Name: Mashiyat Mahjabin Prapty CSE 204

Student ID: 1805117 Offline 7

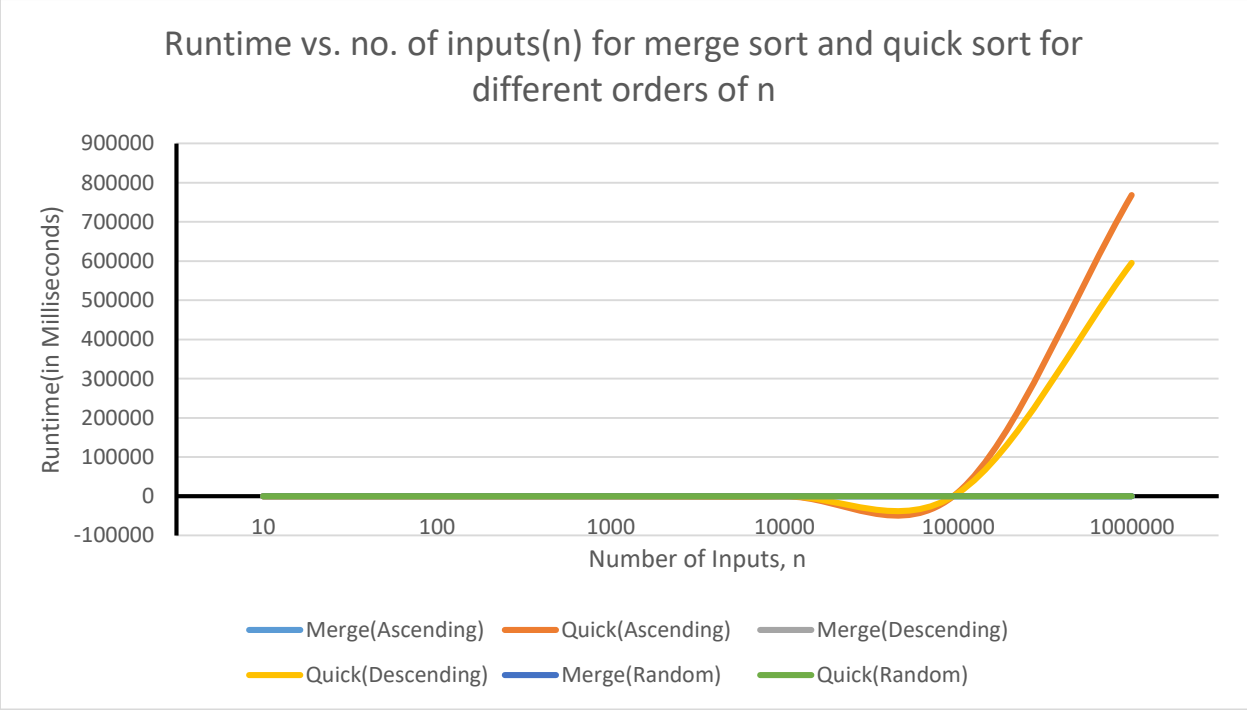
Machine Configuration:

Processor: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Installed RAM: 8.00 GB
System type: 64-bit operating system, x64-based processor
OS: Windows 10 Home Single Language

Table: Average time for sorting n integers in different input orders

Input Order	n = / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	0	0	0	0	2.992	31.212
	Quick	0	0	0	78.107	10192.4	768548
Descending	Merge	0	0	0	0	4.99	56.226
	Quick	0	0	0	78.092	7700.23	595181
Random	Merge	0	0	0	0	15.606	109.412
	Quick	0	0	0	0	15.647	124.971

Runtime vs. number of inputs graph:



Complexity Analysis of Merge Sort:

1) In random order:

$$\frac{15.606}{10^5 \times \log_2 10^5} = \frac{109.412}{10^6 \times \log_2 10^6} \approx constant$$

2) In ascending order:

$$\frac{2.992}{10^5 \times \log_2 10^5} = \frac{31.212}{10^6 \times \log_2 10^6} \approx constant$$

3) In descending order:

$$\frac{4.99}{10^5 \times \log_2 10^5} = \frac{56.226}{10^6 \times \log_2 10^6} \approx constant$$

So, it can be concluded that merge sort can run in O(nlogn) for best/average/worst case.

Complexity Analysis of Quick Sort:

1) In random order:

$$\frac{15.647}{10^5 \times \log_2 10^5} = \frac{124.971}{10^6 \times \log_2 10^6} \approx constant$$

2) In ascending order:

$$\frac{10192.4}{(10^5)^2} = \frac{764538}{(10^6)^2} \approx constant$$

3) In descending order:

$$\frac{7700.23}{(10^5)^2} = \frac{595181}{(10^6)^2} \approx constant$$

Quick sort runs in O(nlogn) in average or best case, but in worst cases such as in case of an ascending or descending sorted array, the algorithm takes O(n^2) time to run.

Comparative Analysis between the Complexities of Merge and Quick sort algorithms:

From the data table as well as the graph, we can conclude that for random inputs, the runtimes of both the algorithms are almost same or near to each other. On the other hand, for sorted arrays (in ascending or descending orders), merge sort is quite efficient compared to quick sort, which should actually be avoided in these cases.

N.B. - Data were taken several times for each number of inputs and the average was taken for the analysis. A rough estimation of complexity was done at first by introducing a count variable inside the loops of the algorithms. From the value of the count the primary complexity was predicted and then the above analysis was done.