Stuart Lech

CS 340

Project 1

9/6/2023

Introduction and Methodology

The role of sorting algorithms in computer science is critical for the performance of many applications, ranging from database queries to user-interface design. Two commonly used sorting algorithms are Insertion Sort and Heap Sort. While both aim to achieve the same end, the mechanisms by which they sort an array are quite different, leading to variances in performance. This report aims to compare these two sorting algorithms based on their time complexity and actual running times under different scenarios. I implemented both Insertion Sort and Heap Sort algorithms and measured their running times in microseconds. Tests were conducted using both pre-sorted and randomly generated arrays of varying sizes, ranging from 100 to 50,000 elements.

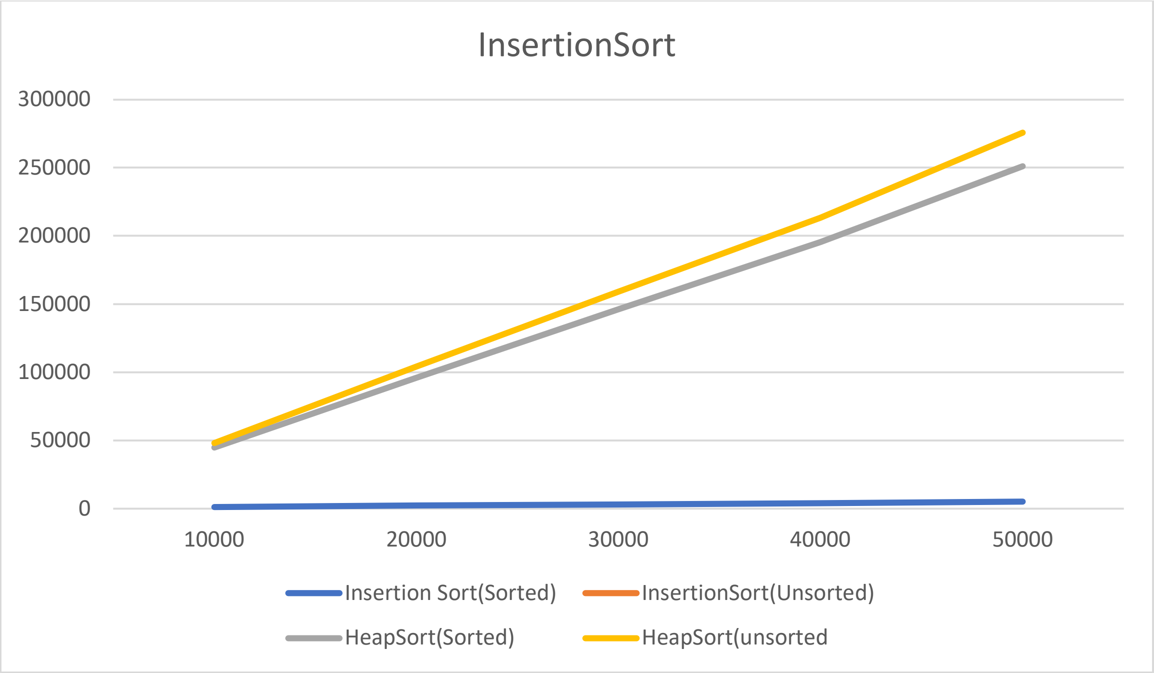
Time Complexity

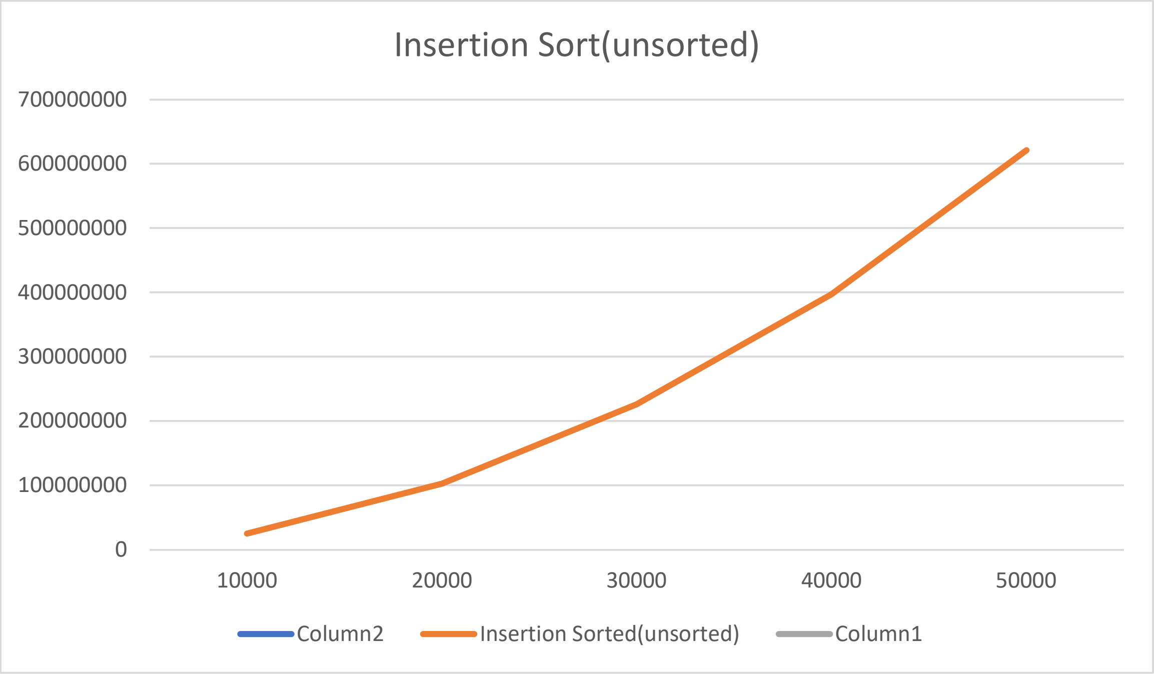
For Insertion Sort, the time complexity varies depending on the initial arrangement of the data set. When the input array is already sorted, the algorithm runs in linear time, denoted as O(n), because each new element essentially requires only a single comparison as it gets inserted into its appropriate position. However, in the case of a randomly ordered or worst-case scenario, the time complexity becomes quadratic, represented as O(n^2). This is due to the need to compare, and possibly shift, each of the already sorted elements for every new element being inserted, leading to a performance that can deteriorate rapidly as the size of the data set increases.

On the other hand, Heap Sort maintains a more consistent performance profile. Regardless of whether the input data is sorted or unsorted, the algorithm operates at a logarithmic-linear time complexity, denoted as O(n log n). The reason for this lies in the inherent structure of the heap data structure, which allows both the building of the heap and the subsequent heapification to occur in logarithmic time. The initial building of the heap is O(n), and each removal from the heap is O(log n), making the overall complexity O(n log n). Therefore, while both Insertion Sort and Heap Sort aim to sort a list of numbers, their time complexities indicate that they will do so with different levels of efficiency, depending on the characteristics of the input data.

