PROJECT NAME: CS 415 Programming Assignment #2

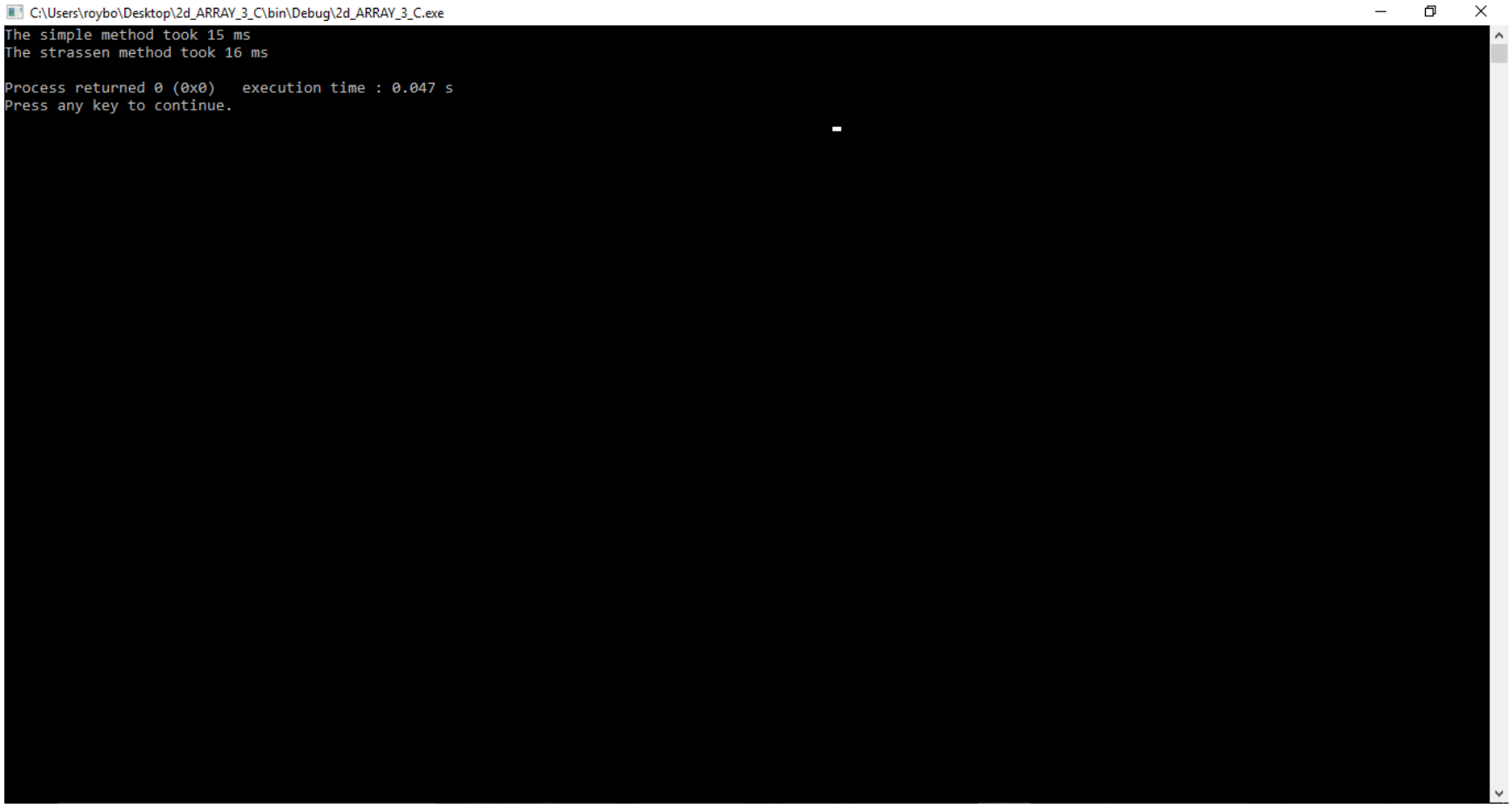
Test Priority (LOW/MED./HIGH): HIGH	**Test Designed By:** Adam Moses
Module Name: 2D Array Matrix processing	
Test Title: Testing Pthread Efficiency	
Description: Test multiple array sizes regarding 2d array matrix multiplication	
Caveats: The following were used for testing and developing: Machine – Dell Inspiron 5000 series (intel core I3) OS – Windows 10 IDE – Code Blocks Language – C	

