International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2025: 7.089

Performance Analysis of Sorting Algorithms in Context of Complexity

Dr. Mohit Kumar Sharma

Associate Professor & Head, Post Graduate Department of Computer Science, J.C. D.A.V. College, Dasuya, Punjab, India

Abstract: - Data structure is a systematic way to organize, manage and store data in a computer's memory so that it can be accessed and modified efficiently. Sorting is a fundamental operation in data structures that deals with arranging the elements of a list or array in a specific order. This order can be numerical, alphabetical, or any user-defined order. In computer science, sorting is not just about arranging numbers, it is about organizing data so that it can be used efficiently. In C programming, sorting algorithms are commonly applied to arrays using loops, functions, pointers, and sometimes recursion. Time complexity describes how the running time of an algorithm increases as the input size increases. Space complexity describes how much additional memory an algorithm needs while sorting. Bubble Sort, Selection Sort, Insertion Sort, and Quick Sort can generally be implemented in-place. The aim of this paper is to analyse a role of different sorting algorithms in context of time and space complexity.

Keywords: Time and Space Complexity: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort and Heap Sort

1. Introduction

Data structure is a systematic way to organize, manage and store data in a computer's memory so that it can be accessed and modified efficiently. It defines the relationship between the data elements and the operations that can be performed on that data [1]. Selecting a right data structure makes a program efficient in terms of both Time and Space. In today's world, every application deals with a huge amount of data - Facebook has billions of users, Google handles millions of searches per second. Without a proper data structure, handling this data is impossible.

A sorting algorithm is a procedure used to arrange data elements in a particular order, usually ascending or descending. Sorting is fundamental in computer science because organized data makes searching, processing, and analysis more efficient.

In C programming, sorting algorithms are commonly applied to arrays using loops, functions, pointers, and sometimes recursion. Time complexity describes how the running time of an algorithm increases as the input size increases. Space complexity describes how much additional memory an algorithm needs while sorting [2].

Bubble Sort, Merge Sort, Selection Sort, Insertion Sort, Heap Sort and Quick Sort can generally be implemented in-place. Sorting is one of the most fundamental and important operations in data structures. It is the process of arranging a collection of data elements in a specific logical order. Data in its raw form is meaningless. Sorting makes it meaningful and useful.

Sorting is important due to following reasons: -

- Fast: - It is much easier to search an element in sorted data. Binary Search only works on sorted data.
- Expands Readability: - Sorted data like a dictionary, phone book, or marksheet is easier for humans to understand.
- Effective Data Processing: - Many other algorithms like merging, finding duplicates, finding min/max work much faster if data is already sorted.
- Basis for other Data Structures: - Sorting is used in databases, file systems, and operating systems.
- Data Analysis: - It becomes easy to find minimum, maximum, median and duplicate values.

In other words, without sorting, data is just unorganized information. Sorting converts it into meaningful and useful information. Sorting is a fundamental operation in data structures that deals with arranging the elements of a list or array in a specific order [3][8]. This order can be numerical, alphabetical, or any user-defined order. In computer science, sorting is not just about arranging numbers, it is about organizing data so that it can be used efficiently.

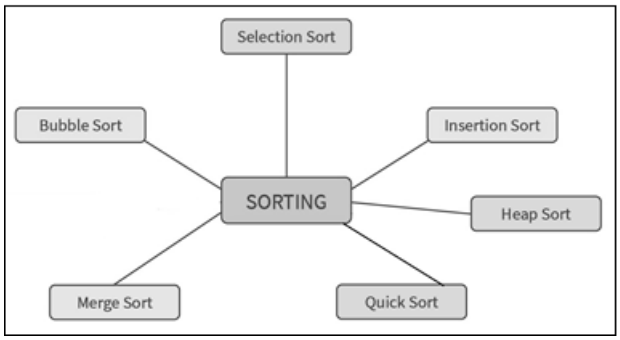

Figure 1.1: Classification of Sorting

C language is well suited for implementing and studying sorting algorithms because it provides direct control over memory, arrays, pointers, and program execution. Sorting algorithms mainly work with arrays and element comparisons, which can be implemented efficiently in C. C is good for sorting algorithms because it is fast, memory-efficient, and provides direct access to arrays and memory. It also makes it easy to implement different sorting techniques and compare their execution time and memory requirements. Therefore, C is widely used for learning, implementing, and analyzing sorting algorithms.

2. Objectives of the Study

1) To study the comparison and relevance of different sorting algorithms in data structures.
2) To analyse a role of different sorting algorithms in context of time and space complexity.

