JDEP 283 Homework 2 Fall 2008

Due Thursday, October 23, 2008

Name _____

Instructions Follow instructions *carefully*. This is an individual homework consisting of two parts; a set of analytical problem from the class text, Levitin, and a programming assignment. For the analytical part, you should solve all of them and submit them individually on or before the due date. Clarity and legibility of presentation of your submission are as important as your answers to problems. You are very strongly advised to typeset your solution using some document processing system. If the grader cannot easily read your writings, you may not be awarded full points even if you claim your answers are correct. Staple this cover page to the front of your assignment for easier grading. For the programming part, be sure to submit all your program files via the webhandin. You should follow JDE coding standard which is available at the course homepage `http://www.cse.unl.edu/~vinod/jdef08/info.html`.

Analytical Problems

Problem	Page	Points	Score
3.2.4	106	5	
3.2.9a	106	5	
3.3.2ab	112	10	
4.2.5ab	134	5	
4.3.6	139	5	
5.2.1	170	5	
5.2.4	171	5	
5.3.9	176	10	
Program			
Correctness		25	
Style/Documentation		5	
Analysis		20	
Total		100	

Programming Assignment The goal of this programming assignment is to implement and compare two matrix multiplication algorithms:

- Brute-Force Multiplication (Levitin 2.3, and)
- Strassen's Multiplication Algorithm (Levitin 4.5)

It is required that you implement the following to recieve full credit.

- A `Matrix` class that takes as aruguement in it's constructor an n by m array of integers. This class will have the following two methods:
  - **multiBF(Matrix multiplier)** which will print to stdout the resulting array and will return a two element array, the first element being the number of additions/subtractions and the second the number of multiplications used in the calculation of the product of the two matices using the brute force multiplication technique.
  - **multiST(Matrix multiplier)** which will print to stdout the resulting array and will again return a two element array, the first element being the number of additions/subtractions and the second the number of multiplications using the Strassen's Algorithm.

  Other methods may be implemented but these two are required. Catching arrays with improper dimensions without terminating (unless it is the last trial) is also required. It is critical that the matrix class handle the issue of deep copies. This is transparent to the driver and the test class.
- A driver class used to test your matrix class on randomly generated arrays that will print out the results of your experiments. This class (are you ready for it) can be called what ever you choose.

- A test class that take as input a file name. This input file will contain the a series of values for arrays preceded by a code as shown in the example provided. The code indicates what is to be done with the following array. This class must be called TestArray. The input file name must be command line input (not by prompt).
  - **base** - the following numbers are used to create a base matrix object.
  - **BF** - the following numbers are used to populate an array to be passed to the last created base matrix and multiplied using the brute force multiplication method.
  - **ST** - the following numbers are used to populate an array to be passed to the last created base matrix and multiplied using the Strassen multiplication algorithm.

  An example input file with the expected output is given at the end of this handout.

As in the last assignment, you will compare both algorithm's performances on various test cases, report the results and discuss these results in your analysis. The choice is yours, but you are highly advised to run as many test runs on as many randomly populated matrices on as many different sizes of matrices as you can. There are no set requirements other than you *must* make them reasonable and justify them in your write up. CPU time should also be compared. Provide a table similar to the one you did for the last assignment.
As before, points will be awarded based on the following categories:

- **Correctness** – Your code must compile and execute as expected. You must implement the correct algorithms, follow naming conventions, etc.
- **Style/Documentation** – Your code should be readable, properly indented and spaced with sufficient comments to explain. Good Object Oriented style should be followed including separation of header/implementation files.
- **Analysis** – Your analysis should reflect your understanding and critical analysis of the algorithms as well as demonstrate that you gave some thought to the assignment beyond mindlessly coding the algorithms. Discuss how each algorithm works, their asymptotic characterization, difficulty in coding, strengths, weaknesses, etc. For each criterion, (additions, multiplications, CPU time) compare and contrast your empirical results with each algorithm's asymptotic characterization, with each other, and with regard to increasing data set size. Were you able to find the "crossing" point at which the speed of the recursive algorithms surpassed the overhead? Which criteria are more relevant and in what context? Are there any other criteria that could be explored? Did the data match the theoretical asymptotic behavior? How and why or why not? Discuss any modifications that you tried to the algorithms and justify why you did so. Discuss anything else that you feel is relevant, pitfalls that you faced, etc. Turn in your analysis as a hard copy with the rest of your assignment.

SAMPLE: the XX in the output should be replaced with the counts your algorimths generated.

Input

```
Base
3       5       11      13
1       2       1       2
BF
4       1
2       8
1       2
2       1
ST
1       2       2
0       1       2
0       1       1
1       2       1
Base
1       2
2       1
1       2
ST
1
2
BF
2       4       4
1       2       1
```

Output

```
59      78
13      21
adds    XX
multi   XX
16      48      40
3       9       9
adds    XX
multi   XX
5
4
5
adds    XX
multi   XX
4       8       6
5       10      9
4       8       6
adds    XX
multi   XX
```