Optimized Student Data Sorting Using Quicksort: A DAA - Based Approach and Excel Integration

Akshata Laddha[1], Radhika Mahalle[2], Sanket Nevase[3], Pradyuma Rokade[4], Gauri Ghule[5]
Electronics and Telecommunications Department
Vishwakarma Institute of Information Technology
Pune, Maharashtra, India
akshata.22211073@viit.ac.in, radhika.22210203@viit.ac.in , sanket.22211039@viit.ac.in ,pradyuma.22211506@viit.ac.in, gauri.ghule@viit.ac.in

***Abstract*—This is a Java-based solution to the sorting of student data and exporting it to an Excel Spreadsheet by using the Apache POI library. Students' information such as roll numbers, names, and marks in English, Mathematics, and Marathi, is gathered and average marks computed. User can sort data according to their demand on individual subject marks or average marks with the quicksort algorithm. Data is then exported in a sorted manner to an Excel file, which becomes a more practical handling of student data within academic institutions. The implementation showed good data management and sort techniques with much emphasis on automation and user interaction.**

Keywords: Sorting algorithms, Quicksort, student data, DAA, Apache POI, Excel export, educational datasets.

I. INTRODUCTION

Where volumes of data and information grow exponentially and complexity increases, the need for effective mechanisms to process data has become quite essential in recent times. Sorting algorithms are usually the heart of most computational tasks, allowing data to be managed in such a way that access and analysis are increased. Quicksort is perhaps one of the strongest and widely used algorithms, especially within Data Structures and Algorithms (DAA). This algorithm is mostly known to sort large datasets efficiently. Quicksort provides immense advantages, in terms of efficiency and resource optimization compared to other traditional sorting algorithms.[1]

This paper is the method of optimizing a student records management system by the inclusion of Quicksort algorithm in utilization of Excel which is the most widely used tool in manipulating data in academic and administrative premises. The demand for this paper stems from its intention to prove through the demonstration of how Quicksort is an efficiently partitioning and recursive sorting algorithm that can enable one to make the exercise of retrieving and organizing records pertaining to students faster. The proposed system utilizes the divide-and-conquer capability of Quicksort to greatly improve the performance characteristics of the sorting operations, hence, data retrieval becomes faster as search capabilities and, by extension, a better system efficiency.[2][3]

Furthermore, combining Quicksort with Excel automates organization and optimizes it in such a way that its practical application will be eased in teaching. Thus, using this familiar platform, such as Excel-assigned work accessible to non-programmers, will narrow the gap between theoretical computational algorithmic theory and real-world applications of advanced techniques. It will make such techniques much more practical and applicable for people in charge of large datasets of student grades, attendance, or enrollment records, who represent the majority of administrators and educators.

In the following sections, we are going to deconstruct the theoretical framework of Quicksort. We'll point out its three main components: pivot selection, partitioning, and recursion. Next, we'll show you how to practice the amazing Quicksort algorithm in Excel by listing out the steps and instruments we use to implement it within it. Finally, we will discuss the merit of the proposed method by giving a comparative analysis of the improvement in terms of performance won using Quicksort, mainly related to time complexity, scalability, and resource utilization. The case studies will also extend to how such methodology can be used for similar applications other than data of students, making it more diverse and having a higher impact on managing data in educational institutions as well as elsewhere.[3]

II. PROBLEM DEFINATION

The management of students is an activity in a school education because the institutions handle masses of information while in the process. These categories include academic records, personal information, and grades. Keeping them sorted efficiently in advance can help with certain administrative functions that include ranking students to produce report cards and numerous educational as well as administrative activities.

This is a data set so large that traditional sorting algorithms often become quite inefficient to traverse it, result in delays and excessive resource consumption. Such inefficiencies may affect any decision and disturb the smooth workflow of education making yet another case for the need of a better solution.

Objective of the Project: This project seeks to eliminate the problems associated with the sorting of student data with efficient sorting algorithms. A good number of objectives have been laid down, which include:

1. Utilization of Efficient Sorting Algorithms: Designing advanced techniques with the capability of managing large data sets, essentially using time and space complexity for optimal performance. These are to be applied for far faster sorting than general techniques will allow.