Test Case	Test Data	Expected Results	Actual results	Status (Pass / Fail)
Matrix size 100	Random digits between 0-9	Time for Strassen and Simple Method	Simple – 15 m(sec) Strassen – 16 m(sec)	P
Matrix size 500	Random digits between 0-9	Time for Strassen and Simple Method	Simple – 1162 m(sec) Strassen – 969 m(sec)	P
Matrix size 1000	Random digits between 0-9	Time for Strassen and Simple Method	Simple – 14314 m(sec) Strassen – 8910 m(sec)	P
Matrix size 5000	Random digits between 0-9	Time for Strassen and Simple Method	Simple – 1087420 m(sec) Strassen – 1860380 m(sec)	P
Matrix size 10000	Random digits between 0-9, ALL with 2	Time for Strassen and Simple Method	ERROR: Process returned 255 (0xFF)	F

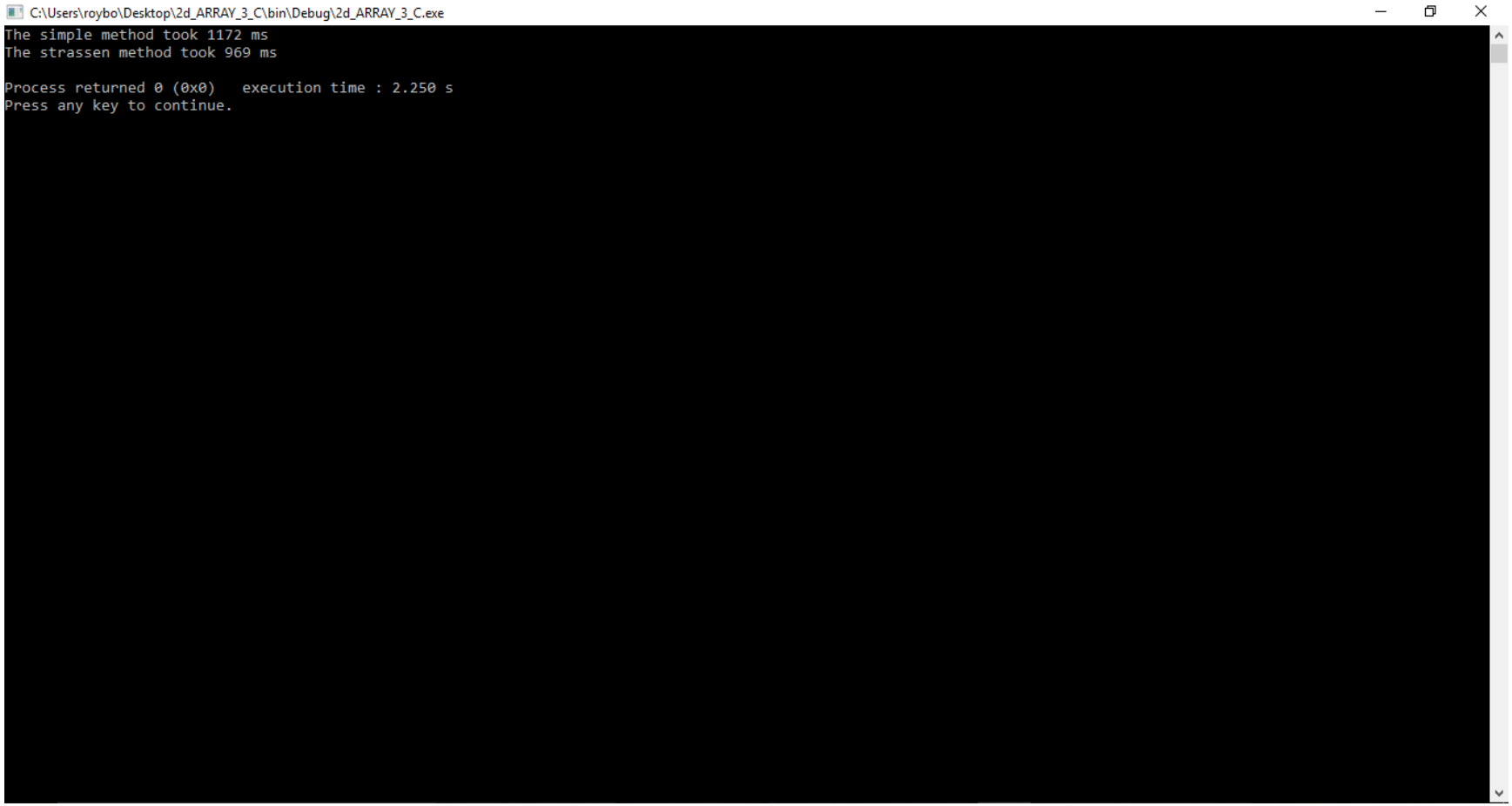
NOTES: For matrix 10000, I tried feeling arrays with random numbers between 0-9 and with ALL numbers being 2. Still giving error message.

