Sorting Algorithms: Comparing Between Insertion, Merge, Heap and Quick Sorting Algorithms and Making a Hybrid Sorting

Hakan Aydin

December 21, 2022

1. Introduction

In this paper I am going to implement insertion sort, quick sort, merge sort and heap sort. After creating these algorithms, I am going to compare them. Once I compare the four algorithms, I am to create a hybrid sorting algorithm based on our findings and compare the hybrid sort with the other sorting algorithms.

2. Insertion Sort:

Is a simple, in-place sorting algorithm that has a time complexity of O (n^2) in the worst case. This means that the time taken to sort the array increases quadratically with the size of the array. Insertion Sort is a stable sort, which means that it preserves the order of equal elements. It is not a very efficient algorithm for large arrays, but it can be useful for small arrays or for sorting partially sorted arrays.

3. Merge Sort:

Is a divide and conquer sorting algorithm that has a time complexity of O (n log n) in the worst case. This means that the time taken to sort the array increases logarithmically with the size of the array. Merge Sort is a stable sort, which means that it preserves the order of equal elements. It is generally more efficient than Insertion Sort, especially for larger arrays. However, it requires additional space to store the subarrays that are being merged, which can be a disadvantage in certain situations.

4. Heap Sort:

Is a comparison-based sorting algorithm that has a time complexity of O (n log n) in the worst case. This means that the time taken to sort the array increases logarithmically with the size of the array. Heap Sort is not a stable sort, which means that it does not preserve the order of equal elements. It is generally more efficient than Insertion Sort, especially for larger arrays. However, it requires additional space to store the heap, which can be a disadvantage in certain situations.

5. Quick Sort

Is a divide and conquer sorting algorithm that has a time complexity of O(n^2) in the worst case, but an average time complexity of O (n log n). This means that the time taken to sort the array increases logarithmically with the size of the array on average, but there is a possibility of worst-case performance that is quadratic. Quick Sort is not a stable sort, which means that it does not preserve the order of equal elements. It is generally more efficient than Insertion Sort, especially for larger arrays. However, it requires additional space to store recursive function calls, which can be a disadvantage in certain situations.

6. ***Methodology***

This is a program that compares the performance of four sorting algorithms: merge sort, quick sort, insertion sort, and heap sort. It does this by generating random arrays of different sizes and measuring the time it takes for each algorithm to sort those arrays. The program then uses this data to determine which algorithm is the fastest for a given array size, and stores this information in the '*fastestAlgorithms*' vector. Finally, the program defines a function '*hybrid_sort'* that uses this information to choose the fastest algorithm for a given array, based on its size.

The program starts by measuring the performance of the sorting algorithms on small arrays (sizes 1 to 100). It does this by generating '*iterations'* (100) random arrays of a given size and measuring the time it takes for each algorithm to sort each array. It then calculates the average time for each algorithm by dividing the total time by '*iterations'*. This process is repeated for each array size in the range.

Next, the program measures the performance of the sorting algorithms on large arrays (sizes 100 to 10000, in steps of 100). It does this in the same way as for small arrays, but only measures the performance for a few selected sizes.

Finally, the program determines which algorithm is the fastest for each array size by finding the algorithm with the lowest average time for that size. It stores this information in the '*fastestAlgorithms'* vector. The '*hybrid_sort*' function can then use this information to choose the fastest algorithm for a given array.

7. ***Results***

Based on my findings when I compare these 4 algorithms Heap Sorting is the fastest and when the larger arrays are the case Insertion Sort is the slowest. When I include Hybrid Sort to the comparison that I created comparison shows that it is slower than Heap Sorting, Merge Sorting and Quick Sorting but faster than Insertion Sorting. (Unfortunately, Hybrid Sorting is not on the graph)