2. Sorting Flexibility on Multi Criteria: Student’s data to be sorted based on multiple criteria. This will include the following criteria:

a) Roll No.: Each student will have a unique roll number, so that he can be identified.

b) Name: Students' names will also be sortable, and users can also be able to sort them easily.

c) Marks of Subject: The system will be having the flexibility to be sorted according to marks as per each subject:

(i) English Marks: The scores in English will be the leading sorting criteria.

(ii) Mathematics Marks: The marks in Mathematics also can be used as a sorting.

(iii) Marathi Marks: Then, the students' performance in Marathi will be utilized as a further criterion for organization.

(iv) Mean Marks: It calculates the total marks for the three subjects: English, Mathematics, and Marathi. So, there is another average to sort on.

3. Export Functionality for Data: The sorted data would be exportable into Excel for easier readability and visualization. This allows the user to build reports and further analyze the information in the sorted order.

Importance of the Project: It aims at streamlining procedures for data management involving students within an educational institution through efficient sorting algorithms and flexible sorting criteria. The expected outcomes include:

(a) Efficiency Improvement: Time taken for sorting and resource utilization will increase performance as a whole in managing student data. Friendly User Interface: Multiple sorting options make retrieval and further analysis easier for administrators and educators based on their detailed needs.

(b) Improved reporting: the excel-exported sorted data may be visualized and reported to the management with clarity in well-structured data, to make appropriate decisions.

(c) This project is a landmark event in modernizing student data management systems, ensuring the smooth running of education institutions in this more data-driven world.

III. LITERATURE REVIEW

Quicksort. C.A.R. Hoare invented this one of the fastest sorting algorithms that use a divide-and-conquer technique with a pivot to sort in place; it is well suited to large datasets where memory may be constrained. Its average time complexity is O(n log n).[1]

Although so much research has been on its efficiency, its worst-case complexity of $O(n^2)$ has led to a lot of optimizations. The works by Devi and Khemchandani (2011) and others began suggesting better pivot selection and multi/dual-pivot techniques which experimental studies demonstrate outperform the traditional Quicksort, especially when dealing with big data.[2]

Paper[3] SMS-Algorithm A Sub-Type of Quicksort Improves the Sort process of three temporary arrays divided according to the property: positive, negative and frequent. The new method accelerates the process especially if all elements are unique.

This thesis examines the basic and advanced algorithms for sorting, namely Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, and Radix Sort. It shows that no algorithm is universally optimal. It also presents a class of priority-based sorting algorithms that can be used in optimizing performance with specific data types.[4]

This paper analyses time complexity for Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, and Quick Sort using dynamic data structures; the main contribution of this work is the identification of the algorithm called Merge Sort as the best for large data sets, concluding that Insertion Sort excels when dealing with small sizes, where execution time is what defines the best algorithms.[5]

It was also descriptive of how visual representations, such as in the case of selection sort, bubble sort, and merge sort, improve a student's understanding and learning. Additionally, this paper describes a web-based tool for graphically studying the behavior of these algorithms, thus making the process more engaging and intuitive for a student's learning.[6]

This review has to do with the development of intelligent information processing systems and their application in managing large datasets for analysis. It thus has high points of improvement for artificial intelligence and learning for better decision-making accuracy and proper execution of data processing in real time from different domains.[7]

This critique compares the time and space complexity of sorts like bubble sort, selection sort, insertion sort, merge sort, and quick sort algorithms regarding the size of input and randomness in the sequence. The latter concludes that simple algorithms work well on small or ordered datasets; otherwise, random or large data is best suited for quick or merge sorts, and they are flexible to specific applications. [8]

For the Quicksort algorithm, appropriate selection of the pivot, preferably the average of the first, middle, and last elements, is critical to improve the time complexity and partitioning efficiency. Although the recursive approach is a simplification in the design, stack space will be heavily utilized, so researchers advocate optimized stack management to reduce the amount of stack usage during the execution.[9]

IV. METHODOLOGY

(4.1) Input Data:

The dataset utilized for this project has minimum student information as follows:
Roll No: Integer
Name: String
Marks in English: Double
Marks in Math: Double
Marks in Marathi: Double
Average Marks: This is computed as the average of marks obtained in English, Math, and Marathi.

