Programming Assignment #1
DUE January 28, 2019

You are required to complete this assignment INDIVIDUALLY.

Complete this programming assignment using a high-level procedural programming language (e.g., Java, C, C++; not MATLAB) of your choice. Carefully read the instructions below regarding printouts and other documentation; note that you do NOT submit an executable copy of your program.

The deliverables for this assignment must be submitted electronically on the Canvas "Assignments" page no later than 11:00 pm on the due date listed above. A single pdf file is the preferred format. Clearly PRINT your name at the top of each sheet. You are responsible for ensuring that your submitted file(s) are readable and successfully uploaded to Canvas.

This assignment requires you to implement two different code segments, both $O(n^2)$, for initializing an [NxN] identity matrix. Assume the elements of the matrix are floating-point numbers. Compare the actual running times for matrices of various sizes; start with 10x10 and increase in multiple steps to 500x500. (If your system cannot handle a matrix that large, then you may need to stop at a smaller size.)

Option A

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        A[i][j] = 0.0;
for (i = 0; i < N; i++)
    A[i][i] = 1.0;
```

Option B

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        if (i == j)
            A[i][j] = 1.0;
            else
            A[i][j] = 0.0;
```

Your program should do the following each time it executes:

(a) Prompt you to enter the size of the matrix.
(b) Execute both algorithms, tracking the execution time for each (see hints below).
(c) Print out the size of the matrix and the execution time for each option.

Submit the following materials, preferably combined into a single .pdf file:

(a) A listing (text) of your program source code; an executable version is NOT required.

(b) The program output (written to a file or captured from your computer screen) for both the smallest and largest matrix sizes that you analyze.

(c) A table summarizing the execution times for both algorithms for all matrix sizes analyzed. This may be handwritten or you may generate the table using a program (e.g., Word or Excel).

(d) A **SINGLE** graph comparing execution time versus matrix size for both algorithms. Since these algorithms are both $O(n^2)$, also plot the curve $\mathbf{y = c\ x^2}$ on this same graph, where you choose **c** to produce a useful comparison to the measured execution times. This graph may be generated using a program (e.g., Excel) or hand-drawn reasonably to scale.

(e) A brief (2 or 3 paragraphs) summary of the results and your observations about the execution times of these two algorithms, compared to each other and to their Big-Oh values.

Guidelines and hints for measuring running time:

One simple method for estimating the execution time for a code segment is to record the clock time immediately before and immediately after the code segment; the difference between the two times is approximately the execution time of that code segment. Most systems provide an appropriate system call (whose format will depend on the language and system) that returns the clock time. **Figure 1** below illustrates this approach

```
|
|  lines of code (not timed)
|
StartTime = clocktime();
     |
     |  code sequence to be timed
     |
ElapsedTime = clocktime() – StartTime;
|
|
```

Figure 1.

```
|
|  lines of code (not timed)
|
NumTimes = 1000000;
StartTime = clocktime();
for (i = 1 to NumTimes) do {
          |
          |  code sequence to be timed
          |
     }
ElapsedTime = clocktime() – StartTime;
TimeForOne = ElapsedTime / NumTimes;
|
```

Figure 2.

This method has several limitations, particularly for measuring very short code segments, including clock granularity and overhead for the function calls. In such cases, a loop may be inserted to repeat the execution of the code segment a large number of times (e.g., 100,000 or 1,000,000). The total elapsed time is divided by the number of iterations to get an approximate time for a single execution of the code segment. This method, shown above in **Figure 2**, may not work if the code sequence in question includes instructions that cannot be repeated without affecting the result.

In either case, for maximum accuracy when measuring performance, minimize extra code (e.g., input or output statements, added function calls) between the two calls to the clock.