Queens College of CUNY
Department of Computer Science
Design and Analysis of Algorithms (CSCI 323)
June 2024

Assignment #3:
"Matrix Multiplication"
Due: June 10, 2024

Overview:

In the first two assignments, we studied the empirical performance of searching and sorting algorithms respectively. In this assignment, we turn our attention to matrix multiplication, one of several problems that we studied in our lecture on 'Numerical Algorithms". In particular, we will consider five approaches:

1. numpy_mult (wrap the function provided by Python's famous NumPy package)
2. listcomp_mult (use Python's nested list comprehension)
3. simple_mult (our first approach, typically taught in Discrete Math and Linear Algebra)
4. divconq_mult (our second approach, basic divide-and-conquer algorithm)
5. strassen_mult (our third approach, Strassen's creative/improved divide-and-conquer algorithm)

For all approaches, do not pass in the dimensions; those can be easily obtained from the matrices themselves. When using code from public sources, please remove extraneous code and comments, and keep in mind that you will already have your own code to generate and print matrices in tasks 2-3.

Submissions:

In the Google form, please submit the following:

- Assignment4.py (source code)
- Assignment4.txt (console output including table)
- Assignment4.png (bar graph)

Tasks:

[0] Please consult Assignments #0, #1, #2 for a program template and structure, general guidelines about best practices, as well as sources of and policies concerning using public code.

[1] If you are programming in Python, you will want to use numpy, a powerful package for numerical and other array-related calculations. Write this:

```
import numpy as np
```

[2] Define a function `random_matrix(mn, mx, rows, cols)` that returns a matrix of random integers in the range(mn, mx) inclusive. You can code a nested loop, or use list-comprehension like

```
matrix = [[random.randint(mn, mx) for col in range(0, cols)] for row in range(0, rows)]
```

For the purpose of this assignment, we will call this function with parameters *`random_matrix(-1, 1, n, n)`* where n is the size of the matrix.

[3] Define a function `print_matrix(matrix)` that nicely prints a matrix. Several suggestions at

https://stackoverflow.com/questions/17870612/printing-a-two-dimensional-array-in-python

[4] You may find it useful to convert the 2-D list to a 2D array. See

https://stackoverflow.com/questions/7717380/how-to-convert-2d-list-to-2d-numpy-array

[5] Define a function numpy_`mult(m1, m2)` that wraps the matrix-multiplication functionality built into Python's numpy package: `np.matmul(m1, m2)`.

See https://numpy.org/doc/stable/reference/generated/numpy.matmul.html

[6] Define a function listcomp_`mult(m1, m2)` that uses nested list comprehension.

See https://geekflare.com/multiply-matrices-in-python/

[7] Define a function `simple_mult(m1, m2)` that uses three nested loops.

See https://www.geeksforgeeks.org/c-program-multiply-two-matrices/

[8] Define a function divconq_`mult(m1, m2)` that uses the basic divide-and-conquer approach.

See https://www.geeksforgeeks.org/strassens-matrix-multiplication/

[9] Define a function `strassen_mult(m1, m2)` that uses Strassen's divide-and-conquer approach.

See https://www.geeksforgeeks.org/strassens-matrix-multiplication/

[10] Define a function `verify_result(m1, m2)` that checks if the product is in fact the correct one

[11] Run each algorithm for matrices of sizes n = 10, 20, 30, …, 100. (If 100 doesn't take that long, you can push higher.)

[12] Summarize the results in a table using Pandas Dataframe or equivalent. Place the algorithms along the left side and the different sizes across the top.

[13] Visualize the statistics in a graph using matplotlib or another graphic package. Place all sorts in the same graph for easy comparison. Use a different color for each sort and create a color legend either in or below the graph.