Project 1 – Matrix Manipulation and Sorting

Presented to: Professor Phillip Servio

Prepared By:

Bizhan Alatif (260907005) – bizhan.alatif@mail.mcgill.ca

Ngan Jennifer Tram Su (260923530) – jennifer.tramsu@mail.mcgill.ca

CHEE 390 – Computational Methods in Chemical Engineering

Department of Chemical Engineering

McGill University

Fall 2021

Table of Contents

1. Objective

The objective of this report was to implement a program that sorts a 3D matrix in two steps: (1) sort each page of the matrix in ascending order of their trace (sum of diagonals), and (2) apply quicksort to sort each individual plane.

2. Flowchart

Figure 1 follows the path of an input 3D matrix through the implemented sorting algorithm.

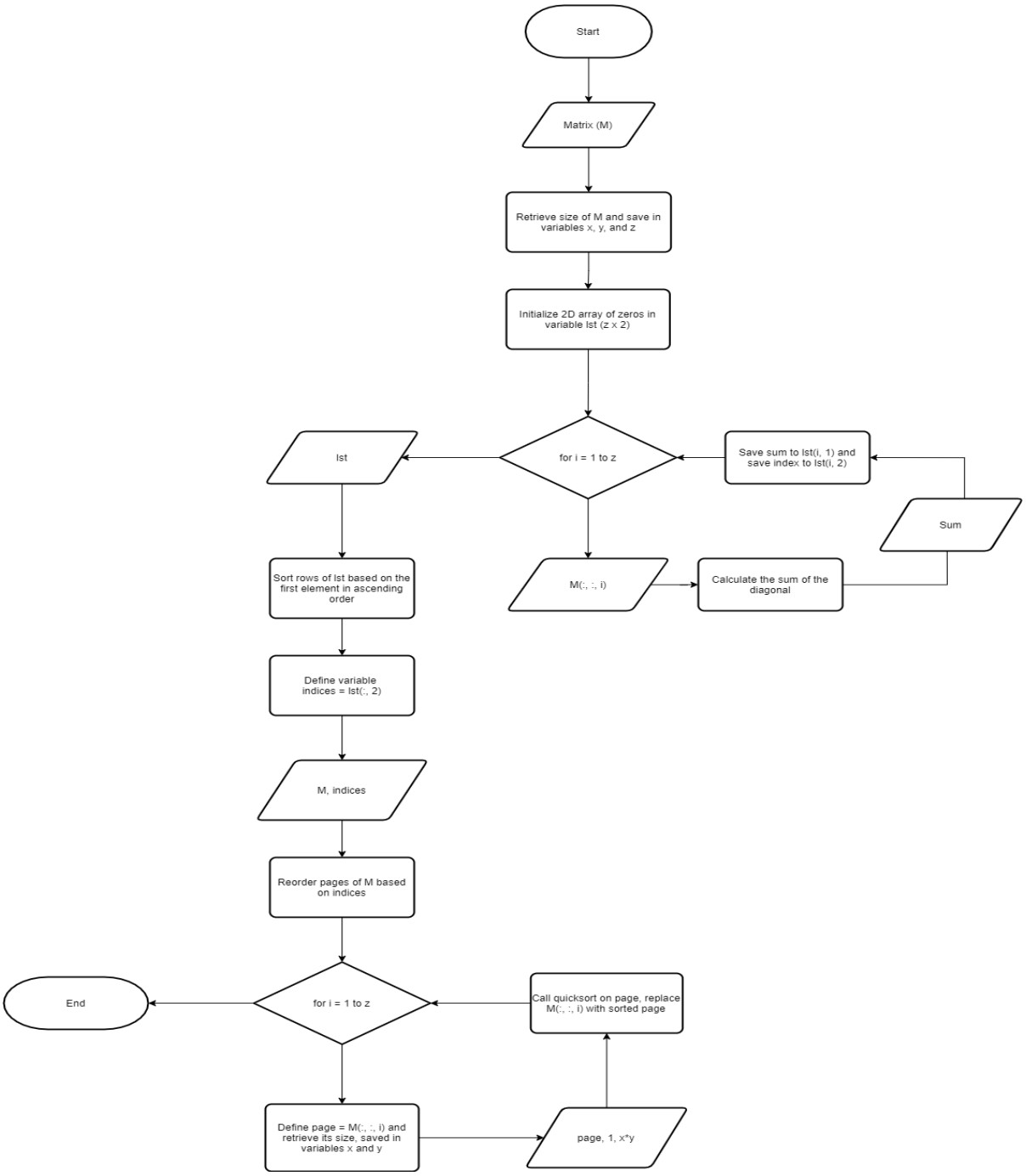

Figure 1 Quicksort Algorithm Flowchart

3. Results and Discussion

3.1 Pivot Selection

In computer programs, time complexity denotes the amount of time, relative to the input size, needed for an algorithm to reach its intended goal. Big-O notation is used alongside complexity to indicate the growth function of the complexity component (i.e. time). For instance, an algorithm with a time complexity of $O(n^2)$ means that for some input size X, the number of steps required to complete the program is X^2. Evidently, for larger input sizes, this time complexity could lead to astronomical computing times; therefore, minimizing it is favorable.