Screenshots:

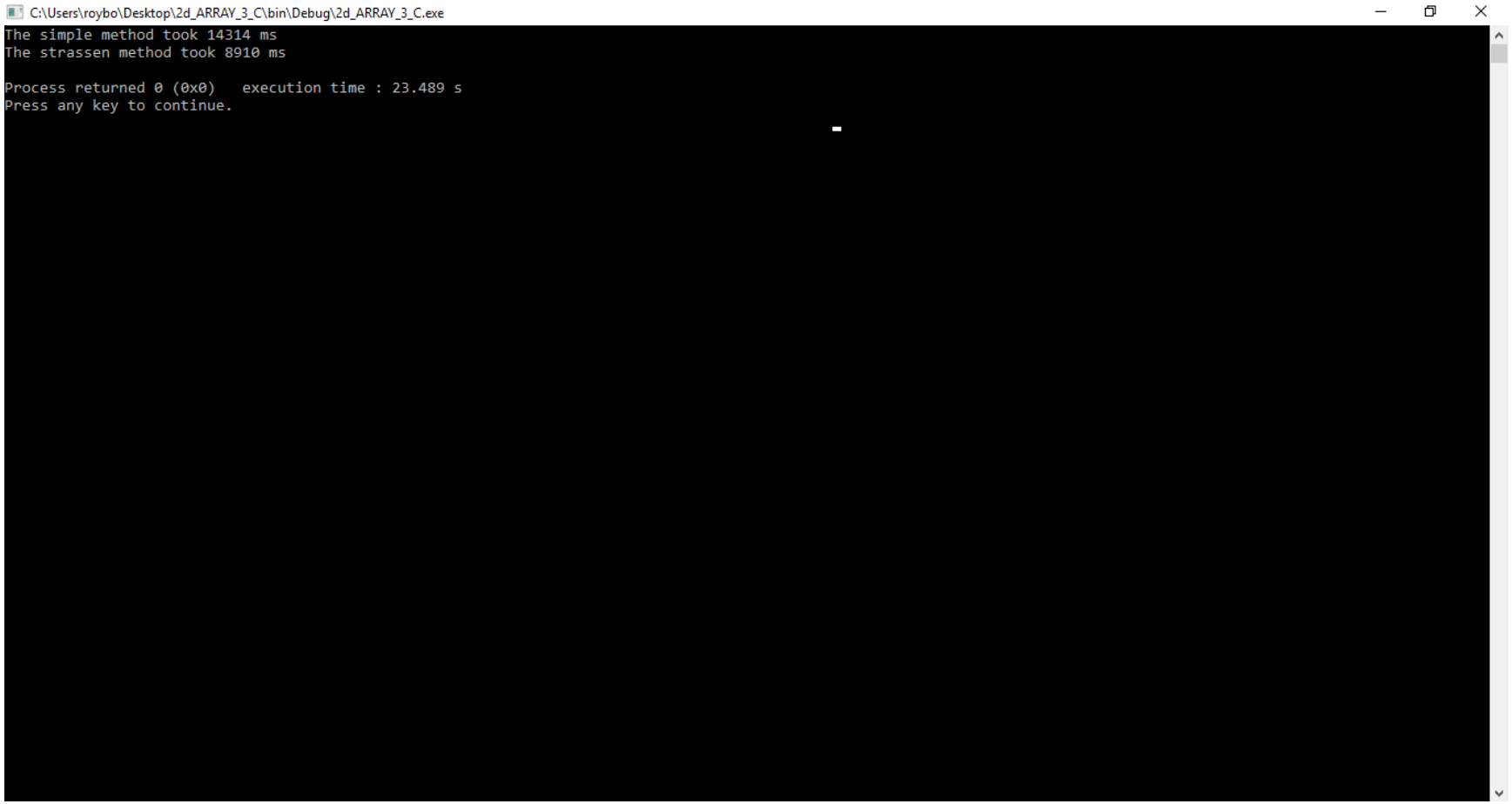
Size – 100 (random numbers between 0-9)