DATA

Input Size	Insertion Sort(Sorted) (microseconds)	InsertionSort(Unsorted)	HeapSort(Sorted)	HeapSort(unsorted
10000	1211	25434898	44963	48264
20000	2273	102565315	95873	104053
30000	3133	226092963	146226	158952
40000	4025	397329041	195484	213200
50000	5278	621506987	251146	275693


InsertionSort
300000
250000
200000
150000
100000
50000
0
10000
20000
30000
40000
50000
Insertion Sort(Sorted)
InsertionSort(Unsorted)
HeapSort(Sorted)
HeapSort(unsorted


Insertion Sort(unsorted)
700000000
600000000
500000000
400000000
300000000
200000000
100000000
0
10000
20000
30000
40000
50000
Column2
Insertion Sorted(unsorted)
Column1

In the line graph plotting running times, it’s evident that Insertion Sort is significantly slower on unsorted data, as the curve steepens sharply as the input size grows. Heap Sort demonstrates a more gradual increase in running time, consistent with its O (n log n) time complexity for both sorted and unsorted data.

Implications

The implications of our findings extend beyond mere academic interest. In real-world applications where sorting large datasets is crucial, the choice of sorting algorithm can significantly impact performance and, by extension, cost. For instance, database management systems often rely heavily on sorting algorithms; thus, choosing an algorithm like Insertion Sort for large, unsorted datasets could result in inefficient use of computational resources. On the other hand, an algorithm like Heap Sort, with its more consistent O (n log n) performance, could save both time and computational power. This emphasizes the need for software engineers and data scientists to carefully consider the characteristics of the data they are working with when selecting a sorting algorithm.

Conclusion

In conclusion, our study has shed light on the practical performance differences between Insertion Sort and Heap Sort, substantiating their theoretical time complexities with empirical data. While Insertion Sort may be suitable for small or nearly sorted datasets, Heap Sort emerges as a more reliable option for larger, unsorted collections of data. The study underscores the importance of aligning algorithmic choices with the specific requirements and constraints of each application, an essential skill in both software development and data analysis.