Assignment - 1

National Institute of Technology Karnataka, Surathkal

Department of Computer Science and Engineering

Algorithms and Complexity

ANALYSIS OF SORTING ALGORITHMS

Submitted To:
Vani M
Associate. Professor

Submitted By :
Sulthan Vishnu Sai
242CS035

1 Introduction

- **Bubble Sort**:
  - Best: $O(n)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$
- **Insertion Sort**:
  - Best: $O(n)$
  - Average: $O(n^2)$
  - Worst: $O(n^2)$
- **Merge Sort**:
  - Best: $O(n \log n)$
  - Average: $O(n \log n)$
  - Worst: $O(n \log n)$
- **Heap Sort**:
  - Best: $O(n \log n)$
  - Average: $O(n \log n)$
  - Worst: $O(n \log n)$
- **Radix Sort**:
  - Best: $O(nk)$
  - Average: $O(nk)$
  - Worst: $O(nk)$
- **Quick Sort**:
  - Best: $O(n \log n)$
  - Average: $O(n \log n)$
  - Worst: $O(n^2)$

2 Data Generation and Experimental Setup

1. **What kind of machine did you use?**

Component	Details
Processor	Intel Core i5-8th Gen
Graphics	NVIDIA GeForce GTX 1450
Memory	8 GB RAM
Storage	512 GB SSD
Operating System	Windows 11 Home

Table 1: System Specifications

2. **What timing mechanism did you use?**
   Used Chrono c++ Standard Timing Mechanism
3. **How many times did you repeat each experiment?**
   Repeated the experiment approximately 45-50 times.
4. **How did you select inputs?**
   Generated Randomly positive integers in the range of 1 and 1000000
   (a) Random numbers
   (b) Increasing numbers
   (c) Decreasing numbers

5. **What times are reported?**

3 Best Case Performance (in ms)

	10000	25000	50000	75000	100000
Bubble sort	0.20718	1.36605	6.10309	10.9284	19.0614
Insertion Sort	3e-02	0.000157	0.000411	0.000562	0.000538
Merge Sort	0.012543	0.011729	0.030424	0.037725	0.051135
Quick sort	0.000494	0.002652	0.004339	0.004871	0.006042
heap Sort	0.002705	0.006864	0.028241	0.03628	0.037392
Radix sort	0.002623	0.003189	0.007033	0.011066	0.070337

Table 2: Best Case Performance for All Sorting Algorithms

Worst Case Performance (in ms)

	10000	25000	50000	75000	100000
Bubble sort	0.528138	3.32506	17.5298	33.1707	59.9983
Insertion sort	0.277589	1.77405	6.35969	15.7773	26.9017
Merge sort	0.006695	0.029246	0.026961	0.037569	0.059406
Quick sort	0.005643	0.021797	0.033806	0.004412	0.007124
Heap sort	0.001686	0.006428	0.013554	0.065634	0.053784
Radix sort	0.002678	0.014137	0.006269	0.032565	0.016748

Table 3: Worst Case Performance for All Sorting Algorithms

Average Case Performance (in ms)

	10000	25000	50000	75000	100000
Bubble sort	0.562817	3.71982	13.9775	32.9866	54.9291
Insertion sort	0.208834	1.085714	2.98445	7.97245	13.9378
Merge sort	0.008626	0.024758	0.040546	0.147155	0.05373
Quick sort	0.002694	0.00463	0.008943	0.065615	0.010627
Heap sort	0.002171	0.004647	0.019914	0.042011	0.039858
Radix sort	0.00204	0.004082	0.007012	0.009451	0.009183

