Comparison of insertion, heap, radix, quick and hybrid sorting algorithms

Samad Hashimzada

Introduction

This is based on comparison of 4 sorting algorithms, insertion, heap, radix and quick sorting algorithms and figuring out hybrid sorting algorithm.

Insertion Sort

Insertion sort is a simple sorting algorithm used to organize an array by gradually inserting elements into an already sorted subarray, resulting in a fully sorted array. The process starts by iterating through the array, beginning from the second element. At each step, the algorithm compares the current element with its preceding elements. If the current element is smaller than its predecessor, a series of swaps occur until the smaller element takes its correct position in the sorted sequence. This swapping process continues until the smaller element is appropriately placed in the sorted array.
The time complexity of Insertion Sort is O(n^2) in the worst-case scenario, in the best-case scenario, when the array is already sorted, the time complexity is O(n), average-case time complexity of Insertion Sort is O(n^2).

Heap Sort

Heap sort is a comparison-based sorting algorithm that sorts an array in ascending or descending order by transforming it into a binary heap data structure. It uses the concepts of max-heap and min-heap, where in max-heap, the largest element is the root node, and in min-heap, the smallest element is the root node. The algorithm uses the binary heap structure to efficiently sort the elements.
Heap Sort has a time complexity of O(n log n) for its average and worst-case and best-case
Advantages: Optimized performance, efficiency, and accuracy are a few of the best qualities of this algorithm. The algorithm is also highly consistent with very low memory usage. No extra memory space is required to work, unlike the Merge Sort or recursive Quick Sort.
Disadvantages: Heap Sort is considered unstable, expensive, and not very efficient when working with highly complex data.

Radix Sort

Radix sort is a sorting algorithm designed for integers, organizing data with integer keys by categorizing the keys based on individual digits that share the same significant position and value (place value). This algorithm utilizes counting sort as a component to arrange an array of numbers. Radix sort extends its applicability beyond integers, as it can also work on other data types, including strings represented as integers through hashing. Not relying on comparisons, radix sort breaks free from the Ω(n log n) lower bound for running time associated with comparison-based sorting algorithms. In fact, radix sort has the potential to achieve linear time complexity.
The time complexity of Radix Sort, when implemented using Counting Sort as a subroutine, is O(d * (n + k)), where:
d is the number of digits in the maximum number in the array, n is the number of elements in the array, and k is the range of values for each digit (typically 10 for decimal numbers). In practice, radix sort is often considered to have linear time complexity, O(n), since d is a constant factor

and can be considered relatively small. This makes radix sort particularly efficient for sorting integers, especially when the range of values is not too large.

Quick Sort

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.
Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets.
The time complexity of Quick Sort is typically O(n log n) on average, making it one of the most efficient sorting algorithms for general-purpose use. However, in the worst-case scenario, Quick Sort can have a time complexity of O(n^2).

Hybrid Sort

After recording performance of other 4 sorting algorithms, as a decision insertion sort has better performance for small size arrays, and quick sort has better performance for large size arrays. According to this, I created hybrid sort where it calls insertion sort function for small size arrays, and quick sort function for large size arrays.

Methodology

Algorithms were implemented in Python programming language, where PyCharm used as IDE and as hardware laptop Asus Rog Strix g15 2022 with Windows 11 pro operating system.