For a quicksort algorithm, the best-case scenario for time complexity is $O(n \log(n))$ and the worst-case scenario is $O(n^2)$. One way to avoid the worst-case scenario is by pivot selection. If a poor pivot is chosen, the algorithm may iterate through each element of the array recursively (e.g. if the input array is practically sorted, a central pivot would significantly increase the number of steps as the algorithm compares nearly every element to the pivot). Consequently, using a randomly selected pivot drastically reduces the likelihood of a worst-case scenario, providing an average-case scenario of $O(n \log(n))$ (i.e. the best-case scenario). This is because randomly assigning a pivot reduces the likelihood of getting a poor pivot for those outlier input arrays (e.g. a central pivot for a nearly sorted array).

3.2 In-Place Sorting

When creating an array in MATLAB, the assigned array is stored in a unique block of memory, with a separate block of memory holding information describing the array (e.g. dimensions) [1]. When a copy of an array is created, no new block of memory is made; however, once changes to this copy are made, the information of this new modified array is stored in a separate memory

block, with an additional memory block storing the array's information. As a result of re-assigning previously created arrays and, subsequently, modifying them, more blocks of memory are filled.

In the case of sorting, recursion is integrated into the algorithm to ensure a fully sorted array. If new sub-array variables were created and sorted separately from the parent array (i.e. new memory blocks filled with information of these sub-arrays), there's a higher likelihood of reaching the worst-case scenario in space complexity (i.e. the algorithm's memory usage relative to the input) [2]. If this occurred, the processing speed would drop, leading to a much slower computing time [3]. As a result, in-place sorting is used, where the parent array is directly modified and sorted, with no new intermediary copies created – minimizing space complexity.

3.3 Arithmetic Overflow

Computers represent information with binary digits, otherwise known as bits. Having a base of 2, these binary digits can be converted to base-10 to represent integers. However, limitations in memory prevent computers from fully expressing every value in the integer domain. These limitations are commonly bounded by 32 or 64 bits, depending on the computer. For example, the largest positive (negative) value that can be expressed with 32 bits is 2,147,483,647 (-2,147,483,648) [4]. Any attempt to express a larger numeric value will result in arithmetic overflow. Simply put, arithmetic overflow occurs when a computer attempts to store a numeric value that exceeds the bounds of its memory.

In the case of quicksort, arithmetic overflow may occur during pivot selection. A common pivot choice is the middle index of the array, whose value can be obtained by equation (1).

$$pivot = \frac{right + left}{2} \quad (1)$$

However, should the matrix be sufficiently large, the summation may result in a numeric value whose binary representation exceeds the computer's memory, leading to arithmetic overflow. Consequently, (assuming only positive values) the program will return a negative index [5] and throw an error as MATLAB does not support negative indexing. Ideally, this must be avoided so that the quicksort algorithm can be generalized for an input 3D matrix of any size. Certain pivot selections can avoid this problem. Arithmetic overflow never occurs when adding two numbers of different signs. Therefore, pivot selection methods that follow this property may be chosen to safeguard against arithmetic overflow (e.g. random selection).

4. Conclusion

Of the available sorting methods, the algorithm presented in this report follows the quicksort method. With a time complexity of $O(nlog(n))$, quicksort is a fast divide-and-conquer method that outperforms many others. Selecting a pivot, this method compares values on either side of it, swapping them until the end condition is met, and recursively calls the method on the two resulting subarrays. To avoid the time complexity worst-case scenario, the implementation presented in this report uses a random pivot selection; this also avoids the case of arithmetic overflow as integers of the same sign are never added. Avoiding the space complexity worst-case scenario is accomplished via in-place sorting, which reduces the quantity of memory usage as no array variables are created in the recursive algorithm.

5. References

[1] *How MATLAB Allocates Memory*. MATLAB & Simulink. (n.d.). Retrieved September 27, 2021, from https://www.mathworks.com/help/matlab/matlab_prog/memory-allocation.html.

[2] Baeldung. (2021, August 2). *Space complexity*. Baeldung on Computer Science. Retrieved September 27, 2021, from https://www.baeldung.com/cs/space-complexity.

[3] DELL. (n.d.). *How random access memory (RAM) affects performance*. How Random Access Memory (RAM) affects performance | Dell Canada. Retrieved September 27, 2021, from https://www.dell.com/support/kbdoc/en-ca/000129805/how-random-access-memory-ram-affects-performance.

[4] IBM Corporation. (n.d.). *Signed and Unsigned Integers*. IBM. Retrieved September 21, 2021, from https://www.ibm.com/docs/en/aix/7.1?topic=types-signed-unsigned-integers

[5] Alam, M., & Alam, B. (2015). *DIGITAL LOGIC DESIGN* (1st ed.). PHI LEARNING.