Table 4: Average Case Performance for All Sorting Algorithms

6. **Did you use the same inputs for all sorting algorithms?**
   yes

4 Analysis of Quicksort

4.1 Graphs for elements sorted in Random order of the input file of size n for all three versions of quick sort

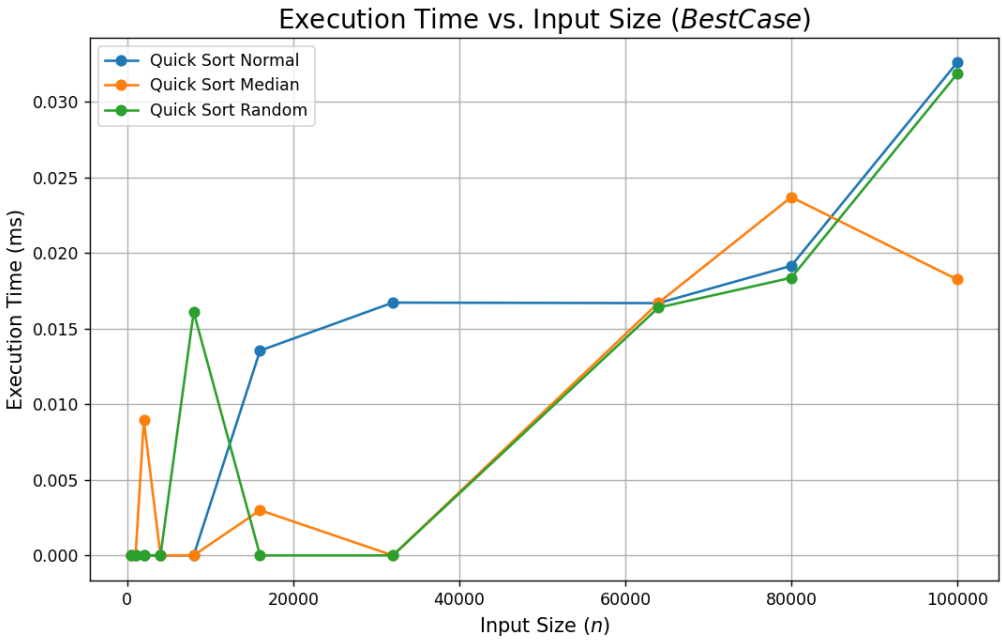

Figure 1: Best Case

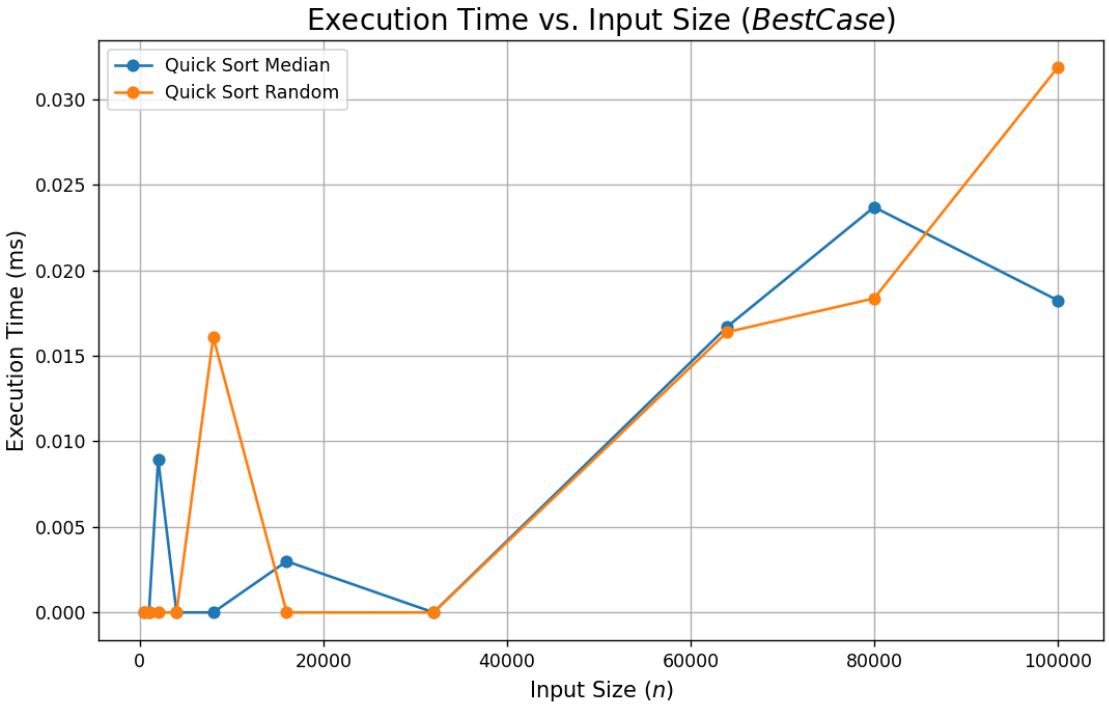

Figure 2: Expanded version of Best case (Excluding Quick sort - Normal pivot)