3. Review of Sorting Complexity

The Parameters for Performance Analysis for sorting algorithms as: -

Time Complexity: Time complexity represents the amount of time required by an algorithm as the input size increases. It is generally expressed using Big-O notation.

- Best Case: Minimum time required.
- Average Case: Expected time for a typical input.
- Worst Case: Maximum time required.

Space Complexity: Space complexity represents the amount of additional memory required by an algorithm during execution.

1) Bubble Sort: This sorting repeatedly compares two adjacent elements. If they are in the wrong order, they are swapped. Bubble Sort is a simple and comparison-based sorting algorithm [4][7]. It is used to arrange a list of elements in ascending or descending order. The basic idea of Bubble Sort is to compare adjacent elements and exchange their positions if they are in the wrong order. This process is repeated for multiple passes until the entire list is sorted.

Table 1: Bubble Sort: Time and Space Complexity

Bubble Sort	Time Complexity	Space Complexity
Best Case	O(n)	0
Average Case	$O(n^2)$	0
Worst Case	$O(n^2)$	0

2) Selection Sort: This sorting divides the array into a sorted and unsorted portion. It searches for the smallest element in the unsorted portion and places it at the beginning [5][6]. Selection Sort is a simple comparison-based sorting algorithm. It sorts an array by repeatedly finding the smallest element from the unsorted part of the array and placing it at the beginning of that unsorted part.

Table 2: Selection Sort: Time and Space Complexity

Selection Sort	Time Complexity	Space Complexity
Best Case	O(n2)	0
Average Case	O(n2)	0
Worst Case	O(n2)	0

3) Insertion Sort: This sorting builds the sorted array one element at a time. Insertion Sort is a simple comparison-based sorting algorithm. It builds the final sorted array one element at a time by taking an element from the unsorted part and inserting it into its correct position in the sorted part. It works similarly to how people arrange playing cards in their hands.

Table 3: Insertion Sort: Time and Space Complexity

Insertion Sort	Time Complexity	Space Complexity
Best Case	O(n)	0
Average Case	O(n2)	0
Worst Case	O(n2)	0

4) Merge Sort: This sorting uses the divide-and-conquer technique. It divides the array into smaller parts, sorts those parts, and then merges them together. Merge Sort is an efficient comparison-based sorting algorithm that follows the Divide and Conquer approach. It divides the array into smaller sub-arrays, sorts them, and then merges them to form a sorted array.

Table 4: Merge Sort: Time and Space Complexity

Merge Sort	Time Complexity	Space Complexity
Best Case	O(nlog n)	O(n)
Average Case	O(nlog n)	O(n)
Worst Case	O(nlog n)	O(n)

5) Quick Sort: This sorting also uses divide and conquer. It selects an element called a pivot and rearranges the array so that elements smaller than the pivot are placed before it. Elements larger than the pivot are placed after it. The process is then recursively applied to the two partitions.

Table 5: Quick Sort: Time and Space Complexity

Quick Sort	Time Complexity	Space Complexity
Best Case	O(nlog n)	O(n)
Average Case	O(nlog n)	O(n)
Worst Case	O(n2)	O(n)

6) Heap Sort: This sorting uses a special tree-based data structure called a heap. For ascending sorting, a max heap is commonly constructed. The largest element is repeatedly removed from the heap and placed at the end of the array. Heap Sort is a comparison-based sorting algorithm that uses a special binary tree structure called a Heap. It is based on the heap data structure and is commonly used to sort elements efficiently. For sorting in ascending order, Heap Sort uses a Max Heap, where the largest element is always present at the root.

Table 6: Heap Sort: Time and Space Complexity

Heap Sort	Time Complexity	Space Complexity
Best Case	O(nlog n)	O(1)
Average Case	O(nlog n)	O(1)
Worst Case	O(nlog n)	O(1)

4. Comparison of Sorting Algorithms

Sorting algorithms are used to arrange data in a particular order, usually ascending or descending. The commonly studied sorting algorithms are Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort and Heap Sort.

Table 7: Comparison Table for Sorting

Sorting	Best Case	Average Case	Worst Case	Space	Stable	In-Place
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	No	Yes
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes	Yes
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes	No*
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	No	Yes
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	No	Yes

From the comparison, Merge Sort, Quick Sort, and Heap Sort generally provide better performance for large datasets than Bubble Sort, Selection Sort, and Insertion Sort. Insertion Sort is efficient for small or nearly sorted data, while Merge Sort provides consistent O(n log n) performance. Quick Sort is often very fast in practice, but its worst-case performance can be $O(n^2)$. Heap Sort guarantees O(n log n) time with O(1) extra space. For general-purpose sorting, Quick Sort or Merge Sort is commonly preferred, depending on memory requirements, stability, and input characteristics.