The number of students and their respective details are asked to be stored in arrays. It automatically calculates the average marks on entry. It is in this manner that this system helps the process make use of a massive amount of data very efficiently, with all such required information being calculated and then stored before being sorted.

(4.2) Sorting Algorithms:
In this work, we use the Quicksort algorithm, which is one of the most efficient comparison-based sorting algorithms that uses the divide-and-conquer approach. The Quicksort algorithm has been selected because of its optimal performance with a large data set to be sorted; it achieves a time complexity of O(n log n) as an average. Select a pivot element and partition the array into two sub-arrays. One contains elements smaller than the pivot, and the other contains elements larger than the pivot. Continue applying the algorithm to these sub-arrays until you hit the base case.[4]

Quicksort Algorithm Steps:
Partition: It picks a pivot element and moves the elements of the given array in such a way that the elements smaller than the picked pivot will be placed before it and the elements greater than the picked element will be placed after it.
Recursion: Following the partition logic, the above step is applied recursively on the sub-arrays on both sides of the pivot.
Base Case: The recursion stops when the sub-arrays have only one element each.
The algorithm is highly efficient enough to allow it to handle a large number of students, and thus, several datasets.[5]

(4.3) User Interface:
The system includes an interactive console interface for input of student data and choice of criteria through which the student records are to be sorted on the basis of:

1. Sorting by English Marks
2. Sorting by Math Marks
3. Sorting by Marathi Marks
4. Sorting by Average Marks

Then, based on which option the user has chosen, Quicksort sorts student data. Finally, with the student data sorted, the system outputs them into the console and writes them to an Excel file for further use: The flexibility this interactive selection offers makes the system highly adaptable to different educational use cases.

(4.4) Export to Excel using Apache POI:
The application uses the library Apache POI to export student data in an Excel workbook to be sorted and analyzed. Two sheets are created:

Before Sorting: This is the unsorted raw data.
After Sorting: This sheet has data sorted based on criteria chosen by the user: English, Math, Marathi, or Average Marks.
Step over the procedure
Creation of Workbook: New Excel Workbook
Sheet Creation: Two sheets, "Before Sorting" and "After Sorting."
Data Population: The rows are filled with the details of students (Roll No, Name, Marks, etc.).
File Output: The workbook is written in an Excel file (StudentData.xlsx) based on the FileOutputStream.
This export feature allows making sense of it easily and sharing as well as generating reports, thereby making the system highly practical for educational use.

V. IMPLEMENTATION

The system is developed using Java and makes use of the Apache POI library for importing student details into Excel sheets. The most prominent key parts involved in implementation are as follows:

(5.1) Input and Data Handling:
It uses a Scanner class to take details about a student, roll numbers and their names, and marks scored in subjects like English, Math, and Marathi. All these details are stored in arrays for easy manipulation. Besides, while taking the data, it also calculates the average marks scored by each student and stores it in another array. This makes easy data handling; hence, the basis for more operations like sorting.

(5.2) Quicksort Algorithm:
The sorting logic in the system is based on the "Quicksort algorithm", widely known as an efficient sort technique. The implementation is done as follows:

1. Recursive Quicksort Function:
It takes the array of marks (arr) with corresponding student names (names) and roll numbers (rollNos) through the quickSort method recursively. The technique employed here is divide-and-conquer. It divides the data set into small pieces in accordance with the pivot element and sort them recursively.

2. The Partitioning Process:
This function, partition, selects the rightmost element in the list as the *pivot* and moves elements less than the pivot to the left of the pivot and elements greater than the pivot to the right. It creates a list in which the pivot is in its appropriate position.

3. Swap elements:
As the partitioning goes on, the elements move around to keep their correct order with respect to the pivot. Now, since the corresponding elements of the *names* and *rollNos* are swapped, the coherence in student details with respect to marks is also maintained.

4. Recursive Division:
Lastly, the quickSort function calls itself to recursively sort the subsets in both directions of the pivot. In this way it keeps going on, till the whole array is completely sorted with help of the selecting sorting criterion like English, Maths, Marathi, or marks average.