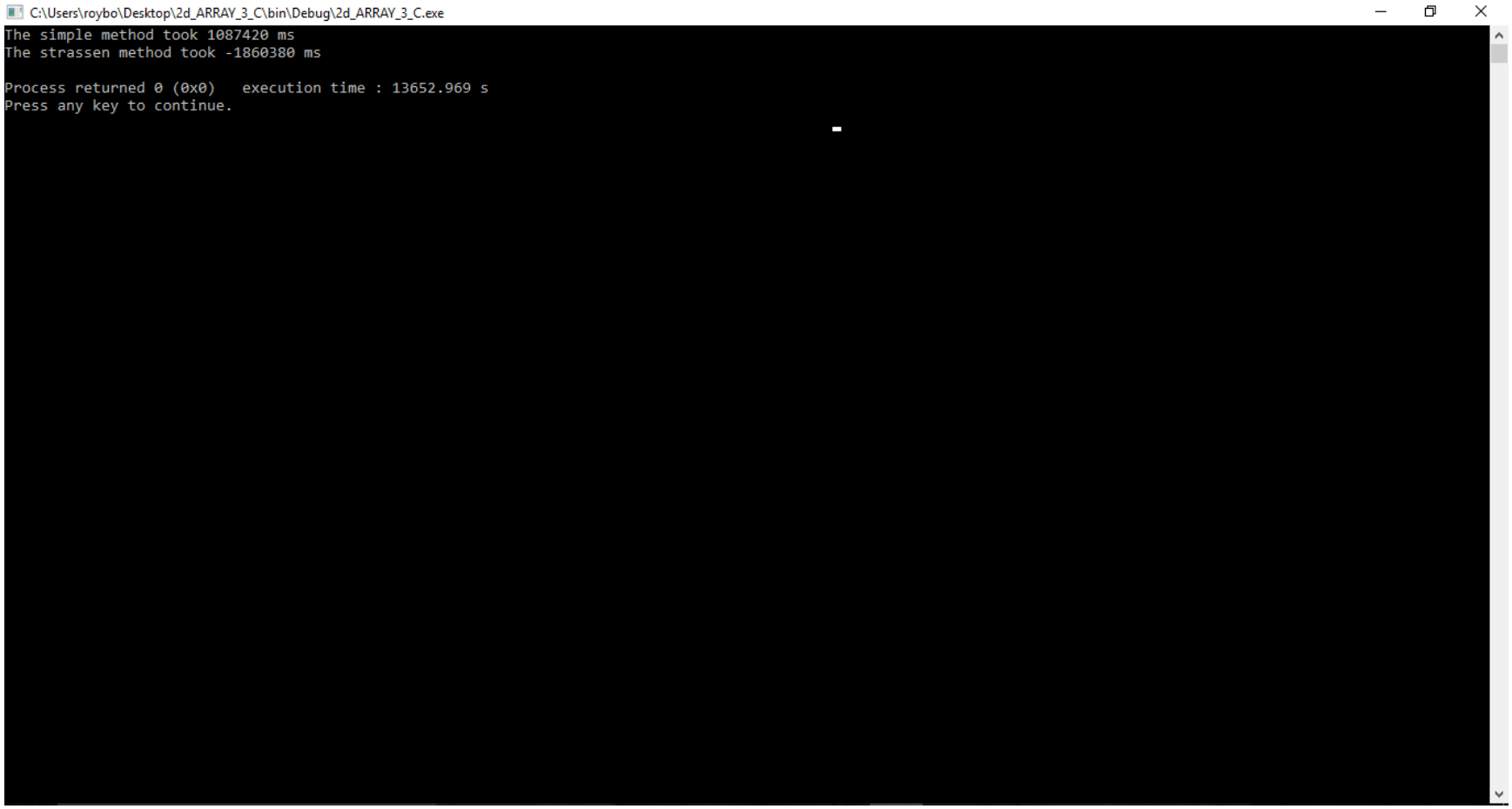
Size – 500 (random numbers between 0-9)