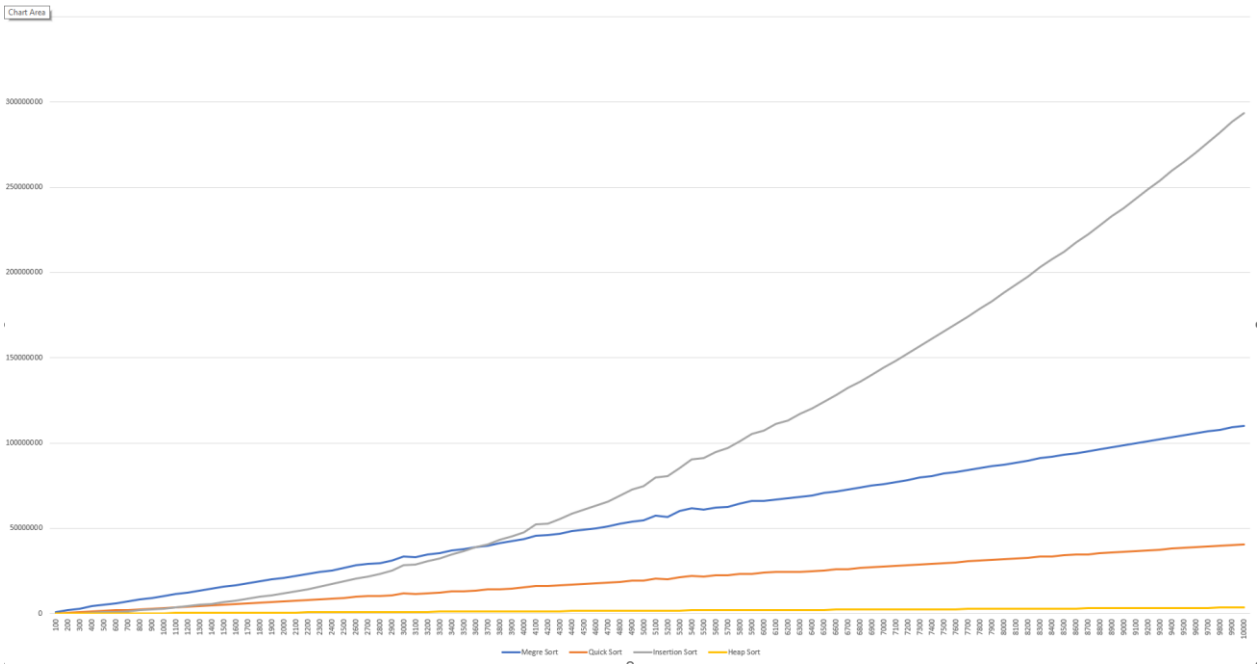
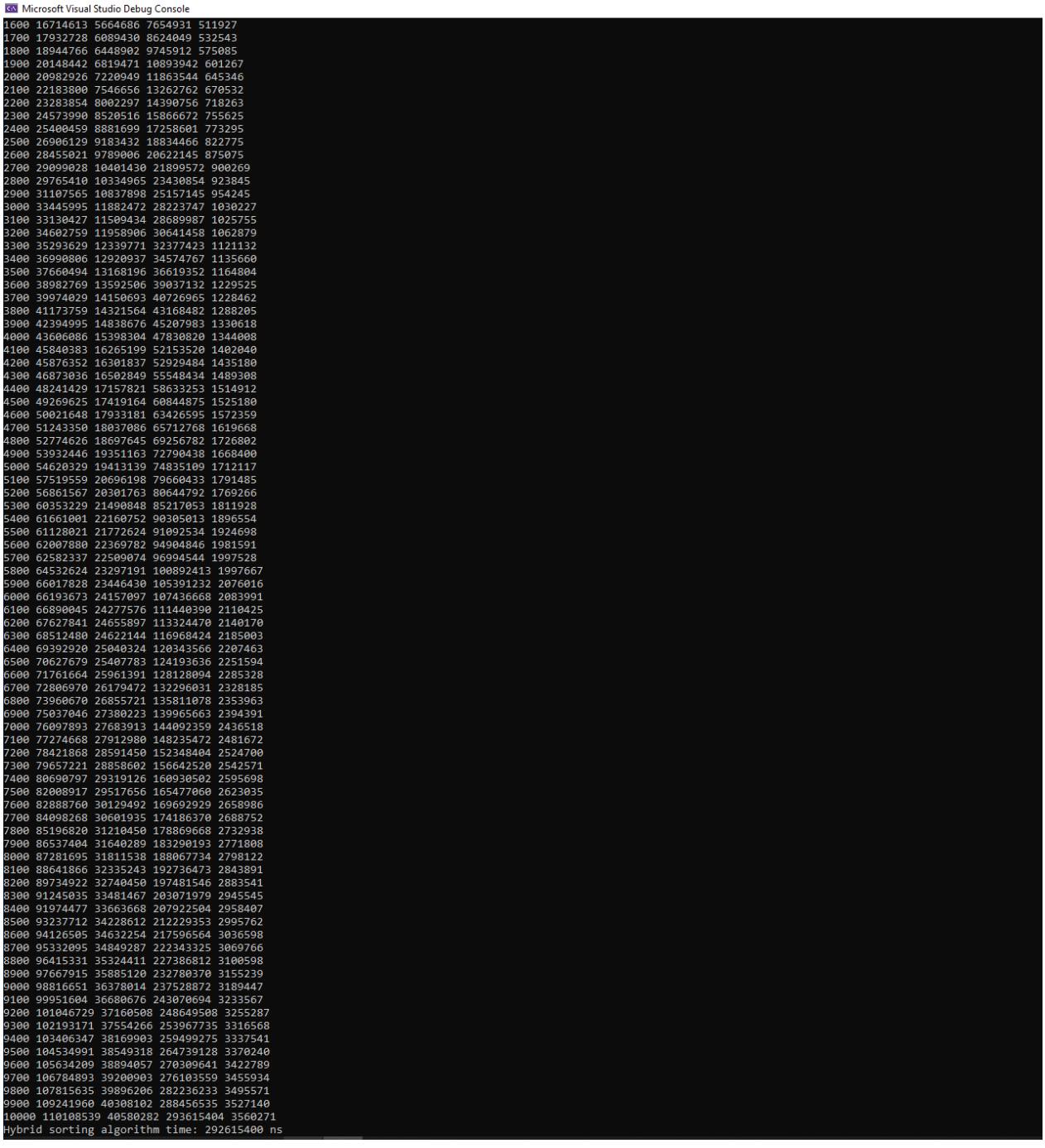


Chart Area
300000000
250000000
200000000
150000000
100000000
50000000
0
Megre Sort
Quick Sort
Insertion Sort
Heap Sort



8. ***Conclusions***

Heap sorting algorithm is best all-around and then second fastest is Quick sort algorithm. Using the Hybrid sorting algorithm its closer to Insertion but not even close to heap sorting.

So I conclude that Heap sorting, Quick sorting, Hybrid sorting, Merge sorting and Insertion sorting algorithm is the order of fastest to slowest.