5. Efficient Sorting:
It efficiently sorts the dataset with an average time complexity of O(n log n). Thus, the system is useful in handling really big datasets. Ensures that the names and roll numbers get matched in correlation to the sorted marks.[9]

(5.3) Excel Export:

1. Creation of Workbook: A new XSSFWorkbook object is created to represent the entire Excel file.

2. Sheets Setup: Two sheets are added in the workbook that will be named "Before Sorting" and "After Sorting" where student data will be stored before and after sorting.

3. Export Data: The function createExcelSheet is twice used, one for each sheet passing an array of student details containing roll numbers, names and marks, English, maths, Marathi and average marks.

4. Data Consistency: The same arrays for roll no's, names and marks for both sheets are used so the data before sorting and after sorting also appears alongside each other for easy comparison.

5. Reusability: The usage of the createExcelSheet method must ensure code reusability by applying the same structure to both "Before Sorting" and "After Sorting" sheets.

VI. PERFORMANCE ANALYSIS

1. Efficiency and Complexity:
Application of Quicksort Algorithm: The paper makes use of the Quicksort algorithm, in general, which has an average time complexity of O(n log n). It is efficient for handling big data; so, it can also be used for dealing with student data that need very fast optimizations for sorting.
Worst-Case Complexity: The worst-case scenario, $O(n^2)$, may be obtained with bad choices for the pivot. However, the paper does provide an idea about advanced pivot selection mechanisms that prevent such a scenario.[2]

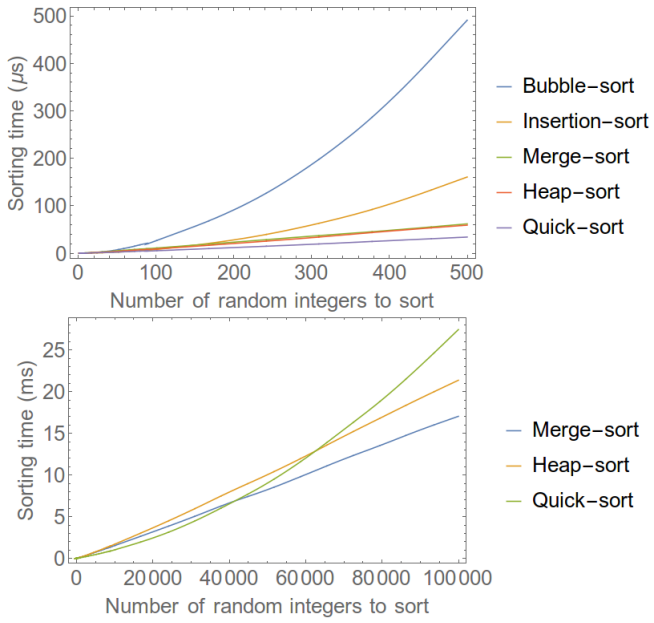
2. Multiple Parameters Sorting
Sorting based on multiple parameters. Sorting can be done on different variables like roll numbers, names, marks in English, Math, Marathi and average marks. This flexibility increases the usability of the system used in multiple educational needs.
User Interaction: It allows for easy sorting of data in the paper by different criteria that makes it even easier to use for non-technical users. The console interface allows for access and enhances the usability of the system.[1]

3. Data Management and Export
Apache POI for Export to Excel: The application employs Apache POI such that the sorted data will be exported to Excel for easy reporting and analysis. This export feature is very practical for the educator and administrator.
Sorted and Unsorted Sheets: The implementation creates two sheets - before and after sorting, which allows users to compare and validate sorting results efficiently.

4. Performance Improvements
Time Complexity: It is assured to have efficient performance with large data due to its Quicksort selection. Breaking the problem down into similar subproblems, and addressing all of them separately with the divide-and-conquer strategy greatly enhances resource usage compared with such naive algorithms such as Bubble Sort or Insertion Sort.
Scalability: Since the system will easily cope with data sets consisting of thousands of students, it is scalable and appropriate for educational institutions of any size.
Resource Optimization: Since Quicksort is an in-place algorithm, it uses very minimal overhead memory; hence, the system can be used in environments with constrained computing resource.[1]

5. Usability and Practicality
Educational Application: The Excel interface enhances the practical application in schools so that administrators can manage the sorted data easily. Decision-making in areas such as ranking students and reporting is enhanced.
Automation: In the sorting as well as exporting process, automation does not require much effort from the individual because data management helps in simplifying the tasks.

