# Assignment 3: Empirical Analysis of Matrix Multiplication Algorithms

**Team:**

1. Shivani Bhalchandra
2. Abhishek Dongare
3. Prathyusha Vadlamannati

## Traditional Matrix Multiplication

**STUDY:**

The traditional Matrix Multiplication is that if Z = XY for an n × m matrix X and an m × p matrix Y, then Z is an n × p matrix with entries

Two matrices can be multiplied only if the no. of rows of the 1st matrix is equal to the no. of columns of the 2nd matrix.
From this, a simple algorithm can be constructed which loops over the indices i from 1 through n and k from 1 through p, computing the above using a nested loop:

1) Input: matrices X and Y of size n*n
2) Let Z be the resultant matrix of the same size n
3) For i ->from 1 to n:
- For j -> from 1 to n:
- Let result = 0
- For k -> from 1 to n:
- Set result ← result + Xik × Ykj
- Set Zij ← result ( This gives a resultant matrix of size n*n )
- Return Zij

5) The time taken to multiply 2 matrices of size n*n in traditional method is O(pow(n,3)) as there are 3 for loops which runs for n times.
6) 8 multiplications are required to calculate the Z(i,j) matrices.

So we have adopted Divide and Conquer strategy for solving this problem

**OBSERVATION:**

```
run:
first matrix
    1    2    3    4    4    7    6
    4    2    5    1  -12   -8   -3
    7    2    9    0   32    6    2
   12   45   61    8   33   66   89
    4   -2    5   10    3  -44   -1
    4    2    5    1  -12   -8   -3
    7    2    9    0   32    6    2

 second  matrix
    1    2    3    4    4    7    2
    4    2    5    1  -12   -8   -4
    7    2    9    0   32    6    0
   12   45   61    8   33   66    8
    4   -2    5   10    3  -44  -23
    4    2    5    1  -12   -8   -4
    7    2    9    0   32    6    1

Matrix multiplication result using traditional method:
Time Taken: 0 ns

  164  210  393   85  328   77  -88
  -42   69    1 -102  149  682  313
  244  -12  320  356  380 -1357 -752
 1734  840 2594  553 3879 -828 -1026
  -20  368  443   80 1035  948  202
  -42   69    1 -102  149  682  313
  244  -12  320  356  380 -1357 -752
```

## ***Pure Strassen***

**STUDY:**

Pure Strassen's Matrix Multiplication:

Pure Strassen's Matrix Multiplication works on the concept of divide and conquer where a matrix size of M x M is divided into multiple matrices of the size of order 2.

Every matrix is then computed and merged with the other matrices.

We iterate this division process n times (recursively) until the submatrices degenerate into numbers (elements of the ring R). The resulting product will be padded with zeroes just like A and B, and should be stripped of the corresponding rows and columns.

Practical implementations of Strassen's algorithm switch to standard methods of matrix multiplication for small enough submatrices, for which those algorithms are more efficient. The particular crossover point for which Strassen's algorithm is more efficient depends on the specific implementation and hardware. Earlier authors had estimated that Strassen's algorithm is faster for matrices with widths from 32 to 128 for optimized implementations.

Let *A*, *B* be two square matrices over a ring *R*. We want to calculate the matrix product *C* as

$$\mathbf{C} = \mathbf{AB} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n \times 2^n}$$

If the matrices *A*, *B* are not of type $2^n \times 2^n$ we fill the missing rows and columns with zeros.

We partition *A*, *B* and *C* into equally sized block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

with

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in R^{2^{n-1} \times 2^{n-1}}.$$

The naive algorithm would be:

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

With this construction we have not reduced the number of multiplications. We still need 8 multiplications to calculate the $C_{i,j}$ matrices, the same number of multiplications we need when using standard matrix multiplication.

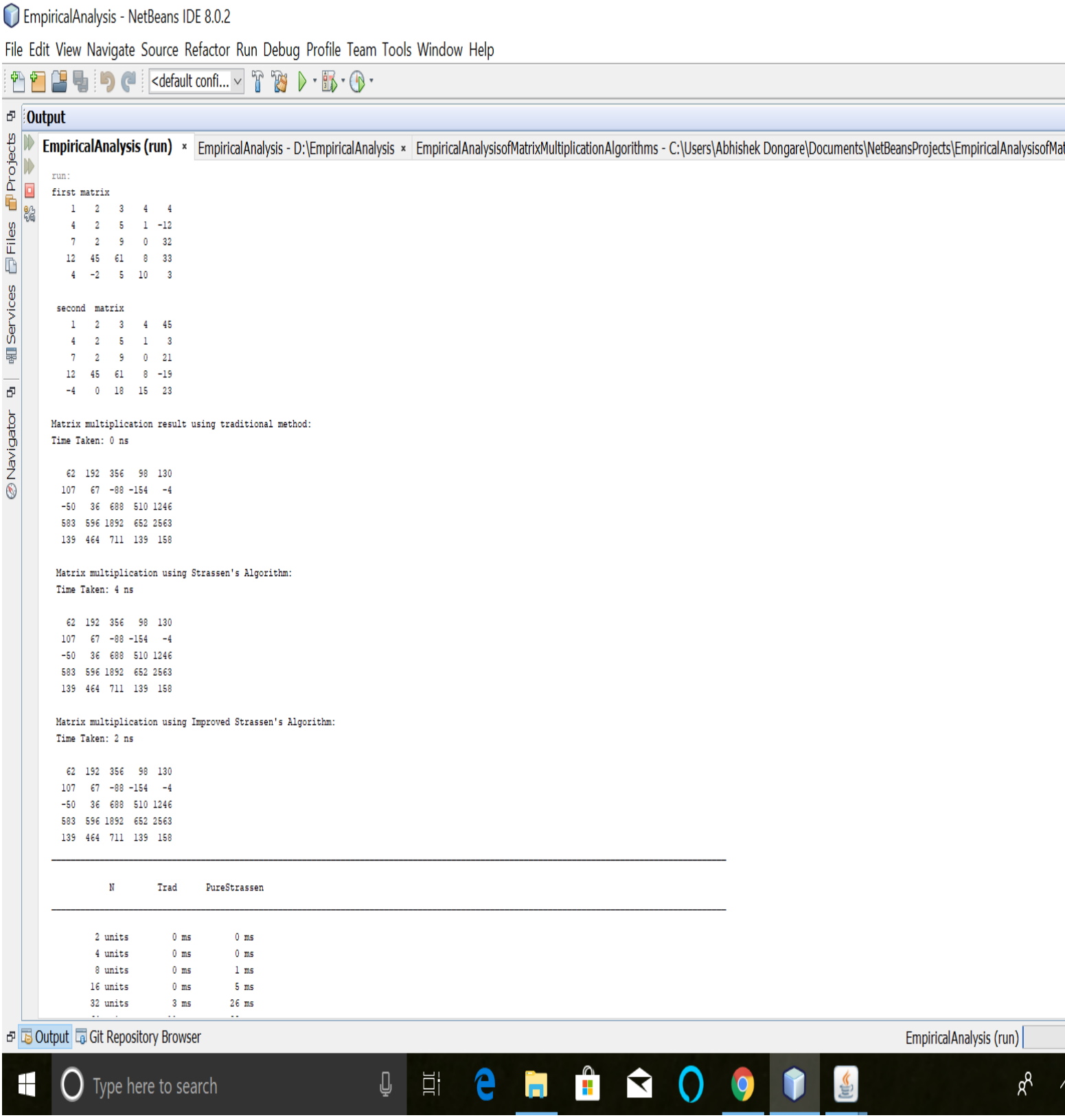
The Strassen algorithm defines instead new matrices:

