = a[i][j] - b[i][j];
        }
    }
    return c;
}

int** strassens_mm(int** a, int** b, int n) {
    if (n == 1) {
        int** c = new_matrix(n);
        c[0][0] = a[0][0] * b[0][0];
        strass_mul_count++;

        return c;
    } else {
        int** c = new_matrix(n);
        int k = n / 2;

        // sub matrices initialization
```

```
    int** A11 = new_matrix(k);
    int** A12 = new_matrix(k);
    int** A21 = new_matrix(k);
    int** A22 = new_matrix(k);
    int** B11 = new_matrix(k);
    int** B12 = new_matrix(k);
    int** B21 = new_matrix(k);
    int** B22 = new_matrix(k);

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            A11[i][j] = a[i][j];
            A12[i][j] = a[i][k + j];
            A21[i][j] = a[k + i][j];
            A22[i][j] = a[k + i][k + j];
            B11[i][j] = b[i][j];
            B12[i][j] = b[i][k + j];
            B21[i][j] = b[k + i][j];
            B22[i][j] = b[k + i][k + j];
        }
    }

    // calculations

    int** P1 = strassens_mm(A11, mat_sub(B12, B22, k), k);
    int** P2 = strassens_mm(mat_add(A11, A12, k), B22, k);
    int** P3 = strassens_mm(mat_add(A21, A22, k), B11, k);
    int** P4 = strassens_mm(A22, mat_sub(B21, B11, k), k);
    int** P5 = strassens_mm(mat_add(A11, A22, k), mat_add(B11, B22, k), k);
    int** P6 = strassens_mm(mat_sub(A12, A22, k), mat_add(B21, B22, k), k);
    int** P7 = strassens_mm(mat_sub(A11, A21, k), mat_add(B11, B12, k), k);

    int** C11 = mat_sub(mat_add(mat_add(P5, P4, k), P6, k), P2, k);
    int** C12 = mat_add(P1, P2, k);
    int** C21 = mat_add(P3, P4, k);
    int** C22 = mat_sub(mat_sub(mat_add(P5, P1, k), P3, k), P7, k);

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            c[i][j] = C11[i][j];
            c[i][j + k] = C12[i][j];
```

```
                c[k + i][j] = C21[i][j];
                c[k + i][k + j] = C22[i][j];
            }
        }
        return c;
    }
}

int main() {
    int n;
    int** a;
    int** b;

    cout << "Enter matrix dimension: ";
    cin >> n;

    if (!is_power_of_two(n)) {
        cout << "The order of matrix must be a power of 2!\n";
        exit(EXIT_FAILURE);
    }

    cout << "\nGenerating random matrix A: \n";
    a = get_random_matrix(n);
    print_matrix(a, n);

    cout << "\nGenerating random matrix B: \n";
    b = get_random_matrix(n);
    print_matrix(b, n);

    int** c_n = normal_mm(a, b, n);

    cout << "\nResultant Matrix AB using normal multiplication: \n";
    print_matrix(c_n, n);

    cout << "\nResultant Matrix AB using strassen's multiplication: \n";
    int** c_s = strassens_mm(a, b, n);
    print_matrix(c_s, n);

    cout << "\n\nMultiplication required for normal multiplication: "
         << normal_mul_count << endl;
    cout << "Multiplication required for strassens multiplication: "
```