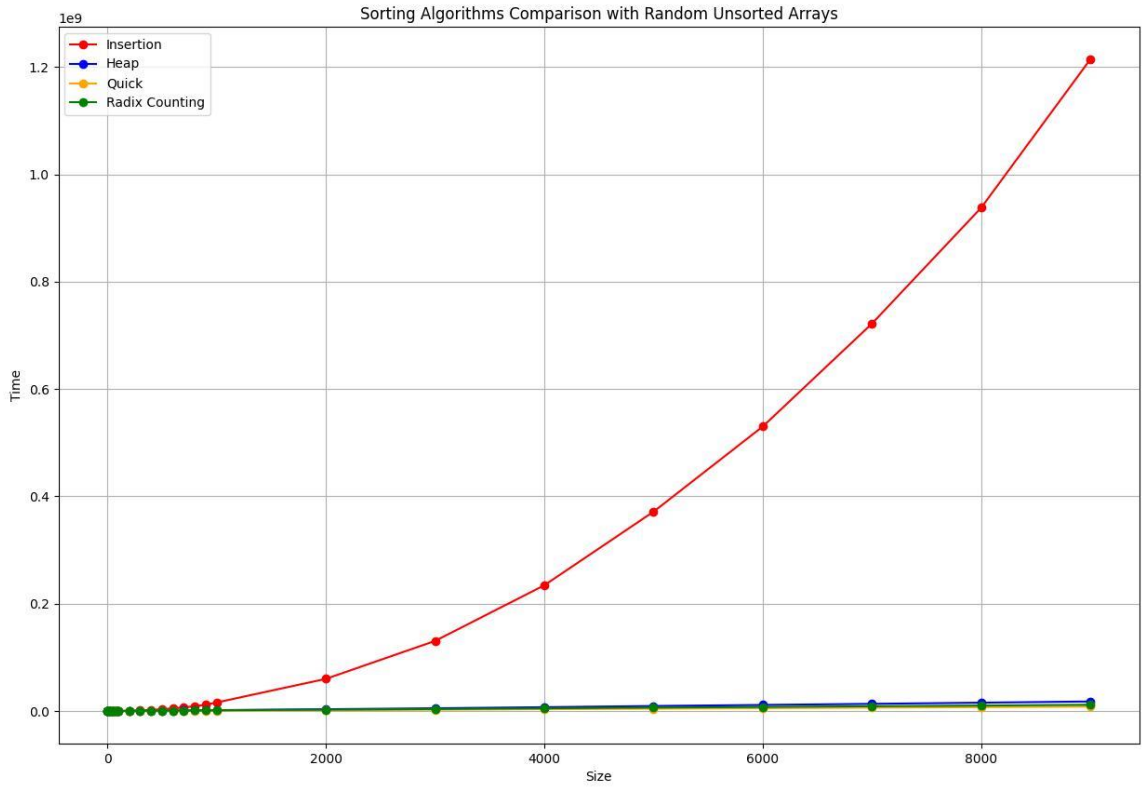
Algorithms were tested with arrays sized 1 to 10, 10 to 100 (as tens increment), 100 to 1000 (as 100 increment) and 1000 to 9000 (as 1000 increment), where arrays randomly generated between 0 and 1000000, and every array repeated 100 times.

