ISSN: 2393-8048

An International Conference on
"Educational Technology and Computers"
10th-11th May 2014

Analysis of Sorting and Searching Algorithms Implemented in C

Amit Singla, IT Head & HOD (Computer Science Department) Seth G. L. Bihani S. D. PG College, Sri Ganganagar

Abstract

Sorting and searching are fundamental operations in computer science, playing a crucial role in data organization, retrieval efficiency, and overall algorithmic performance. This study presents a comprehensive analysis of key sorting and searching algorithms implemented in the C programming language, focusing on their logic, computational complexity, memory usage, and practical applicability. The algorithms evaluated in this review include Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort for sorting, and Linear Search and Binary Search for searching operations. Using C as the implementation medium provides a low-level, memory-efficient environment that enables precise observation of algorithmic behavior, pointer manipulation, and execution efficiency.

The study highlights that simple sorting algorithms such as Bubble Sort and Selection Sort are easy to implement and suitable for small datasets but exhibit poor performance with an average and worst-case time complexity of $O(n^2)$. Insertion Sort demonstrates relatively improved performance on nearly sorted datasets due to its adaptive nature. In contrast, advanced algorithms like Merge Sort and Quick Sort show significantly better performance, operating in $O(n \log n)$ time. Merge Sort offers stable sorting and predictable complexity but requires additional memory due to its divide-and-conquer approach. Quick Sort generally performs fastest on large datasets due to efficient partitioning but may degrade to $O(n^2)$ in unfavorable pivot conditions.

For searching, Linear Search provides an intuitive approach suitable for unsorted datasets but suffers from $O(n)$ time complexity. Binary Search, implemented on sorted arrays, dramatically improves search performance with $O(\log n)$ time complexity, making it ideal for large, ordered datasets. Through C-based implementation, the study evaluates execution time, iteration count, and recursion depth, providing practical insights into how algorithm design impacts real-time performance. The findings emphasize that algorithm selection must be based on dataset size, memory constraints, and required efficiency rather than implementation simplicity alone.

Overall, this analysis concludes that understanding algorithmic behavior through C implementation not only enhances problem-solving abilities but also deepens conceptual knowledge of computational complexity and optimization. The study reinforces that efficient sorting and searching are foundational to advanced computing tasks such as database indexing, information retrieval, and system-level programming. Future work may explore hybrid algorithms, parallel implementations, and performance optimization techniques using modern C standards.