Bangladesh University of Engineering and Technology

Course No: CSE 204

Course Title: Data Structure and Algorithm – 1

Offline Assignment – 7

<u>Sorting Algorithm (Merge Sort vs Quick Sort)</u>

Name: Md. Azizur Rahman Anik

Student Id: 1805115

Department: CSE

Section: B2

Date of Submission: 11-06-2021

Data Collection

Average Run time in milliseconds for sorting n integers in different input order using Merge sort and Quick sort.

Input Order	n= / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	1.2	1.44	2.50	5.09	27.04	119.54
	Quick	0.98	1.43	8.09	108.03	6017.04	801006.03
Descending	Merge	1.33	1.31	2.40	5.46	27.13	116.8
	Quick	1.05	1.18	6.72	102.58	5682.0	809576.88
Random	Merge	1.24	1.42	2.74	5.72	30.0	223.25
	Quick	0.96	1.12	1.71	3.68	27.89	115.0

Graph Plotting

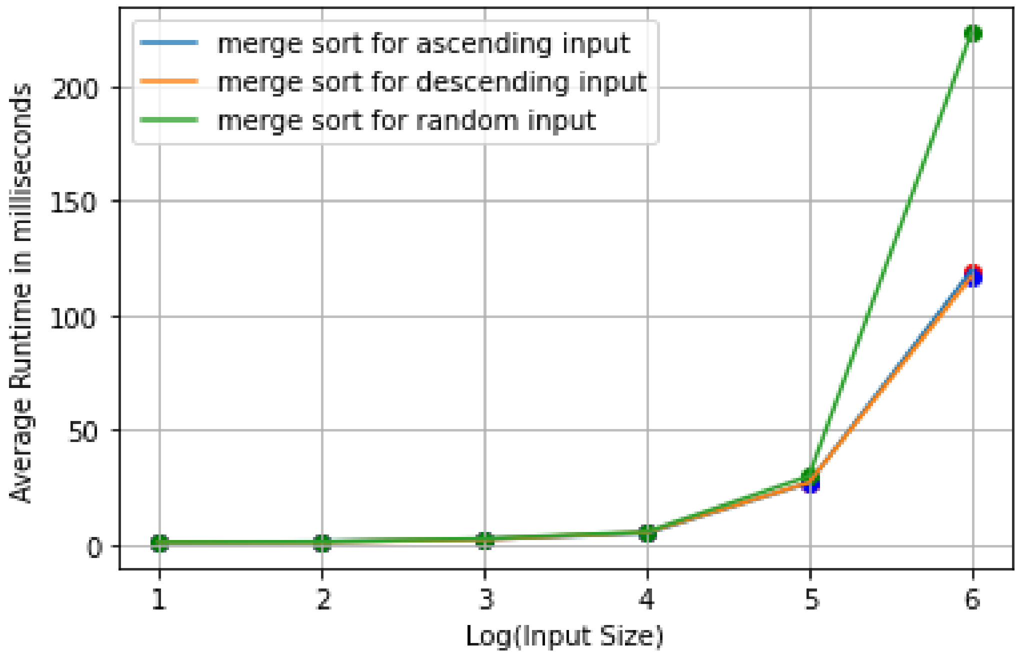

Fig-1: Merge Sort Statistics graph

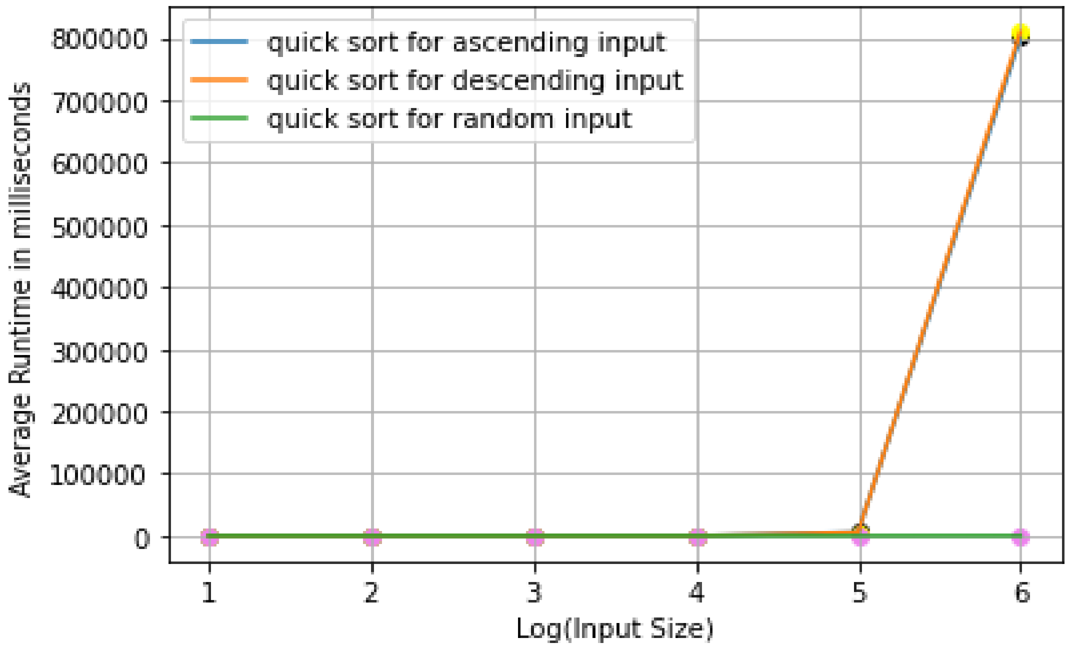

Fig-2: Quick Sort Statistics graph

Machine Configuration

Processor: Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz 3.30 GHz

Installed RAM: 8.00 GB

System type: 64-bit operating system, x64-based processor

OS: Windows 10 Pro

Complexity Analysis

Merge Sort:

From data table and the Fig-1, we see that as the input size increases the average running time increases in a non-linear fashion. Theoretically the time complexity of Merge sort is **O(nlogn)** where n= input size and whatever the input order is, i.e, **random, ascending or descending.**

But from the graph, we observe that, for random order input, average running time is increasing more than those of ascending or descending order. Moreover, if we observe the increase ratio in average running time, for example, for random order input of 10000 and 100000 numbers, the average running time increasing ratio is 30/5.72 = 5.24 which theoretically should be 12.5. In case of 1000 to 10000 inputs, the ratio is 2.09 which theoretically should be 13.33.

