LIBRARY ALGORITHMS

ALGORITHMS ANALISYS OF THE PROJECT

TABLE OF CONTENT

INTRODUCTION

Librarytech contacted the company because their e-library repository has grown, both in people who come in to look for a book and in the entry of more books to their library, they need a new method of organization for their books and enhance the experience of their users.

For us the most important thing is to meet the requirements of the client, which we will implement the respective necessary solutions, for this we will use the java technologies with the Apache NetBeans IDE software and we will discuss the time and space complexity of the algorithms and sort methods that have been chosen for the implementation of this software as well as its performance.

PART.1: EXPLICATION OF PROCESS SOLVING THE PROBLEM

To start the development of this software, the first step is to understand all the requirements from the client about the software. We start separating every requirement into smaller pieces, imagining the outputs that should be showing on the command line and the general structure that our code should have, to make it easier to handle and to give a clearer representation of how the user can interact with it.

1.1. How the data is process

Whoever uses the software could enter the name of the data file with unknown number of entries to be processed, this data should be processed in 2 types:

- Book titles
- Page numbers

To solve the first point, we created a new class in Java called *CA_1_Methods* where all the general methods used within our software will be located and implemented in certain parts of it. Once inside this class we created the *sizeData method* where the file is read and returns the number of entries that contains, then we created another method called *readFile* which using the command *promt* asks the user for their file name. If this is found we use the *sizeData* method which defines the exact size of our arrays where we store the information.

For this we process the file again but this time we used the *lastIndexOf* that is a Java String Method gives the exact position of the last time a specific character appears, taking as character (",") the last comma it finds. Once identified, all text before comma is stored in our first array using *substring* which takes a specific part of another string and trim which removes what won´t be used.

Thus, storing the name of the books in this array, same is done but with the part after the comma, removing what is before it. After this we proceed to clean the information of the book names to give a better appearance when presenting them in the command line, with this we finish our method which returns our information in 2 arrays of strings, to properly process this, we will pass the one that contains the information of the page numbers to the type of data integers with the help of our *intTrans* method in this way we have an array of strings that contains the names of the books(data[0]) and another array of integers where the numbers of the pages(*numberPages*) of these are found.

1.2 Information and user interaction

To give to the user the freedom to choose what likes to do, a MENU was made with a switch of 4 different cases in our main class of the software, using a *while loop*, that is shown when a data file has been processed correctly, otherwise our software will end. These four cases are as follows:

1. Option of sorting books by titles, it shows another menu that is inside the Java Class SelectionSort, this has the same structure but now with 3 cases asking the information to be displayed on ascending or descending order or return to the previous menu. When is decided the type of sorting, our software automatically processes the 3 methods simultaneously sited at the Java classes with the same name *selectionSort*, *QuickSort* and *BubbleSort* respectively, these are different each other but works similarly using the data of the book titles as a reference and mainly use the *compareToIgnoreCase* Java String Method which compares strings lexicographically ignoring case differences. In the menu structure it has been settle starting and ending timers, this to measure the

performance of them and to show the individual processing times and printing the best method that performs this task, it takes the first 50 elements from the array that has been sorted as per user selection and are shown on the screen. For this a method was created in *CA_1_Methods* called *printArray* where the two arrays run together inside a *For loop* using a suitable structure so that when the result is shown to the user in a good-looking way to enhance their experience.

2. Exactly the same tasks are performed but according to the number of pages that books have just adapting the 3 sorting algorithms for integers instead of string. To do that the conditions are changed to compare just the numbers itself without using a Java method all of this to achieve the client requirements.

3. A searching option for a specific book that the user is looking for, the *searchingMethods* class was used, where respectively are the methods to perform *LinearSearch* and *BinarySearch*, these ask the user for the name of the book he is looking for. The title entered has to be exact since these methods take the input of the user and uses the *equalsIgnoreCase* method which compares two strings, character by character, ignoring whether it is uppercase or not and take it as the target to look for and returns the position of it, showing on screen the book title, the position and the number of pages it has. Both algorithms are processed simultaneously with the same input from the user to record the times and to show which one performs the task better.

4. Last case, used to exit the software, also uses a method located in the class of *CA_1_Methods* called *performanceResult*, which uses the results obtained on the previous processes and uses them to show us the general performance of these methods as a final result, these results are only shown if the previous processes have been used, otherwise it will only show an exit output message, this to solve the last requirement of the LibraryTech company.