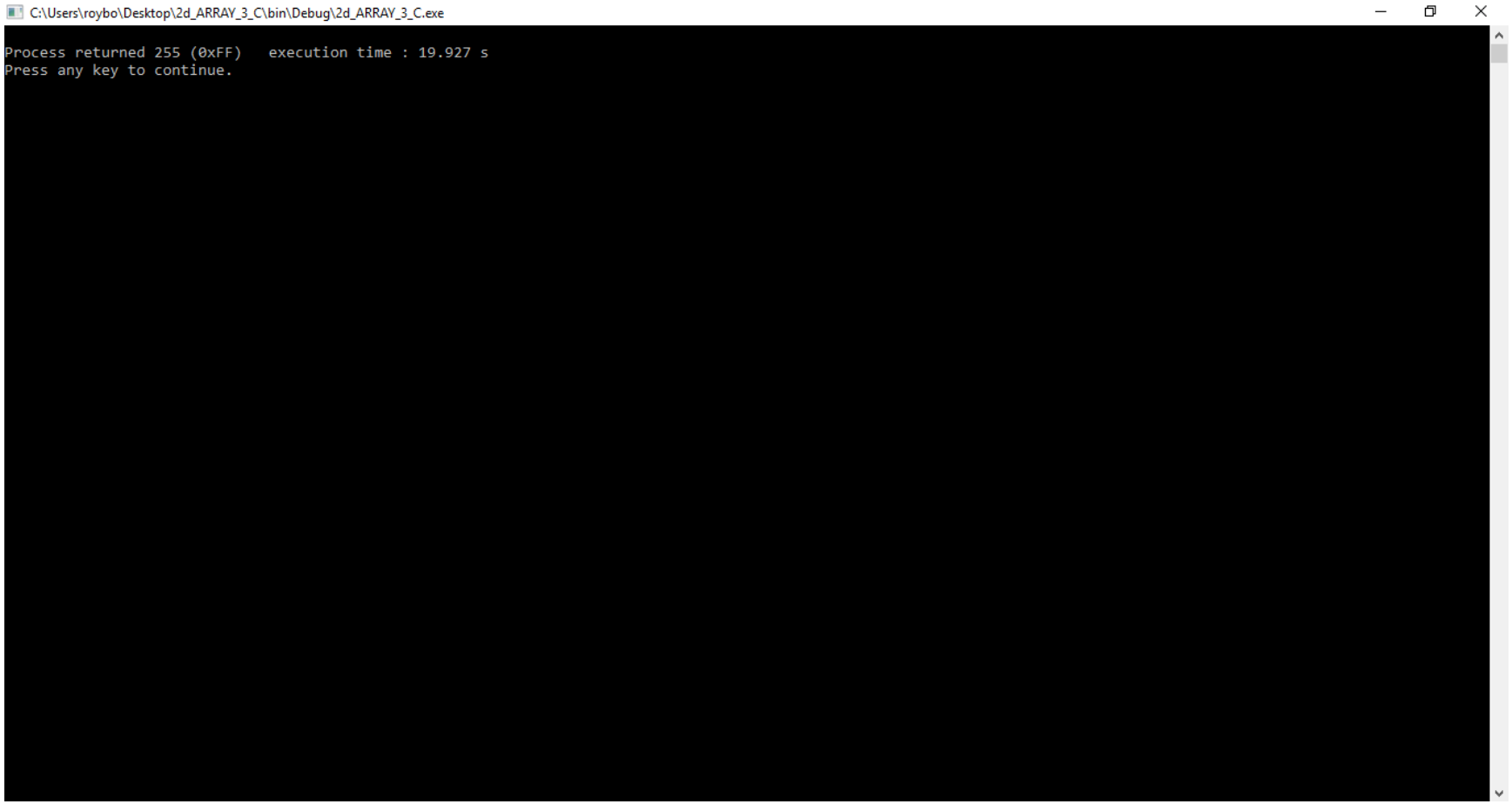
Size – 1000 (random numbers between 0-9)