Result

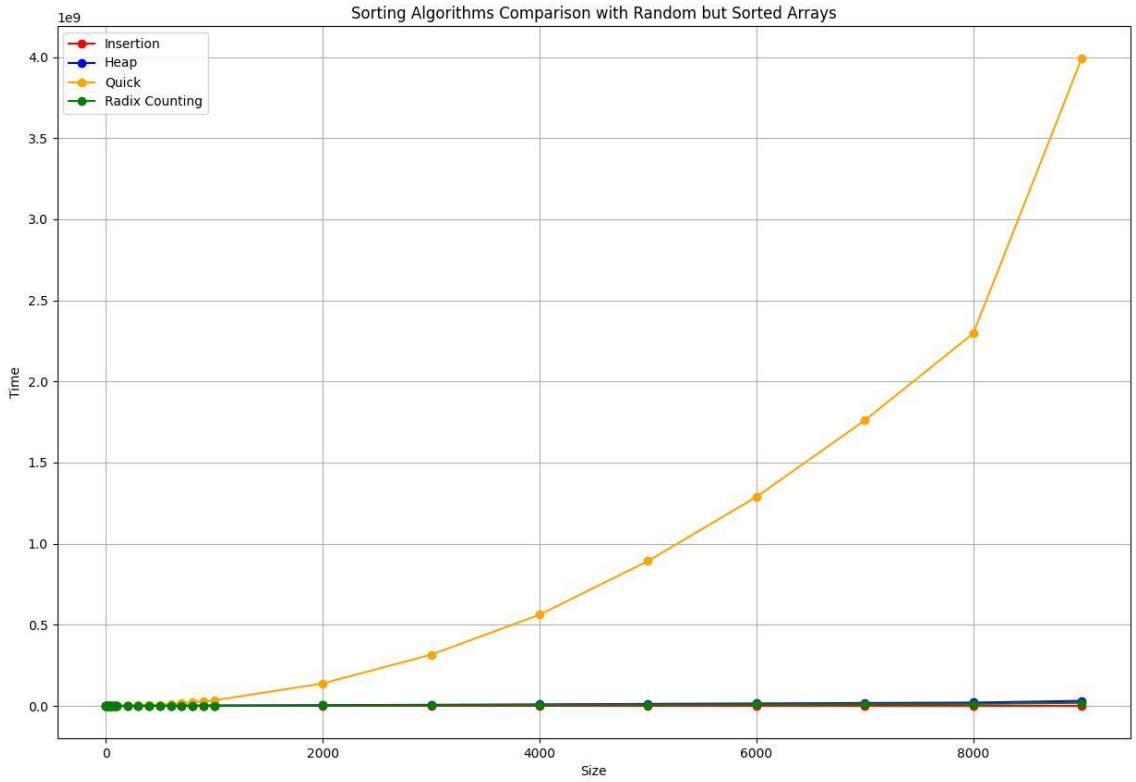
Below graphs represent comparison of each sorting algorithm together and separately with hybrid sort. As a result, hybrid sort is best for time measurement. Other sorting algorithms can differ for size of arrays.

Conclusion

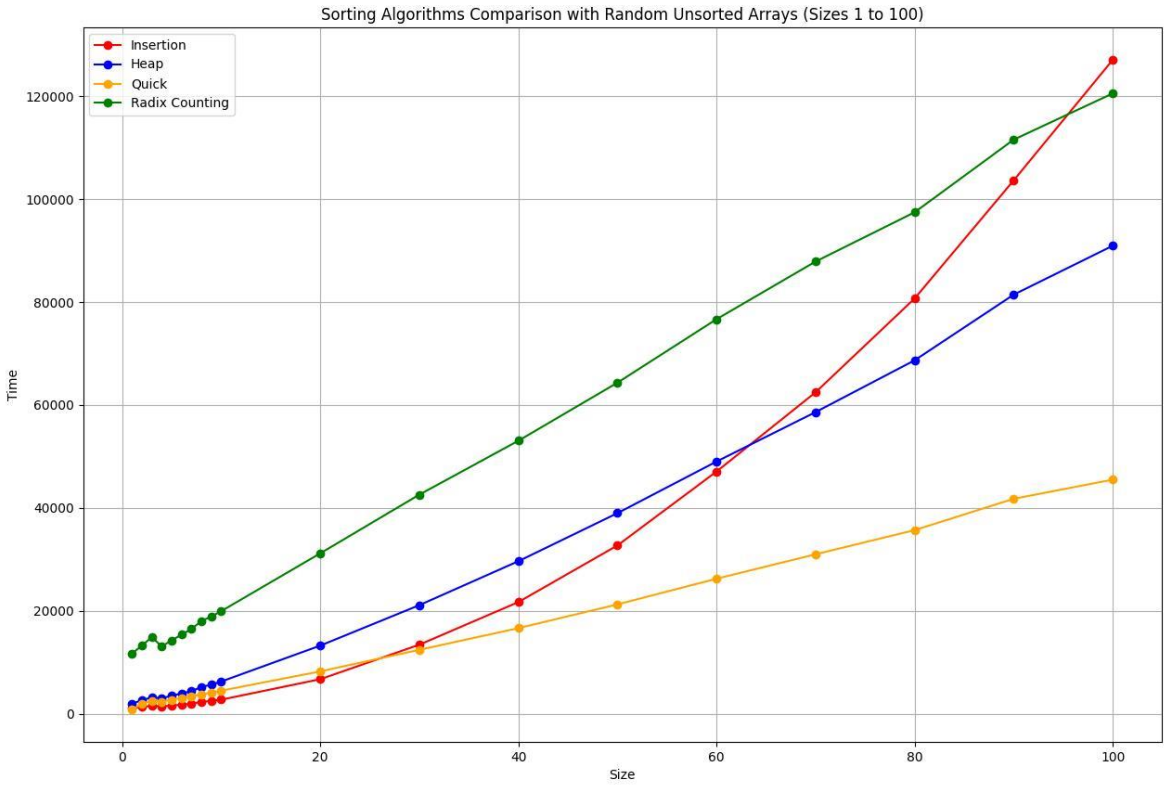
As a conclusion, hybrid sort is best in out of 5 sorting algorithms for time measurement.



