`

Name: Iftekhar E Mahbub Zeeon

ID: 1805038

Course: CSE204

Offline on Sorting Algorithm(Quick Sort and Merge Sort)

Data Table:

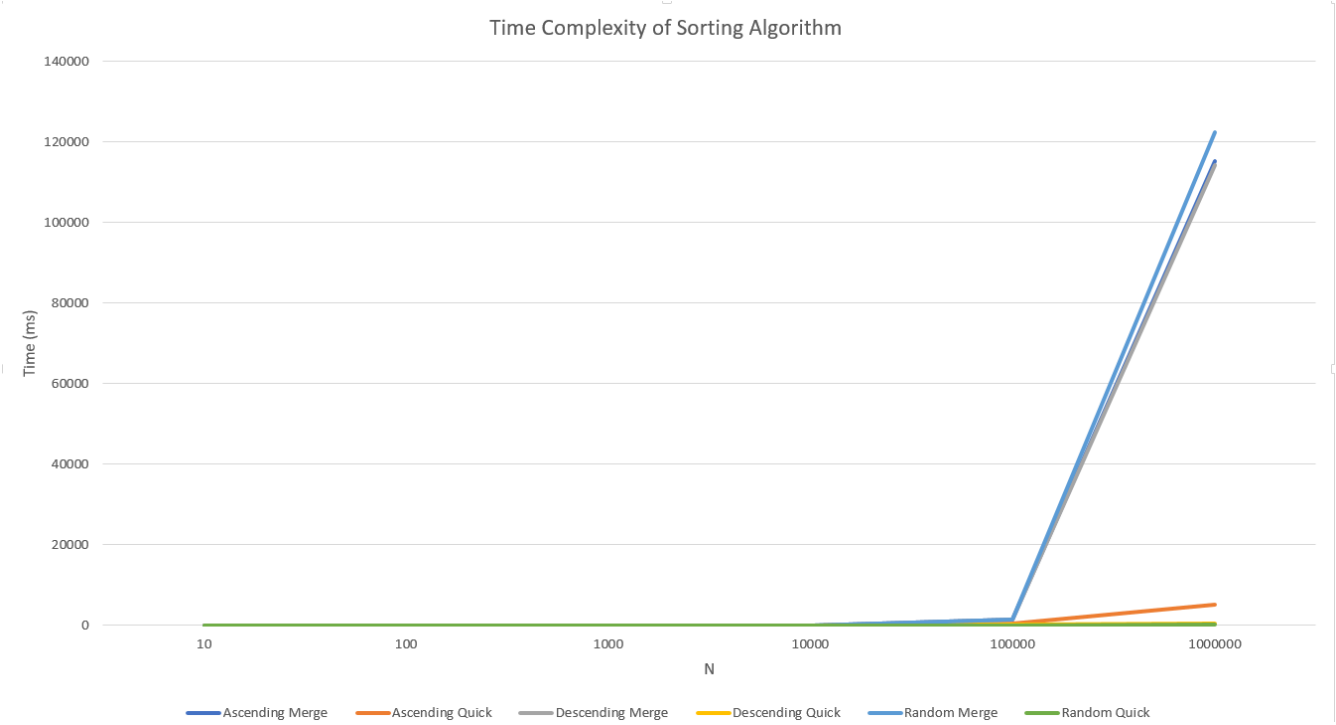
*Table: **Average time** for sorting **n integers** in different **input orders***

Input Order	N = / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	1.3748	1.522	3.3293	51.5911	1534.1331	115294.3992
	Quick	0.0173	0.2358	6.585	23.6454	398.3575	5209.8398
Descending	Merge	1.3873	1.5386	3.8572	55.4266	1384.2887	114251.592
	Quick	0.0198	0.405	6.1934	37.8165	154.1828	356.4776
Random	Merge	1.2995	1.48	3.2975	53.9234	1380.8468	122381.9985
	Quick	0.0183	0.0754	0.4676	1.9392	16.217	128.8368