Size – 5000 (random numbers between 0-9)



Size – 10000 (random numbers between 0-9)



Source Code: (header file for “timer.h”)

```
#ifndef TIMER_H_INCLUDED
#define TIMER_H_INCLUDED
#include <time.h>
#include <stdlib.h>
#include <stdio.h>

clock_t start, diff;
// Starts timer and resets the elapsed time
void timerStart(){
  start = clock();
}

// Stops the timer and returns elapsed time in msec
long timerStop(){
  diff = clock() - start;
  return (diff * 1000) / CLOCKS_PER_SEC;
}

#endif // TIMER_H_INCLUDED
```

==========

Source code:

```
//Adam Moses
//CS 415 Spring 2017
//Program #2
//Sources: https://randu.org/tutorials/threads/
//      : https://athens.blackboard.com/webapps/blackboard/execute/
//        displayLearningUnit?course_id=_162737_1&content_id=_1576768_1&framesetWrapped=true
//        Module Parts A - D
//      : https://athens.blackboard.com/webapps/blackboard/execute/
//        displayLearningUnit?course_id=_162737_1&content_id=_1577462_1&framesetWrapped=true
//      : https://github.com/bryanmills/hpc-course/tree/master/hw3

#include "timer.h"
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define NUM_THREADS 10

//Global matrix
```

```
int **A;
int **B;
int **C;
// Reference matrix, call simpleMethod to populate.
int **R;

//Parts for dealing with strassen
int **A11;
int **A12;
int **A21;
int **A22;
int **B11;
int **B12;
int **B21;
int **B22;

// M matrixes
int **M1; // (A_1,1 + A_2,2)*(B_1,1 + B_2,1)
int **M2; // (A_2,1 + A_2,2)*B_1,1
int **M3; // A_1,1 * (B_1,2 - B_2,2)
int **M4; // A_2,2 * (B_2,1 - B_1,1)
```

```
int **M5; // (A_1,1 + A_1,2) * B_2,2
int **M6; // (A_2,1 - A_1,1) * (B_1,1 + B_1,2)
int **M7; // (A_1,2 - A_2,2) * (B_2,1 + B_2,2)

// C matrixes
int **C11;
int **C12;
int **C21;
int **C22;

int howBig;

//Simple method
void simpleMethod(int **a, int **b, int **r, int N) {
  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      for (int k=0; k<N; k++) {
            r[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

```
}

void simpleAdd(int **a, int **b, int **r, int N) {
  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      r[i][j] = a[i][j] + b[i][j];
    }
  }
}

void simpleSub(int **a, int **b, int **r, int N) {
  for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
      r[i][j] = a[i][j] - b[i][j];
    }
  }
}

void makeParts(int N) {
  int half = N/2;
  for (int row = 0; row < half; row++) {
```

```
    for (int col = 0; col < half; col++) {
      A11[row][col] = A[row][col];
      A12[row][col] = A[row][half+col];
      A21[row][col] = A[half+row][col];
      A22[row][col] = A[half+row][half+col];
      B11[row][col] = B[row][col];
      B12[row][col] = B[row][half+col];
      B21[row][col] = B[half+row][col];
      B22[row][col] = B[half+row][half+col];
    }
  }
}
// Allocate square matrix.
int **allocMatrix(int size) {
  int **matrix;
  matrix = (int **)malloc(size * sizeof(int *));
  for (int row = 0; row < size; row++) {
    matrix[row] = (int *)malloc(size * sizeof(int));
  }
  for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
```