4.2 Graphs for elements sorted in Decreasing order of the input file of size n for all three versions of quick sort

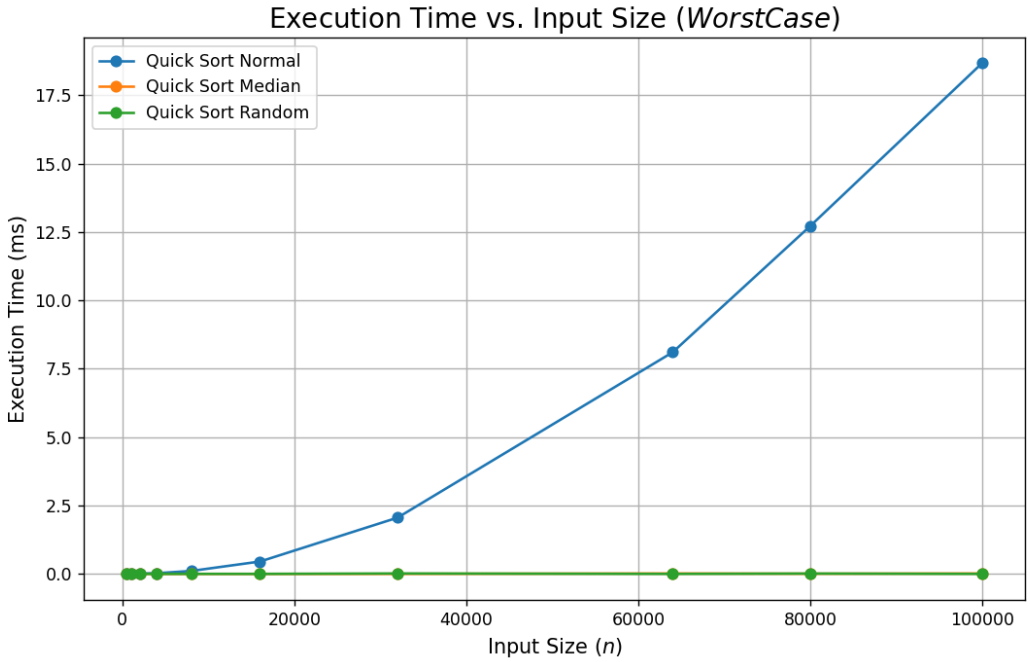

Figure 3: Worst Case

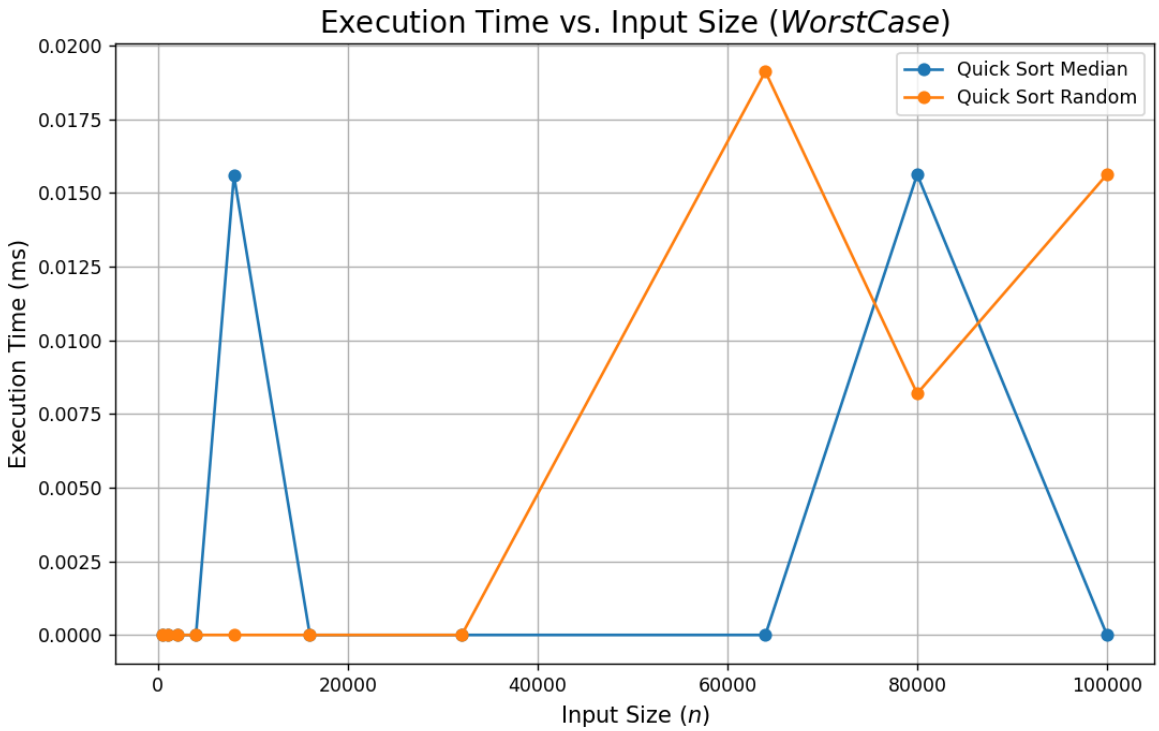

Figure 4: Expanded version of Worst case (Excluding Quick sort - Normal pivot

4.3 Graphs for randomly generated elements of the input file of size n for all three versions of quick sort

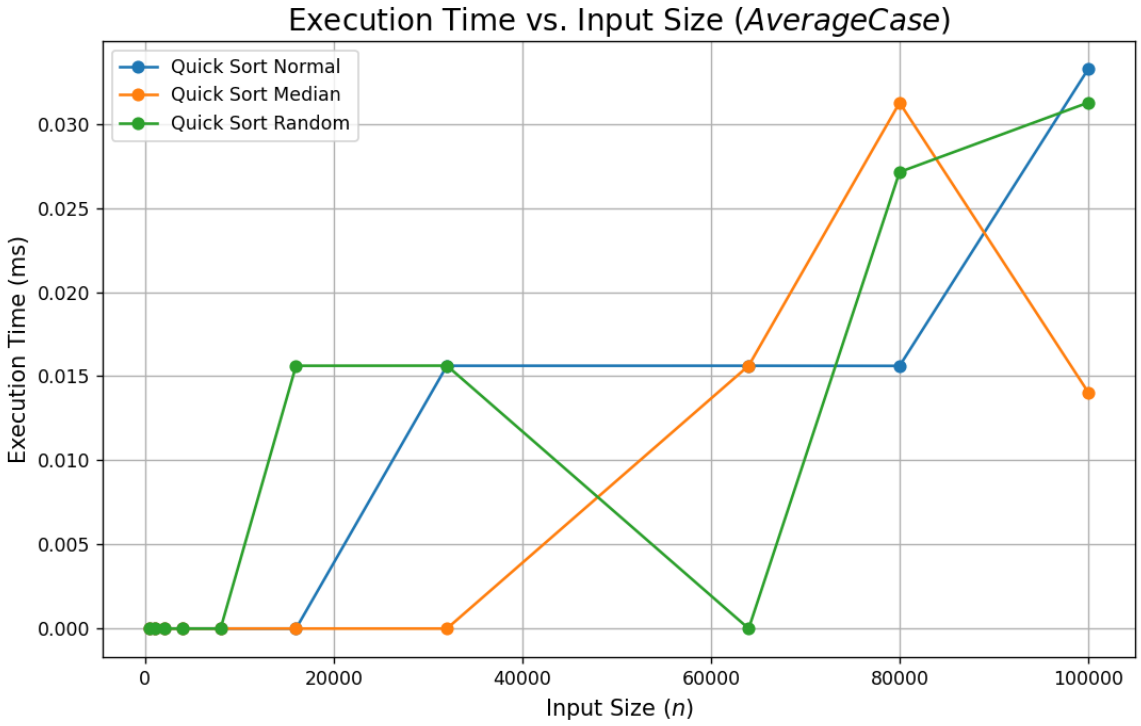

Figure 5: Average Case

Summary:

- In this section, we've explored three versions of Quicksort using different input orders: increasing, decreasing, and random illustrating how Quicksort's performance varies with different pivot selection strategies and input types.
- From the observation Increasing and Decreasing order input file gives the worst time but the Random element order input file gives the Best time.
- The best case for Quick Sort occurs when the pivot consistently splits the array into two nearly equal halves. This happens when the pivot is the median of the array or close to it.
- The plots likely show the time complexity for different input sizes, the median-pivot Quick Sort performs the best across all cases due to its ability to balance partitions effectively and its time complexity O(nlogn).
- The random-pivot version shows good average performance but with some variability. In contrast, the pivot-first version performs well in the Average case but poorly in the worst case because of its time complexity O(n^2).

