Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 45 No. 2 (2024)

Analysis and Comparison of Various Sorting Algorithms

Yashvi Singh [1], Minal Verma [2], Ishika Pandey [3], Anshika Saini [4], Prerna Chawla [5], Vandana Niranjan [6]

[1, 2, 3, 4, 5, 6] Department of Electronics and Communication, IGDTUW, Delhi-110006, India

Abstract:- Sorting algorithms are crucial in various aspects due to their significant role one of the most commonly used techniques. As we deal with data and handle large datasets on a day-to-day basis for various purposes such as analysis, comparison display (in ascending or descending order), and data processing, the need arises for data to be sorted efficiently. This allows for the fast and efficient retrieval of useful information from large collections of data. Sorting large datasets enhances data accessibility, improves system performance, and enables better decision-making processes. Sorting algorithms play a vital role in many areas like data mining and managing databases and information retrieval. In this study, we provide the core advantages and disadvantages of each modern sorting algorithm, such as Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, and Heap Sort, which are frequently used to sort large datasets in order to retrieve information efficiently and quickly. Through this analysis, we aim to identify the most appropriate algorithm for a given scenario. Each technique's specific functionality, importance, advantages, and disadvantages have been discussed in detail. We also provide the approximate time taken by each algorithm, known as time complexity, to provide a better understanding of their performance. Additionally, we demonstrate the practical implementation of different algorithms and calculate the time taken by each to display data in sorted order. Sorting algorithms facilitate faster search operations, enable easier data analysis, and support more effective data processing workflows. Overall, the ability of sorting algorithms to handle large datasets efficiently is essential for optimizing data management and analysis tasks in modern computing environments.

Keywords*:* Time Complexity (TC), approximate (approx), environments (env), frequently (freq), implementation (impl.), algorithm (algo).

1. Introduction

Getting a loan should be an easy and convenient task, especially in this era of technology.

However, currently getting a loan requires the manual comparison of the interest rate from multiple banks, which can be tedious and time-consuming. And comparing interest rates from multiple banks without the proper knowledge and experience can be challenging as well. To tackle this problem an innovative solution has been built known as Lender which is a bank recommendation system based on the interest rates provided by various banks for various kinds of loans. This model along with providing the comparison of various interest rates also compares the various sorting algorithms such as Selection sort, Bubble Sort, Insertion Sort, Merge Sort, Quick Sort and Heap Sort. The model compares the time complexity of various sorting algorithms to sort the interest rates and then uses the method which takes the least time to sort the data.

In this paper we are presenting the pros and cons of each sorting algorithm along with the comparison of their time complexity using the model.

2. Various Sorting Algorithms

A. Bubble Sorting Algorithm

Bubble sort presents a straightforward approach to sorting a list by repeatedly checking neighbouring items in a list. If they're out of order, it swaps them [1]. This continues until the entire list is in the correct order, like putting your groceries away one by one, with smaller elements gradually moving to their correct positions, akin to bubbles rising to the surface. The algorithm employs two nested loops [2]. The outer loop traverses the list, starting at the beginning and stopping at the element one position before the end. Nested within this outer loop, the inner loop iterates from the beginning of the list up to, but not including, the current index of the outer loop. This ensures each element is compared to its subsequent neighbours.

Advantage: Bubble sort is renowned for its simplicity, making it an excellent starting point for those new to sorting algorithms. Its straightforward nature allows for easy understanding and implementation, serving as a foundational concept in algorithmic studies [1]. Beyond its simplicity, Bubble Sort boasts a memory-frugal design [2]. Unlike some algorithms, it operates directly on the original data set, minimizing the need for additional storage allocations. Furthermore, Bubble Sort exhibits a degree of adaptability. In scenarios where the input data is already partially sorted, the algorithm can leverage this inherent order, potentially reducing the overall number of comparisons required for complete sorting, potentially achieving a linear time complexity in these cases. This adaptability gives it an edge in scenarios where data is already somewhat ordered.

Disadvantage: Despite its simplicity and adaptability, bubble sort suffers from inefficiency when dealing with larger datasets [2]. Its runtime complexity presents a significant limitation. The time required to complete the sorting process scales quadratically with the size of the data set (O(n^2)). This translates to a rapid decline in performance as the number of elements to be sorted increases. Additionally, its performance is inconsistent, particularly struggling with reverse-sorted lists. This lack of scalability and poor performance on larger datasets limit the practical use of bubble sort in favour of more efficient sorting algorithms like quicksort or merge sort.