```
      matrix[i][j] = 0;
    }
  }
  return matrix;
}

// (A_1,1 + A_2,2)*(B_1,1 + B_2,1)
void* calcM1(void* nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleAdd(A11, A22, temp1, howBig/2);
  simpleAdd(A11, A22, temp2, howBig/2);
  simpleMethod(temp1, temp2, M1, howBig/2);
  free(temp1);
  free(temp2);
  return 0;
}

// (A_2,1 + A_2,2)*B_1,1
```

```
void* calcM2(void* nothing) {
  int **temp1;
  temp1 = allocMatrix(howBig/2);
  simpleAdd(A21, A22, temp1, howBig/2);
  simpleMethod(temp1, B11, M2, howBig/2);
  free(temp1);
  return 0;
}
// A_1,1 * (B_1,2 - B_2,2)
void* calcM3(void* nothing) {
  int **temp1;
  temp1 = allocMatrix(howBig/2);
  simpleSub(B12, B22, temp1, howBig/2);
  simpleMethod(A11, temp1, M3, howBig/2);
  free(temp1);
  return 0;
}

// A_2,2 * (B_2,1 - B_1,1)
void* calcM4(void* nothing) {
  int **temp1;
```

```
  temp1 = allocMatrix(howBig/2);
  simpleSub(B21, B11, temp1, howBig/2);
  simpleMethod(A22, temp1, M4, howBig/2);
  free(temp1);
  return 0;
}
// (A_1,1 + A_1,2) * B_2,2
void* calcM5(void *nothing) {
  int **temp1;
  temp1 = allocMatrix(howBig/2);
  simpleAdd(A11, A12, temp1, howBig/2);
  simpleMethod(temp1, B22, M5, howBig/2);
  free(temp1);
  return 0;
}
// (A_2,1 - A_1,1) * (B_1,1 + B_1,2)
void* calcM6(void* nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
```

```
  simpleSub(A21, A11, temp1, howBig/2);
  simpleAdd(B11, B12, temp2, howBig/2);
  simpleMethod(temp1, temp2, M6, howBig/2);
  free(temp1);
  free(temp2);
  return 0;
}
// (A_1,2 - A_2,2) * (B_2,1 + B_2,2)
void* calcM7(void *nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleSub(A12, A22, temp1, howBig/2);
  simpleAdd(B21, B22, temp2, howBig/2);
  simpleMethod(temp1, temp2, M7, howBig/2);
  free(temp1);
  free(temp2);
  return 0;
}
// (A_1,1 * B_1,1) + (A_1,2 * B_2,1)
```

```
void* calcC11(void* nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleMethod(A11, B11, temp1, howBig/2);
  simpleMethod(A12, B21, temp2, howBig/2);
  simpleAdd(temp1, temp2, C11, howBig/2);
  free(temp1);
  free(temp2);
  return 0;
}
// (A_1,1 * B_1,2) + (A_1,2 * B_2,2)
void* calcC12(void* nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleMethod(A11, B12, temp1, howBig/2);
  simpleMethod(A12, B22, temp2, howBig/2);
  simpleAdd(temp1, temp2, C12, howBig/2);
```

```
  free(temp1);
  free(temp2);
  return 0;
}

// (A_2,1 * B_1,1) + (A_2,2 * B_2,1)
void* calcC21(void* nothing) {
  int **temp1;
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleMethod(A21, B11, temp1, howBig/2);
  simpleMethod(A22, B21, temp2, howBig/2);
  simpleAdd(temp1, temp2, C21, howBig/2);
  free(temp1);
  free(temp2);
  return 0;

// (A_2,1 * B_1,2) + (A_2,2 * B_2,2)
void* calcC22(void * nothing) {
  int **temp1;
```