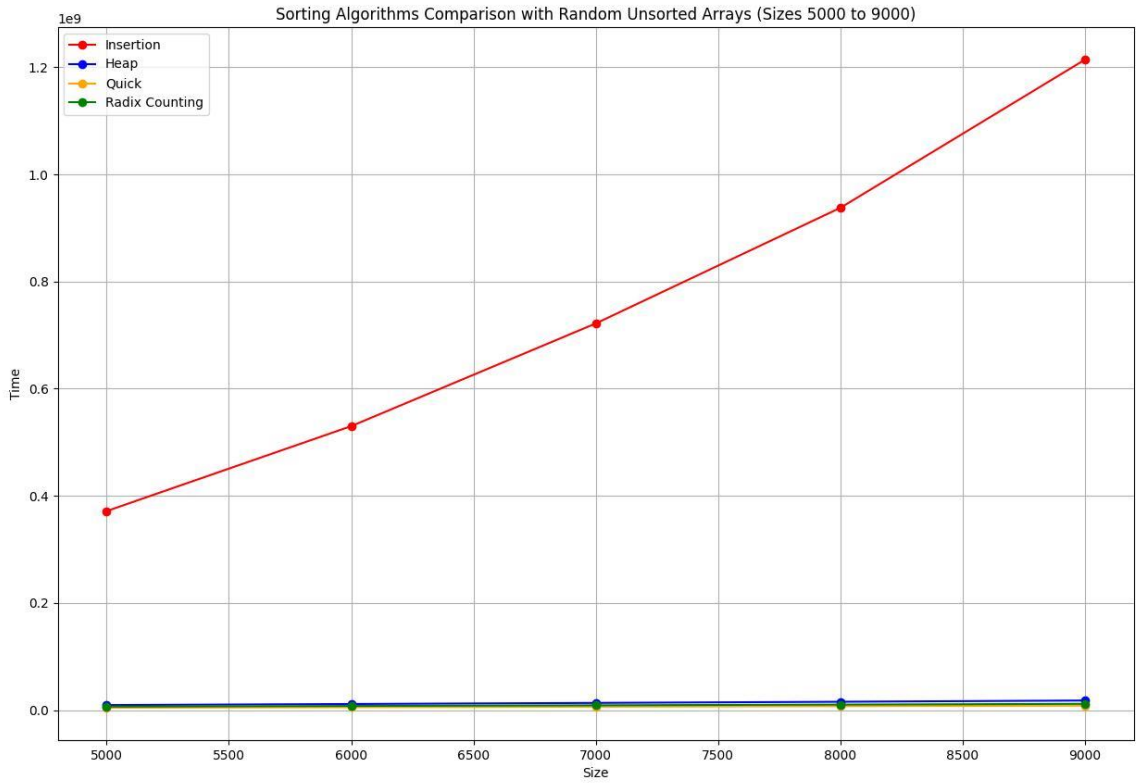
Time of each algorithm takes to sort randomly generated unsorted arrays