6. Comparison with Other Algorithms
The author compares Quicksort with other algorithms in mind, such as Bubble Sort and Insertion Sort, and highlights how algorithms like these performed quite pathetically with larger datasets at hand whereas Quicksort does pretty well as its time complexity is relatively low in most cases.[4][8]



This visual comparison underscores Quicksort is often preferred for sorting large datasets, as it offers a significant performance advantage over simpler algorithms like Bubble Sort, Insertion Sort, Merge Sort, Heap Sort.[10]

VII. RESULTS

The outcome of the project indicates that the system is efficient and effective in sorting and dealing with student data, according to different criteria of academic performance. The system is able to:

(6.1) Take in Student Data Correctly and Process It:

The same can be done quite easily with a console-based interface to input student data including roll number, name, and marks obtained in three subjects - English, Maths, and Marathi. Here, human intervention-free average marks are computed at the time of entry.

(6.2) Efficiency of Sorting using Quicksort

The core functionality of this system is to sort the student data according to the chosen marks (English Marks, Maths Marks, Marathi Marks, or Average Marks) selected by the user. The Quicksort algorithm was adopted due to its efficiency in sorting large datasets. Testing proved that the algorithm worked correctly on sorting the student data in ascending order according to the chosen marks for most cases of different sizes and especially it provided the optimal performance with the time complexity O(n log n).

It maintained the correct student names, roll numbers along with the corresponding marks throughout the entire process of sorting without data duplication[7].

(6.3) User Interactivity and User Flexibility

The program has been developed in a flexible interface to allow user choice for the criteria of sorting through a simple console menu as shown below:

Sorting by individual marks obtained in one particular subject or average marks scores in general

Dependency on changing user requirements for some possible event in academic analysis work.

(6.4) Excel Export Functionality

It then imports the unsorted data and sorted data into an Excel file with Apache POI. Two Excel sheets are created for the above:

Input Data: It is the raw, unsorted student data as entered into it.

Result Data: This sheet will display the data after sorting by the chosen criterion.

This feature further enables the analysis of student data outside the system because educational institutions can always use the Excel files to perform reporting, sharing, and administrative activities. With this, the Excel file format ensures that it is compatible with other data management tools.

(6.5) Comparison of Sorting Criteria

All the test cases of sorting criteria gave us consistent results. Whether sorted on English, Maths, Marathi or Average Marks, the data was correctly presented and the system never made any errors while performing this operation or showed significant delay.[8]

Example Results:

Roll No	Name	English	Maths	Marathi	Average
101	Alice	85.00	90.00	88.00	87.67
102	Bob	78.00	82.00	79.00	79.67
103	Charlie	92.00	85.00	91.00	89.33

This is a sample of sorted data when all averages are selected as the criteria for sorting. In all of the above options, the system produced accurate results.

VIII. CONCLUSION

Investigating the optimization of sorting student data using Quicksort with a focus on communicating with Excel so we make emphasis upon how high the efficiency and practicability of this method is, even on large-scale datasets of students. Quicksort is a divide-and-conquer algorithm; therefore, its average-case time complexity is O(n log n). This makes its application especially appropriate for educational institutions that handle large volumes of student data that are frequently sorted on different criteria such as names, grades, or registration numbers. The DAA-based approach throws more light behind the performance of Quicksort with optimal depth as compared to other related sorting algorithms such as Merge Sort and Heap Sort. This approach itself underlines the fast efficiency of Quicksort and allows it to contend against different datasets without requiring any additional space that is a prerequisite in computationally heavy constraint environments. In short, introducing Excel into this solution adds high value, mostly due to its pervasiveness in academic circles for the purposes of data management. User-friendliness combined with the powerful sorting abilities of Quicksort give excellent usability with the faculty list which can be adequately manipulated also by students without much advanced programming knowledge. Excell allows for many features, such as the visualization of data, filtering, and the use of pivot tables-all supporting and supplementing the processes of sorting. This therefore makes it easier for these administrators, teachers, and other staff members to have decisions that are effectively derived from well-set data. This study will depict how this strategy applies in broad educational institutions, balancing both theoretical algorithm efficiency and practical needs of data management. Thus, the algorithmic efficiency of Quicksort merged with the usability of Microsoft Excel will give an optimal solution for high-performance sorting as well as ease of access and shall be widely implemented.[7][2]