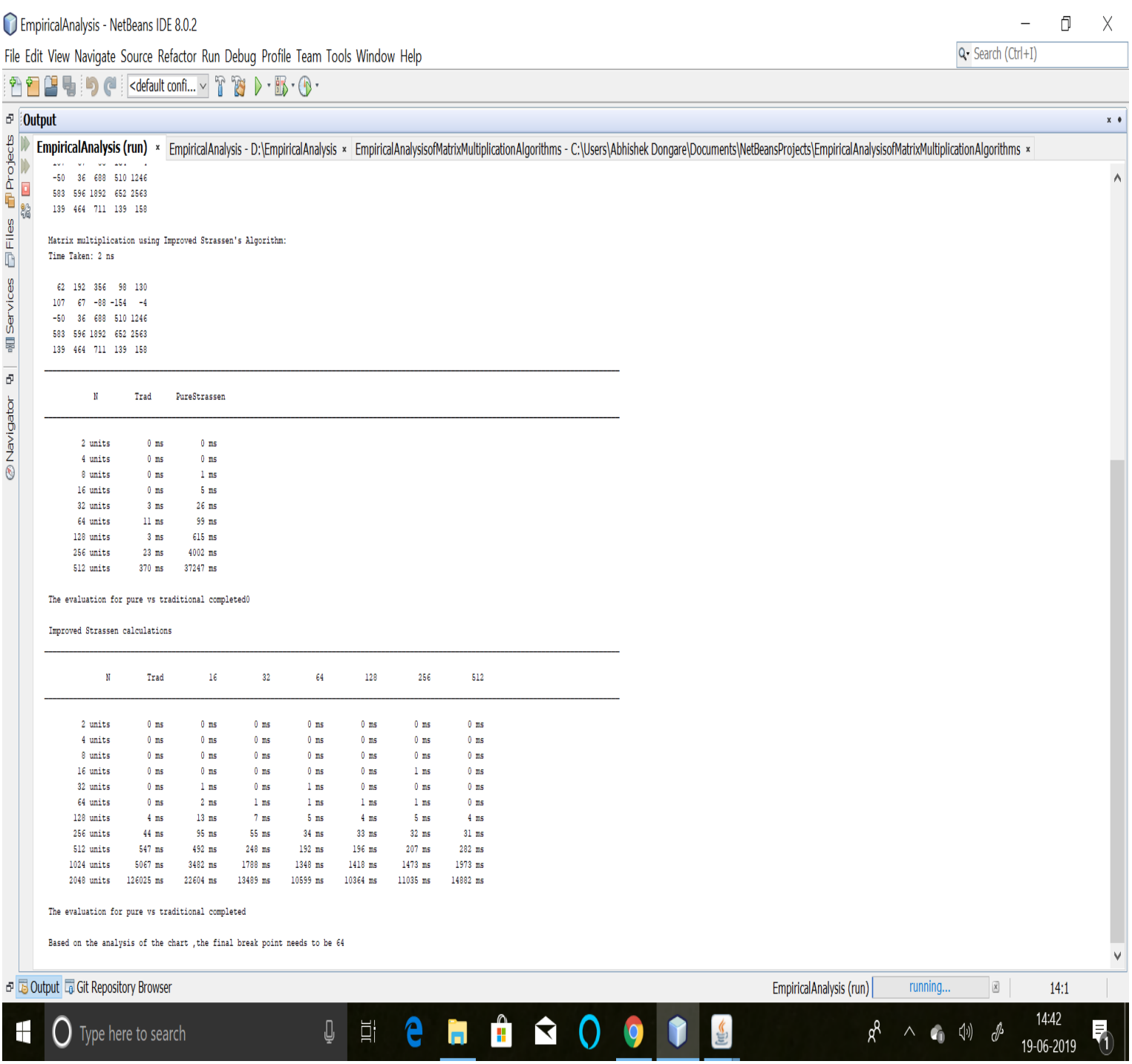
$$\begin{aligned} \mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\ \mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\ \mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

only using 7 multiplications (one for each $M_k$) instead of 8. We may now express the $C_{i,j}$ in terms of $M_k$:

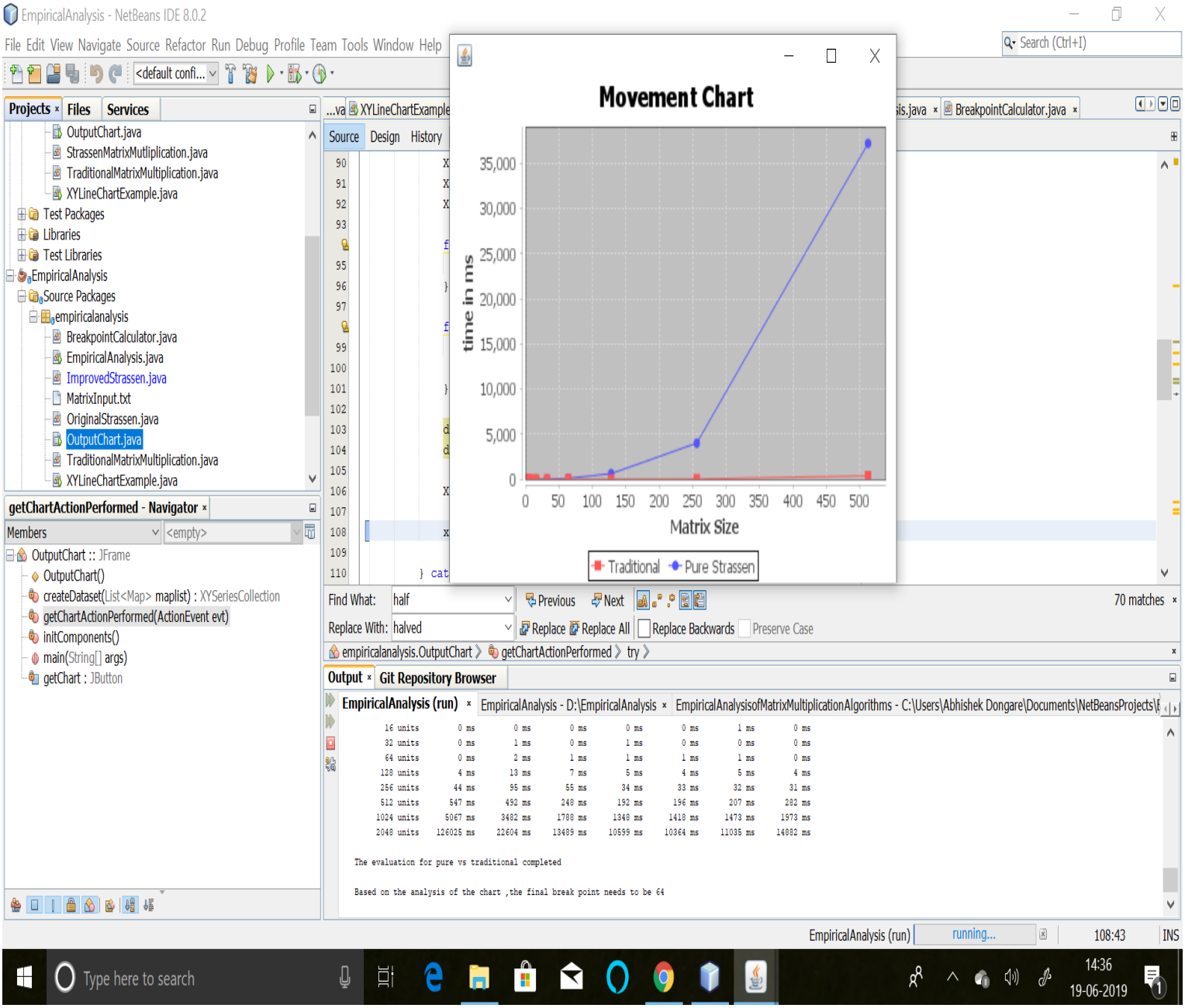
$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{aligned}$$

**OBSERVATION:**




EmpiricalAnalysis - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default confi...
Output
EmpiricalAnalysis (run)
EmpiricalAnalysis - D:\EmpiricalAnalysis
EmpiricalAnalysisofMatrixMultiplicationAlgorithms - C:\Users\Abhishek Dongare\Documents\NetBeansProjects\EmpiricalAnalysisofMatrixMultiplicationAlgorithms
-50 36 688 510 1246
583 596 1892 652 2563
139 464 711 139 158
Matrix multiplication using Improved Strassen's Algorithm:
Time Taken: 2 ns
62 192 356 98 130
107 67 -88 -154 -4
-50 36 688 510 1246
583 596 1892 652 2563
139 464 711 139 158
N Trad PureStrassen
2 units 0 ms 0 ms
4 units 0 ms 0 ms
8 units 0 ms 1 ms
16 units 0 ms 5 ms
32 units 3 ms 26 ms
64 units 11 ms 99 ms
128 units 3 ms 615 ms
256 units 23 ms 4002 ms
512 units 370 ms 37247 ms
The evaluation for pure vs traditional completed0
Improved Strassen calculations
N Trad 16 32 64 128 256 512
2 units 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms
4 units 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms
8 units 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms 0 ms
16 units 0 ms 0 ms 0 ms 0 ms 0 ms 1 ms 0 ms
32 units 0 ms 1 ms 0 ms 1 ms 0 ms 0 ms 0 ms
64 units 0 ms 2 ms 1 ms 1 ms 1 ms 1 ms 0 ms
128 units 4 ms 13 ms 7 ms 5 ms 4 ms 5 ms 4 ms
256 units 44 ms 95 ms 55 ms 34 ms 33 ms 32 ms 31 ms
512 units 547 ms 492 ms 248 ms 192 ms 196 ms 207 ms 282 ms
1024 units 5067 ms 3482 ms 1788 ms 1348 ms 1418 ms 1473 ms 1973 ms
2048 units 126025 ms 22604 ms 13489 ms 10599 ms 10364 ms 11035 ms 14882 ms
The evaluation for pure vs traditional completed
Based on the analysis of the chart ,the final break point needs to be 64
Output
Git Repository Browser
EmpiricalAnalysis (run)
running...
14:1
Type here to search
14:42
19-06-2019

