

# KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

## Report

### Department of Computer Science and Engineering

**Course Title:** Data Structures and Algorithms Laboratory

**Course No:** CSE 2106

**Topic:** Sorting Algorithms Implementation and Time Complexity Comparison



<b>Name</b>	: Kazi Rifat Al Muin
<b>Roll</b>	: 2107042
<b>Group</b>	: A2
<b>Year</b>	: 2 <sup>nd</sup>
<b>Semester</b>	: 1 <sup>st</sup>
<b>Date of Submission</b>	: 25/04/2024

## Objectives

- To know about different 8 different sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort, Radix Sort, Counting Sort and Bucket Sort.
- To Learn how to implement these 9 sorting algorithms and how they work.
- To compare their theoretical time complexities and empirical time complexities obtained from the runtime measurements.
- To analyze if any unexpected result is found and discuss its reason
- To know the best use case of every algorithm and efficiently choose the algorithm.

## Introduction

Sorting algorithms are crucial in computer science, providing efficient means to organize data. Bubble Sort compares adjacent elements and swaps them until the list is sorted, while Insertion Sort builds the final array by inserting each element in its correct position among the already sorted elements. Selection Sort divides the list into sorted and unsorted parts, selecting the smallest element for sorting and repeatedly swapping it with the first unsorted element. Quick Sort employs a divide-and-conquer strategy, selecting a pivot to partition the array into two sub-arrays and recursively sorting them. Merge Sort divides the array into halves, sorts them independently, and then merges them to produce a single sorted array, ensuring stable sorting. Heap Sort utilizes a binary heap to repeatedly extract the root element, maintaining the heap property until the array is sorted. Radix Sort sorts elements by grouping digits and sorting based on significant positions, while Count Sort orders elements by counting their frequencies without comparison. Bucket sort divides elements into buckets, sorts each individually, and then concatenates them to achieve overall sorting, often employed when the input is uniformly distributed across a range. Sorting algorithms offer versatile techniques to efficiently organize data, each with its unique approach tailored to specific requirements and constraints, facilitating various applications in computer science and beyond.

## Implementation

Attached is a C++ file containing implementations of eight sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, Heap Sort, Radix Sort, Count Sort, and Bucket Sort. Each algorithm is thoroughly commented on to explain its operation. The code utilizes a random number generation function to create arrays of varying sizes, and the C++ chrono library measures the execution time of each sorting algorithm for each input size. This approach allows for a standardized evaluation of the efficiency and effectiveness of these sorting techniques.

## Time Complexity Analysis

### Theoretically

Here are the theoretical best-case, average-case, and worst-case time complexities for the implemented sorting algorithms :