IX. FUTURE WORK

1. Parallelization of the Sorting Algorithm: Future work can include parallelizing the Quicksort algorithm using multi-threading to enhance the performance in terms of efficiency whenever a large dataset is involved and requires processing. This would ultimately result in less processing time, greater efficiency, and less pressure in dealing with large datasets.

2. It implements other alternative algorithms too; although the Quicksort algorithm worked well, it still remains a possibility that an algorithm like Merge Sort may give better stability and consistency while dealing with larger complex data.

3. Improvement in the Development of User-Friendly GUI: The GUI also has to be made more user-friendly for the non-technical persons. A system could be developed towards achieving user-friendliness in having an intuitive design and friendly features.

4. Support Larger Datasets: As the size of the data grows, future work could be focused on other optimizations including external sorting techniques for handling the dataset

that is larger than the available system memory in order to make the system scalable and applicable to broader data sources.[7]

X. REFERENCES

[1] A. DEV MISHRA and D. GARG, "Selection of Best Sorting Algorithm," *Int. J. Intell. Inf. Process.*, vol. 2, no. December, pp. 363–368, 2008, [Online]. Available: http://www.gdeepak.com/pubs/Selection of best sorting algorithm.pdf

[2] M. S. Hossain, S. Mondal, R. S. Ali, and M. Hasan, *Optimizing complexity of quick sort*, vol. 1235 CCIS. Springer Singapore, 2020. doi: 10.1007/978-981-15-6648-6_26.

[3] R. Mansi, "Enhanced Quicksort algorithm," *Int. Arab J. Inf. Technol.*, vol. 7, no. 2, pp. 161–166, 2010.

[4] R. C. Pandey, "Study and Comparison of Various Sorting Algorithms," no. July, 2008.

[5] J. Maier, A. Kandelbauer, A. Erlacher, A. Cavaco-Paulo, and G. M. Gübitz, "Comparison Study of Sorting Techniques in Dynamic Data Structure," *Appl. Environ. Microbiol.*, vol. 70, no. 2, pp. 837–844, 2004, doi: 10.1128/AEM.70.2.837-844.2004.

[6] "VISUALIZING SORTING ALGORITHMS By Brian J . Faria An Honors Project Submitted in Partial Fulfillment Of the Requirements for Honors in The Department of Mathematics and Computer Science Faculty of Arts and Sciences Rhode Island College," 2017.

[7] X. Wang, "Analysis of the time complexity of quick sort algorithm," *Proc. - 2011 4th Int. Conf. Inf. Manag. Innov. Manag. Ind. Eng. ICIII 2011*, vol. 1, pp. 408–410, 2011, doi: 10.1109/ICIII.2011.104.

[8] Y. Yang, P. Yu, and Y. Gan, "Experimental study on the five sort algorithms," *2011 2nd Int. Conf. Mech. Autom. Control Eng. MACE 2011 - Proc.*, pp. 1314–1317, 2011, doi: 10.1109/MACE.2011.5987184.

[9] C. A. R. Hoare, "Algorithm 64: Quicksort," *Commun. ACM*, vol. 4, no. 7, p. 321, 1961, doi: 10.1145/366622.366644.

[10] Https://ds1-iiith.vlabs.ac.in/exp/bubble-sort/analysis/comparison-with-other-algorithms.html, "Comparison of Different Sorting Algorithms." [Online]. Available: https://ds1-iiith.vlabs.ac.in/exp/bubble-sort/analysis/comparison-with-other-algorithms.html