

Name: Pratyay Kumar

Date: 11/10/2022

Program Lab: 8

## **Program #8: Concurrency**

### **Program Description:**

- The program takes 1 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 takes 1 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 :  $\text{range} * \text{Random}() + \text{start\_value}$

\* 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:

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER SQL CONSOLE
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 javac concurrency.java
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 java concurrency 2

Matrix size: 2
Range: ( 536870912 - 1073741824 )

===== Matrix =====
782974208 572135109
778548890 737571151

===== Statistics =====
Thread      Minimum      Maximum      Average
0           572135109    782974208    677554688.000000
1           737571151    778548890    758060032.000000

Matrix total result: Minimum: 572135109, Maximum: 782974208, Average: 717807360.000000
Time for calculation: 1007580208 nano sec = 1007ms

~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 java concurrency 4

Matrix size: 4
Range: ( 134217728 - 268435456 )

===== Matrix =====
163345509 201201856 167728364 204950199
261179449 187313572 161233437 219653974
230038691 174652897 214368619 165007889
248157824 189759629 219527385 158119344

===== Statistics =====
Thread      Minimum      Maximum      Average
0           163345509    204950199    184306480.000000
1           161233437    261179449    207345104.000000
2           165007889    230038691    196017024.000000
3           158119344    248157824    203891040.000000

Matrix total result: Minimum: 158119344, Maximum: 261179449, Average: 197889920.000000
Time for calculation: 1010211250 nano sec = 1010ms

~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8
```

## Matrix of Size 8:

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER SQL CONSOLE
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 java concurrency 8
Matrix size: 8
Range: ( 8388608 - 16777216 )

===== Matrix =====
15254240 12562871 10563465 14561329 14497243 12063741 11713268 14863278
10271766 12018749 10657752 13271796 9277616 10454145 15321968 15154140
13972429 15059790 11101949 15204486 14325279 13156989 8867512 16632235
16432887 9839180 12625967 10267408 9377605 9228328 11019496 9690784
9889757 10556201 8698303 9282727 12076490 9305498 9709253 12300907
13609857 11355804 11824978 11497007 15548707 13877386 12839240 8638376
15134638 14212910 11338336 11032670 12282253 13714271 15799900 16665932
15794498 13269945 12188525 10621287 10966256 12572233 9204538 13428567

===== Statistics =====
Thread    Minimum      Maximum      Average
0         10563465     15254240     13259929.000000
1         9277616      15321968     12053491.000000
2         8867512      16632235     13540083.000000
3         9228328      16432887     11060206.000000
4         8698303      12300907     10238642.000000
5         8638376      15548707     12398919.000000
6         11032670     16665932     13772613.000000
7         9204538      15794498     12255731.000000

Matrix total result: Minimum: 8638376, Maximum: 16665932, Average: 12322452.000000
Time for calculation: 1011614333 nano sec = 1011ms

~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8
```

Ln 74, Col 46 Spaces: 4 UTF-8 LF {} Java

## Matrix of Size 16:

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER SQL CONSOLE
~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 java concurrency 16 01:40:07 PM

Matrix size: 16

Range: ( 32768 - 65536 )