**ANALYSIS:**

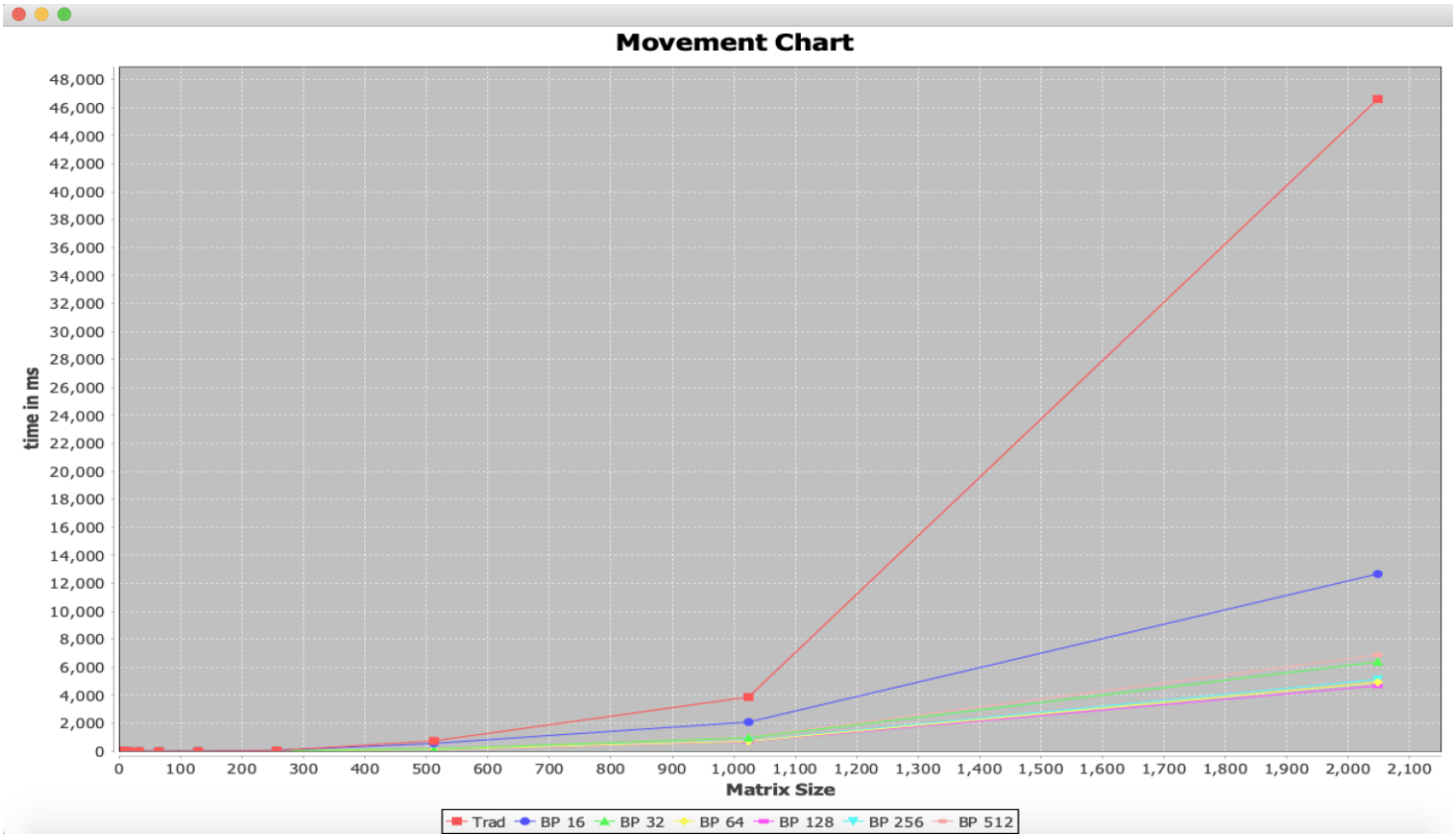



After analyzing the above output, it can be seen that Strassen's matrix multiplication does not work efficiently for some matrices of specific size. Hence in order to improve the Strassen's matrix multiplication efficiency, we designed an improved version of the Strassen's matrix multiplication algorithm.

***Improved Strassen:***

**ANALYSIS:**

The following JFreeChart shows the time taken by Traditional Matrix Multiplication method and time taken by our improved algorithm.




**OBSERVATION:**

In the improved Strassen we have used a value of breakpoint to optimize the performance of Pure Strassen algorithm. This breakpoint is the size of the matrix after which the Strassen breakdown stops, and the matrices are multiplied using traditional multiplication method. The above graph shows that the Traditional takes very high time as compared to all the breakpoint condition.

For the purpose of empirical analysis we assumed the starting breakpoint as 16 and we multiplied it by a factor of 2 ( in order to maintain the even sizes of the matrix). The result of multiplying increased sizes of matrix starting from 2*2 to 2048*2048 matrices against different break point conditions are clearly seen in this graph.

From the graph we can see that breakpoint 64 and breakpoint 128 have the quickest solution times. The timing outputs are almost similar. We verified this by printing the following values in the output console:

Improved Strassen calculations

| N | Trad | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| 2 units | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| 4 units | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| 8 units | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| 16 units | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms | 0 ms |
| 32 units | 0 ms | 1 ms | 0 ms | 0 ms | 1 ms | 0 ms | 0 ms |
| 64 units | 0 ms | 3 ms | 1 ms | 1 ms | 1 ms | 0 ms | 0 ms |
| 128 units | 2 ms | 14 ms | 6 ms | 4 ms | 4 ms | 4 ms | 3 ms |
| 256 units | 35 ms | 54 ms | 26 ms | 17 ms | 21 ms | 26 ms | 29 ms |
| 512 units | 723 ms | 544 ms | 147 ms | 110 ms | 103 ms | 108 ms | 150 ms |
| 1024 units | 3862 ms | 2072 ms | 955 ms | 701 ms | 682 ms | 726 ms | 979 ms |
| 2048 units | 46613 ms | 12661 ms | 6373 ms | 4917 ms | 4664 ms | 5120 ms | 6869 ms |

The evaluation for pure vs traditional completed

**CONCLUSION:**

We can see drastic improvement in the performance first in the breakpoint 64 table . Then the values remain almost similar for 128 and 256 bp. Hence from all above analysis and console output and calculations we can say that we get the best performance using smallest breakpoint: 64

