INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 8.854 Volume 13, issue 04 (2026)

THE INSERTION SORT METHOD FOR ARRAY SORTING IN PYTHON PROGRAMMING

Fathiddinov Saidma'ruf Lazizxo'ja ugli,
Department of General Technical Sciences, Asia International University

Abstract: Insertion sort is a simple and intuitive algorithm used for sorting arrays in Python programming. It works by dividing the array into two parts: a sorted portion and an unsorted portion. At each step, the algorithm takes one element from the unsorted part and inserts it into its correct position within the sorted part. This process continues until all elements are arranged in the desired order.

In Python, insertion sort is typically implemented using loops and conditional statements. The algorithm iterates through the array starting from the second element, compares it with the previous elements, and shifts larger elements one position ahead to make space for insertion. This approach makes insertion sort efficient for small datasets or nearly sorted arrays.

The time complexity of insertion sort is O(n^2) in the worst and average cases, while in the best case (when the array is already sorted), it performs at O(n) Despite its simplicity, insertion sort is not suitable for large datasets due to its quadratic time complexity. However, its advantages include ease of implementation, stability, and low memory usage since it is an in-place sorting algorithm.

Keywords: Insertion Sort, Array Sorting, Python Programming, Sorting Algorithms, Data Structures, Algorithm Efficiency, Time Complexity, In-place Sorting, Stable Sorting, Iterative Methods

Introduction

Sorting is one of the fundamental operations in computer science and plays a crucial role in data processing, searching, and optimization tasks. Efficient organization of data allows algorithms to perform faster and more accurately, making sorting techniques an essential topic in programming. Among various sorting methods, insertion sort is considered one of the simplest and most intuitive algorithms, especially for beginners learning Python programming.

Insertion sort works by gradually building a sorted portion of an array. It takes elements one by one and inserts each element into its correct position within the already sorted part of the array. This step-by-step approach makes the algorithm easy to understand and implement, as it closely resembles the way people sort items manually, such as arranging playing cards in hand.

In Python programming, insertion sort is commonly implemented using loops and conditional statements, which makes it a practical example for understanding algorithm design and control structures. Although insertion sort is not the most efficient algorithm for large datasets due to its quadratic time complexity, it performs well for small or nearly sorted arrays.

Research Methods

This study employs a qualitative and practical approach to analyze the insertion sort method for array sorting in Python programming. The research is primarily based on algorithm analysis, implementation, and experimental observation.

Firstly, a theoretical analysis of the insertion sort algorithm is conducted. This includes studying its working principles, step-by-step procedure, and computational complexity. Key

aspects such as time complexity in best, average, and worst cases, as well as space complexity, are examined to understand the algorithm's efficiency.

Secondly, the algorithm is implemented using Python programming. Standard programming constructs such as loops, conditional statements, and array manipulation techniques are used to develop a functional insertion sort program. This implementation helps to observe how the algorithm behaves in real execution.

Thirdly, experimental testing is carried out on different types of datasets, including randomly generated arrays, sorted arrays, and reverse-ordered arrays. The performance of the algorithm is evaluated by measuring execution time and the number of operations required for sorting.

Finally, a comparative analysis method is applied by briefly comparing insertion sort with other basic sorting algorithms such as Bubble Sort and Selection Sort. This allows for identifying the advantages and limitations of insertion sort in practical scenarios.

Results

The experimental results show that the insertion sort algorithm is effective for sorting small arrays and datasets that are already partially sorted. During testing, the algorithm demonstrated faster performance in cases where minimal element shifting was required, confirming its efficiency in best-case scenarios.

When applied to an already sorted array, insertion sort performed with linear time complexity O(n) requiring only one comparison per element. In contrast, for randomly arranged arrays, the algorithm showed average performance with time complexity O(n^2), due to repeated comparisons and element shifts. The worst performance was observed when the array was sorted in reverse order, where the number of operations increased significantly.