===== Matrix =====

53238 41251 35753 42383 44170 49478 62250 38815 38145 37962 61272 56187 50589 54154 41264 58693
52989 51419 60743 33859 65390 57493 49370 50437 41849 59312 62980 47488 37609 48852 48077 52771
33197 61307 54289 33622 37707 33915 54372 46833 53551 63753 46368 46367 41851 60875 47964 49105
63518 33273 56429 43804 46777 53863 54371 48743 49421 55839 53712 50182 33053 39643 57430 46012
51063 59255 64981 44803 34570 64276 60699 45520 51302 50279 38892 56441 40388 49747 60247 42711
33732 50076 39079 36309 43495 54521 41808 60959 49521 42204 34414 42532 64298 57386 62211 51289
35827 48541 61468 64273 53205 35615 51176 38602 51593 34832 48533 64443 52451 54709 43052 59055
35566 62237 43821 36962 37519 63353 55122 33694 44536 60788 51565 39940 34783 61018 45897 59889
53499 64933 47003 34346 33388 65455 53190 35689 35475 51431 33790 43437 37472 47678 38443 60046
59451 45568 43502 62009 62875 35699 51534 39565 56543 47484 35319 55451 34626 46585 48403 45943
34631 32811 62881 52098 60491 60242 55947 50273 36605 48247 64745 59355 49092 63032 43525 48772
50461 39714 38951 45596 40144 50917 38240 48767 48436 58252 36373 51012 38593 52835 58739 64088
45327 53082 37430 47660 64167 59822 42584 62684 47398 54617 36036 34714 38980 43326 40731 59515
45567 44780 51696 47375 52059 34111 57813 42430 36284 40736 62233 62560 37006 35463 43609 58101
53172 64153 45360 54153 40054 49162 51736 46250 34482 40622 44174 63032 44436 61577 49312 38956
51554 61041 51935 52256 44354 42883 55567 41727 65243 35729 37090 40175 42454 36022 46417 35878

===== Statistics =====
Thread      Minimum      Maximum      Average
0           35753        62250        47850.000000
2           33197        63753        47817.000000
1           33859        65390        51289.000000
3           33053        63518        49129.000000
4           34570        64981        50948.000000
5           33732        64298        47739.000000
6           34832        64443        49835.000000
7           33694        63353        47918.000000
8           33388        65455        45954.000000
9           34626        62875        48159.000000
10          32811        64745        51421.000000
11          36373        64088        47569.000000
12          34714        64167        48004.000000
13          34111        62560        46988.000000
14          34482        64153        48789.000000
15          35729        65243        46270.000000

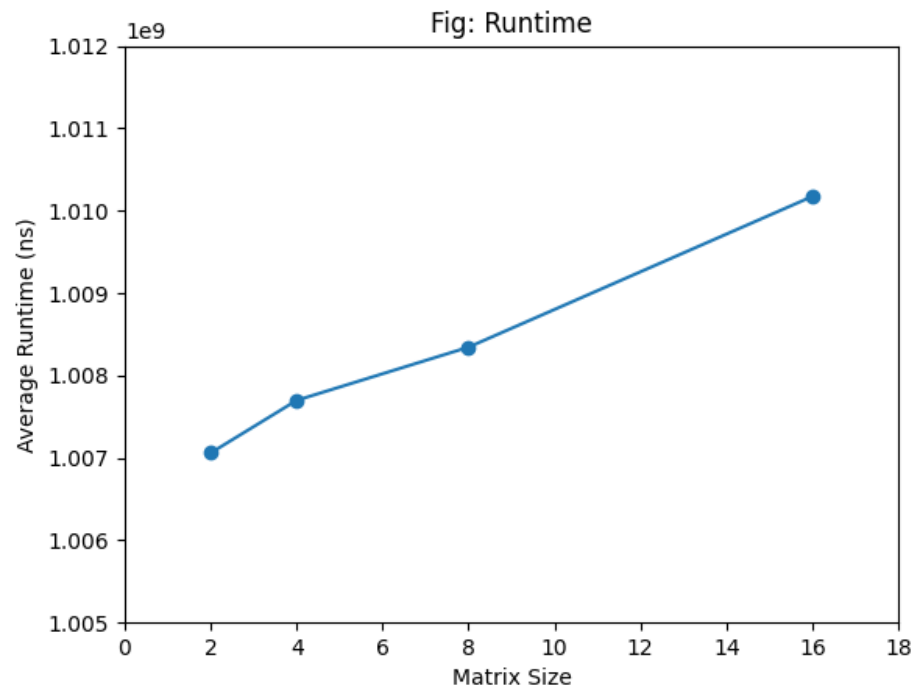
Matrix total result: Minimum: 32811, Maximum: 65455, Average: 48479.937500
Time for calculation: 1008376500 nano sec = 1008ms

~/Desktop/Fall 2022/CS 471 Programing Language Structures 1/Assignment/8 01:40:10 PM

0 4 Connect Ln 74, Col 46 Spaces: 4 UTF-8 LF {} Java
```

**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.