Comparative Analysis Between Selection Sort and Merge Sort Algorithm

Nahid Niaz, Jobayer Mansur Mufti, Md. Shakibul Hasan, Md. Abdullah Al Mamun, Abir Chandra Das, and Eazdan Mostafa Rafin

Department of Computer Science and Engineering, Rajshahi University of Engineering and Technology

January 26, 2024

Abstract

In the realm of Computer Science, datasets are needed to sort according to different criteria.In order to solve these sorting problems, different sorting algorithms have been developed such as selection sort and merge sort.in this paper selection and merge sort algorithms were implemented on an octa-core processing machine using the *std* :: *chrono*() function and C++ language to compare their respective execution time.The acquired results indicate that merge sort is more efficient and takes less execution time than selection sort, after testing several datasets of different sizes on the machine. For small input datasets, selection sort may be faster than merge sort. The central processing unit (CPU) plays a significant role in the performance of both algorithms.

Keywords: Algorithms, Sorting, Merging, Complexity, Execution Time.

Introduction

Sorting is the process of arranging data in a statistical order, which might be rising, decreasing, or lexicographical. In contrast, merging uses a divide-and-conquer strategy to sort a given array of components.

There are different types of sorting algorithms. Such as - insertion sort, bubble sort, selection sort, merge sort, quick sort, shell sort, tim sort, heap sort, and others. Each of them has a unique method for rearranging the elements, which improves the effectiveness and performance of the real-world applications while also making each one less time-consuming. A number of considerations need to be made when comparing different sorting algorithms. first of all, time complexity need to be considered as it establishes the maximum time that an algorithm can operate [1]. Many sorting algorithms, some of them are inefficient and excessively slow, depending on the size of data that needs to be rearranged. When expressing an algorithm's time complexity, big $O(n)$ notation is typically used, where O stands for complexity and n for the number of basic operations the algorithm performs [2]. Then comes stability, which refers to the algorithm's capacity to maintain elements with equal values in the same relative order in the output as they did in the input [3]. Lastly, memory space is evaluated; recursive algorithms require more copies of sorting data, which affects memory space [4].

For situations involving small data sets, selection sort works especially well when there are few components and time complexity is not a major problem. When it comes to embedded systems—which have limited memory and processing power—its efficiency and simplicity make it a sensible option. In such contexts, the small code footprint and clear implementation of this technique are beneficial. Selection sort is also useful because of its simplicity as a tool for benchmarking and testing alternative sorting algorithms. For more complex sorting techniques in particular, it provides a baseline comparison. Furthermore, the method performs very well when used with partially sorted data since it saves needless swaps in segments that have already been sorted, which speeds up the sorting process. When changing components results in a substantial expense, the selection sort's efficiency is further enhanced in certain scenarios, including handling small keys and huge records, by its ability to reduce the number of swaps. [5]

Merge sort is commonly used to efficiently sort big arrays of integers. Its versatility extends to external sorting, where it excels at managing enormous datasets that cannot fit into memory by separating and sorting smaller portions before merging them.

Merge sort is useful in database operations for sorting data before searches, combining datasets, and optimizing queries. Its parallel processing capabilities allow it to be partitioned into sub-operations, enabling simultaneous execution and improving overall efficiency in computational tasks. Furthermore, merge sort is important in computer graphics, notably in-depth sorting, where it controls the order in which objects are drawn to build a 3D image, ultimately improving the quality of computer-generated visuals [6].

Background Study

Selection sort

Selection sort is a simple sorting algorithm.It works by repeatedly finding the minimum or maximum element in the unsorted part of the array and placing it in the sorted part.In each pass, one element will be sorted. For n elements, n_1 passes are required. Firstly the algorithm sets the first element as minimum. Compare the minimum with the second element. If the second element is smaller than the minimum, assign the second element as the minimum. Compare minimum with the third element. Again, if the third element is smaller, then assign the minimum to the third element otherwise do nothing. Repeat this process until the last element. After each pass, minimum is placed in the front of the unsorted list. After each pass indexing starts from the first unsorted element. The algorithm can sort the array in ascending or descending order, depending on whether it finds the minimum or maximum element in each iteration [7]. This algorithm is not suitable for large data sets as its average and worst-case complexities are of $O(n^2)$, where n is the number of items and the space complexity is $O(1)$ [8].