The Python implementation confirmed that insertion sort is memory-efficient, as it does not require additional storage beyond the original array (in-place sorting). It also maintained stability, preserving the relative order of equal elements.

Furthermore, comparison with other basic sorting algorithms indicated that insertion sort performs better than Bubble Sort in many practical cases, especially when the dataset is nearly sorted. However, it is less efficient than more advanced algorithms for large-scale data.

Discussion

The findings of this study highlight both the strengths and limitations of the insertion sort algorithm in Python programming. One of the most significant advantages of insertion sort is its simplicity and ease of implementation. This makes it particularly useful for educational purposes, as it helps beginners understand fundamental concepts such as iteration, comparison, and element shifting within arrays.

The results confirm that insertion sort performs efficiently when dealing with small datasets or nearly sorted arrays. In such cases, the algorithm minimizes unnecessary operations, achieving near-linear performance. This adaptive behavior makes it suitable for real-world scenarios where data is often partially ordered.

However, the study also reveals clear limitations. Insertion sort becomes inefficient when applied to large or randomly ordered datasets due to its quadratic time complexity O(n^2). The repeated comparisons and shifts significantly increase execution time, making it less practical compared to more advanced algorithms such as Quick Sort or Merge Sort.

Another important aspect discussed is the stability and in-place nature of insertion sort. The algorithm preserves the relative order of equal elements, which is beneficial in applications


where data consistency matters. Additionally, it does not require extra memory, making it space-efficient.

Conclusion

In conclusion, the insertion sort method is a simple and effective algorithm for sorting arrays in Python programming, especially in cases involving small or nearly sorted datasets. Its straightforward logic and ease of implementation make it an ideal choice for beginners to understand the fundamental principles of sorting and algorithm design.

The study shows that insertion sort performs efficiently in best-case scenarios with linear time complexity O(n), while its performance decreases to O(n^2) in average and worst cases. Despite this limitation, the algorithm remains valuable due to its stability and in-place nature, which allow it to sort data without requiring additional memory and while preserving the order of equal elements.

Although insertion sort is not suitable for handling large datasets compared to more advanced algorithms, it still plays an important role in computer science education and in specific practical situations where simplicity and low overhead are preferred.

References

1. M. Shahzad, M. Shakeel, A. U. Rehman, and M. U. Shoukat. Review on Sorting Algorithms - A Comparative Study, International Journal of Innovative Science and Modern Engineering, vol. 5, no. 1, p. 4, 2017.
2. Ali, I., H, Nawaz, I.K., Ameen, M., Chhajro, M. and Maitlo, A. Performance Comparison between Merge and Quick Sort Algorithms in Data Structure, in International Journal Of Advanced Computer Science And Applications, 9(11), pp.192-195, 2018.
3. J. P. Ocampo. An empirical comparison of the runtime of five sorting algorithms, p. 26.
4. A. Alotaibi, A. Almutairi, and H. Kurdi. OneByOne (OBO): A Fast Sorting Algorithm, in Procedia Comput. Sci., vol. 175, pp. 270–277, 2020. doi: 10.1016/j.procs.2020.07.040.
5. J. Alnihoud and R. Mansi. An Enhancement of Major Sorting Algorithms, Int. Arab J. Inf. Technol. vol. 7, no. 1, pp 55-62, 2010.
6. N. Yadav and S. Kumari. Sorting Algorithms, in International Research Journal of Engineering and Technology, vol. 3, no. 2, pp. 528-531, 2016.
7. F. G. Furat. A Comparative Study of Selection Sort and Insertion Sort Algorithms, in International Research Journal of Engineering and Technology (IRJET), vol. 03, no. 12, pp. 336–330, Dec. 2016.
8. W. Min. Analysis on Bubble Sort Algorithm Optimization, in 2010 International Forum on Information Technology and Applications, Kunming, China, Jul. 2010, pp. 208–211. doi: 10.1109/IFITA.2010.9.