5 Plots of all the six sorting algorithms

5.1 Graphs for elements sorted in increasing order of the input file of size n for 6 Alogorithms

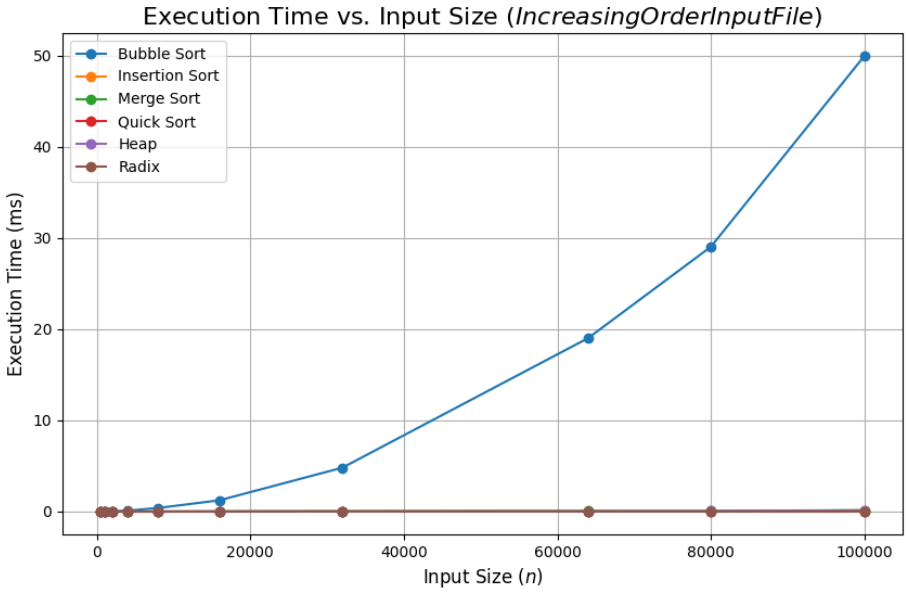

Figure 6: Best Case

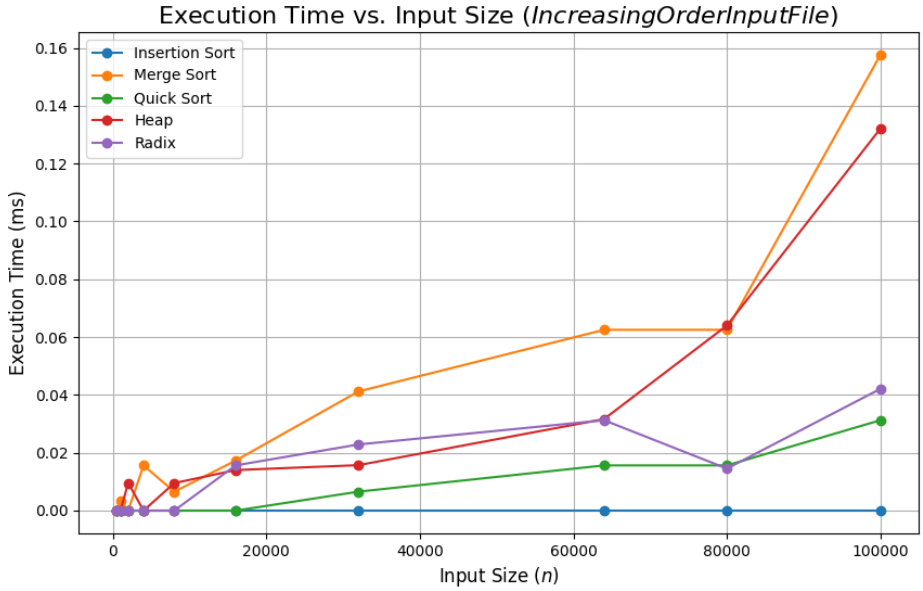

Figure 7: Expanded version of Best Case (Excluding Bubble sort)

