King Fahd University of Petroleum & Minerals
Information and Computer Science Department

ICS 353: Design and Analysis of Algorithms

Term: 181

Implementations of Matrix Multiplication Algorithms Programming Assignment Report

Group #15

Doctor in Charge:
Dr. Wasfi AlKhteeb

Group Members:

Name	ID
Yousef Majeed	201568070
Abdulrahman Abdulmohsen	201475900

Monday, December 10, 2018

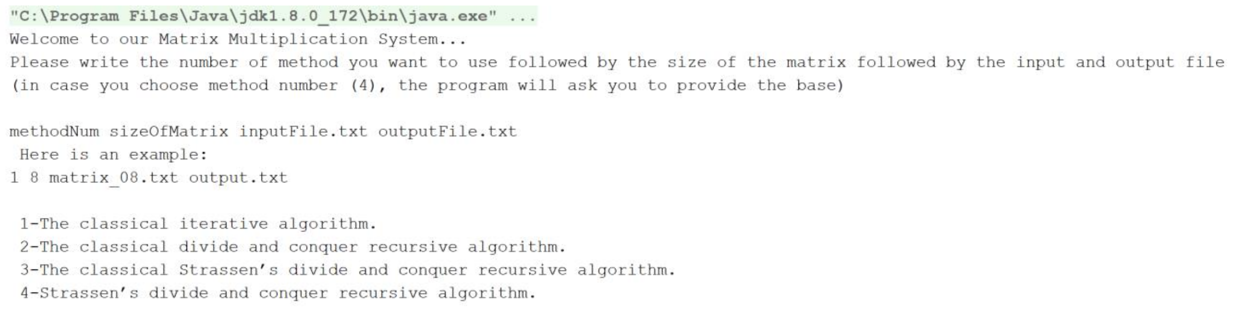
"Group #15" ® - *"Success is a journey not a destination!"*

Table of Contents

Implementations of Matrix Multiplication Algorithms

1. How to compile and run the code

Frist you have to open the java file that will be attached using any IDEs and start running the program then the program will prompt a menu that you can choose which method you want to use also give you instructions of how you will enter the size of the matrices you want to run the algorithms in, the input file and the output file. The image attached below will illustrate more.



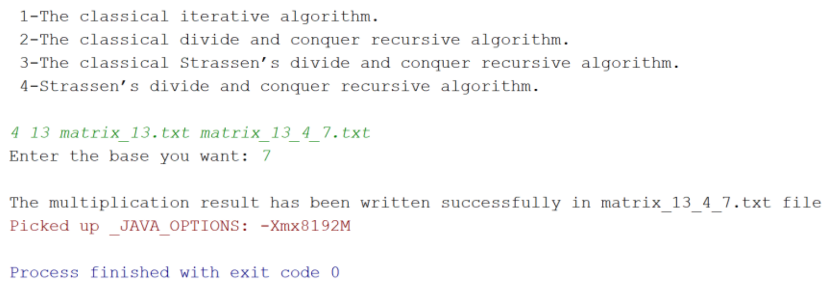
```
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe" ...
Welcome to our Matrix Multiplication System...
Please write the number of method you want to use followed by the size of the matrix followed by the input and output file
(in case you choose method number (4), the program will ask you to provide the base)

methodNum sizeOfMatrix inputFile.txt outputFile.txt
 Here is an example:
1 8 matrix_08.txt output.txt

 1-The classical iterative algorithm.
 2-The classical divide and conquer recursive algorithm.
 3-The classical Strassen's divide and conquer recursive algorithm.
 4-Strassen's divide and conquer recursive algorithm.
```

As you have seen in the above image this will be displayed after running the compiler.

- First, write the number of methods that you want to use then add space.
- Second, write the size of the matrices that you want to multiply (n which will be 2^n inside the method), then add space.
- Third, write the input file name with extension (txt) and it must be saved in the input folder inside the project file, then add space.
- Forth, write the output file name with extension (txt), and it will be saved automatically after the program finish in the project directory.
- Fifth, click enter, in case you choose method number (4) the program will ask you to enter the base that you want to procced.
- Finally, the program will run, and the result will be written with the execution time in the output file and the command will show you a successful message as shown below.



```
 1-The classical iterative algorithm.
 2-The classical divide and conquer recursive algorithm.
 3-The classical Strassen's divide and conquer recursive algorithm.
 4-Strassen's divide and conquer recursive algorithm.

4 13 matrix_13.txt matrix_13_4_7.txt
Enter the base you want: 7

The multiplication result has been written successfully in matrix_13_4_7.txt file
Picked up _JAVA_OPTIONS: -Xmx8192M

Process finished with exit code 0
```

As shown above, the green line is the user input that have to be followed. Also, if you are testing matrices with large input n, you have to configure your maximum heap size of your environment, visit the link below to see how https://youtu.be/x75szwof54w