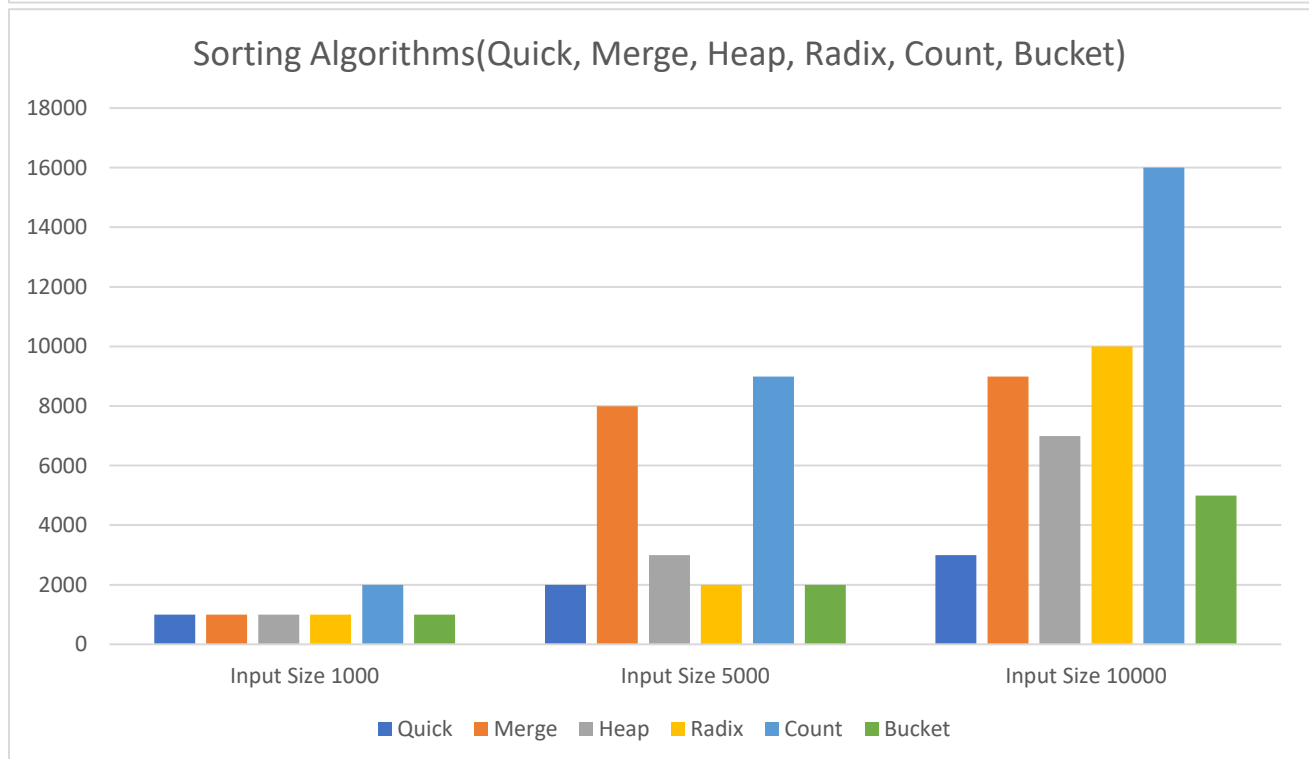
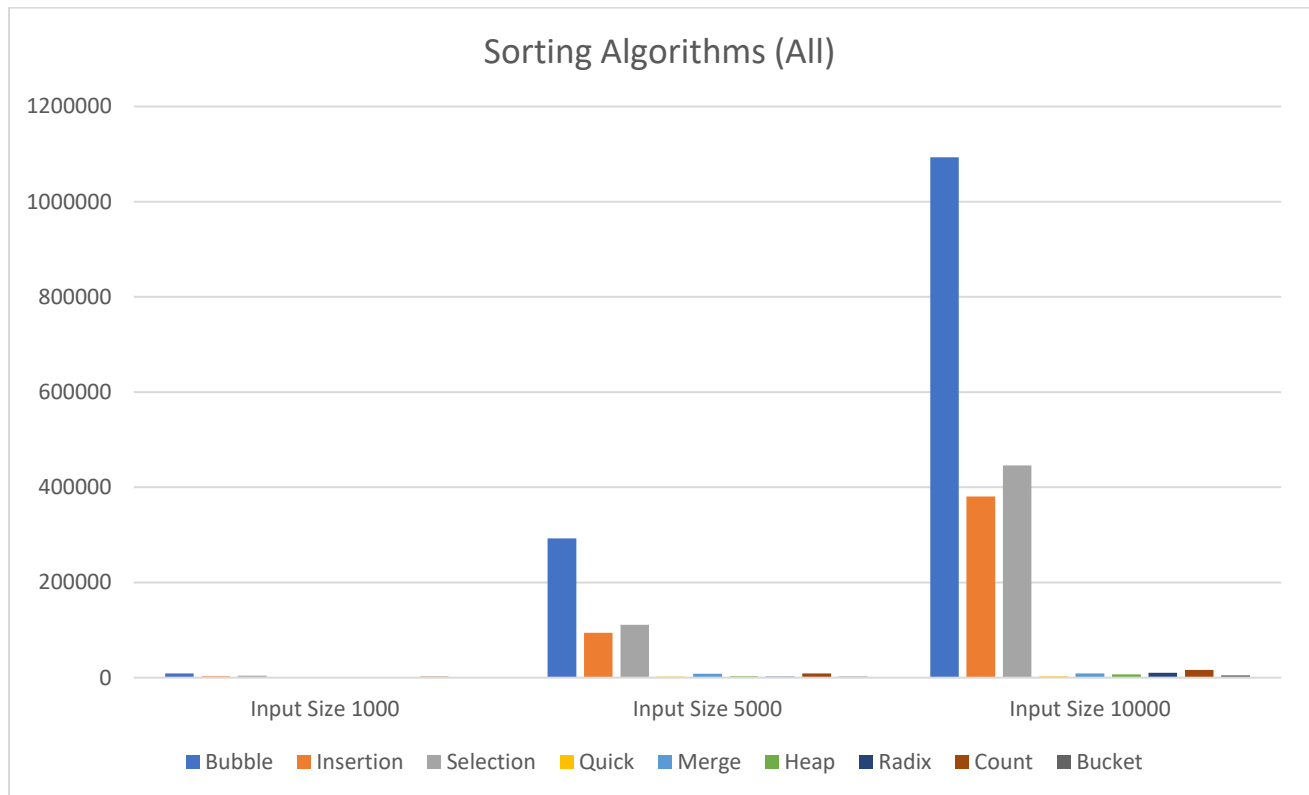
Algorithm	Worst Case	Average Case	Best Case
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Radix Sort	$O(n * d)$	$O(n * d)$	$O(n * d)$
Count Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$
Bucket Sort	$O(n^2)$	$O(n + k)$	$O(n)$