5.2 Graphs for elements sorted in Decreasing order of the input file of size n for 6 Alogorithms

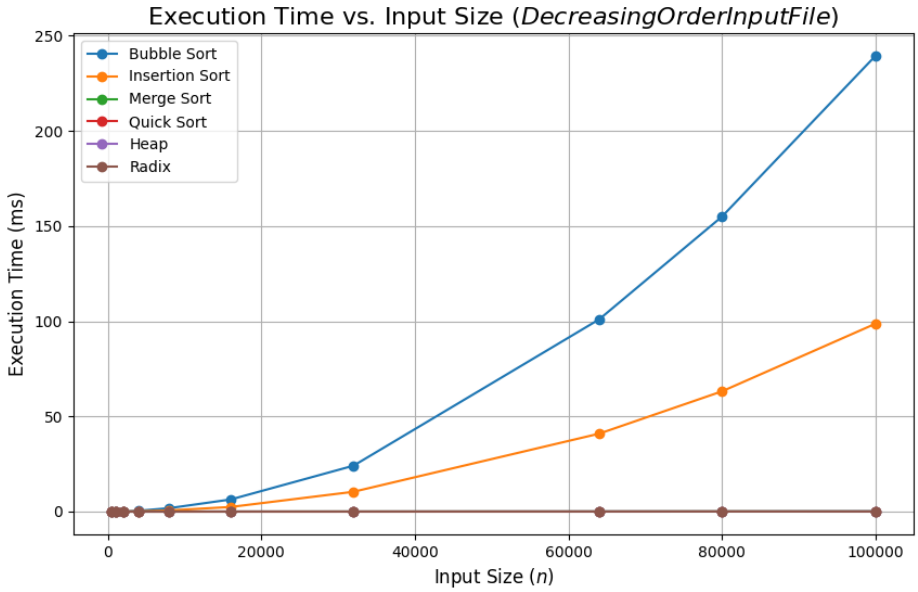

Figure 8: Worst Case

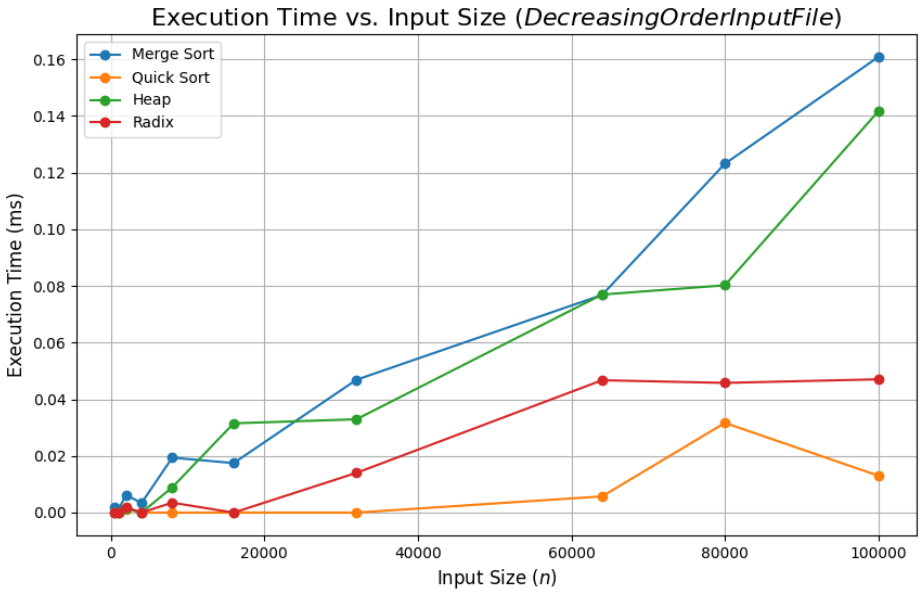

Figure 9: Expanded version of Worst Case (Excluding Bubble and Insertion sort)