2. Documentation of all experiments in the form of a comparative table.

Execution Time	Size of the Matrix 2^n					
	n = 2	n = 4	n = 5	n = 6	n = 7	n = 8
Classical Iterative	0.0s	0.0s	0.002s	0.005s	0.016s	0.047s
Classical Recursive	0.0s	0.0s	0.008s	0.046s	0.291s	1.786s
Classical Strassen	0.0s	0.0s	0.009s	0.031s	0.141s	0.516s
Strassen (Base = 2)	---	0.0s	0.005s	0.016s	0.062s	0.203s
Strassen (Base = 3)	---	0.0s	0.001s	0.015s	0.015s	0.076s
Strassen (Base = 4)	---	---	0.0s	0.008s	0.031s	0.046s
Strassen (Base = 5)	---	---	---	0.0s	0.016s	0.047s
Strassen (Base = 6)	---	---	---	---	0.016s	0.031s
Strassen (Base = 7)	---	---	---	---	---	0.031s

Execution Time (s)	Size of the Matrix 2^n					
	n = 9	n = 10	n = 11	n = 12	n = 13	n = 14
Classical Iterative	0.427s	6.999s	1.575m	15.561m	2.502h	
Classical Recursive	12.373s	1.596m	13.21m	1.71h	15.03h	>24h
Classical Strassen	3.174s	19.936s	2.33m	18.88m	2.05h	
Strassen (Base = 2)	1.16s	6.502s	46.033s	5.63m	38.93m	
Strassen (Base = 4)	0.312s	1.431s	9.068s	1.116m	7.873m	
Strassen (Base = 5)	0.25s	1.067s	7.466s	54.993s	6.384m	
Strassen (Base = 7)	0.25s	1.139s	8.060s	1.013m	7.040m	
Strassen (Base = 9)	---	1.815s	13.048s	1.572m	12.878m	
Strassen (Base = 11)	---	---	---	10.739m	1.581h	

3. Analysis of the results present in the table

The result tables in section 2 show that the classical divide and concur algorithm is the worst algorithm in time performance in all experiments. This is expected because of the overhead of the recursive concept. As we know that recursive functions affect the performance negatively because of the high number of function calls and the overhead of handling and passing the parameters in use. Also, because the overhead that comes of the stack parameters.

The Strassen's divide and conquer recursive algorithm with base case n >1 gave the best result in performance with a big difference in results compared to the other algorithms especially when the matrices are big. These results are expected since this algorithm takes the advantage of reducing the number of multiplications by one and also it minimizes the recursive calls by using the iterative algorithm in its base case. We are going to make analysis of the bases in the next section.

We expect that the classical Strassen's divide and conquer recursive algorithm will give better results than the iterative in all experiments since it reduces the number of multiplications by one. However, the results of the experiments show that in small size matrices the iterative algorithm is faster than the classical Strassen. This because of the overhead of the recursion. As we increase the size of the matrices the difference between the classical Strassen and the iterative in time execution gets smaller. As you can see from the table when n =13 the classical Strassen was faster than the iterative by about half an hour but in all previous results the iterative was faster. The reduction of the number of multiplications in the classical Strassen overcome the overhead caused by the recursion when n is large.

4. Analysis of the results for different base values for Strassen's algorithm

As the result tables show that when the base of the Strassen algorithm is small the execution is slow. This is because of the more recursive calls that are done when the base is small. Also, when the base is large, the execution time is slow. This is because we use the iterative version in the base case which is not efficient in multiplications of large size matrices. The results show that Strassen algorithm performs the best results when the base case value of n is around 5.

5. Conclusions

5.1 Of when to use each algorithm

If the matrices have small sizes n < 5 we should use the iterative algorithm. However, for n>=5 the best choice is Strassen's divide and conquer recursive algorithm with base case value of n=5. In addition, if the matrices are big n >= 13 it is better to use classical Strassen than the iterative but of course is not better than Strassen's divide and conquer with base case value of n=5. The classical divide and concur algorithm is not recommended at all.

5.2 About the best Strassen's base case

As the result tables show that in almost all experiments the base case value of n = 5 gave the best results in time execution.

6. Work Distribution

TASK RECORD

Date	Member	Task Details
29 November 2018	Yousef	Starting the project, do some most initial functions
6 December 2018	Abdulrahmen	Implement classical divide and concur algorithm and Strassen with base n >1 algorithm.
6 December 2018	Yousef	Implement classical iterative algorithm and classical Strassen algorithm.
7 December 2018	Yousef	Running all the test cases for analysing
10 December 2018	Abdulrahmen	Analyse the test case and document the analysis results
10 December 2018	Both	Finalize Work and Finalize the Report

Team Member	%
Yousef Majeed	50%
Abdulrahman Abdulmuhsen	50%

Note: both team members were aware of what is happing and verify the work.