***CODE:***

```
 /*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package empiricalanalysis;

import java.awt.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;
import java.util.Random;

/**
 *
 * @author shivani bhalchandra
 */
public class BreakpointCalculator {

    Map<Integer, Long> tradmap = new HashMap<Integer, Long>();
    Map<Integer, Long> pureStrasmap = new HashMap<Integer, Long>();
```

```
Map<Integer, Long> tradmap2 = new HashMap<Integer, Long>();
Map<Integer, Long> map16 = new HashMap<Integer, Long>();
Map<Integer, Long> map32 = new HashMap<Integer, Long>();
Map<Integer, Long> map64 = new HashMap<Integer, Long>();
Map<Integer, Long> map128 = new HashMap<Integer, Long>();
Map<Integer, Long> map256 = new HashMap<Integer, Long>();
Map<Integer, Long> map512 = new HashMap<Integer, Long>();

public Map<Integer, Long> getTradmap() {
   return tradmap;
}

public Map<Integer, Long> getTradmap2() {
   return tradmap2;
}
public Map<Integer, Long> getPureStrasmap() {
   return pureStrasmap;
}

//calculateBreakPoint
public int calculateBreakPoint() {

   TraditionalMatrixMultiplication tmm = new TraditionalMatrixMultiplication();

   OriginalStrassen ds = new OriginalStrassen();
   int size = 1;
   Random rand = new Random();

   String header2 = String.format("%13s", "N") + String.format("%13s", "Trad") + String.format("%18s",
      "PureStrassen");
   System.out.println(String.format("%140s\n", " ").replace(" ", "_"));
   System.out.println(header2);
   System.out.println(String.format("%140s\n", " ").replace(" ", "_"));

   while (size <= 256) {

      size *= 2;

      int M1[][] = new int[size][size];
      int M2[][] = new int[size][size];
      int res[][];
      for (int i = 0; i < M1.length; i++) {
         for (int j = 0; j < M1.length; j++) {
            M1[i][j] = rand.nextInt(200) + 100;
            M2[i][j] = rand.nextInt(200) + 100;
         }
      }

      String OutputValues = String.format("%10d units", size);
      long starttime1 = System.currentTimeMillis();
      if (size <= 2048) {

         res = tmm.MultiplyMatrixTraditionally(M1, M2);
         long stoptime1 = System.currentTimeMillis() - starttime1;
```

```
            String tradTime = String.format("%10d ms", stoptime1);
            tradmap.put(size, stoptime1);
            OutputValues += tradTime;
        }


        int requiredLength = extraLength(M1.length);
        if (requiredLength > M1.length) {
            M1 = applyPadding(M1, requiredLength);
            M2 = applyPadding(M2, requiredLength);
        }

        long starttime2 = System.currentTimeMillis();
        res = ds.SendMatrixForStrassen(M1, M2, size);
        long stoptime2 = System.currentTimeMillis() - starttime2;
        String time = String.format("%10d ms", stoptime2);
        pureStrasmap.put(size, stoptime2);
        OutputValues += time;
        System.out.println(OutputValues);
    }
    int breakPoint = 0;
    return breakPoint;

}

//table for pure strassen
public int impStrassenBP() {
    TraditionalMatrixMultiplication tmm = new TraditionalMatrixMultiplication();
    ImprovedStrassen imp = new ImprovedStrassen();

    Queue<Integer> bpQueue = new LinkedList<Integer>();
    int size = 1;
    Random rand = new Random();

    String header = String.format("%16s", "N") + String.format("%13s", "Trad") + String.format("%13s", "16")
        + String.format("%13s", "32") + String.format("%13s", "64") + String.format("%13s", "128") +
        String.format("%13s", "256") + String.format("%13s", "512");
    System.out.println(String.format("%140s\n", " ").replace(" ", "_"));
    System.out.println(header);
    System.out.println(String.format("%140s\n", " ").replace(" ", "_"));

    while (size <= 1024) {

        size *= 2;
        int M1[][] = new int[size][size];
        int M2[][] = new int[size][size];
        int res[][];
        for (int i = 0; i < M1.length; i++) {
            for (int j = 0; j < M1.length; j++) {
                M1[i][j] = rand.nextInt(200) + 100;
                M2[i][j] = rand.nextInt(200) + 100;
            }
        }
```

```
    String OutputValues = String.format("%10d units", size);
    long nanos = System.currentTimeMillis();
    if (size <= 2048) {
        res = tmm.MultiplyMatrixTraditionally(M1, M2);
        long t1 = System.currentTimeMillis() - nanos;
        String tradTime = String.format("%10d ms", t1);
        tradmap2.put(size, t1);
        OutputValues += tradTime;
    } else {
        String tradTime = String.format("%10s   ", "NM1");
        OutputValues += tradTime;
    }

    int assumedBreakPoint = 16;
    int requiredLength = extraLength(M1.length);
    if (requiredLength > M1.length) {
        M1 = applyPadding(M1, requiredLength);
        M2 = applyPadding(M2, requiredLength);
    }
    while (assumedBreakPoint < 1024) {
        nanos = System.currentTimeMillis();
        res = imp.SendMatrixForImprovedStrassen(M1, M2, size, assumedBreakPoint);
        long timeRequired = System.currentTimeMillis() - nanos;
        String time = String.format("%10d ms", timeRequired);

        switch (assumedBreakPoint) {

            case 16:
                map16.put(size, timeRequired);
                break;
            case 32:
                 map32.put(size, timeRequired);
                break;
            case 64:
                 map64.put(size, timeRequired);
                break;
            case 128:
                 map128.put(size, timeRequired);
                break;
            case 256:
                 map256.put(size, timeRequired);
                break;

            case 512:
                 map512.put(size, timeRequired);
                break;
            default:
        }
        OutputValues += time;
        assumedBreakPoint *= 2;
    }
    System.out.println(OutputValues);
}
int breakPoint = 0;
```

```
    return breakPoint;

}

// table for improved stassen
//to make a matrix into even x even
public int[][] applyPadding(int[][] Given, int size) {
    int NewM[][] = new int[size][size];

    for (int i = 0; i < NewM.length; i++) {
        for (int j = 0; j < NewM.length; j++) {
            if (i < Given.length && j < Given.length) {
                NewM[i][j] = Given[i][j];
            } else {
                NewM[i][j] = 0;
            }
        }
    }
    return NewM;
}

public int extraLength(int currentLength) {
    int requiredLength = 0;
    int divisor = 1;
    while (currentLength % divisor != currentLength) {
        divisor *= 2;
        if (divisor == currentLength) {
            return divisor;
        }
    }
    requiredLength = divisor;
    return requiredLength;
}

public Map<Integer, Long> getMap16() {
    return map16;
}

public Map<Integer, Long> getMap32() {
    return map32;
}

public Map<Integer, Long> getMap64() {
    return map64;
}

public Map<Integer, Long> getMap128() {
    return map128;
}

public Map<Integer, Long> getMap256() {
    return map256;
}
```

```
    public Map<Integer, Long> getMap512() {
        return map512;
    }

}
```

---

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package empiricalanalysis;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

/**
 *
 * @author shivani bhalchandra
 */
public class EmpiricalAnalysis {

    /**
     * @param args the command line arguments
     */
    Map<Integer, Long> tradmap = new HashMap<>();

    Map<Integer, Long> pureStrasmap = new HashMap<Integer, Long>();

    Map<Integer, Long> tradmap2 = new HashMap<Integer, Long>();
    Map<Integer, Long> map16 = new HashMap<Integer, Long>();
    Map<Integer, Long> map32 = new HashMap<Integer, Long>();
    Map<Integer, Long> map64 = new HashMap<Integer, Long>();
    Map<Integer, Long> map128 = new HashMap<Integer, Long>();
    Map<Integer, Long> map256 = new HashMap<Integer, Long>();
    Map<Integer, Long> map512 = new HashMap<Integer, Long>();
    List<Map> maplist = new ArrayList<>();

    public Map<Integer, Long> getTradmap() {
        return tradmap;
    }

    public Map<Integer, Long> getPureStrasmap() {
        return pureStrasmap;
    }
```

```
public static void main(String[] args) {
    // TODO code application logic here
    EmpiricalAnalysis empAn = new EmpiricalAnalysis();
    //Arrays.fill(timeLead, 0);
    try {
        empAn.start();
    } catch (FileNotFoundException e) {

    }
}

public void start() throws FileNotFoundException {

    Scanner scanner = new Scanner(new File("MatrixInputExample.txt"));

    //accepted the first integer from the file for calculation of matrix size
    int size = scanner.nextInt();

    //create two n*n matrix
    int M1[][] = new int[size][size];
    int M2[][] = new int[size][size];
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            M1[i][j] = scanner.nextInt();
        }
    }
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            M2[i][j] = scanner.nextInt();
        }
    }
    scanner.close();

    //printing two input matrix
    System.out.println("first matrix");
    printMatrix(M1);
    System.out.println("\n second  matrix");
    printMatrix(M2);

    //calculate the result by traditional method and print output
    TraditionalMatrixMultiplication tmm = new TraditionalMatrixMultiplication();
    long startTime = System.currentTimeMillis();
    int result[][] = tmm.MultiplyMatrixTraditionally(M1, M2);
    System.out.println("\nMatrix multiplication result using traditional method:\nTime Taken: " +
      (System.currentTimeMillis() - startTime) + " ns\n");
    printMatrix(result);

    int required = extraLength(M1.length);
    //System.out.println("test " + M1.length);

    if (required > M1.length) {
        M1 = applyPadding(M1, required);
        M2 = applyPadding(M2, required);
    }
```

