Name: Pratyay Kumar
Date: 11/10/2022
Program Lab: 8

Program #8: Concurrency

Program Description:

- The program takes1 input from the user (or on the command line)
    1. The dimension of the matrix- N
- Your program will do the following
    2. Create an NxN two-dimension INTEGER matrix
    3. Randomly assign INTEGER values to each element between 2^(32-N) and 2^(31-N).
- START TIMER NOW
    4. You will create N threads; each thread is responsible for one row of the matrix
    5. Each thread will calculate the max, min, and what it needs to report for summation/average
    6. You should add a common set of arrays in the main object class to allow each thread to copy values back to the main thread.
    7. Your main thread will wait on all the children's threads and then calculate the overall max, Min, and Average
- STOP TIMER:
    8. You will need to calculate and report the following from the matrix
    9. The maximum value
    10. The minimum value
    11. The average of all of the values in the matrix
    12. The time it took to do parts a-c

Code: (Java):

```
/*
 * Author: Pratyay Kumar
 * Date:
```

```
 * Purpose: A simple concurrency program in JAVA.
 * Description:
               * The program takes1 input from the user (or on the command line)
                   1)  the dimension of the matrix- N
               * Your program will do the following
                   1)  Create an NxN two dimension INTEGER matrix
                   2)  Randomly assign INTEGER values to each element in the range of  between 2^(32-N) and 2^(31-N).
                   This is basically :    range * Random() + start_value
               * START TIMER NOW
                   3)  You will create N threads, each thread is responsible for one row of the matrix
                   4)  each thread will calculate the max, min and what it needs to report for summation/average
                   5)  You should add a common set of arrays in the main object class to allow each thread to copy
values back to the main thread.
                   6)  Your main thread will wait on all of the children threads, and then calculate the overall max,
Min and average
               * STOP TIMER
                   You will need to calculate  and report the following from the matrix
                   a)  The maximum value
                   b)  The minimum value
                   c)  The average of all of the values in the matrix
                   d)  the time it took to do parts a-c
 */

import java.util.*;

// Class concurrency
public class concurrency {
    // Constants
    public static final int BASE = 2;
    public static final int MAX_EXPONENT    = 32;
    public static final int MIN_EXPONENT    = 31;

    // Non-constants
    public static int[][] matrix;
```

```
    public static int size = 0;
    private static ArrayList<Thread> arrThreads = new ArrayList<Thread>();
    public static ArrayList<ThreadClass> arrData    = new ArrayList<ThreadClass>();

    // Randomly assign values to each element of matrix in the range 2^(31-N) and 2^(32-N)
    public static void InitializeMatrix() {
        // Calculate maximum range
        int max         = BASE;
        int maxLimit    = MAX_EXPONENT - size;
        for( int i = 1; i < maxLimit; i++ ) {
            max = max * BASE;
        }

        // Calculate minimum range
        int min         = BASE;
        int minLimit    = MIN_EXPONENT - size;

        for( int i = 1; i < minLimit; i++ ) {
            min = min * BASE;
        }

        System.out.println( "\nRange: ( " + min + " - " + max + " )" );

        // Initialize matrix with random numbers between min and max values.
        Random randomNumber = new Random();

        for( int i = 0; i < size; ++i ) {
            for( int j = 0; j < size; ++j ) {
                matrix[ i ][ j ] = randomNumber.nextInt(( max - min ) + 1 ) + min;
            }
        }
    }

    // Display the elements of matrix.
```

```
    public static void PrintMatrix() {
        System.out.println( "\n----- Matrix -----" );

        for( int i = 0; i < size; ++i ) {
            for( int j = 0; j < size; ++j ) {
                System.out.printf( "%10d", matrix[ i ][ j ] );
            }
            System.out.print( "\n");
        }
    }

    // Main function
    public static void main( String[] args ) {
        int matrixMin           = 0;
        int matrixMax           = 0;
        float matrixSum         = 0;
        float matrixAvg         = 0;
        long startTime          = 0;
        long endTime            = 0;
        long timeElapsed        = 0;

        try {
            if( 1 == args.length ) {
                // Parse argument to get matrix size.
                size = Integer.parseInt( args[ 0 ] );
                System.out.println( "\nMatrix size: " + size);

                // Create matrix of required size.
                matrix = new int[ size ][ size ];

                // Initialize and display matrix.
                InitializeMatrix();
                PrintMatrix();
```

```
                // Start the timer for statistics.
                 startTime = System.nanoTime();

                // Create size number of threads, each thread is responsible for one row of matrix
                for( int i = 0; i < size; i++ ) {
                    Thread threadObj = new Thread( new ThreadClass( i ) );
                    threadObj.start();

                    // Add threads to arraylist.
                    arrThreads.add( threadObj );
                }

                // This for loop will not stop execution of any thread, only it will come out when all threads are
executed.
                for( int i = 0; i < arrThreads.size(); i++ ) {
                    arrThreads.get( i ).join();
                }

                // Calculate matrix total minimum, maximum, and average using results from each thread
                matrixMin = arrData.get( 0 ).threadMin;
                matrixMax = arrData.get( 0 ).threadMax;
                matrixSum = arrData.get( 0 ).threadAvg;
                for( int i = 1; i < arrData.size(); i++ ) {
                    if( matrixMin > arrData.get( i ).threadMin ) {
                        matrixMin = arrData.get( i ).threadMin;
                    }
                    if( matrixMax < arrData.get( i ).threadMax ) {
                        matrixMax = arrData.get( i ).threadMax;
                    }
                    matrixSum = matrixSum + arrData.get( i ).threadAvg;
                }

                matrixAvg = matrixSum / size;
```