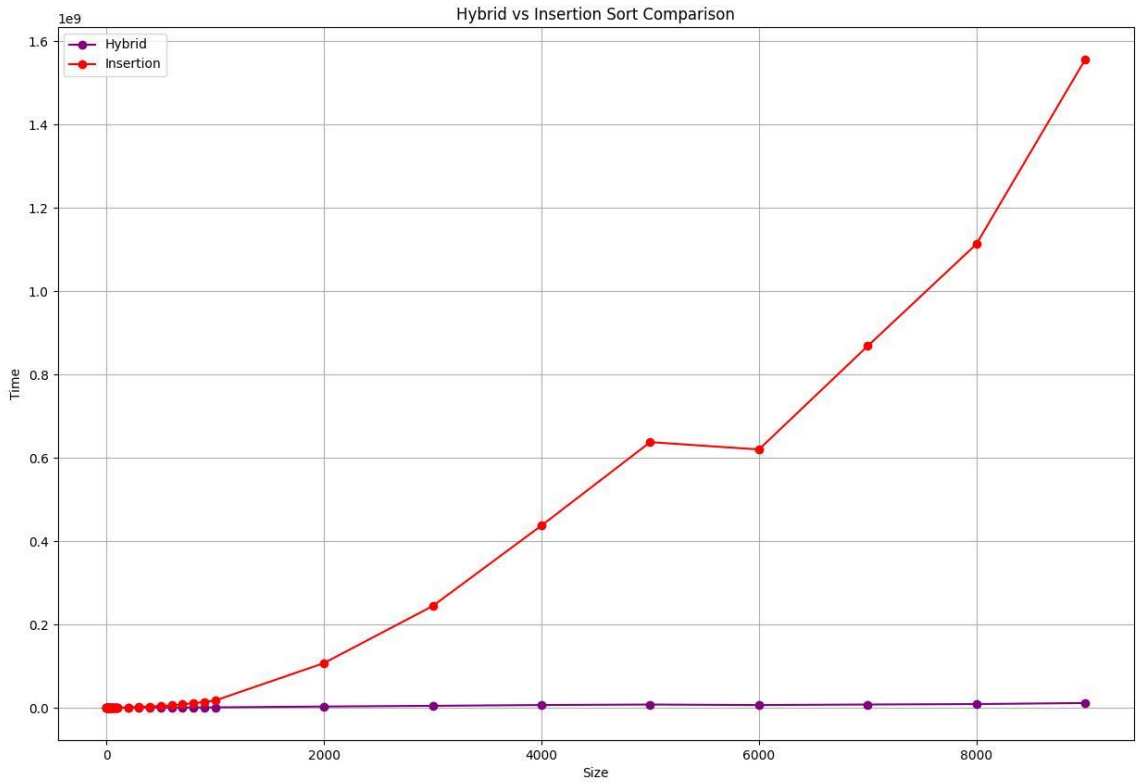
Time of each algorithm takes to sort randomly generated sorted arrays