```
    //calculate pure strassen
    long startTime2 = System.currentTimeMillis();
    OriginalStrassen Ds = new OriginalStrassen();
    int res2[][] = Ds.SendMatrixForStrassen(M1, M2, result.length);
    long stoptime = System.currentTimeMillis() - startTime2;
    System.out.println("\n Matrix multiplication using Strassen's Algorithm: \n Time Taken: " + stoptime +
       " ns\n");
    printMatrix(res2);

    //improved strassen
    long startTime3 = System.currentTimeMillis();
    ImprovedStrassen imp = new ImprovedStrassen();
    int res3[][] = imp.SendMatrixForImprovedStrassen(M1, M2, result.length, -1);
    long stoptime3 = System.currentTimeMillis() - startTime3;
    System.out.println("\n Matrix multiplication using Improved Strassen's Algorithm: \n Time Taken: " +
       stoptime3 + " ns\n");
    printMatrix(res3);

    //calculating breakpoint for pure vs traditional
    BreakpointCalculator bpc = new BreakpointCalculator();
    int calulatedbreakPoint = bpc.calculateBreakPoint();
    tradmap = bpc.getTradmap();
    pureStrasmap = bpc.getPureStrasmap();
    System.out.println("\n The evaluation for pure vs traditional completed" + calulatedbreakPoint);

    //calculating breakpoint for improved vs traditional
    System.out.println("\n Improved Strassen calculations");
    int impbpc = bpc.impStrassenBP();
    maplist.add(tradmap2 = bpc.getTradmap2());
    maplist.add(map16 = bpc.getMap16());
    maplist.add(map32 = bpc.getMap32());
    maplist.add(map64 = bpc.getMap64());
    maplist.add(map128 = bpc.getMap128());
    maplist.add(map256 = bpc.getMap256());
    maplist.add(map512 = bpc.getMap512());
    System.out.println("\n The evaluation for pure vs traditional completed ");
    System.out.println("\n Based on the analysis of the chart ,the final break point needs to be 64");
}

//method to print the given matrix
public void printMatrix(int[][] M) {
    for (int i = 0; i < M.length; i++) {
        for (int j = 0; j < M.length; j++) {
            System.out.print(String.format(" %4d", M[i][j]));
        }
        System.out.print("\n");
    }
}

//to make a matrix into even x even
public int[][] applyPadding(int[][] Given, int size) {
    int NewM[][] = new int[size][size];
```

```
    for (int i = 0; i < NewM.length; i++) {
        for (int j = 0; j < NewM.length; j++) {
            if (i < Given.length && j < Given.length) {
                NewM[i][j] = Given[i][j];
            } else {
                NewM[i][j] = 0;
            }
        }
    }

    return NewM;
}

public int extraLength(int currentLength) {
    int requiredLength = 0;

    int divisor = 1;
    while (currentLength % divisor != currentLength) {
        divisor *= 2;
        if (divisor == currentLength) {
            return divisor;
        }
    }
    requiredLength = divisor;
    return requiredLength;
}

public Map<Integer, Long> getTradmap2() {
    return tradmap2;
}

public void setTradmap2(Map<Integer, Long> tradmap2) {
    this.tradmap2 = tradmap2;
}

public Map<Integer, Long> getMap16() {
    return map16;
}

public void setMap16(Map<Integer, Long> map16) {
    this.map16 = map16;
}

public Map<Integer, Long> getMap32() {
    return map32;
}

public void setMap32(Map<Integer, Long> map32) {
    this.map32 = map32;
}

public Map<Integer, Long> getMap64() {
    return map64;
}
```

```
    public void setMap64(Map<Integer, Long> map64) {
        this.map64 = map64;
    }

    public Map<Integer, Long> getMap128() {
        return map128;
    }

    public void setMap128(Map<Integer, Long> map128) {
        this.map128 = map128;
    }

    public Map<Integer, Long> getMap256() {
        return map256;
    }

    public void setMap256(Map<Integer, Long> map256) {
        this.map256 = map256;
    }

    public Map<Integer, Long> getMap512() {
        return map512;
    }

    public void setMap512(Map<Integer, Long> map512) {
        this.map512 = map512;
    }

    public List<Map> getMaplist() {
        return maplist;
    }

    public void setMaplist(List<Map> maplist) {
        this.maplist = maplist;
    }



}
```

---

```
/*
 * To change this license header, choose License Xeaders in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package empiricalanalysis;

/**
 *
 * @author Abhishek Dongare
 */
public class ImprovedStrassen{
```

```
public int[][] SendMatrixForImprovedStrassen(int[][] A, int[][] B, int orignalLength, int
      assumedBreakPoint) {
   int D[][] = new int[orignalLength][orignalLength];

   int C[][] = strassenImp(A, B, A.length / 2, assumedBreakPoint);
   for (int i = 0; i < D.length; i++) {
      for (int j = 0; j < D.length; j++) {
         D[i][j] = C[i][j];
      }
   }

   return D;
}

public int[][] strassenImp(int[][] M1, int[][] M2, int halvedLength, int asumedBreakPoint) {
   int C[][] = new int[M1.length][M1.length];

   if (halvedLength == 0) {
      C[0][0] = M1[0][0] * M2[0][0];
      return C;
   }

   int a[][] = new int[halvedLength][halvedLength];
   int d[][] = new int[halvedLength][halvedLength];

   int e[][] = new int[halvedLength][halvedLength];
   int h[][] = new int[halvedLength][halvedLength];

   int X1[][] = new int[halvedLength][halvedLength];
   int X2[][] = new int[halvedLength][halvedLength];
   int X3[][] = new int[halvedLength][halvedLength];
   int X4[][] = new int[halvedLength][halvedLength];
   int X5[][] = new int[halvedLength][halvedLength];
   int X6[][] = new int[halvedLength][halvedLength];
   int X7[][] = new int[halvedLength][halvedLength];
   int X8[][] = new int[halvedLength][halvedLength];
   int X9[][] = new int[halvedLength][halvedLength];
   int X10[][] = new int[halvedLength][halvedLength];

   for (int i = 0; i < halvedLength; i++) {
      for (int j = 0; j < halvedLength; j++) {
         a[i][j] = M1[i][j];
         d[i][j] = M1[halvedLength + i][halvedLength + j];

         e[i][j] = M2[i][j];
         h[i][j] = M2[halvedLength + i][halvedLength + j];

         X1[i][j] = M2[i][halvedLength + j] - M2[halvedLength + i][halvedLength + j];
         X2[i][j] = M1[i][j] + M1[i][halvedLength + j];
         X3[i][j] = M1[halvedLength + i][j] + M1[halvedLength + i][halvedLength + j];
         X4[i][j] = M2[halvedLength + i][j] - M2[i][j];//g-e
         X5[i][j] = M1[i][j] + M1[halvedLength + i][halvedLength + j];//a+d
         X6[i][j] = M2[i][j] + M2[halvedLength + i][halvedLength + j];//e + h
         X7[i][j] = M1[i][halvedLength + j] - M1[halvedLength + i][halvedLength + j];//b-d
```

