In class, we worked on a piece of code that looked at declaring and using a 2D array that was declared two different ways: ([Link to Video](https://coastal.yuja.com/V/Video?v=9656207&node=42622868&a=99178340&autoplay=1)) 1) all at once, and 2) one row at a time (the code used to generate this is given at the end of this document).

We then timed the two says, and observed:

A white background with black text

Description automatically generated

Here we see that way 2 was superior to way 1, based on the analysis we did, especially as the same of the array increased.

As mentioned in class, since the times are relatively small even for some of the larger matricies, we need to add another parameter ‘w’ that represents the amount of work that the usage part of the code will perform each time the program is run. For example, an outer loop could be used around the work section, with w representing the number of iterations to perform that loop. Then the times reported could be the running sum of cumulative time spent doing that work, and thus making the analysis easier.

Secondly, in class, it was hard to tell how the overall time was being spent, so we need to update the code into parts A) time spent declaring the arrays B) time spent using / working with them.

As such, the new interface and functionality would be like this

java -Xmx60000m make\_data\_2D.java 10000 10000 10

Here, the last 10 is the w parameter.

Then the output would be something like this:

W1 = 1.0E-01 + 6.0E-02 = **1.6E-01**, W2 = 7.0E-02 + 6.0E-02 = **1.5E-01**

Where YELLOW is the time spent declaring the structures, the BLUE is the time spent doing the work on the structures, and the bolded is the overall time. (note, I made up these example outputs, the actual numbers will vary)

Once you do this, I would like for you to gather some data and make some plots / graphs that show the three constituent parts for each method as a function of the array size (NxN, square would be appropriate for this).

So that means that you will have a plot with the Y axis being in units of time and the X axis is N. There will be 6 lines on the graph, one for each of the constituent parts of both methods.

That way we can observe which is better and how much time spent, and how it varies for this system. N should range from 1000 to 15000, at least, with a step size of 1000, so the where should be 15 separate X values – each line in the graph should have 15 data points.

Make sure the phots are well-formatted, and have labeled axes and also that there is a key that identifies what each line is for (maybe have the lines have different colors and shapes (dotted, dashed, etc --- for accessibility reasons)

* Write and document the code.
* Write a report that indicates the thought process how it all works, and that shows it in operation for gathering all the data used above
* Include all data, graphs, and plots, tables in the report

You ZIP file should have this folder structure, with a top level, and three sub folders:

/hw04-your-name

./code // put the code here – java source

./report // put the report here, Word DOC and PDF

./data // raw data and plots and plot-generating items (Excel, python, whatever)

**Submission:**

ZIP and compress and submit.

**Skeleton of code – with some comments to help point out what you need to add:**

**public** **class** make\_data\_2D {

**public** **static** **void** main(String[] args) {

// **TODO** -- update to take one more parameter w

// w is the number of times to do the "work" portion of the code

**if**(args.length!=2) {

System.***out***.println("usage: java \_\_\_ <#r> <#c>");

System.*exit*(1);

}

**int** rows = Integer.*parseInt*(args[0]);

**int** cols = Integer.*parseInt*(args[1]);

// way 1 -- normal way -- all at once

**long** start1 = System.*nanoTime*();

// this is the create part for way 1

**double**[][] a = **new** **double**[rows][cols];

// this is the work part of the code for way 1

// **TODO** -- add outer loop to do the work part "w" times

**for**(**int** i=0; i<a.length; i++) {

**for**(**int** j=0; j<a[i].length; j++) {

a[i][j] = 42.5;

}

}

**for**(**int** i=0; i<a.length; i++) {

**for**(**int** j=0; j<a[i].length; j++) {

a[i][j] \*= (i+j);

}

}

**long** end1 = System.*nanoTime*();

// way 2 -- row by row

**long** start2 = System.*nanoTime*();

// this is the create part for way 2

**double**[][] b = **new** **double**[rows][];

**for**(**int** i=0; i<b.length; i++) {

b[i] = **new** **double**[cols];

}

// this is the work part for way 2

// **TODO** -- add outer loop to do the work part "w" times

**for**(**int** i=0; i<b.length; i++) {

**for**(**int** j=0; j<b[i].length; j++) {

b[i][j] = 42.5;

}

}

**for**(**int** i=0; i<b.length; i++) {

**for**(**int** j=0; j<b[i].length; j++) {

b[i][j] \*= (i+j);

}

}

**long** end2 = System.*nanoTime*();

// **TODO** -- update this part to measure

// overall time for ways 1 and 2 -- which is what the code did in class

// time spent just in creating way 1

// time spent just in creating way 2

// time spent just in using way 1 (the work)

// time spent just in using way 2 (the work)

**double** elapsed1 = (end1 - start1) / 1E9;

**double** elapsed2 = (end2 - start2) / 1E9;

// **TODO** -- so this will print 6 numbers here when done

System.***out***.printf("W1 = %.1E, W2 = %.1E \n", elapsed1, elapsed2);

}

}

**CODE FROM CLASS – as it was when we were in class :**

**public** **class** make\_data\_2D {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**if**(args.length!=2) {

System.***out***.println("usage: java \_\_\_ <#r> <#c>");

System.*exit*(1);

}

**int** rows = Integer.*parseInt*(args[0]);

**int** cols = Integer.*parseInt*(args[1]);

// way 1 -- normal way -- all at once

**long** start1 = System.*nanoTime*();

**double**[][] a = **new** **double**[rows][cols];

**for**(**int** i=0; i<a.length; i++) {

**for**(**int** j=0; j<a[i].length; j++) {

a[i][j] = 42.5;

}

}

**for**(**int** i=0; i<a.length; i++) {

**for**(**int** j=0; j<a[i].length; j++) {

a[i][j] \*= (i+j);

}

}

**long** end1 = System.*nanoTime*();

// way 2 -- row by row

**long** start2 = System.*nanoTime*();

**double**[][] b = **new** **double**[rows][];

**for**(**int** i=0; i<b.length; i++) {

b[i] = **new** **double**[cols];

}

**for**(**int** i=0; i<b.length; i++) {

**for**(**int** j=0; j<b[i].length; j++) {

b[i][j] = 42.5;

}

}

**for**(**int** i=0; i<b.length; i++) {

**for**(**int** j=0; j<b[i].length; j++) {

b[i][j] \*= (i+j);

}

}

**long** end2 = System.*nanoTime*();

**double** elapsed1 = (end1 - start1) / 1E9;

**double** elapsed2 = (end2 - start2) / 1E9;

System.***out***.printf("W1 = %.1E, W2 = %.1E \n", elapsed1, elapsed2);

}

}