Tony Maldonado
CS 471
November 15, 2020

Programming # 8 – Concurrency

Code:

```
//
// // from http://www.letmeknows.com/2017/04/24/wait-for-threads-to-finish-java/ //
// This is a very small set up to get people started on using threads
//
//
//
//
//  Adopted by Shaun Cooper
//  last updated November 2020
//
//  We need static variable pointers in the main class so that
//  we can share these values with the threads.
//  the threads are address separate from us, so we need to share
//  pointers to the objects that we are sharing and updating

/*
* Updated by: Tony Maldonado
* On date: November 15, 2020
* Input: N, the size of the matrix
* Output: Min, Max, Avg, and running time
* Preconditions: N/A
* Postconditions: N/A
*/

import java.util.*;
import java.math.*;

public class Concurrency {

    private static ArrayList<Thread> arrThreads = new ArrayList<Thread>();

    // we use static variables to help us connect the threads
    // to a common block
    public static int N = 0;
    public static int[][] A;

    // Create the 1D arrays to store the max, min, and avg
    public static int[] Max;
    public static int[] Min;
    public static float[] Avg;
```

Tony Maldonado
CS 471
November 15, 2020

```
//main entry point for the process
public static void main(String[] args) {
   try {

      int localMin = 0;
      int localMax = 0;
      float localAvg = 0;

      // Input from the user:
      Scanner scan = new Scanner(System.in);
      int size = scan.nextInt();
      N = size;

      // create the array from input
      A = new int[size][size];
      Min = new int[size];
      Max = new int[size];
      Avg = new float[size];

      // Get the max and min range
      int max = (int) (Math.pow(2, (32 - N)));
      int min = (int) (Math.pow(2, (31 -N)));
      int range = max - min;

      // Now fill the array with the random values
      for (int i = 0; i < A.length; i++){
         for (int k = 0; k < A.length; k++) {
            A[i][k] = (int)(range * Math.random() + 1);
         }
      }

      // Start the timer
      long start = System.nanoTime();

      // create N threads to work on each row
      for (int i = 0; i < size; i++) {
         Thread T1 = new Thread(new ThreadTest(i));
         T1.start();              // standard thread start
         arrThreads.add(T1);
      }

      // wait for each thread to complete
      for (int i = 0; i < arrThreads.size(); i++) {
```

```
            arrThreads.get(i).join();
        }

        // Stop the timer
        long end = System.nanoTime();

        // Set the localMin to the first index of Min
        localMin = Min[0];

        // For loops to find the min, max, and avg
        for (int i = 0; i < N; i++) {
            // Min
            if (Min[i] < localMin) {
                localMin = Min[i];
            }

            // Max
            if (Max[i] > localMax) {
                localMax = Max[i];
            }

            // Avg
            localAvg = localAvg + Avg[i];
        }

        // Print how long the calculations took the complete
        System.out.println("Time to calculate: " + (end - start) + " nanoseconds");

        // Print out the min, max, and avg
        System.out.println("Min: " + localMin + " Max: " + localMax + " Avg: " +
localAvg);

        // All the threads are done, do final calculations
        System.out.println("Main Thread has N as value " + N);

        //This for loop will not stop execution of any thread,
        //only it will come out when all thread are executed

        System.out.println("Main thread exiting ");
      } catch (Exception e) {
        System.out.println(e.getMessage());
      }
    }
}
```

Tony Maldonado
CS 471
November 15, 2020

```
// each thread should access its row based on "ind"
// and leave results I would suggest in a static array that you need
// to create  in MythreadTest

class ThreadTest implements Runnable {
    private int i;

    // Some local variables for min, max and avg
    private int lMin = 0;
    private int lMax = 0;
    private float lAvg = 0;

    ThreadTest(int ind) {
      i = ind;
    }

    public void run() {
      try {

          lMin = Concurrency.A[i][0];
          System.out.println("Thread is started " + i + " Array is " + Concurrency.A[i][0]);

          for (int x = 0; x < Concurrency.N; x++) {
             // Find the min
             if (Concurrency.A[i][x] < lMin) {
                lMin = Concurrency.A[i][x];
             }

             // Find the max
             if (Concurrency.A[i][x] > lMax) {
                lMax = Concurrency.A[i][x];
             }

             // Find the avg
             lAvg = lAvg + (Concurrency.A[i][x] / (Concurrency.N * Concurrency.N));
          }

          // Store the values in the global vars
          Concurrency.Min[i] = lMin;
          Concurrency.Max[i] = lMax;
          Concurrency.Avg[i] = lAvg;

          // Thread.sleep(1000);
```