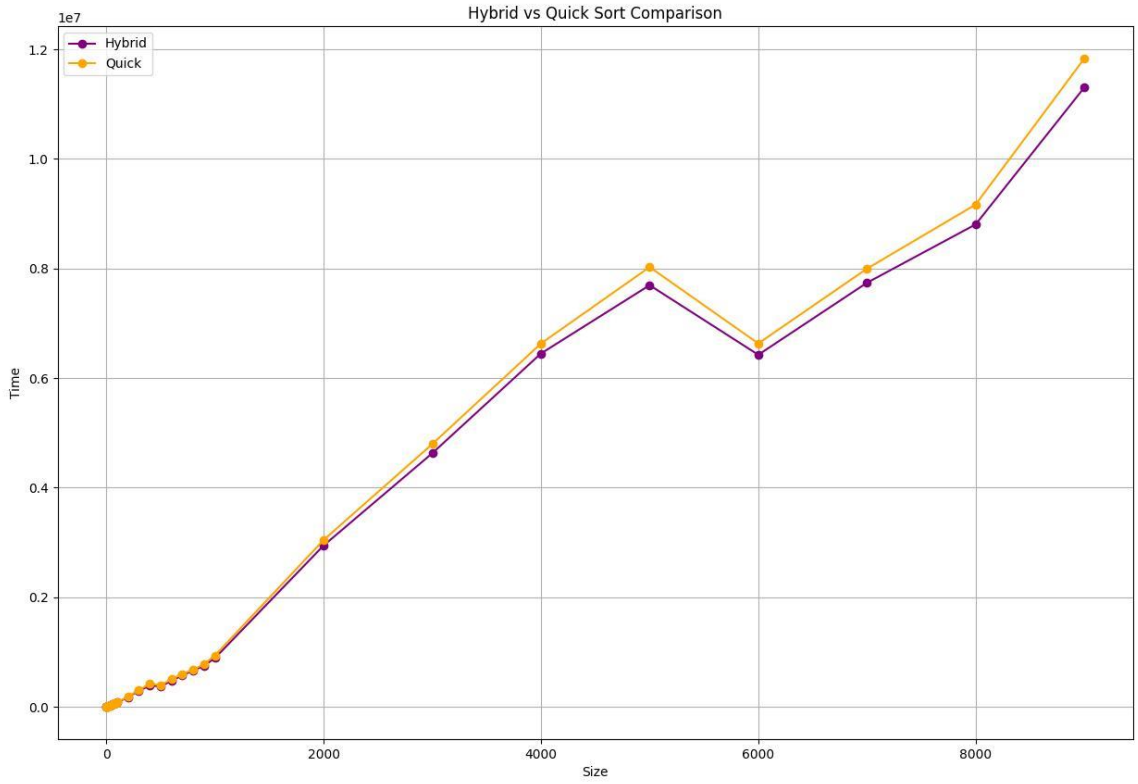
Time of each algorithm takes to sort randomly generated unsorted small size(size 1 to 100) arrays



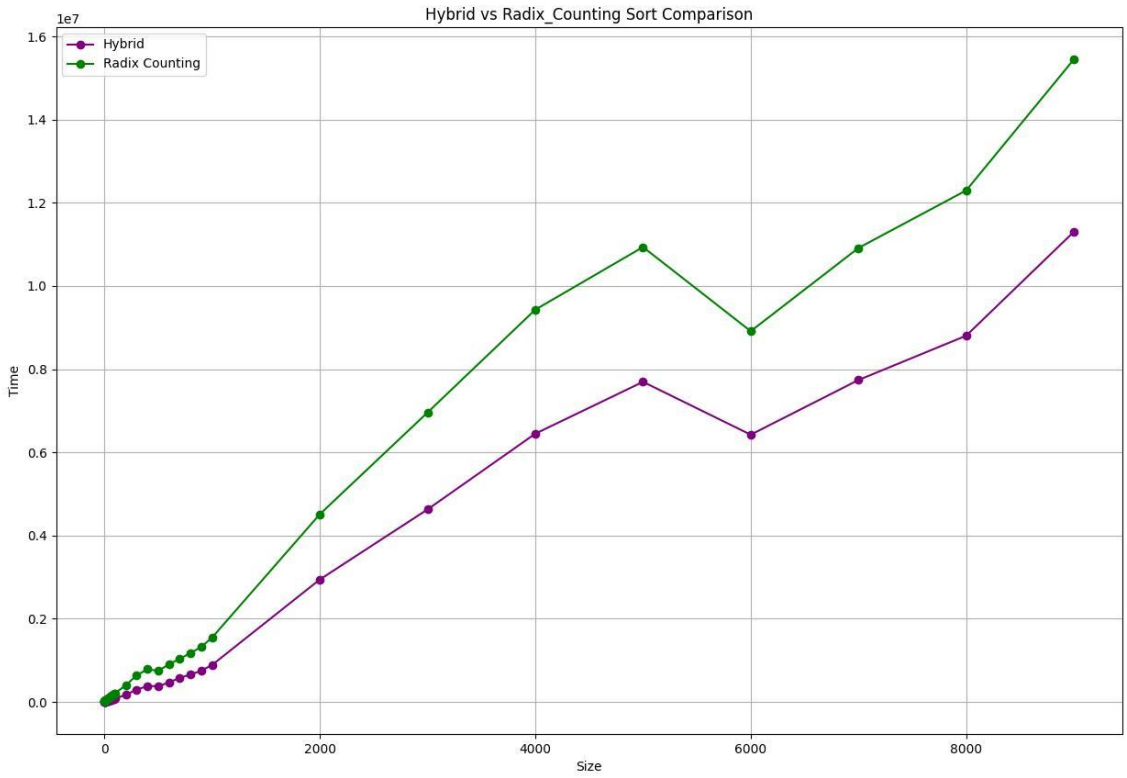
Time of each algorithm takes to sort randomly generated unsorted large size (size 5000 to 9000) arrays



