Merge-Binary Insertion Sort Algorithm: Implementation and Experimentation

Dario Bonfiglio, Inyan Fornaroli e Alessandra Gulli

December 19, 2023

1 Introduction

The Merge-Binary Insertion Sort algorithm is a hybrid sorting algorithm that combines the principles of merge sort and binary insertion sort to achieve efficient sorting for large datasets. This report aims to explain the workings of the algorithm, its key components, and its time complexity.

2 Algorithm Overview

The algorithm is implemented in C and consists of three main functions: `merge_binary_insertion_sort`, `mergeSort`, and `insertionSort`. The primary sorting logic is implemented in the `mergeSort` function, which recursively divides the input array into smaller subarrays until a base case is reached. When the subarrays become sufficiently small (determined by the parameter k), the algorithm switches to using the `insertionSort` function for sorting.

2.1 Binary Insertion Sort

The `insertionSort` function is a standard binary insertion sort implementation. It iterates through the array, for each element performing a binary search to find the correct position for insertion. This reduces the number of comparisons compared to a linear search and is particularly efficient for small subarrays.

2.2 Merge Sort

The `mergeSort` function divides the array into two halves, recursively applies `mergeSort` to each half, and then merges the sorted halves using the `merge` function. This divide-and-conquer strategy ensures that smaller subarrays are efficiently sorted using `insertionSort`, while larger subarrays benefit from the merging step.

2.2.1 Merge Step

The `merge` function combines two sorted subarrays into a single sorted array. It uses three pointers to iterate through the left and right subarrays, comparing and merging elements in a manner similar to the merge step in the standard merge sort algorithm.

3 Complexity Analysis

The time complexity of the algorithm is influenced by the balance between `mergeSort` and `insertionSort`. For large subarrays, the algorithm behaves like merge sort with a time complexity of $O(n \log n)$. For small subarrays, the time complexity is dominated by `insertionSort`, which has a best-case time complexity of $O(n)$ for already sorted input.

4 Algorithm Analysis

To analyze the performance of the Merge-Binary Insertion Sort algorithm, a series of tests were conducted on different datasets with varying values of the parameter k. The data below shows the execution time (in seconds) for each test:

The tests were conducted to find the optimal value for k that balances the benefits of both merge sort and binary insertion sort.

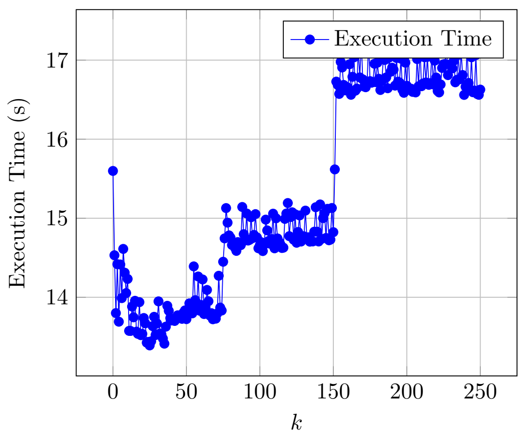


Figure 1: Execution time on Strings

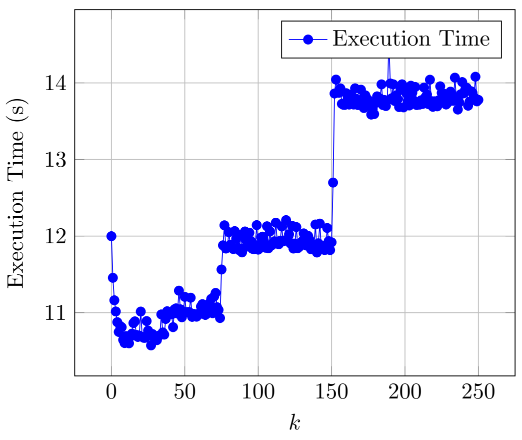
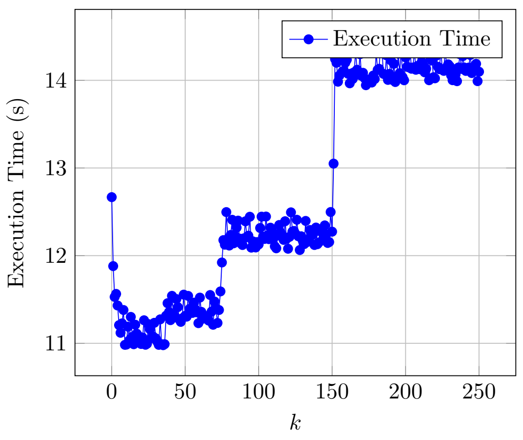


Figure 2: Execution time on Integers



[ht]

Figure 3: Execution time on Floating Point

The graphs provide insights into the algorithm's behavior under different k values, aiding in the selection of an optimal k for specific use cases.

5 Conclusion

The Merge-Binary Insertion Sort algorithm combines the strengths of merge sort and binary insertion sort to efficiently handle different-sized subarrays. By dynamically choosing between the two sorting methods based on the size of the subarray, the algorithm achieves good performance across a wide range of input sizes.