```
            X8[i][j] = M2[halvedLength + i][j] + M2[halvedLength + i][halvedLength + j]; //g+h
            X9[i][j] = M1[i][j] - M1[halvedLength + i][j];//a-c
            X10[i][j] = M2[i][j] + M2[i][halvedLength + j];//e+f
        }
    }

    int Y1[][];// = strassen(a, X1, halvedLength/2);
    int Y2[][];// = strassen(X2, h, halvedLength/2);
    int Y3[][];// = strassen(X3, e, halvedLength/2);
    int Y4[][];// = strassen(d, X4, halvedLength/2);
    int Y5[][];// = strassen(X5, X6, halvedLength/2);
    int Y6[][];// = strassen(X7, X8, halvedLength/2);
    int Y7[][];// = strassen(X9, X10, halvedLength/2);

    if (M1.length <= asumedBreakPoint) {
        TraditionalMatrixMultiplication tmm = new TraditionalMatrixMultiplication();
        Y1 = tmm.MultiplyMatrixTraditionally(a, X1);
        Y2 = tmm.MultiplyMatrixTraditionally(X2, h);
        Y3 = tmm.MultiplyMatrixTraditionally(X3, e);
        Y4 = tmm.MultiplyMatrixTraditionally(d, X4);
        Y5 = tmm.MultiplyMatrixTraditionally(X5, X6);
        Y6 = tmm.MultiplyMatrixTraditionally(X7, X8);
        Y7 = tmm.MultiplyMatrixTraditionally(X9, X10);
    } else {
        Y1 = strassenImp(a, X1, halvedLength / 2, asumedBreakPoint);
        Y2 = strassenImp(X2, h, halvedLength / 2, asumedBreakPoint);
        Y3 = strassenImp(X3, e, halvedLength / 2, asumedBreakPoint);
        Y4 = strassenImp(d, X4, halvedLength / 2, asumedBreakPoint);
        Y5 = strassenImp(X5, X6, halvedLength / 2, asumedBreakPoint);
        Y6 = strassenImp(X7, X8, halvedLength / 2, asumedBreakPoint);
        Y7 = strassenImp(X9, X10, halvedLength / 2, asumedBreakPoint);
    }

    for (int i = 0; i < Y1.length; i++) {
        for (int j = 0; j < Y1.length; j++) {
            C[i][j] = Y5[i][j] + Y4[i][j] - Y2[i][j] + Y6[i][j];//C11[i][j];
            C[i][Y1.length + j] = Y1[i][j] + Y2[i][j];//C12[i][j];
            C[Y1.length + i][j] = Y3[i][j] + Y4[i][j];//C21[i][j];
            C[Y1.length + i][Y1.length + j] = Y1[i][j] + Y5[i][j] - Y3[i][j] - Y7[i][j];//C22[i][j];
        }
    }

    return C;

  }

}
```

---

```
package empiricalanalysis;

/**
 *
 * @author prathyusha
```

```
 */
public class OriginalStrassen {

   //This method will remove the extra padding we added to make the any given matrix into a factor of 2
   public int[][] SendMatrixForStrassen(int[][] A, int[][] B, int givenLength) {
      int D[][] = new int[givenLength][givenLength];

      int C[][] = strassenMultiplication(A, B);
      for (int i = 0; i < D.length; i++) {
         for (int j = 0; j < D.length; j++) {
            D[i][j] = C[i][j];
         }
      }

      return D;
   }

   // this method recursively divides the matrix by a factor of /2
   private static int[][] strassenMultiplication(int[][] A, int[][] B) {

      int halflen = A.length;

      int[][] result = new int[halflen][halflen];

      // if the input matrix is 1x1
      if (halflen == 1) {
         result[0][0] = A[0][0] * B[0][0];
      } else {

         // Dividing the length of first matrix into half recursively
         int[][] a = new int[halflen / 2][halflen / 2];
         int[][] b = new int[halflen / 2][halflen / 2];
         int[][] c = new int[halflen / 2][halflen / 2];
         int[][] d = new int[halflen / 2][halflen / 2];

         // Dividing the length of second matrix into half recursively
         int[][] e = new int[halflen / 2][halflen / 2];
         int[][] f = new int[halflen / 2][halflen / 2];
         int[][] g = new int[halflen / 2][halflen / 2];
         int[][] h = new int[halflen / 2][halflen / 2];

         // Dividing matrix A into 4 parts
         arrayDivision(A, a, 0, 0);
         arrayDivision(A, b, 0, halflen / 2);
         arrayDivision(A, c, halflen / 2, 0);
         arrayDivision(A, d, halflen / 2, halflen / 2);

         // Dividing matrix B into 4 parts
         arrayDivision(B, e, 0, 0);
         arrayDivision(B, f, 0, halflen / 2);
         arrayDivision(B, g, halflen / 2, 0);
         arrayDivision(B, h, halflen / 2, halflen / 2);

         /**
```

```
         * Equations to be formed: p1 = (a + d)(e + h) p2 = (c + d)e p3 =
         * a(f - h) p4 = d(g - e) p5 = (a + b)h p6 = (c - a) (e + f) p7 = (b
         * - d) (g + h)
         *
         */
        int[][] p1 = strassenMultiplication(matricesAddition(a, d), matricesAddition(e, h));
        int[][] p2 = strassenMultiplication(matricesAddition(c, d), e);
        int[][] p3 = strassenMultiplication(a, subMatrices(f, h));
        int[][] p4 = strassenMultiplication(d, subMatrices(g, e));
        int[][] p5 = strassenMultiplication(matricesAddition(a, b), h);
        int[][] p6 = strassenMultiplication(subMatrices(c, a), matricesAddition(e, f));
        int[][] p7 = strassenMultiplication(subMatrices(b, d), matricesAddition(g, h));

        /**
         * Operations to perform to get resultant matrix C11 = p1 + p4 - p5
         * + p7 C12 = p3 + p5 C21 = p2 + p4 C22 = p1 - p2 + p3 + p6
         *
         */
        int[][] C11 = matricesAddition(subMatrices(matricesAddition(p1, p4), p5), p7);
        int[][] C12 = matricesAddition(p3, p5);
        int[][] C21 = matricesAddition(p2, p4);
        int[][] C22 = matricesAddition(subMatrices(matricesAddition(p1, p3), p2), p6);

        // adding all subarray back into one
        copySubArray(C11, result, 0, 0);
        copySubArray(C12, result, 0, halflen / 2);
        copySubArray(C21, result, halflen / 2, 0);
        copySubArray(C22, result, halflen / 2, halflen / 2);
    }
    return result;
}