PART.2: TIME AND SPACE COMPLEXITY OF ALGORITHMS

In order to meet with the requirements mentioned above, some algorithms were implemented which their performance was reviewed to have an idea of which one is the best to process this type of data files. In the table below is showed the data collected of each algorithm:

SORTING ALGORITHMS TABLE				
ALGORITHM	PERFORMANCE TIME (nanoseconds)	BEST CASE	AVERAGE CASE	WORSE CASE
Bubble Sort	18515100	O(N)	O (N^2)	O (N^2)
Selection Sort	13005600	O (N^2)	O (N^2)	O (N^2)
Quick Sort	1261900	(N log (N))	(N log (N))	O (N^2)

To organize our information within the software, the 3 methods shown in the table above have been implemented, to record the time, 30 tests have been carried out as a sample and the average of these has been taken. With the information we got from these algorithms and knowing how they work we can discuss further about their time and space complexity.

The first of the algorithms is *Bubble Sort*, which performs the work in a time of 18515100 nanoseconds, taking into account that due to it works, the average of its cases is O(N^2), even though is very versatile, for this sorting huge arrays takes a long time. This is because as the array size increases, the algorithm must compare each element in the array with every other element.

The second is *Selection Sort* which performs the work in 13005600 nanoseconds processing the information in any case in O (N^2) this gives it an advantage to process the information of some databases efficiently but like *Bubble Sort* this one is not so suitable for huge databases of data, but if it has a better performance for databases not as small as in this case that it has a better performance with respect to *Bubble Sort* but not against the third *Quick Sort* which performs the sorting in 1261900 nanoseconds processing the information in the average case in (N log (N)).

Taking all these parameters into account, *Quick Sort* reduces the number of comparisons needed. This optimized approach results in faster sorting times and is ideal for handling extensive datasets without excessive strain on the system's memory and processor, in contrast, *Bubble Sort* places a huge stress on computer resources when dealing with large datasets and selection sort takes more time to achieve the same task.

SEARCHING ALGORITHMS TABLE				
ALGORITHM	PERFORMANCE TIME (nanoseconds)	BEST CASE	AVERAGE CASE	WORSE CASE
Linear Search	14718000	O (1)	O(N)	O (N)
Binary Search	3265500	O (1)	O (log N)	O (log N)

For the -Searching- task on our software as we can see in the table above, *Linear Search* and *Binary Search* algorithms were used, which 30 records were taken as sample with an element in random position, for the first *Linear Search* we found that it performs the search in 14718000 nanoseconds and we took the average of the O(N) cases which is how it performs most of the time in our system.

On the other hand, we have *Binary Search*, which is an algorithm that in the average of its cases behaves as O (log N), what makes it more efficient because it has the data sorted already and divides the data set, checking if the searched value is located at that midpoint, if is not found there, it moves the start or end values depending on whether the divided point is. This is why with a time of 3265500 nanoseconds this algorithm is our best option as our -Searching- algorithm.

CONCLUSION

To sum up, and as was mentioned before on the previous sections, with taking every aspect of this algorithms *Quick Sort* and *Binary Search* are the quicker and effectives methods that can process the data for LibraryTech company with the best performance using less resources, memory, and time than the others algorithms method analysed before.

COMPUTER AND SOFTWARE ASPECTS

COMPUTER MODEL NAME: ASUS – TUF GAMING A15

PROCESSOR: AMD Ryzen 7 5800H with Radeon Graphics, 3201 Mhz, 8 Core(s), 16 Logical Processor(s)

RAM: 32 GB

SYSTEM TYPE: x64-based pc

OS NAME: Microsoft Windows 11 Home

Version 10.0.22621 Build 22621

Software: Apache NetBeans IDE 15

JAVA FILES UPLOADED FOR THIS SOFTWARE:

- CA_1
- CA_1_Methods
- SelectionSort
- QuickSort
- BubbleSort
- SearchingMethods

All of this were created in CA_1 package.

REFERENCE

baeldung, 2020. *Space Complexity | Baeldung on Computer Science*. [online] Available at: <https://www.baeldung.com/cs/space-complexity> [Accessed 20 October 2023].

Oracle Java, 2023. *API reference for Java Platform, Standard Edition*. [online] Available at: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html> [Accessed 14 October 2023].

Oracle Java Documentation, 2023. *Returning a Value from a Method (The Java™ Tutorials > Learning the Java Language > Classes and Objects)*. [online] Available at: <https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html> [Accessed 14 October 2023].

StudySmarter, 2023. *Algorithms: Definition, Examples & Data Structures | StudySmarter*. [online] StudySmarter UK. Available at: <https://www.studysmarter.co.uk/explanations/computer-science/algorithms-in-computer-science/> [Accessed 20 October 2023].

GeekForGeeks, 2021. Time Complexity and Space Complexity. *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/time-complexity-and-space-complexity/> [Accessed 17 October 2023].

GeekForGeeks, 2014. QuickSort - Data Structure and Algorithm Tutorials. *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/quick-sort/> [Accessed 17 October 2023].