```
  int **temp2;
  temp1 = allocMatrix(howBig/2);
  temp2 = allocMatrix(howBig/2);
  simpleMethod(A21, B12, temp1, howBig/2);
  simpleMethod(A22, B22, temp2, howBig/2);
  simpleAdd(temp1, temp2, C22, howBig/2);
  free(temp1);
  free(temp2);
  return 0;
}
void copyC(int N) {
  int half = N/2;
  for (int row = 0; row < half; row++) {
    for (int col = 0; col < half; col++) {
      C[row][col] = C11[row][col];
      C[row][half+col] = C12[row][col];
      C[half+row][col] = C21[row][col];
      C[half+row][half+col] = C22[row][col];
    }
  }
}
```

```
//*************************************************************************
//*************************************************************************
//Strassen work
//It's pthreading time!!
void strassenMethod(int N) {
  pthread_t ids[NUM_THREADS];
  int i_s[NUM_THREADS];
  makeParts(N);
  pthread_create(&ids[0], NULL, calcM1, NULL); // calcM1(NULL)
  pthread_create(&ids[1], NULL, calcM2, NULL); // calcM2(NULL)
  pthread_create(&ids[2], NULL, calcM3, NULL); // calcM3(NULL)
  pthread_create(&ids[3], NULL, calcM4, NULL); // calcM4(NULL)
  pthread_create(&ids[4], NULL, calcM5, NULL); // calcM5(NULL)
  pthread_create(&ids[5], NULL, calcM6, NULL); // calcM6(NULL)
  pthread_create(&ids[6], NULL, calcM7, NULL); // calcM7(NULL)
  for (int i = 0; i < 7; i++)
    pthread_join(ids[i], NULL);
  pthread_create(&ids[0], NULL, calcC11, NULL); // calcC11(NULL)
  pthread_create(&ids[1], NULL, calcC12, NULL); // calcC12(NULL)
  pthread_create(&ids[2], NULL, calcC21, NULL); // calcC21(NULL)
  pthread_create(&ids[3], NULL, calcC22, NULL); // calcC22(NULL)
```

```
  for (int i = 0; i < 4; i++)
    pthread_join(ids[i], NULL);
  copyC(N);
}
//*********************************************************************
//*********************************************************************
// Allocate memory
void initMatrixes(int N) {
  A = allocMatrix(N); B = allocMatrix(N); C = allocMatrix(N); R = allocMatrix(N);
  int half = N/2;
  A11 = allocMatrix(half); A12 = allocMatrix(half); A21 = allocMatrix(half); A22 = allocMatrix(half);
  B11 = allocMatrix(half); B12 = allocMatrix(half); B21 = allocMatrix(half); B22 = allocMatrix(half);
  M1 = allocMatrix(half); M2 = allocMatrix(half); M3 = allocMatrix(half); M4 = allocMatrix(half);
  M5 = allocMatrix(half); M6 = allocMatrix(half); M7 = allocMatrix(half);
  C11 = allocMatrix(half); C12 = allocMatrix(half); C21 = allocMatrix(half); C22 = allocMatrix(half);
}

// Free up matrixes.
void cleanup() {
  free(A);
  free(B);
```

```
  free(C);
  free(R);
}
//---------------------------------------------------------------------------------------
//---------------------------------------------------------------------------------------
// Main method
int main(int argc, char* argv[]) {

  //How big is the array matrix
  howBig=10000;
  initMatrixes(howBig);

    //Fill arrays with random number between 0-9
    for (int i=0; i<howBig; i++) {
      for (int j=0; j<howBig; j++) {
            A[i][j] = rand() % 10;
            B[i][j] = rand() % 10;
      }
    }
  //Exercise the simple method and output the time it took.
  timerStart();
```

```
  simpleMethod(A, B, R, howBig);
  printf("The simple method took %ld ms\n", timerStop());

  //Exercise the strassen method and output the time it took.
  timerStart();
  strassenMethod(howBig);
  printf("The strassen method took %ld ms\n", timerStop());
//Clean house
  cleanup();
  return 0;
}
//-------------------------------------------------------------------------------
//-------------------------------------------------------------------------------
```