Algorithm 1 Selection Sort

```
1:  procedure SelectionSort(a, n)
2:      for i ← 1 to n do
3:          j ← i
4:          for k ← i + 1 to n do
5:              if a[k] < a[j] then
6:                  j ← k
7:              end if
8:          end for
9:          t ← a[i]
10:         a[i] ← a[j]
11:         a[j] ← t
12:     end for
13: end procedure
```

Merge Sort

Merge Sort is a divide-and-conquer algorithm that works by dividing the array into two halves, sorts each half independently, and then recombines them. The array is first split in half, and then each half is subjected to a recursive procedure until no more elements can be split. The sorting procedure starts after the array is split up into its smallest components. Every element is compared to its neighbor; if the elements do not match in order, they are exchanged. Finally, the sorted halves are combined once more, and the procedure is continued until the array is sorted in its whole. The average time complexity of merge sort is $O(n \log n)$ where n is the number of items [9]. The auxiliary space complexity of this algorithm is $O(n)$ [10].

Algorithm 2 Merge Sort

```
1: procedure MergeSort(low, high)
2:     if low < high then
3:         mid ← ⌊(low + high)/2⌋
4:         MergeSort(low, mid)
5:         MergeSort(mid + 1, high)
6:         Merge(low, mid, high)
7:     end if
8: end procedure
```

Result and Analysis

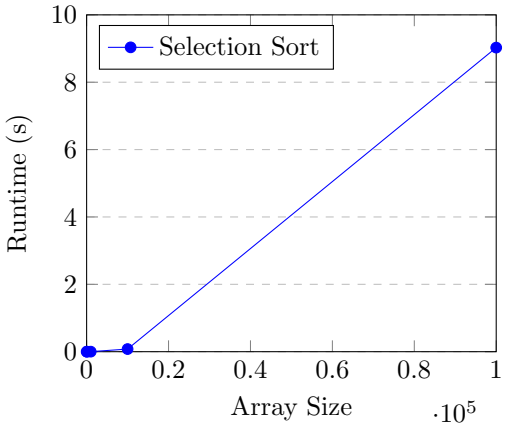
Running time of selection sort and merge sort algorithm using a different number of array sizes is shown in Table 1 and 2 respectively. And the graphical representation is shown in figure 1 and 2.

Selection Sort

Table 1: Running time of selection sort algorithm

Array Size	Runtime (s)
10	0.0000008
100	0.0000124
1000	0.001009
10000	0.0779707
100000	9.02775

Figure 1: Performance of selection sort algorithm



Merge Sort

Table 2: Running time of merge sort algorithm

Array Size	Runtime (s)
10	0.000001
100	0.0000055
1000	0.0000055
10000	0.0007045
100000	0.0809197

Selection Sort shows a steady rise in execution time with larger data sets.As its time complexity is $O(n^2)$.Merge Sort displays more steady and predictable behavior with a comparatively lower execution time rise as data set size grows, as it has better complexity than Selection Sort which is $O(n \log n)$. Table 1 and Figure 1 show that the running times of selection sort rise as the size of the array increases [11]. As a result, the durations required to sort a specific element in an array using selection sort are proportional to the number of elements in the array. When the data size is 10000, the time required for selection sort is 0.0779707 seconds, but the time required for merge sort is 0.0007045 seconds, which is 110 times less than the time required for selection sort. When the data size was extended to 100000, the execution time for the selection sort increased to 9.02775 and the execution time for the merge sort increased to 0.0809197, which is 111 times smaller than the execution time for the selection sort. Figure 3 shows that the Merge sort algorithm outperforms the Selection sort algorithm on both smaller and bigger array sizes throughout the sorting process.The performance of algorithms is affected by the array's input size [12].