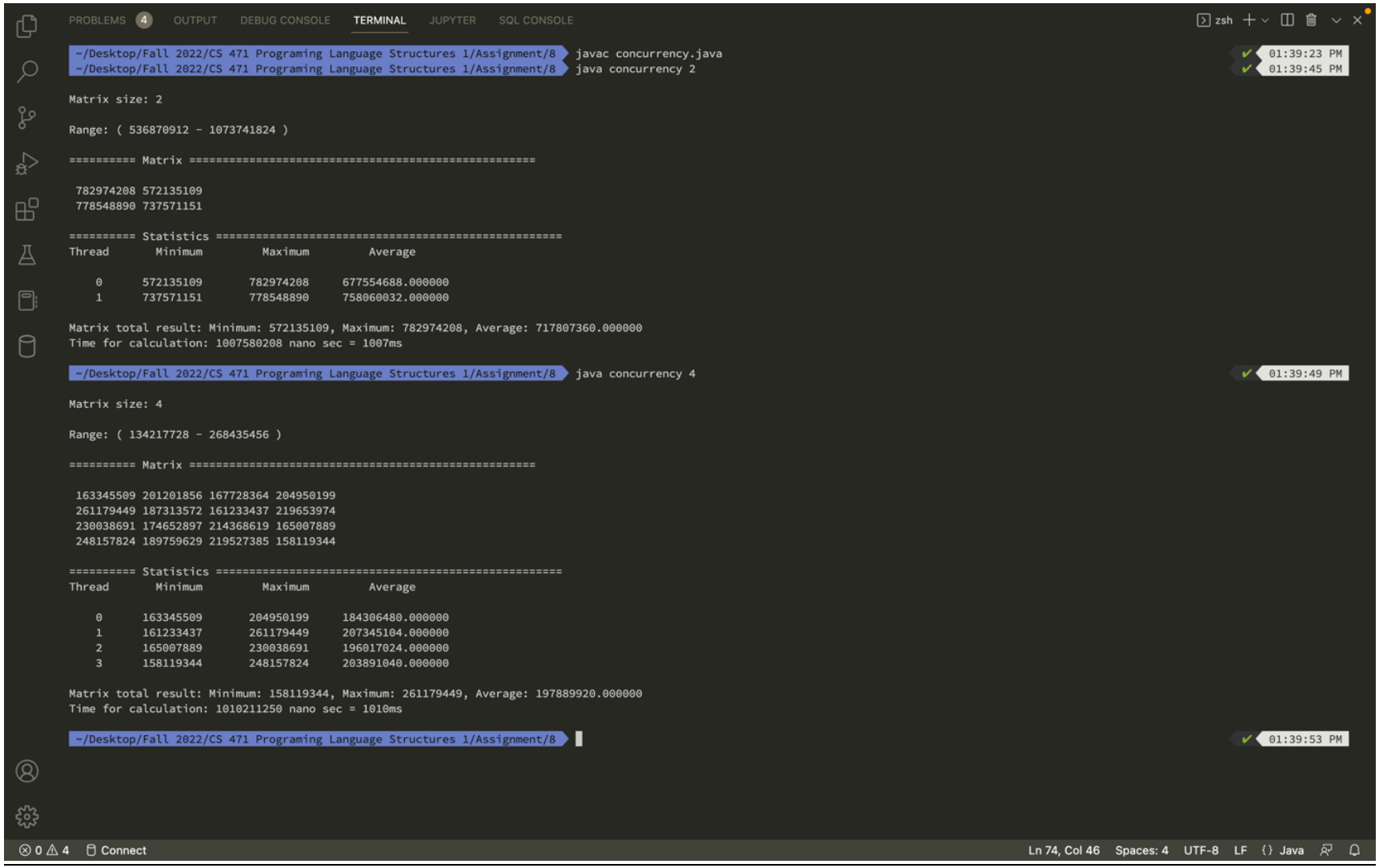
```
                // Stop the timer after calculation.
                endTime = System.nanoTime();
                timeElapsed = endTime - startTime;

                // Display statistics for each thread and then matrix total.
                System.out.println( "\n---------- Statistics ----------");
                System.out.printf( "\n%s %13s %15s %15s\n", "Thread", "Minimum", "Maximum", "Average" );


                for( int i = 0; i < arrData.size(); i++ ) {
                    System.out.printf( "\n%5d %14d %15d %20f",
                            arrData.get( i ).index,
                            arrData.get( i ).threadMin,
                            arrData.get( i ).threadMax,
                            arrData.get( i ).threadAvg );
                }

                System.out.printf( "\n\nMatrix total result: Minimum: %d\tMaximum: %d\tAverage: %f", matrixMin,
matrixMax, matrixAvg );

                System.out.println( "\nTime for calculation: " + timeElapsed + " nano sec" + " = " + timeElapsed /
1000000 + "ms\n" );
            }
            else {
                // If matrix size is not provided as command line argument, exit program.
                System.out.println( "Usage : java Concurrency <matrix size>" );
                System.exit( 1 );
            }
        }
        catch ( Exception e ) {
            // Catching exception
            System.out.println ( "Exception!!!!" );
        }
    }
```

```
}

// Class implementing the runnable interface
class ThreadClass implements Runnable {
    public int index;
    public int threadMin;
    public int threadMax;
    public float threadAvg;

    // Use the index as thread-id and index of matrix.
    ThreadClass( int tid ) {
        this.index       = tid;
        this.threadMin   = 0;
        this.threadMax   = 0;
        this.threadAvg   = 0;
    }

    //Override run() method
    public void run() {
        try {
            //System.out.println( "\nRunning Thread: " + index );

            threadMin = concurrency.matrix[ index ][ 0 ];
            threadMax = concurrency.matrix[ index ][ 0 ];
            int sum = 0;

            // Calculate row minimum, maximum and average.
            for( int i = 0; i < concurrency.size; i++ ) {
                if( threadMin > concurrency.matrix[ index ][ i ] ) {
                    threadMin = concurrency.matrix[ index ][ i ];
                }
                if( threadMax < concurrency.matrix[ index ][ i ] ) {
                    threadMax = concurrency.matrix[ index ][ i ];
                }
```