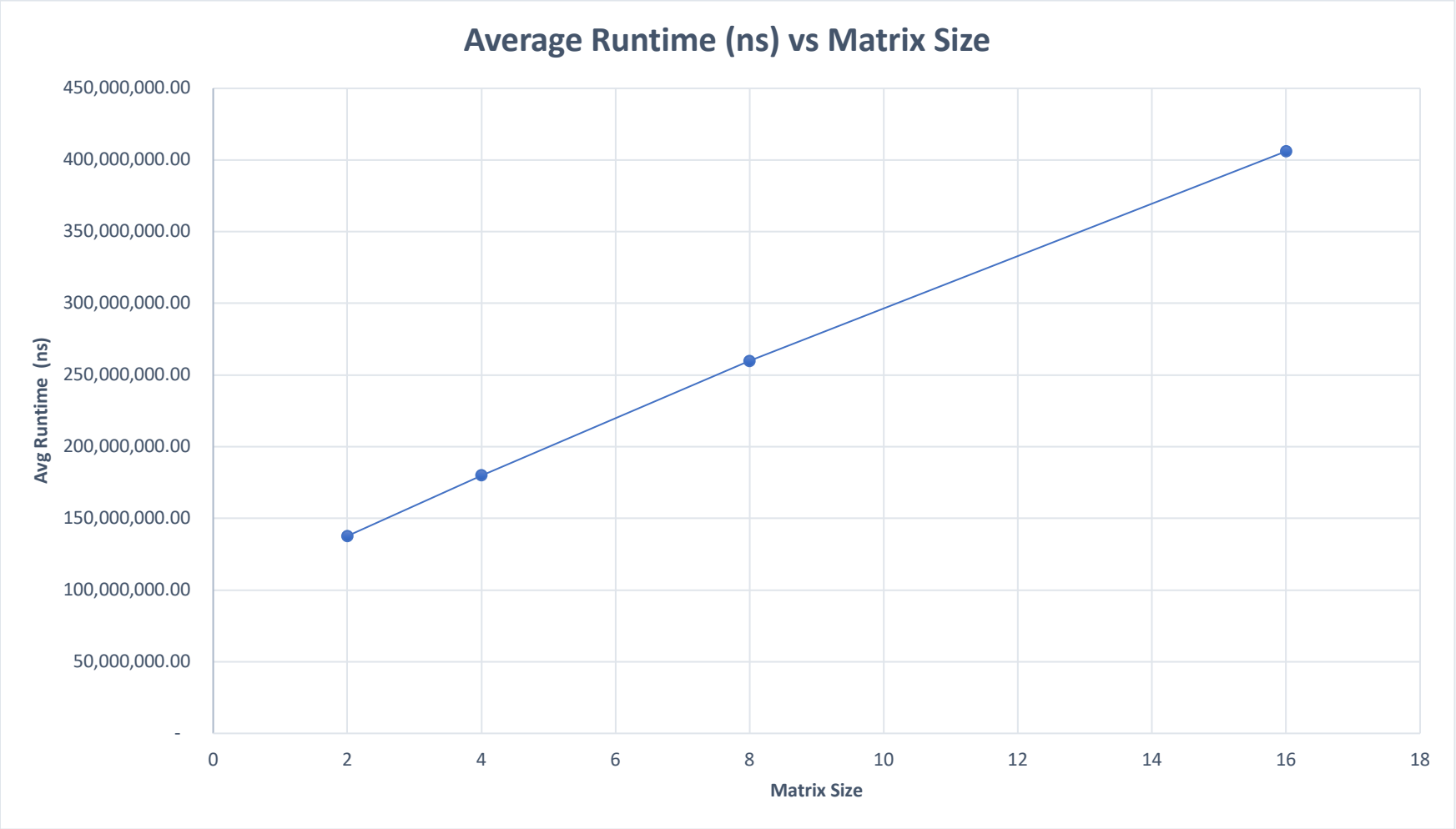
```
            System.out.println("Thread is exiting " + i);

        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Tables/Graphs:

Matrix Size N	2	4	8	16
Runtime 1 (ns)	120,810,211.00	161,123,395.00	271,609,167.00	364,440,408.00
Runtime 2 (ns)	151,364,848.00	166,160,561.00	313,030,922.00	408,950,637.00
Runtime 3 (ns)	118,797,518.00	199,650,466.00	241,227,872.00	423,627,319.00
Runtime 4 (ns)	154,804,759.00	197,245,818.00	223,140,890.00	381,222,237.00
Runtime 5 (ns)	142,595,783.00	175,572,372.00	250,248,923.00	452,040,418.00
AVG	137,674,623.80	179,950,522.40	259,851,554.80	406,056,203.80
STDEV	16,925,093.21	17,684,788.59	34,471,044.19	34,569,748.90

Matrix Size N	Average Runtime (ns)
2	137,674,623.80
4	179,950,522.40
8	259,851,554.80
16	406,056,203.80

Tony Maldonado
CS 471
November 15, 2020



Analysis:

- Based on the graph, we can conclude that the runtime for the different matrix sizes increases linearly.
- Based on the outputs of min, max, and avg for each matrix size, we can also conclude that as N increases, the minimum, maximum, and average decrease because the range [(2^(32-N)) and (2^(31-N))] of random integers that fill the arrays increases as N increases

Tony Maldonado
CS 471
November 15, 2020

Output of Code:

```
newton cs471/program8> java Concurrency
2
Thread is started 0 Array is 153149075
Thread is started 1 Array is 106437310
Thread is exiting 0
Thread is exiting 1
Time to calculate: 153498736 nanoseconds
Min: 106437310 Max: 238012603 Avg: 1.78893328E8
Main Thread has N as value 2
Main thread exiting
newton cs471/program8>
```

```
newton cs471/program8> java Concurrency
4
Thread is started 0 Array is 90845190
Thread is started 3 Array is 12841764
Thread is started 1 Array is 14823506
Thread is started 2 Array is 73547809
Thread is exiting 1
Thread is exiting 2
Thread is exiting 0
Thread is exiting 3
Time to calculate: 204406000 nanoseconds
Min: 4776117 Max: 116427532 Avg: 4.2498E7
Main Thread has N as value 4
Main thread exiting
newton cs471/program8>
```

Tony Maldonado
CS 471
November 15, 2020

```
[newton cs471/program8> java Concurrency
8
Thread is started 5 Array is 3732559
Thread is started 0 Array is 7282864
Thread is started 3 Array is 6776364
Thread is started 7 Array is 7117765
Thread is started 6 Array is 4257520
Thread is started 2 Array is 2271235
Thread is started 4 Array is 4298070
Thread is started 1 Array is 6403435
Thread is exiting 0
Thread is exiting 2
Thread is exiting 3
Thread is exiting 1
Thread is exiting 5
Thread is exiting 6
Thread is exiting 7
Thread is exiting 4
Time to calculate: 302560016 nanoseconds
Min: 116973 Max: 8326603 Avg: 4432605.0
Main Thread has N as value 8
Main thread exiting
[newton cs471/program8>
```

Tony Maldonado
CS 471
November 15, 2020

```
newton cs471/program8> java Concurrency
16
Thread is started 4 Array is 2136
Thread is started 15 Array is 26185
Thread is started 5 Array is 28529
Thread is started 13 Array is 32707
Thread is started 14 Array is 21863
Thread is started 0 Array is 24484
Thread is started 11 Array is 8504
Thread is started 12 Array is 25893
Thread is started 10 Array is 32346
Thread is started 3 Array is 3674
Thread is started 9 Array is 4342
Thread is started 6 Array is 22729
Thread is started 7 Array is 31258
Thread is started 2 Array is 25539
Thread is started 1 Array is 16802
Thread is started 8 Array is 7005
Thread is exiting 5
Thread is exiting 3
Thread is exiting 14
Thread is exiting 1
Thread is exiting 10
Thread is exiting 2
Thread is exiting 7
Thread is exiting 13
Thread is exiting 6
Thread is exiting 8
Thread is exiting 0
Thread is exiting 11
Thread is exiting 12
Thread is exiting 9
Thread is exiting 4
Thread is exiting 15
Time to calculate: 456107960 nanoseconds
Min: 324 Max: 32707 Avg: 16142.0
Main Thread has N as value 16
Main thread exiting
newton cs471/program8>
```