B. Insertion Sorting Algorithm

Insertion sort is a well-established sorting technique that relies on an iterative approach to build the final sorted array incrementally [3]. It works by systematically examining each element in the unsorted portion of the data set. This element is then compared with the elements in the already sorted portion of the array. The algorithm strategically inserts the element into its rightful position within the sorted section, potentially shifting existing elements to the right to maintain order. This process ensures that the array remains sorted after each insertion, ultimately leading to a fully sorted array at the conclusion.

Advantage: Insertion sort offers a significant advantage in its inherent understandability and straightforward implementation [4]. The algorithm is straightforward to understand, making it accessible for beginners and useful for educational purposes. Insertion sort thrives on pre-sorted or small datasets, leveraging existing order for efficient sorting. In such cases, insertion sort can outperform more complex algorithms due to its minimal overhead and low memory requirements. Insertion sort exhibits adaptability, capitalizing on inherent order in partially sorted data to minimize comparisons and swaps during the sorting process.

Disadvantage: Despite its simplicity and efficiency for small datasets, insertion sort has several disadvantages. While insertion sort excels with smaller or pre-sorted datasets due to its adaptive nature, its performance deteriorates for large collections. The time complexity of O(n^2) translates to a significant increase in comparisons and swaps as the number of elements grows [3]. This inefficiency makes it less suitable for large-scale sorting tasks where faster algorithms like quicksort or merge sort would be preferable. Additionally, insertion sort lacks stability when dealing with duplicate elements, potentially altering their relative order after sorting. This instability might be a concern for specific applications where preserving the original order of duplicates is crucial.

C. Selection Sorting Algorithm

Selection sorting is an iterative sorting algorithm that operates by repeatedly identifying the minimum element from the unsorted portion of an array and placing it at the beginning [5]. In this algorithm, the array is divided into two sub-arrays: the sorted sub-array at the beginning and the unsorted sub-array at the end. The basic concept behind selection sort involves selecting the smallest element from the unsorted sub-array and swapping it with the element in the current position. The sorting continues until the array is completely sorted [6].

Advantage: most Selection sorting is simple and easy to understand, making it an accessible sorting algorithm for everyone. Additionally, selection sort performs well with small datasets where it has a quadratic time complexity of O(n^2) which may not pose a significant performance issue [5].

Disadvantage: The algorithm exhibits a worst-case time complexity of O(n^2), indicating that its performance degrades significantly with larger datasets [6]. It is not an adaptive sorting algorithm, as it does not adjust its strategy based on the initial order of elements, leading to a consistent number of comparisons and swaps regardless of the input [5].

D. Quick Sorting Algorithm

Quick Sort is an effective sorting method in which we select a pivot element, and divides the other elements in the array into two sub-arrays [7]. The array is divided based on whether elements are smaller or larger than the pivot element. This procedure is applied repeatedly to the sub-arrays until the entire array is organized.

Advantage: Quick Sort stands out for its speed and memory efficiency. On average, it sorts data in O(n log n) time, placing it among the top sorting algorithms in terms of sorting speed[7]. This efficiency makes it a popular choice for many situations. Additionally, Quick Sort also excels in memory usage because it sorts data "in place." This means it sorts the data within the original array, minimizing the need for extra memory to store temporary copies. This combination of speed and memory efficiency makes Quick Sort a highly versatile and valuable sorting algorithm.

Disadvantage: worst-case time complexity of quick sort can go to O(n^2). If the pivot element selection is not good it can lead it to unbalanced partitions [8]. This sorting algorithm performance is heavily dependent on the choice of pivot element. Also, its recursive nature can lead to stack overflow errors with large datasets. While Quick Sort excels in sorting large datasets quickly and efficiently, it may not be the best choice for small arrays or when stability is a crucial requirement.

E. Merge Sorting Algorithm

The merge sort algorithm is known for its efficiency, simplistic implementation and effectiveness. It is a simpler-to-understand algorithm compared to any other divide-and-conquer algorithm, having its own wide-ranged applications. Working on the principle of divide and conquer, a paradigm where a problem is broken down, into smaller, more manageable sub-problems until they can be easily solved [10]. The key steps of the merge sort algorithm are as follows:

Divide: The unsorted list is divided into two equal halves recursively until each sublist contains only one element. This is the base case of the recursion.

Conquer: Once the base case is reached, the merging phase begins. Pairs of adjacent sublists are repeatedly merged into larger sorted sublists until the entire list is sorted.

Merge: During the merge phase, the sorted sublists are merged together by comparing the elements from each sublist and arranging them in the correct order.

