Florent Dondjeu Tschoufack
COSC 320 – 002
Project 1
Mar 15, 2020

Instructions: In order to compile and run the program, type "make" and two executables, driver and iomodel, will be created. To run driver, simply type "./driver" and to run iomodel, type "./iomodel [name of data file]". For iomdel, you must pass in the 2nd arguments or the program will not work.

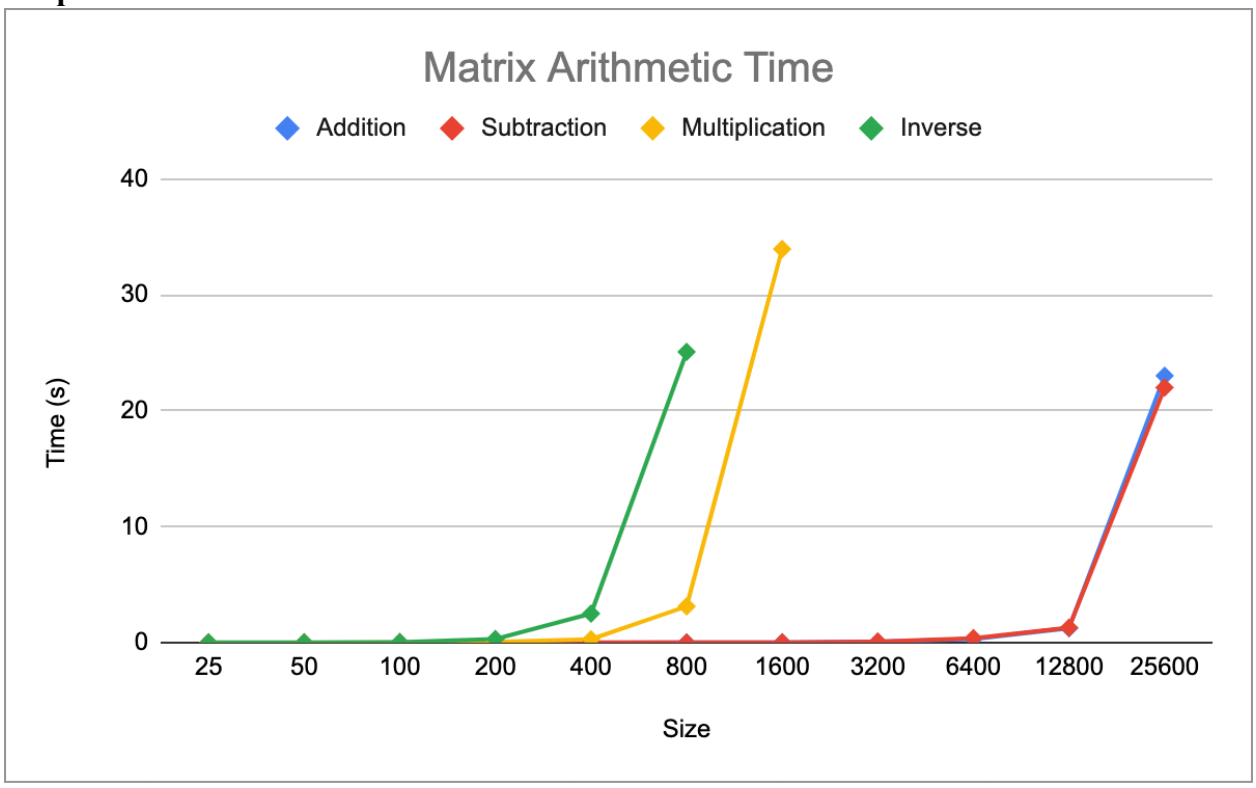
Driver

Procedure:
In order to collect my data for both addition and subtraction, I used a for loop in which the size ranges from 100 to 25600, where size is doubled after each loop. Inside the loop I created two randomly generated matrices A & B and I used <chrono> to time C=A+B or C=A-B. Similar to addition and subtraction, for multiplication, size ranged from 50 to 1600, where size is doubled after each loop, and <chrono> is used to time C=A*B. To time inverse operation, size ranged from 25 to 800, where size is doubled after each loop and one randomly generated matrix A is created and <chrono>is used to time A^{-1} = A.Inverse().

Table:

Size	Addition	Subtraction	Multiplication	Inverse
25	0	0	0	0.01
50	0	0	0.001	0.004
100	0	0	0.005	0.032
200	0.001	0.003	0.051	0.305
400	0.002	0.005	0.282	2.497
800	0.005	0.005	3.125	25.079
1600	0.021	0.018	33.954	
3200	0.071	0.083		
6400	0.306	0.375		
12800	1.241	1.283		
25600	23.016	22.001		

Graph:



Discussion:

In order to implement both Matrix addition and subtraction, I used double nested for loops so I could conduct the necessary operations to find the sum or difference of two matrices. With the use of two nested for loops, the time complexity for each matrix multiplication is $\theta(n^2)$. My graph does express this this time complexity. In order to implement Matrix multiplication, I used triple nested for loops unlike the addition and subtraction. The third loop is used to help us multiply (A*B) row in A by column in B. This gives us a time complexity of $O(n^3)$. When looking at the graph there is a distinct notice on how fast the time increases as the size of the matrices increase as well. In order to implement Matrix inverse, I used a multitude of variations of addition, subtraction and multiplication thus, bringing the time complexity to $\Omega(n^3)$ if we do not take into account all of the comparisons which run at a constant time.

In conclusion, for all of the arithmetic equations described above, the running time is directly proportional to the input size. Not only will it take a significant amount of time to compute two huge matrices, but it will also take a lot of time to generate each dynamically allocated element in the matrices.

Iomodel

Procedure:
If there are no arguments passed in or there are more than two, and error message will be outputted, and the program will end. If the argument passed in is empty, the program will output an error message and the program will end. If there was no prior error and we assume the data file syntax is correct, the program will store the names of the products into a key with the first name having the highest priority. Depending on how many names where present, a square matrix is created with its dimensions being the number products that were counted. Then the matrix is filled. Next an identify matrix is made with the same dimension as the previous matrix. Then another matric with column size 1 and row equal to the number of products counted. The matrix is then filled and the necessary arithmetic operation to determine how much product is needed is handled. The names in the queue are then dequeued into an array and the result is then output accordingly to the console.

Discussion:
This program should not have any problems or limitations because I can easily store many products names using priority queue. As the number of input increase, so will the time to calculate the amount of each product need.