The changes in runtime ratio between theoretical analysis and practical data analysis is due to the device's configuration. For different types and sizes of inputs, the operating system may provide different CPU time and comparing, swapping etc. operations may take different times based on the categories of inputs which are considered constant time in theoretical analysis.

Quick Sort:

From data table and Fig-2, we see that the average running time for different types of input doesn't show any similar type of behavior. This is because, the core quick sort algorithm works in a manner that, for **random inputs**, the time complexity is **O(nlogn)** which is the **best case** for Quick sort and for **ascending or descending order inputs**, the time complexity is **O(n^2)** which is the **worst case** for Quick sort. The worst case can be overcome by taking the pivot in a random position and keep the time complexity O(nlogn) for all types of input.

Like we've shown in the merge sort analysis, the average runtime increase ratio doesn't match with theoretical analysis for quick sort too. This is also caused due to the device's configuration and constant time assumption for some operations in theoretical analysis.

Merge Sort vs Quick Sort:

For Random Order Inputs:

If we compare the complexity of Merge sort and Quick sort in basis of space and time, both algorithm have a space complexity of O(n) where n=size of input. But in case of time complexity, for random inputs, both algorithm have O(nlogn) theoretically. But from fig-3, we see that, in practical usage, quick sort takes less time as the input size gets larger.