5.3 Graphs for randomly generated elements of the input file of size n for 6 Alogorithms

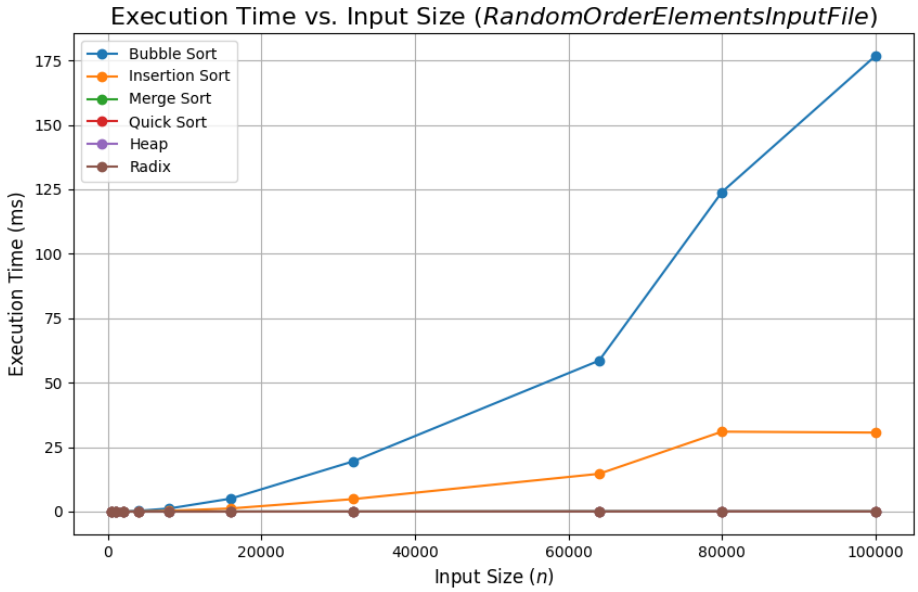

Figure 10: Average Case

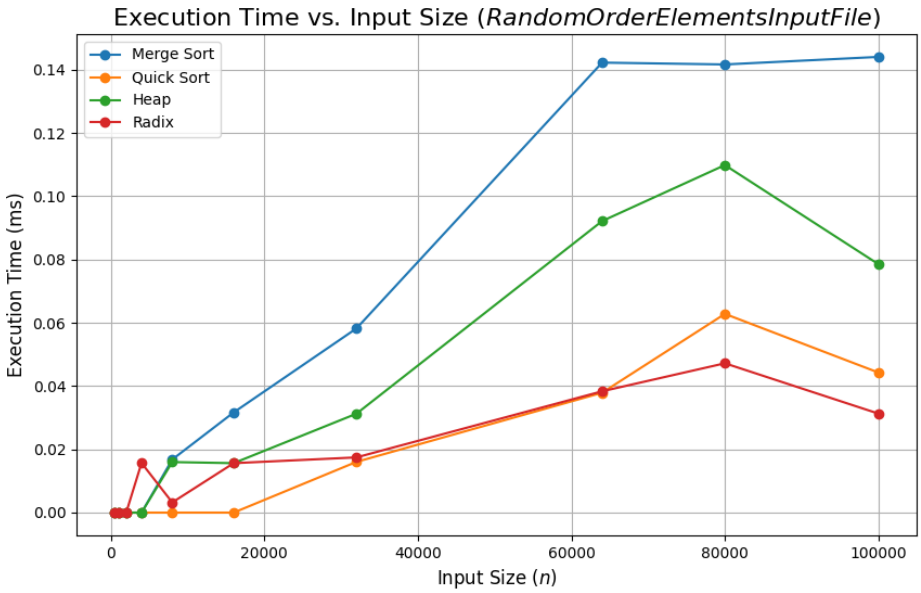

Figure 11: Expanded version of Average Case (Excluding Bubble and Insertion sort)

6 Analysis

6.1 Analysis of all the six sorting algorithms

- **Best case:** In O(N), the insertion sort works best. Thus, the same is followed by Merge sort, Quick sort, Heap sort, and Radix.
- **Worst case:** Quick sort works best in O(NlogN). Merge sort, Heap sort, and Radix sort follow Quick sort.
- **Average case:** The Radix sort works best in O(N * k), where the number's length affects the result. Radix comes first, followed by Merge, Heap, and Quick sorts; Bubble and Insertion sorts come last.

