**Name**: Ravi Jamanbhai Makwana

**Roll No**: 21BCP418

Div-6, G-12

**Assignment-5**

**Aim:**

Multiplication of Matrices(nxn) by using the Divide and Conquer Algorithm.

**Theory:**

Matrix multiplication is based on a divide and conquer-based approach. Here we divide our matrix into a smaller square matrix, solve that smaller square matrix and merge into larger results. For larger matrices this approach will continue until we recurse all the smaller sub matrices.

**Algorithm:**

1. DIVIDE the problem into a number of subproblems that are smaller instances of the same problem.
2. CONQUER the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.

COMBINE the solutions to the subproblems into the solution for the original problem.

**Program:**

import java.io.\*;

import java.util.\*;

class Main {

static int ROW\_1 = 4,COL\_1 = 4, ROW\_2 = 4, COL\_2 = 4; public static void printMat(int[][] a, int r, int c){ for(int i=0;i<r;i++){ for(int j=0;j<c;j++){

System.out.print(a[i][j]+" ");

}

System.out.println(""); }

System.out.println("");

} public static void print(String display, int[][] matrix,int start\_row, int start\_column, int end\_row,int end\_column) {

System.out.println(display + " =>\n");

for (int i = start\_row; i <= end\_row; i++) { for (int j = start\_column; j <= end\_column; j++) {

System.out.print(matrix[i][j]+" ");

}

System.out.println(""); }

System.out.println("");

}

public static void add\_matrix(int[][] matrix\_A,int[][] matrix\_B,int[][] matrix\_C, int split\_index)

{ for (int i = 0; i < split\_index; i++){ for (int j = 0; j < split\_index; j++){

matrix\_C[i][j] = matrix\_A[i][j] + matrix\_B[i][j];

}

} }

public static void initWithZeros(int a[][], int r, int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

a[i][j]=0;

}

} }

public static int[][] multiply\_matrix(int[][] matrix\_A,int[][] matrix\_B)

{ int col\_1 = matrix\_A[0].length; int row\_1 = matrix\_A.length; int col\_2 = matrix\_B[0].length; int row\_2 = matrix\_B.length; if (col\_1 != row\_2)

{

System.out.println("\nError: The number of columns in Matrix A must be equal to the number of rows in Matrix B\n"); int temp[][] = new int[1][1];

temp[0][0]=0;

return temp;

}

int[] result\_matrix\_row = new int[col\_2]; Arrays.fill(result\_matrix\_row,0);

int[][] result\_matrix = new int[row\_1][col\_2]; initWithZeros(result\_matrix,row\_1,col\_2); if (col\_1 == 1){

result\_matrix[0][0] = matrix\_A[0][0] \* matrix\_B[0][0];

} else {

int split\_index = col\_1 / 2;

int[] row\_vector = new int[split\_index]; Arrays.fill(row\_vector,0); int[][] result\_matrix\_00 = new int[split\_index][split\_index]; int[][] result\_matrix\_01 = new int[split\_index][split\_index]; int[][] result\_matrix\_10 = new int[split\_index][split\_index]; int[][] result\_matrix\_11 = new int[split\_index][split\_index];

initWithZeros(result\_matrix\_00,split\_index,split\_index); initWithZeros(result\_matrix\_01,split\_index,split\_index); initWithZeros(result\_matrix\_10,split\_index,split\_index); initWithZeros(result\_matrix\_11,split\_index,split\_index); int[][] a00 = new int[split\_index][split\_index]; int[][] a01 = new int[split\_index][split\_index]; int[][] a10 = new int[split\_index][split\_index]; int[][] a11 = new int[split\_index][split\_index]; int[][] b00 = new int[split\_index][split\_index]; int[][] b01 = new int[split\_index][split\_index]; int[][] b10 = new int[split\_index][split\_index]; int[][] b11 = new int[split\_index][split\_index]; initWithZeros(a00,split\_index,split\_index); initWithZeros(a01,split\_index,split\_index); initWithZeros(a10,split\_index,split\_index); initWithZeros(a11,split\_index,split\_index); initWithZeros(b00,split\_index,split\_index); initWithZeros(b01,split\_index,split\_index); initWithZeros(b10,split\_index,split\_index);

for (int i = 0; i < split\_index; i++)

{

for (int 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 (int i = 0; i < split\_index; i++)

{

for (int 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];

}

} }

return result\_matrix;

}

public static void main (String[] args)

{

int[][] matrix\_A = { { 1, 2, 3, 4 },

{ 1, 2, 3, 4 },

{ 1, 2, 3, 4 },

{ 1, 2, 3, 4 } };

System.out.println("Array A =>"); printMat(matrix\_A,4,4);

int[][] matrix\_B = { { 1, 2, 3, 4 }, { 1, 2, 3, 4 },

{ 1, 2, 3, 4 },

{ 1, 2, 3, 4 } }; System.out.println("Array B =>"); printMat(matrix\_B,4,4);

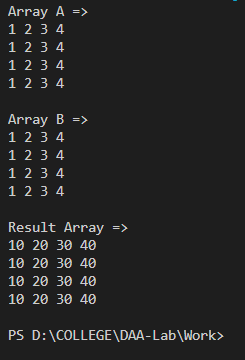
int[][] result\_matrix = multiply\_matrix(matrix\_A, matrix\_B);

System.out.println("Result Array =>"); printMat(result\_matrix,4,4);

}

}

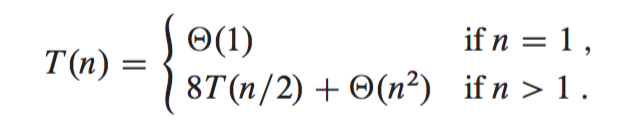
**Output:**



**Analysis:**

**Time Complexity Analysis:**

For multiplying two matrices of size n x n, we make 8 recursive calls above, each on a matrix/subproblem with size n/2 x n/2. Each of these recursive calls multiplies two n/2 x n/2 matrices, which are then added together. For the addition, we add two matrices of size n2/4, so each addition takes Θ(n2/4) time. We can write this recurrence in the form of the following equations:



The master theorem is a way of figuring out the runtime complexity of algorithms that use the divide and conquer approach, where subproblems are of equal size.

For our block partitioning approach, we saw that we had 8 recursive calls (a=8), where each subproblem was of size n/2 x n/2 (b=2). Outside of the recursive calls, we were performing additions that were of the order n2/4 since each quadrant matrix had those many entries. So we were doing work of the order of Θ(n2) outside of the recursive calls (d=2). This corresponds to case 3 of the master theorem since a(8) is more than b2 (22 = 4).

Using the master theorem, we can say that the runtime complexity is big O(nlogba), which is big O(nlog28) or big O(n3). This is no better than the straightforward iterative algorithm.