```
         << strass_mul_count << endl;
    return 0;
}
```

2) Code for analysis Comparison

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

long normal_mul_count = 0;
long strass_mul_count = 0;

int** new_matrix(int n) {
    int** m = new int*[n];
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
    }
    return m;
}

int** get_random_matrix(int n) {
    int** m = new_matrix(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            m[i][j] = rand() % 10;
        }
    }

    return m;
}

int** normal_mm(int** a, int** b, int n) {
    int** c = new_matrix(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int sum = 0;
            for (int k = 0; k < n; k++) {
```

```
                sum += a[i][k] * b[k][j];
                normal_mul_count++;
            }
            c[i][j] = sum;
        }
    }

    return c;
}

// basic operation required for strassens
int** mat_add(int** a, int** b, int n) {
    int** c = new_matrix(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}

int** mat_sub(int** a, int** b, int n) {
    int** c = new_matrix(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = a[i][j] - b[i][j];
        }
    }
    return c;
}

int** strassens_mm(int** a, int** b, int n) {
    if (n == 1) {
        int** c = new_matrix(n);
        c[0][0] = a[0][0] * b[0][0];
        strass_mul_count++;

        return c;
    } else {
        int** c = new_matrix(n);
        int k = n / 2;
```

```
    // sub matrices initialization
    int** A11 = new_matrix(k);
    int** A12 = new_matrix(k);
    int** A21 = new_matrix(k);
    int** A22 = new_matrix(k);
    int** B11 = new_matrix(k);
    int** B12 = new_matrix(k);
    int** B21 = new_matrix(k);
    int** B22 = new_matrix(k);

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            A11[i][j] = a[i][j];
            A12[i][j] = a[i][k + j];
            A21[i][j] = a[k + i][j];
            A22[i][j] = a[k + i][k + j];
            B11[i][j] = b[i][j];
            B12[i][j] = b[i][k + j];
            B21[i][j] = b[k + i][j];
            B22[i][j] = b[k + i][k + j];
        }
    }

    // calculations

    int** P1 = strassens_mm(A11, mat_sub(B12, B22, k), k);
    int** P2 = strassens_mm(mat_add(A11, A12, k), B22, k);
    int** P3 = strassens_mm(mat_add(A21, A22, k), B11, k);
    int** P4 = strassens_mm(A22, mat_sub(B21, B11, k), k);
    int** P5 = strassens_mm(mat_add(A11, A22, k), mat_add(B11, B22, k), k);
    int** P6 = strassens_mm(mat_sub(A12, A22, k), mat_add(B21, B22, k), k);
    int** P7 = strassens_mm(mat_sub(A11, A21, k), mat_add(B11, B12, k), k);

    int** C11 = mat_sub(mat_add(mat_add(P5, P4, k), P6, k), P2, k);
    int** C12 = mat_add(P1, P2, k);
    int** C21 = mat_add(P3, P4, k);
    int** C22 = mat_sub(mat_sub(mat_add(P5, P1, k), P3, k), P7, k);

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

```
                c[i][j] = C11[i][j];
                c[i][j + k] = C12[i][j];
                c[k + i][j] = C21[i][j];
                c[k + i][k + j] = C22[i][j];
            }
        }
        return c;
    }
}

int main() {
    // sequentially increase n with respect to power of 2
    // store the number of multiplication required in csv file
    int n = 1;
    ofstream fout("../csv/multiplication_analysis.csv");

    fout << "n,normal,strassens\n";

    cout << "Starting analysis!\n";

    for (int i = 0; i < 9; i++) {
        normal_mul_count = 0;
        strass_mul_count = 0;

        cout << "Calculating multiplication of order " << n << endl;

        int** a = get_random_matrix(n);
        int** b = get_random_matrix(n);

        normal_mm(a, b, n);
        strassens_mm(a, b, n);

        fout << n << "," << normal_mul_count << "," << strass_mul_count << "\n";
        n = n * 2;
    }

    cout << "\nAnalysis Data stored in csv/multiplication_analysis.csv\n";