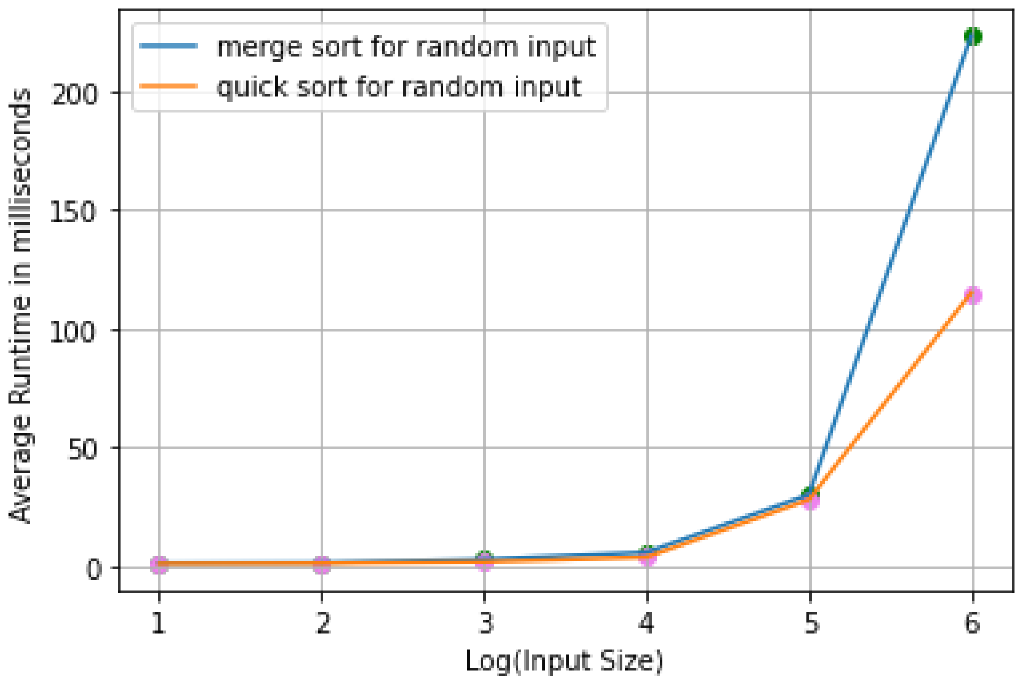

Fig-3: Merge Sort vs Quick Sort for Random Order Inputs

<u>**For Ascending or Descending Order Inputs:**</u>

For ascending or descending order inputs, theoretically the time complexity of quick sort is O(n^2) and merge sort is O(nlogn). And from this we can guess that, if the inputs are already sorted in ascending or descending order, then for larger input size, quick sort will take huge time compared to merge sort and so is the result if we observe the data table and Fig-4 and Fig-5.