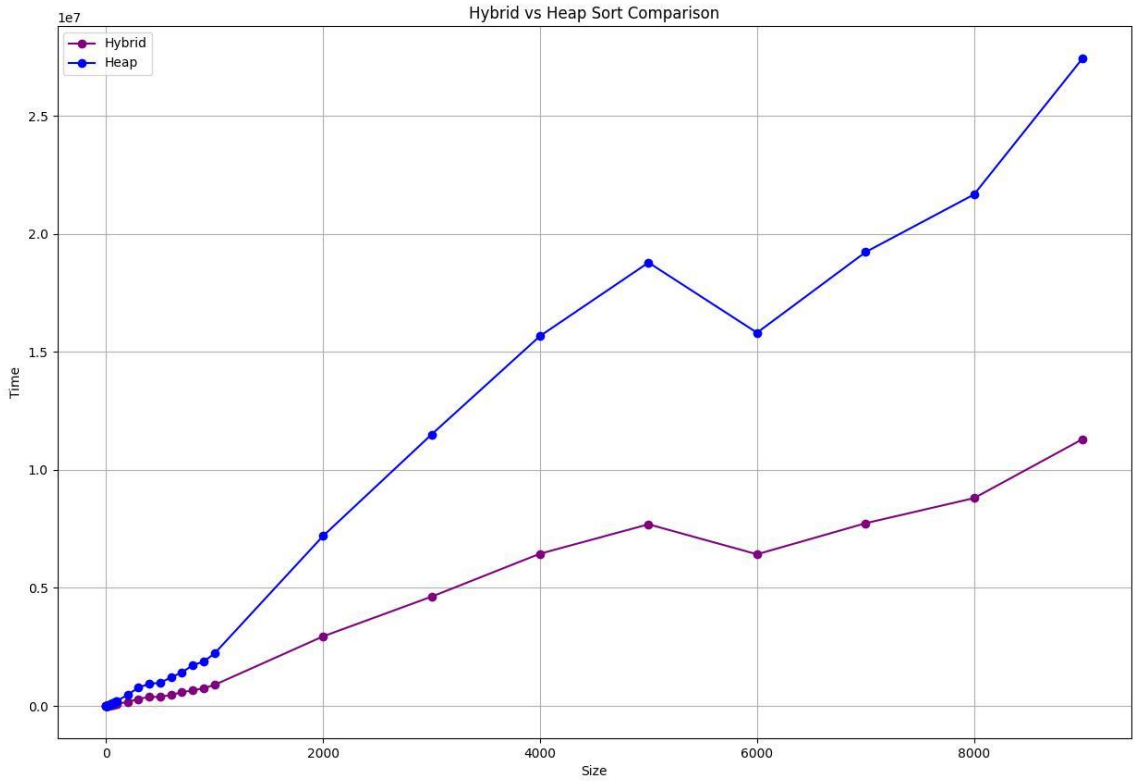
Comparison of hybrid and insertion sort



Comparison of hybrid and quick sort



Comparison of hybrid and radix sort



Comparison of hybrid and heap sort