    return 0;
}
```

Output:

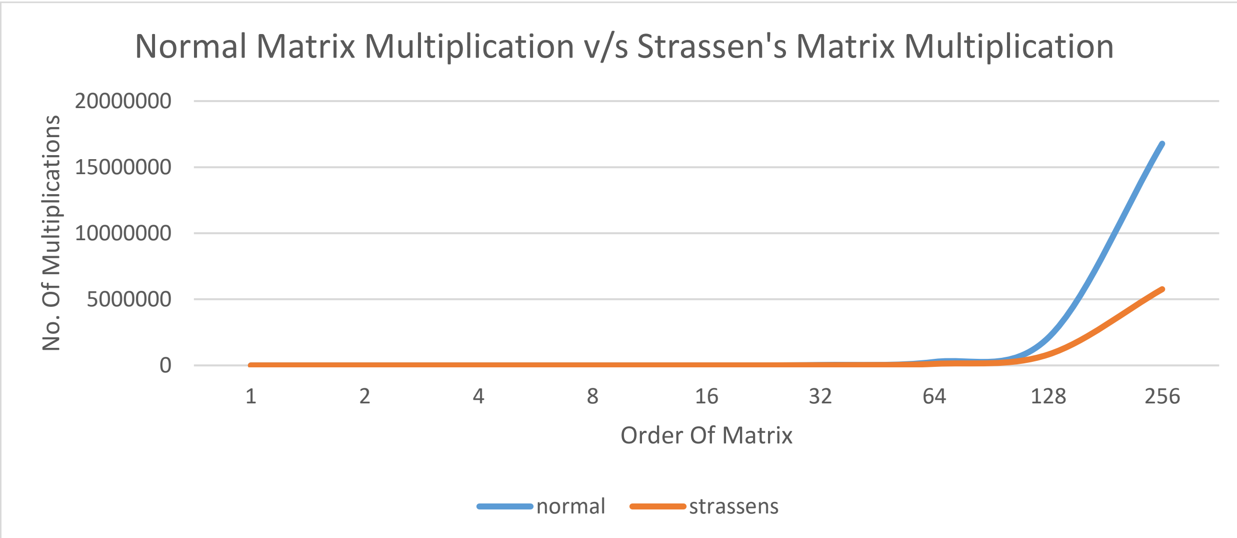
1) Strassen's Matrix Multiplication

```
PS D:\Tejas\clg\daa\Experiment 03\code> g++ .\strassens.cpp
PS D:\Tejas\clg\daa\Experiment 03\code> ./a
Enter matrix dimension: 4

Generating random matrix A:
1     7     4     0
9     4     8     8
2     4     5     5
1     7     1     1

Generating random matrix B:
5     2     7     6
1     4     2     3
2     2     1     6
8     5     7     6

Resultant Matrix AB using normal multiplication:
20    38    25    51
129   90    135   162
64    55    62    84
22    37    29    39

Resultant Matrix AB using strassen's multiplication:
20    38    25    51
129   90    135   162
64    55    62    84
22    37    29    39


Multiplication required for normal multiplication: 64
Multiplication required for strassens multiplication: 49
```

2) Analysis data for Strassen's Matrix Multiplication

```
PS D:\Tejas\clg\daa\Experiment 03\code> g++ .\strass_analysis.cpp
PS D:\Tejas\clg\daa\Experiment 03\code> ./a
Starting analysis!
Calculating multiplication of order 1
Calculating multiplication of order 2
Calculating multiplication of order 4
Calculating multiplication of order 8
Calculating multiplication of order 16
Calculating multiplication of order 32
Calculating multiplication of order 64
Calculating multiplication of order 128
Calculating multiplication of order 256

Analysis Data stored in csv/multiplication_analysis.csv
PS D:\Tejas\clg\daa\Experiment 03\code>
```

Chart:



Observations:

1) Initially both algorithms take almost the same number of multiplications to carry out the operation
2) However as value of n i.e Order of Matrix increases, Normal matrix multiplication starts taking more and more multiplications
3) It is evident especially after order of matrix exceeds 128
4) Normal matrix multiplication can be implemented iteratively, but Strassen's needs to be implemented recursively thus requiring call stack.
5) Strassen's algorithm require more number of intermediate matrices

Analysis:

1) Normal Matrix Multiplication
   - Normal Matrix Multiplication consists of three nested for loop of size n
   - Therefore its time complexity in all cases can be derived as

   $T(n) = O(n^3)$

2) Strassen's Matrix Multiplication

The recurrence relation of Strassen's matrix multiplication is:

$T(n) = 7T(n/2) + n^2$

For all cases.

Using Master's Method to solve (first case),

$T(n) = O(n^{\log 7})$

Conclusion:

After conducting this experiment, I have learnt the implementation of Strassen's Matrix Multiplication. I have also learnt to compute the asymptotic bounds of Strassen's Matrix Multiplication. I conclude that if a given recurrence is in form *T(n) = aT(n/b) + f(n),* decrease in value of *a* can cause massive improvement in performance.