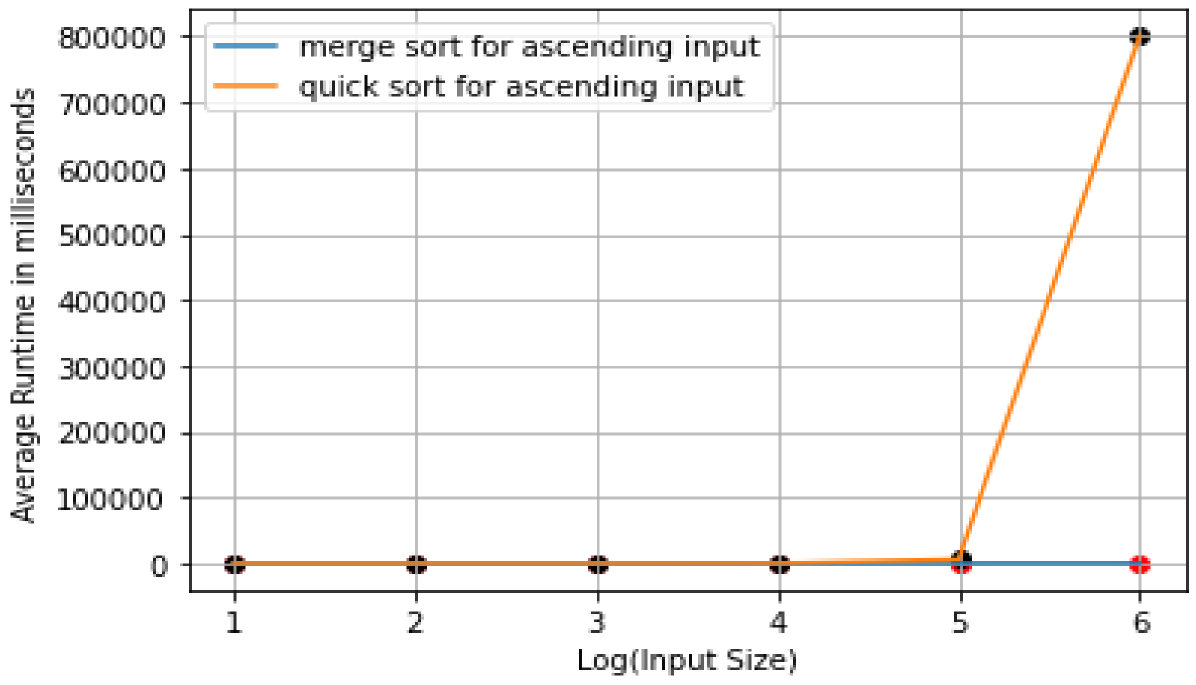

Fig-4: Merge Sort vs Quick Sort for Ascending Order Inputs

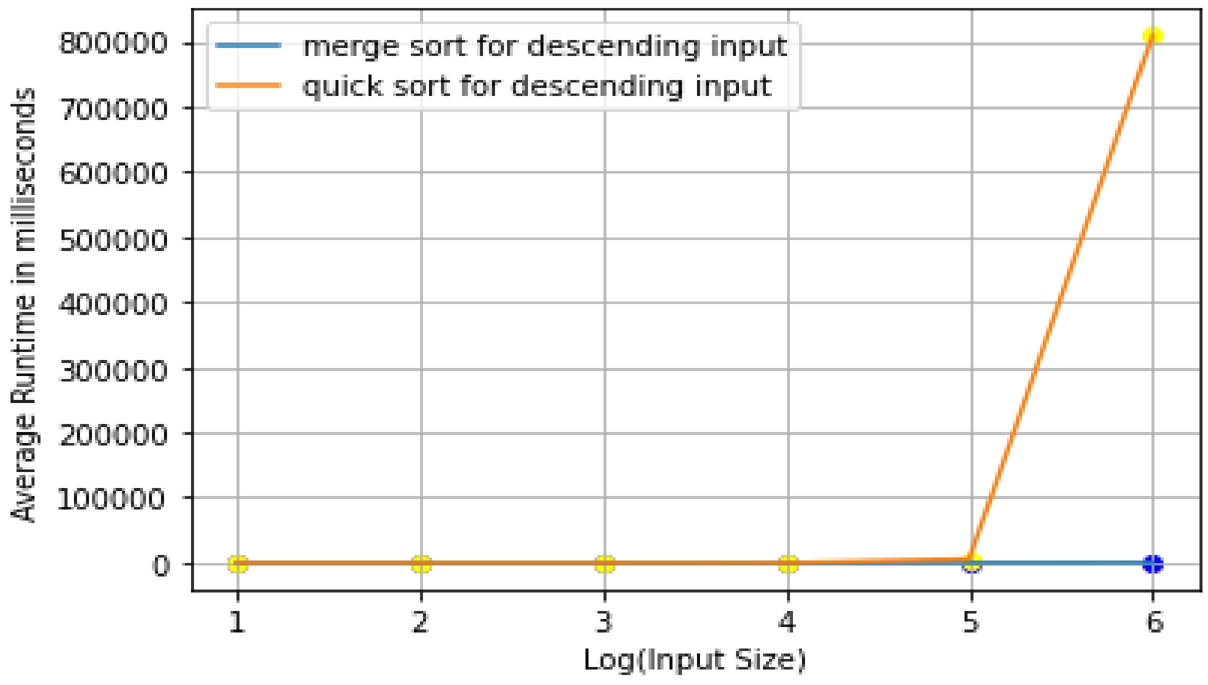

Fig-5: Merge Sort vs Quick Sort for Descending Order Inputs

Therefore, we can conclude that, if there’s an assurance that ascending or descending order inputs will never occur, the obviously quick sort is the better option as it requires less time for random inputs for as the input size increases rapidly. On the other hand, if there’s a possibility of having ascending or descending order input, then merge sort is always the better option.