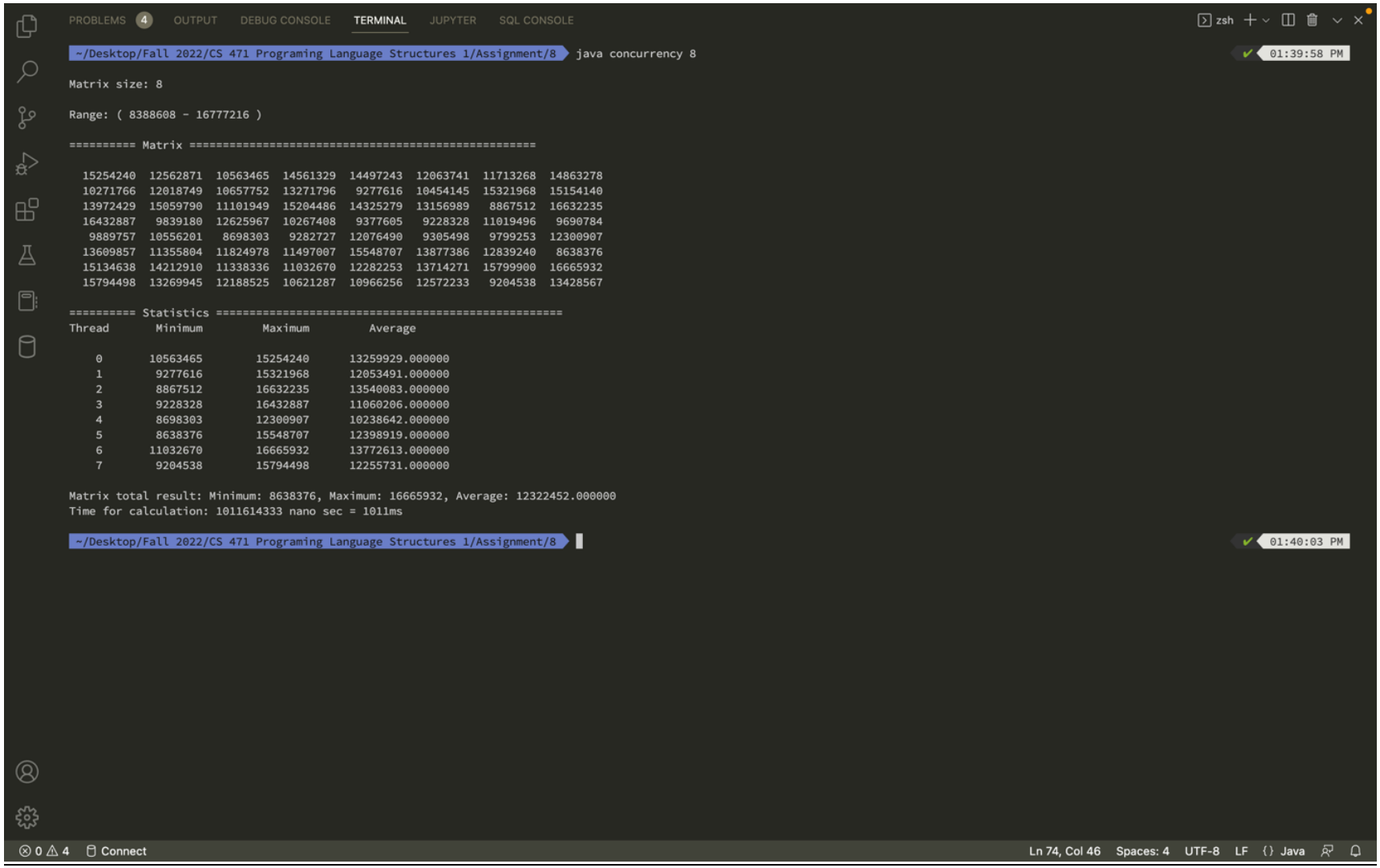
```
                sum = sum + concurrency.matrix[ index ][ i ];

            }
            threadAvg = sum / concurrency.size;

            // Store the data in array list.
            concurrency.arrData.add( this );

            Thread.sleep( 1000 );
        }
        catch ( Exception e ) {
            // Catching exception
            System.out.println ( "Exception!!!!" );
        }
    }
}
```