6.2 correlations

- **Quicksort:** correlation values for 0.8400, 0.9498,0.8824 in the best, 0.9758, 0.1887, 0.8582 in the worst case and in the average case, it is showing 0.9278, 0.9275, 0.8759 for all 3 types of quicksort variations and from all Algorithms Average case is the fastest algorithm
- **Merge and Heap sort:** Merge sort and heap sort are showing the same tendency in all the cases
- **Insertion Sort & Bubble sort:** A stronger correlation with n but with higher execution times compared to Merge and Heap Sort

6.3 Tables and its Observations are described below

- **Observations for Best case comparisons:**
  Highest: Bubble sort has the highest comparisons because of its time complexity.
  Least: Insertion sort and quick sort
  Insertion sort has the lowest number of comparisons since its best-case time complexity is O(n), which occurs when the elements are already sorted.

  Observations for worst case comparisons:
  Highest: Bubble sort has the highest comparisons because of its time complexity.
  Least: Quick sort has the least comparisons.
- **Observations for Average case comparisons:**
  Highest: Bubble sort, with a time complexity of O(n^2), has the most number of comparisons among the six methods.
  Least: Quick sort takes the least number of comparisons because of the time complexity of O(nlogn).

7 Number of comparisons for all Algorithms

Input Size	Bubble Sort	Insertion Sort	Merge Sort	Quick Sort	Heap Sort
500	124750	499	2272	3498	-2.7057e+08
1000	499500	999	5044	7988	1.94511e+09
2000	1.999e+06	1999	11088	17964	-1.48379e+09
4000	7.998e+06	3999	24176	39918	-2.03436e+08
8000	3.1996e+07	7999	52352	87832	-8.14244e+08
16000	1.27992e+08	15999	112704	191690	-3.98579e+07
32000	5.11984e+08	31999	241408	415485	-5.51602e+08
64000	2.04797e+09	63999	514816	896207	-2.38991e+08
80000	-1.09501e+09	79999	672064	1.15174e+06	2.07617e+09
100000	7.04983e+08	99999	853904	1.47008e+06	-3.50239e+08

Table 5: Number of Comparisons for Best case

Input Size	Bubble Sort	Insertion Sort	Merge Sort	Quick Sort	Heap Sort
500	124750	125248	2216	3499	1.56027e+09
1000	499500	500498	4932	7987	1.83129e+09
2000	1.999e+06	2.001e+06	10864	17966	-6.50258e+07
4000	7.998e+06	8.00199e+06	23731	39919	-8.28441e+06
8000	3.1996e+07	3.2004e+07	51465	87830	1.43076e+09
16000	1.27992e+08	1.28008e+08	110989	191685	-3.14779e+07
32000	5.11984e+08	5.12015e+08	238081	415446	-7.49897e+07
64000	2.04797e+09	2.04803e+09	508698	895271	1.57704e+09
80000	-1.09501e+09	-1.09493e+09	638714	1.14938e+06	-1.8037e+08
100000	7.04983e+08	7.05078e+08	818212	1.46957e+06	-2.16068e+08

Table 6: Number of Comparisons for Worst Case

Input Size	Bubble Sort	Insertion Sort	Merge Sort	Quick Sort	Heap Sort
500	124750	64143	3861	4114	-3.39755e+08
1000	499500	254297	8721	9300	-1.00684e+08
2000	1.999e+06	1.00357e+06	19415	20146	-1.15389e+08
4000	7.998e+06	4.0328e+06	42867	46445	-7.56965e+08
8000	3.1996e+07	1.60509e+07	93589	100572	-8.30499e+08
16000	1.27992e+08	6.37888e+07	203286	221932	-9.80502e+07
32000	5.11984e+08	2.55336e+08	438463	480722	-3.29141e+08
64000	2.04797e+09	1.02414e+09	941020	1.04991e+06	-3.4815e+08
80000	-1.09501e+09	1.59509e+09	1.20367e+06	1.30375e+06	-1.49623e+09
100000	7.04983e+08	-1.80742e+09	1.53651e+06	1.69852e+06	-1.41989e+07

Table 7: Number of Comparisons for Average Case

- Radix sort uses counting sort instead of comparison sorting since the components are sorted according to the digit values of the numbers because the range of the numbers is narrow. Thus, in every case, there are exactly zero comparisons overall.