// Helper methods
// Adding 2 matrices
public static int[][] matricesAddition(int[][] a, int[][] b) {
    int n = a.length;
    int[][] result = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = a[i][j] + b[i][j];
        }
    }
    return result;
}

// Subtracting 2 matrices
public static int[][] subMatrices(int[][] a, int[][] b) {
    int len = a.length;
    int[][] result = new int[len][len];
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            result[i][j] = a[i][j] - b[i][j];
        }
    }
```

```
        return result;
    }

    // Divides the array
    public static void arrayDivision(int[][] P, int[][] C, int iB, int jB) {
        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++) {
            for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++) {
                C[i1][j1] = P[i2][j2];
            }
        }
    }

    // copies
    public static void copySubArray(int[][] C, int[][] P, int iB, int jB) {
        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++) {
            for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++) {
                P[i2][j2] = C[i1][j1];
            }
        }
    }

}
```

---

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package empiricalanalysis;


import java.io.FileNotFoundException;

import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;


import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

/**
 *
 * @author shivani bhalchandra
 */
public class OutputChart extends javax.swing.JFrame {

    /**
     * Creates new form OutputChart
     */
    public OutputChart() {
```

```
    initComponents();
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    getChart = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    getChart.setText("GetChart");
    getChart.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            getChartActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(60, 60, 60)
            .addComponent(getChart)
            .addContainerGap(559, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(56, 56, 56)
            .addComponent(getChart)
            .addContainerGap(240, Short.MAX_VALUE))
    );

    pack();
}// </editor-fold>

private void getChartActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    EmpiricalAnalysis emp = new EmpiricalAnalysis();
    try {
        //start the analysis
        emp.start();
        //data for each size of matrix multiplied traditionally
        Map<Integer, Long> tmap = emp.getTradmap();
        //System.out.println("tmap" + tmap.size());
```

```
        //data for each size of matrix multiplied using original strassen
        Map<Integer, Long> pureStrasmap = emp.getPureStrasmap();
        //System.out.println("pireStmap" + pureStrasmap.size());

        //all maps of diffrent breakpoints tested against all sizes of matrix
        //for first chart
        List<Map> outlist = emp.getMaplist();
        XYSeriesCollection dataset2 = createDataset(outlist);
        XYSeriesCollection dataset1 = new XYSeriesCollection();
        XYSeries series1 = new XYSeries("Traditional ");
        XYSeries series2 = new XYSeries("Pure Strassen");
        for (Map.Entry<Integer, Long> map : tmap.entrySet()) {
            series1.add(map.getKey(), map.getValue());
        }
        for (Map.Entry<Integer, Long> map : pureStrasmap.entrySet()) {
            series2.add(map.getKey(), map.getValue());
        }
        dataset1.addSeries(series1);
        dataset1.addSeries(series2);

        //for second chart
        XYLineChartExample xs = new XYLineChartExample();
        //outputing the chart by creating the dataset for different breakpointss
        xs.XYLineChartExample(dataset2);

    } catch (FileNotFoundException ex) {
        Logger.getLogger(OutputChart.class.getName()).log(Level.SEVERE, null, ex);
    }
}

//creating dataset for different breakpoints
private XYSeriesCollection createDataset(List<Map> maplist) {

    XYSeriesCollection dataset = new XYSeriesCollection();
    XYSeries[] serieslist = new XYSeries[10];
    int breappoint = 16;
    for (int i = 0; i < maplist.size(); i++) {
        if (i == 0) {
            serieslist[i] = new XYSeries("Trad");

        } else {
            serieslist[i] = new XYSeries("BP " + breappoint);
            breappoint *= 2;
        }
        Map<Integer, Long> outmap = maplist.get(i);
        for (Map.Entry<Integer, Long> map : outmap.entrySet()) {
            serieslist[i].add(map.getKey(), map.getValue());
        }
        dataset.addSeries(serieslist[i]);

    }

    return dataset;
```

```
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
            javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;

                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(OutputChart.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(OutputChart.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(OutputChart.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(OutputChart.class
                    .getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new OutputChart().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton getChart;
    // End of variables declaration
}
```

---

```
/*
 * To change this license header, choose License Headers in Project Properties.
```

```
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
Prathyusha
*/
package empiricalanalysis;

public class TraditionalMatrixMultiplication {

    //method to multiply two matrix using traditional method
    //O(N^2)
    public int[][] MultiplyMatrixTraditionally(int[][] m1, int[][] m2){
        int out[][] = new int[m1.length][m1.length];
        for(int i = 0; i < m1.length; i++){
            for(int j = 0; j < m1.length; j ++){
                for(int k = 0; k < m1.length; k++){
                    out[i][j] += m1[i][k]*m2[k][j];
                }
            }
        }
        return out;
    }
}
```

---

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package empiricalanalysis;

import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYSeriesCollection;

/**
 *
 * @author shivani bhalchandra
 */
public class XYLineChartExample extends JFrame {

    public void XYLineChartExample(XYSeriesCollection dataset) {
```

```
        JPanel chartPanel = createChartPanel(dataset);
        add(chartPanel, BorderLayout.CENTER);

        setSize(500, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);

    }

    private JPanel createChartPanel(XYSeriesCollection li) {
        // creates a line chart object
        // returns the chart panel

        String chartTitle = " Movement Chart";
        String xAxisLabel = "Matrix Size";
        String yAxisLabel = "time in ms";

        XYSeriesCollection dataset = li;
        // System.out.println("tesst count " + dataset.getSeriesCount());

        //XYDataset dataset = createDataset(li);
        JFreeChart chart = ChartFactory.createXYLineChart(chartTitle, xAxisLabel, yAxisLabel, dataset);
        XYPlot plot = chart.getXYPlot();
        XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
        plot.setRenderer(renderer);

        return new ChartPanel(chart);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new XYLineChartExample().setVisible(true);
            }
        });
    }

}
```

MatrixInputExample:

4

1 12 31 40 4
4 21 5 1 -1
71 2 19 0 32
12 45 61 8 33

11 2 3 4 35
4 21 15 1 3
7 2 94 30 21
12 45 1 82 -19

REFERENCES:


https://en.wikipedia.org/wiki/Strassen_algorithm
https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm
https://walkccc.github.io/CLRS/Chap04/4.2/
https://medium.com/human-in-a-machine-world/strassen-s-algorithm-for-matrix-multiplication-8aada6cda2fd
https://www.geeksforgeeks.org/strassens-matrix-multiplication/
https://www.geeksforgeeks.org/strassens-matrix-multiplication-algorithm-implementation/
http://www.java2s.com/Code/Java/Chart/JFreeChartLineChartDemo6.htm