Output:

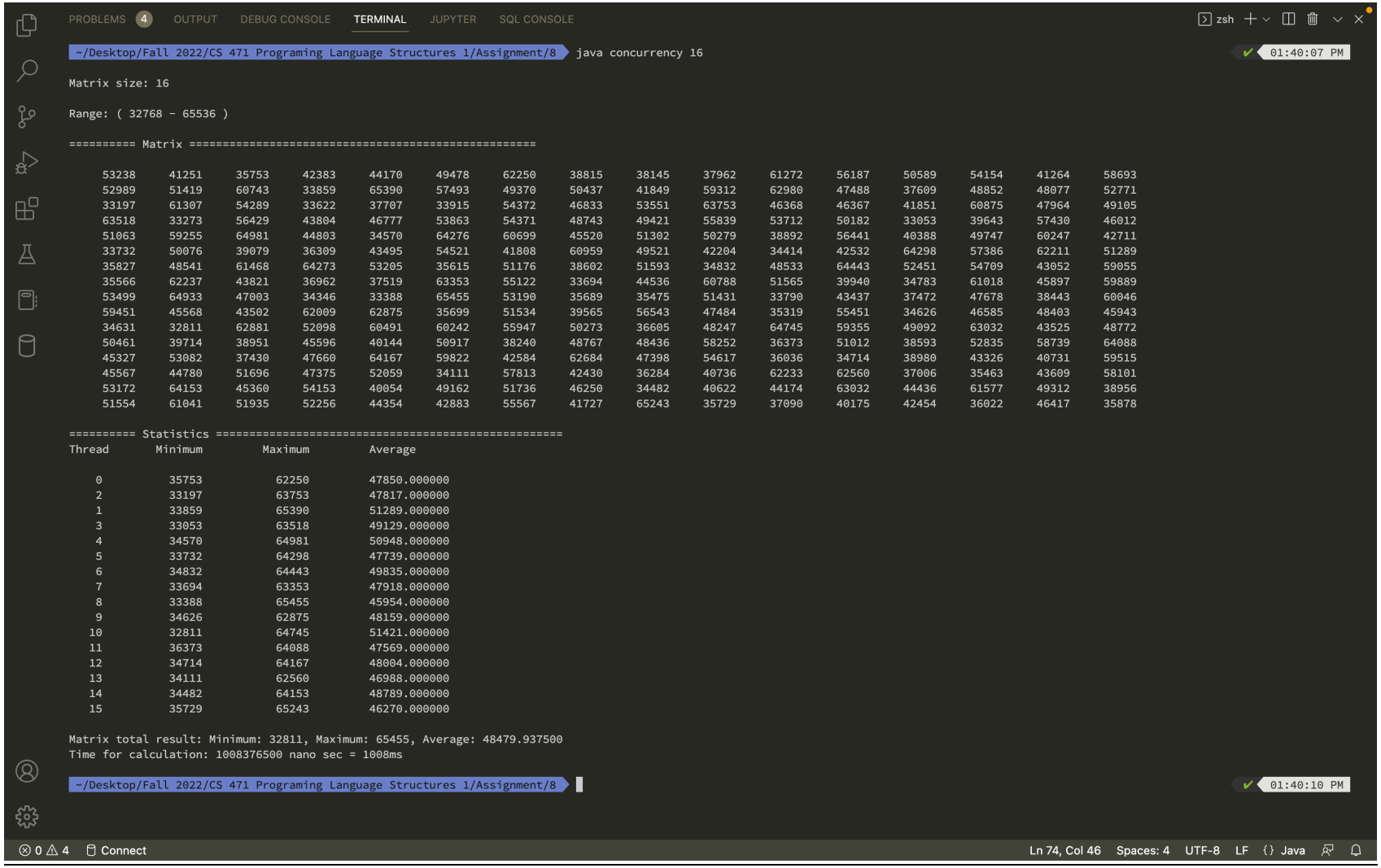
Matrix of Size 2 and 4:



Matrix of Size 8:

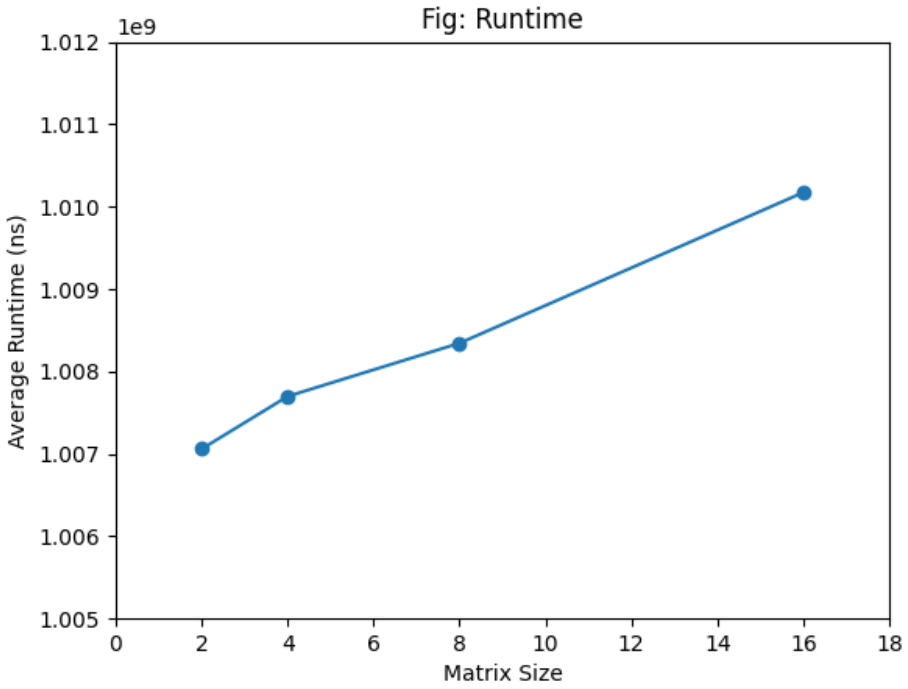


Matrix of Size 16:



Plot (time in nanoseconds): The following time has been calculated on my personal laptop with 8GB Ram, and apple M1 silicon processor.

Matrix Size	Run1	Run2	Run3	Run4	Run5	Average	Standard Deviation
N = 2	1006975542	1008312250	1004090958	1008428917	1007488750	1007059283.4	1577861.29
N = 4	1006402750	1006676667	1006037958	1009755167	1009614042	1007697316.8	1635829.07
N = 8	1007544417	1008701542	1008325708	1009215875	1007927917	1008343091.8	583230.79
N = 16	1008406291	1013187083	1011787875	1007059750	1010419583	1010172116.4	2216614.67



Description of Plot:

The plot's x-axis represents different matrix sizes 2, 4, 8, and 16. The y-axis represents runtime in nanoseconds. The graph shows that the runtime is directly proportional to the matrix size. As matrix size increases, runtime also increases.