## Empirical Analysis :

Here is the dataset from the implemented program for different input array sizes:

Input Size & Elements	Algorithm	Average Execution Time ( $\mu$ S)
Input size: 1000 Elements up to 100000	Bubble Sort	8994
	Insertion Sort	2997
	Selection Sort	3996
	Quick Sort	998
	Merge Sort	1000
	Heap Sort	998
	Radix Sort	999
	Count Sort	1996
	Bucket Sort	999
Input size: 5000 Elements up to 500000	Bubble Sort	292817
	Insertion Sort	93940
	Selection Sort	110930
	Quick Sort	1997
	Merge Sort	7994
	Heap Sort	2997
	Radix Sort	1999
	Count Sort	8994
	Bucket Sort	1998
Input size: 10000 Elements up to 1000000	Bubble Sort	1093320
	Insertion Sort	380764
	Selection Sort	445723
	Quick Sort	2998
	Merge Sort	8994
	Heap Sort	6996
	Radix Sort	9994
	Count Sort	16000
	Bucket Sort	4996

## Graphs



## Discussion and Analysis

From the analysis of actual execution time, we can see that the bubble sort gives the worst performance. Insertion sort gives better performance than the other two sorting algorithms with  $O(n^2)$  time complexity.

The remaining 5 algorithms give far better performance. From all, the quick sort amazingly gives excellent performance. Despite having the worst-case complexity of  $O(n^2)$ , most of the time quick sort gives the surprisingly best result.

Merge sort gives a consistent performance in large input sizes. Heap sort gives a slightly better result than merge sort. Radix sort gives very good performance in smaller input sizes and increases in larger input sizes. The count sort gives worse performance than these other 4 and takes more time with a bigger input size.

The unexpected result was from quick sort, which worked so much faster than expected. The bucket sort also gives a very good performance than expected especially for larger sizes. However, In practice, the quick sort algorithm performed the best of all.

## Conclusion

According to input size and data size, the performance of every algorithm differs. All algorithms are important in different cases. We need to understand when and why the algorithms perform better or worse. According to that, We can pull off the best result from the sorting algorithms.