Figure 2: Performance of merge sort algorithm

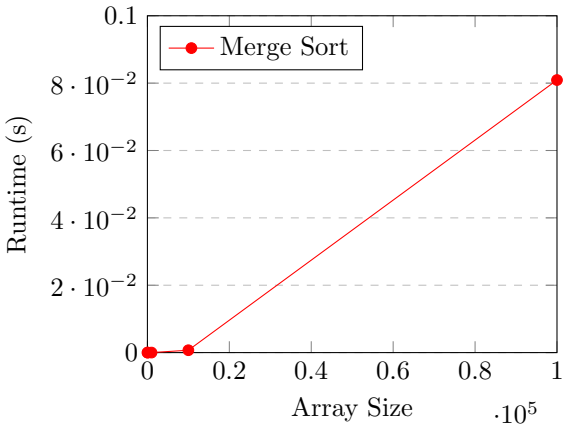
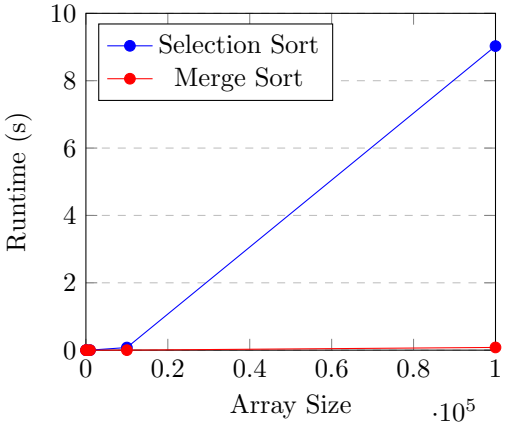


Figure 3: Performance comparison of selection sort and merge sort



Conclusion

According to the study's findings, Merge Sort is more efficient and takes less time to execute than Selection Sort for large datasets. Selection Sort, on the other hand, may be faster than Merge Sort for small input datasets. The order of complexity of an algorithm affects its efficiency. The central processing unit (CPU) is critical to the performance of both methods. If the number of CPU cores is increased, the running time of both algorithms would decrease. As a result, these and other popular algorithms require further investigation to check if they provide the same results when tested on different machines.

References

[1] M. Goodrich and R. Tamassia, "Data structures and algorithms in java," *John Wiley & Sons*, pp. 970–975, 2010.

[2] S. Jadoon, S. Solehria, S. Rehman, and H. Jan, "Design and analysis of optimized selection sort algorithm," vol. 11, no. 1, pp. 16–21, Feb 2011. [Online]. Available: http://www.ijens.org/IJECS%20Vol%2011%20Issue%2001.html

[3] H. Ahmed, H. Mahmoud, and N. Alghreimil, "A stable comparison-based sorting algorithm with worst-case complexity of $o(nlogn)$ comparisons and $o(n)$ moves," *World Academy of Science, Engineering and Technology*, vol. 22, pp. 970–975.

[4] M. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java*, 4th ed. John Wiley & Sons, 2010.

[5] "Selection sort in data structures." [Online]. Available: https://www.scholarhat.com/tutorial/datastructures/selection-sort-in-data-structures

[6] "Merge sort in data structures." [Online]. Available: https://www.scholarhat.com/tutorial/datastructures/merge-sort-in-data-structures

[7] "Selection sort - dsa tutorial." [Online]. Available: https://www.programiz.com/dsa/selection-sort

[8] "Selection sort algorithm - tutorialspoint." [Online]. Available: https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm

[9] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, 1998.

[10] "Time and space complexity analysis of merge sort." [Online]. Available: https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-merge-sort/

[11] R. et al., 2018.

[12] R. et al., Aliyu, and Zirra, 2020.