Combine: Finally, the sorted sublists are combined to form the fully sorted list.

Advantage: Various databases and file storage Systems have a wide range of applications for merge sort. In order to remove the burdens of programmers and to give them quick access to items a modified version of the technique is used. When incorporating large arrays merge sort outperform insertion sort with time complexity of O(n logn). It even have the capacity to outperform Quick Sort, if dealing with a reversed list [9]. Merge sort's ability to gracefully handle large datasets ensures that no matter the size of the task at hand, it can efficiently organize and arrange data with precision and speed[10]. By disintegrating the sorting into smaller modules it minimizes the strain on computational resources while maintaining high performance and inturn making the entire process more manageable. Merge sort's ability to gracefully handle large datasets ensures that no matter the size of the task at hand, it can efficiently organize and arrange data with precision and speed[10].By breaking down the sorting process into smaller, more manageable steps, merge sort minimizes the strain on computational resources while maintaining high performance.

Disadvantage: When traversing through small lists merge sort consumes more time, thus making it slower and utilizes more stack space. Having O(n) complexity. Firstly, it requires additional space proportional to the input size, impacting memory usage and scalability. Secondly, being non-in-place, it demands extra memory for storing sorted subarrays, unsuitable for memory-constrained environments. Additionally, its stability incurs overhead, affecting performance, especially for small datasets. The recursive nature of merge sort also introduces computational overhead, potentially slowing performance [10]. Lastly, it lacks adaptivity to input order, unlike some other sorting algorithms, limiting its versatility in certain scenarios.

F. Heap Sorting Algorithm

Heap Sort is a classical sorting algorithm. Heap sort algorithms are faster than other sorting algorithms. when the input size is very large [11]. Heap sort stands out as a robust sorting technique, particularly adept at efficiently managing extensive data set. Heap sort's ability to leverage the hierarchical structure of heaps enables it to efficiently manage and sort large datasets, providing a valuable tool for data processing and analysis tasks.

In the realm of data, heap sort operates similarly [12].It constructs a binary heap from the unsorted data, leveraging the heap property to efficiently identify the maximum (or minimum) element and place it at the appropriate position The iterative process continues until the data reaches a fully sorted state[12]. By utilizing this hierarchical nature of heaps, heap sort minimizes the number of comparisons and movements required, making it particularly well-suited for handling large volumes of data.

Advantage: Heap sort emerges as a compelling choice for large-scale data sorting due to its confluence of advantages. Notably, its time complexity of O(n log n) guarantees efficient performance even with massive datasets[11]. Furthermore, heap sort exhibits robustness, maintaining consistent performance irrespective of the initial data order. This reliability proves invaluable in real-world applications with diverse input characteristics. Finally, heap sort operates in-place, minimizing additional memory requirements - a critical factor when dealing with large datasets and potential memory limitations [12]. Its stability and adaptability further enhance its utility, providing a robust solution for various sorting tasks on extensive datasets.

Heap sort typically requires O(log n) auxiliary space due to recursion, but an iterative approach can optimize it to O(1), ideal for memory-constrained environments or handling large datasets efficiently.

Disadvantage: Relatively While heap sort offers notable advantages, it also presents some challenges, especially when sorting large datasets [11]. Heap sort's memory footprint increases with the data size due to the additional space needed for the internal heap structure [12]. This additional memory overhead can become prohibitive for extremely large datasets, potentially limiting its scalability. Furthermore, despite its consistent time complexity, heap sort's performance may degrade in practice due to its non-adaptive nature, as it does not adjust its sorting strategy based on the characteristics of the input data. Heap sort lacks stability, potentially altering the original order of elements with equal values, which might be undesirable for specific applications. These limitations should be carefully considered when evaluating heap sort for sorting large datasets.

3. Representation of Analysis

Table-1: Sorting Algorithm vs Time

Sr.no.	Sorting Algorithms	Time
1	Selection sort	0.20000001788139343 ms
2	Insertion sort	0.19999998807907104 ms
3	Bubble sort	0.10000002384185791 ms
4	Heap sort	0.10000000894069672 ms

Sr.no.	Sorting Algorithms	Time
5	Merge sort	0.09999999403953552 ms
6	Quick sort	0.09999999403953345 ms