5. Results and Discussion

There is no single sorting algorithm that is best for every situation. Bubble Sort is best for learning basic sorting concepts, not large datasets. Selection Sort is useful for understanding selection-based sorting and minimizing swaps. Insertion Sort is excellent for small or nearly sorted data. Merge Sort is good when predictable O(n log n) performance and stability are important. Quick Sort is excellent practical choice for arrays because of its speed and relatively low memory requirements. Heap Sort is useful when guaranteed O(n log n) worst-case performance and O(1) auxiliary space are important. Performance analysis shows that Bubble Sort, Selection Sort, and Insertion Sort have $O(n^2)$ average-case complexity and are generally suitable for small datasets. Merge Sort, Quick Sort, and Heap Sort are more efficient for large datasets because they generally provide O(n log n) performance. Merge Sort provides predictable performance and is stable but requires additional memory. Quick Sort is usually very efficient in practice but can degrade to $O(n^2)$ with unfavourable pivot choices. Heap Sort guarantees O(n log n) worst-case performance while requiring only O(1) auxiliary space. Therefore, the choice of sorting algorithm depends on the size and nature of the input, available memory, stability requirements, and expected performance. C language is well suited for implementing and studying sorting algorithms because it provides direct control over memory, arrays, pointers, and program execution. Sorting algorithms mainly work with arrays and element comparisons, which can be implemented efficiently in C. C is good for sorting algorithms because it is fast, memory-efficient, and provides direct access to arrays and memory. It also makes it easy to implement different sorting techniques and compare their execution time and memory requirements. Therefore, C is widely used for learning, implementing, and analyzing sorting algorithms.

6. Conclusion

Sorting converts it into meaningful and useful information. Sorting is a fundamental operation in data structures that deals with arranging the elements of a list or array in a specific order. This order can be numerical, alphabetical, or any user-defined order. In computer science, sorting is not just about arranging numbers, it is about organizing data so that it can be used efficiently. In C programming, sorting algorithms are commonly applied to arrays using loops, functions, pointers, and sometimes recursion. Time complexity describes how the running time of an algorithm increases as the input size increases. Space complexity describes how much additional memory an algorithm needs while sorting. Bubble Sort, Merge Sort, Selection Sort, Insertion Sort, Heap Sort and Quick Sort can generally be implemented in-place.

References

[1] Schaum Lipschutz, "Data Structures with C", Schaums' Outlines Series, Indian Edition, 2nd Edition, 2017, Tata McGraw Hill.
[2] Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed, "Fundamentals of Data Structures in C", 2nd Edition, 2018, W.H. Freeman and Company
[3] Yashavant Kanetkar, "Data Structures Through C", 4th Edition, 2022, BPB publication
[4] R. S. Salaria, "Data Structures and Algorithms using C", 2nd Edition, 2022, Khanna Publishing
[5] E Balagurusamy, "Data Structures Using C", 1st Edition, 2017, Tata McGraw Hill
[6] Gilberg and Forouzan, "Data Structure-A pseudocode approach with C", 2nd Edition, 2005, Cengage Learning
[7] K. S. Al-Kharabsheh, I. M. AlTurani, A. M. I. AlTurani, and N. I. Zanoon, "Review on sorting algorithms: A comparative study," International Journal of Computer Science and Security, vol. 7, no. 3, pp. 120–126, 2013
[8] N. Akhter, M. Idrees, and F. Ur-Rehman, "Sorting algorithms- A comparative study," International Journal of Computer Science and Information Security, vol. 14, no. 12, pp. 930–936, Dec. 2016

Author Profile

Dr. Mohit Kumar Sharma (Ph.D., M.Phil., M.Sc.) is working as Associate Professor & Head, Post Graduate Department of Computer Science, J.C. D.A.V. College, Dasuya, Punjab, India. He has 18 years teaching and research experience. He has 23 research publications with book chapters in international refereed journals/publishers, 20 paper presentations at international/national level conferences/ seminars, 2 books authored and 3 edited books. He is also member of undergraduate and postgraduate board of studies, Computer Science and Applications in Panjab University, Chandigarh. His research interests are in Software Engineering and Object-Oriented Programming