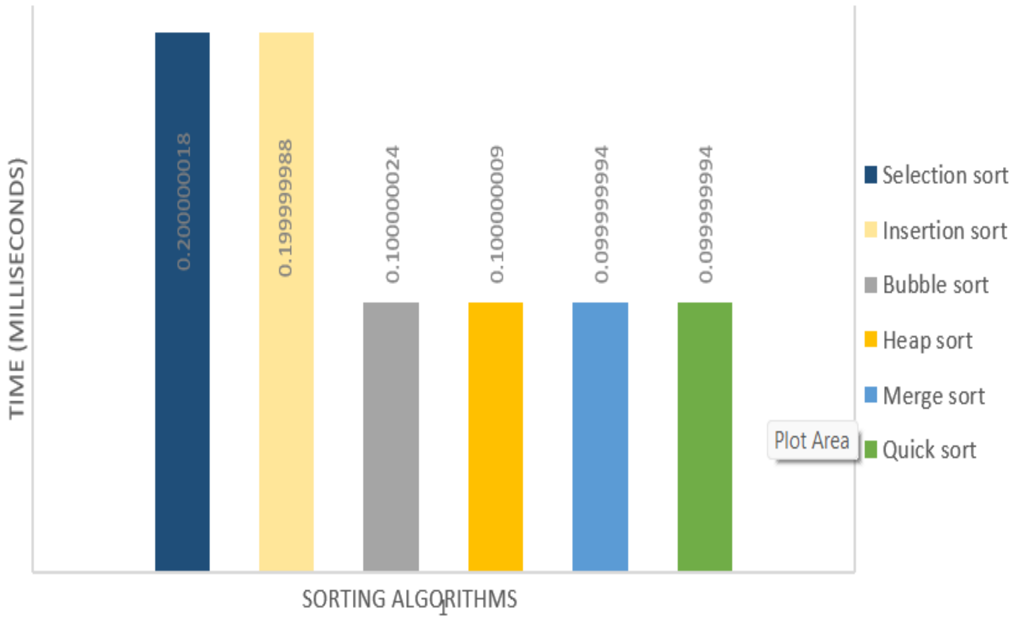

Fig.1. Graphical representation of Sorting Algorithm and Time

4. Conclusions

With the help of this model of comparing various interest rates of various banks, we have made a solid comparison of the various sorting algorithms. Though each sorting algorithm has its own pros and cons, some work better than others depending upon the type of data, the way data is arranged and also the amount of data used for the sorting. After careful analysis we conclude that Bubble sort is recommended for input data. Whereas if we consider large data input data, efficiency required for sorting increases. For large size input Bubble sort is not more efficient. Merge sort is very complex and does not work well on small data input. But perform well for large dataset. Insertion sort works well with both large and small data sets in best cases. For variable data selection sort in not good. Quick sort is highly complex but works well with large dataset. Thus we have finally concluded various sorting algorithms using our model of comparing interest rates of various banks.

References

[1] “Enhanced Efficiency in Sorting: Unveiling the Optimized Bubble Sort Algorithm”, Sep. 2023.
[2] Wang Min, “Analysis on Bubble Sort Algorithm Optimization”, 2010 International Forum on Information Technology and Applications pp. 1-1, June. 2010, doi: 10.1109/IFITA.2010.9.
[3] Ali Rauf, “Insertion sort”, National Library of Medicine, vol. 8,no. 3,pp. 153-153, January. 2015,doi: 291262143.
[4] Tarundeep Singh Sodhi, Surmeet Kaur, Snehdeep Kaur, “International Forum on Information Technology and Applications”,vol. 64, no. 21 , pp. 1-1, Feb. 2013.
[5] Sahil Mattoo, “Selection Sort Algorithm: Definition, Implementation, and Complexities”,Feb. 2024.
[6] Ravikiran A S,“What Is Selection Sort Algorithm In Data Structures?”, Oct. 2023.
[7] Harshil Patel, “QuickSort Algorithm: An Overview”, Dec2023.
[8] Simplilearn,” What is Quick Sort Algorithm: How does it Work and its Implementation”, Mar2020.

[9] Joella Lobo, “Performance Analysis of Merge Sort Algorithms”,”2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)”, July 2020,doi: 10.1109/ICESC48915.2020. 9155623.

[10] Smita Paira, Sourabh Chandra, Sk Safikul Alam,” Enhanced Merge Sort- A New Approach to the Merging Process”,Dec 2016, doi: 10.1016/j.procs.2016.07.292.

[11] Naeem Akhter, Muhammad Idrees, Furqan-ur-Rehman,” Sorting Algorithms – A Comparative Study”,” International Journal of Computer Science and Information Security”, vol. 14, no.12, Dec 2016.

[12] R.Schaffer, R. Sedgewick,”The Analysis of Heapsort”,”J. Algorithms”, July 1993,doi: 10.1006/JAGM. 1993.1031.