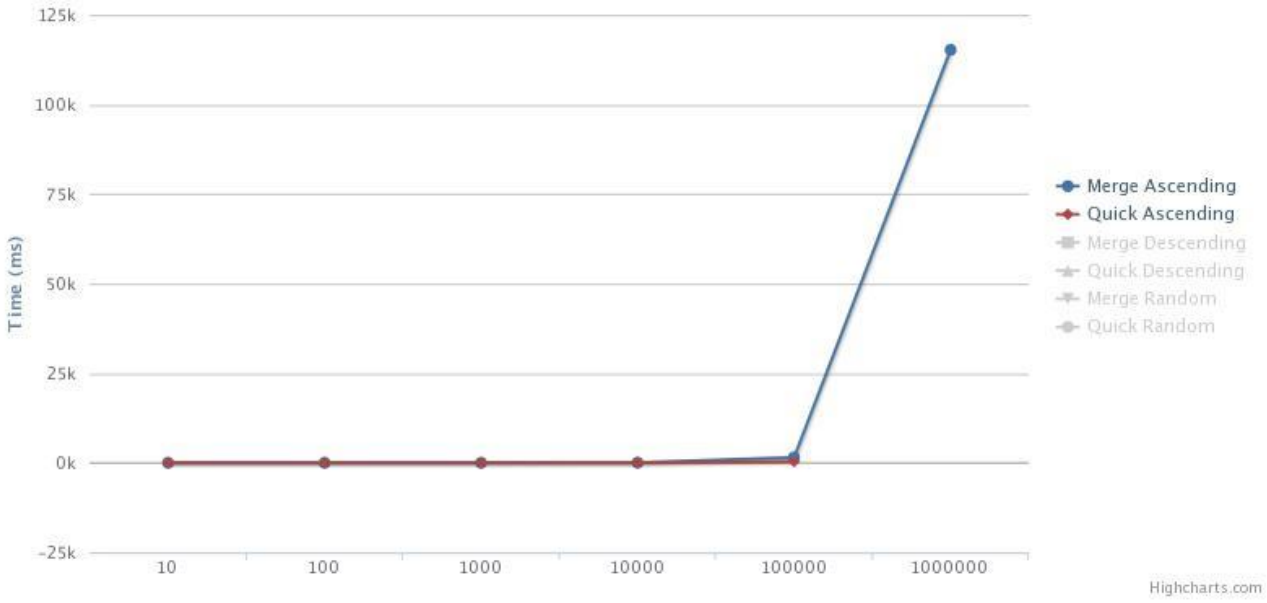
`

Complexity Analysis Graphs:

*Time vs N graph for **Both Algorithm***

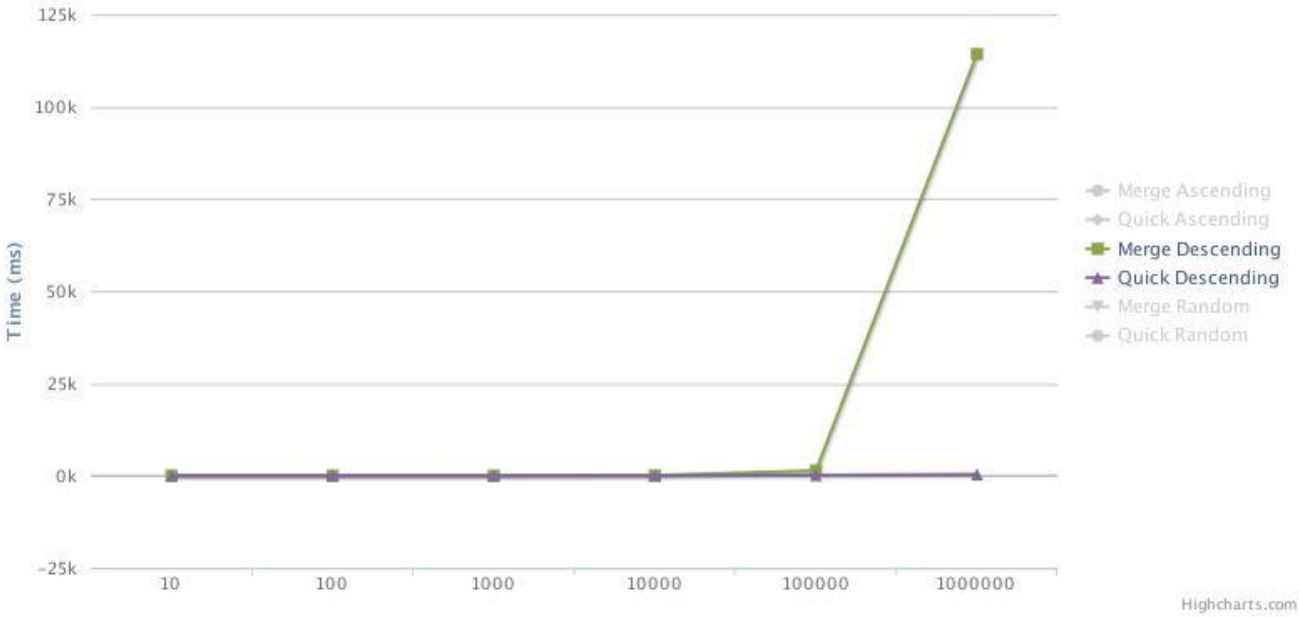


*Time vs N graph for **Ascending** input order*

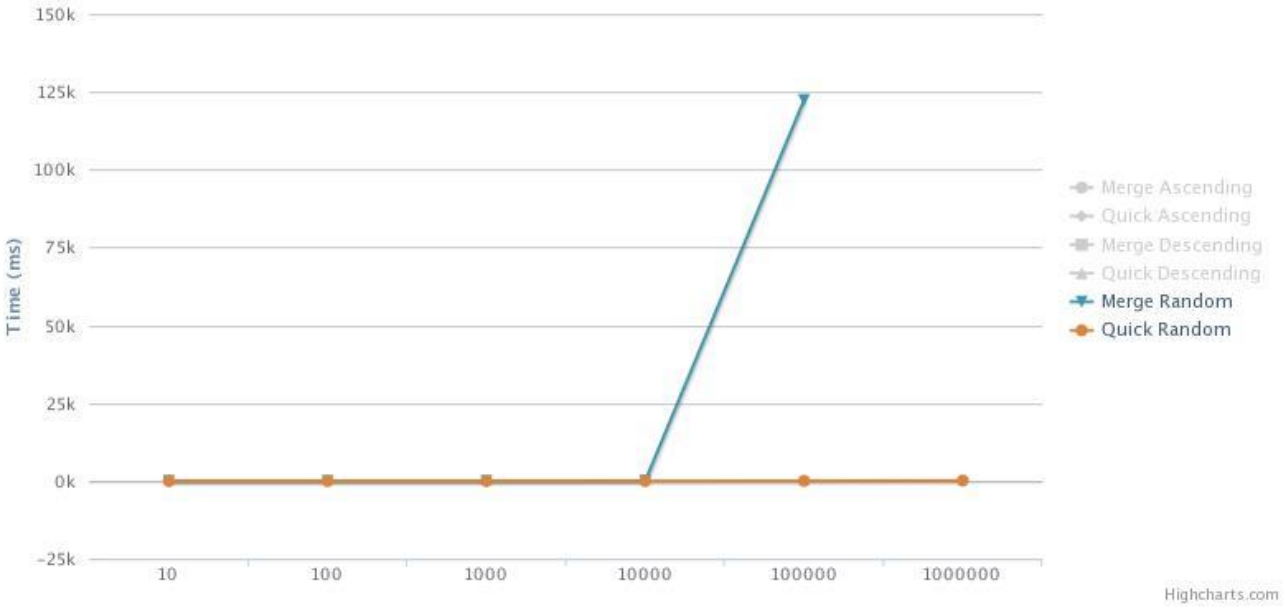


`

*Time vs N graph for **Descending** input order*

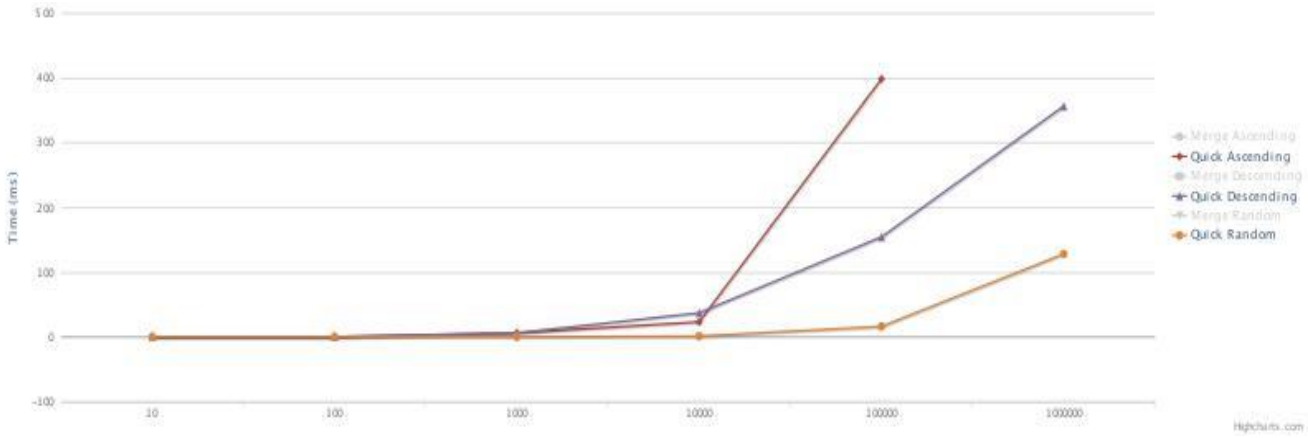


*Time vs N graph for **Random** input order*

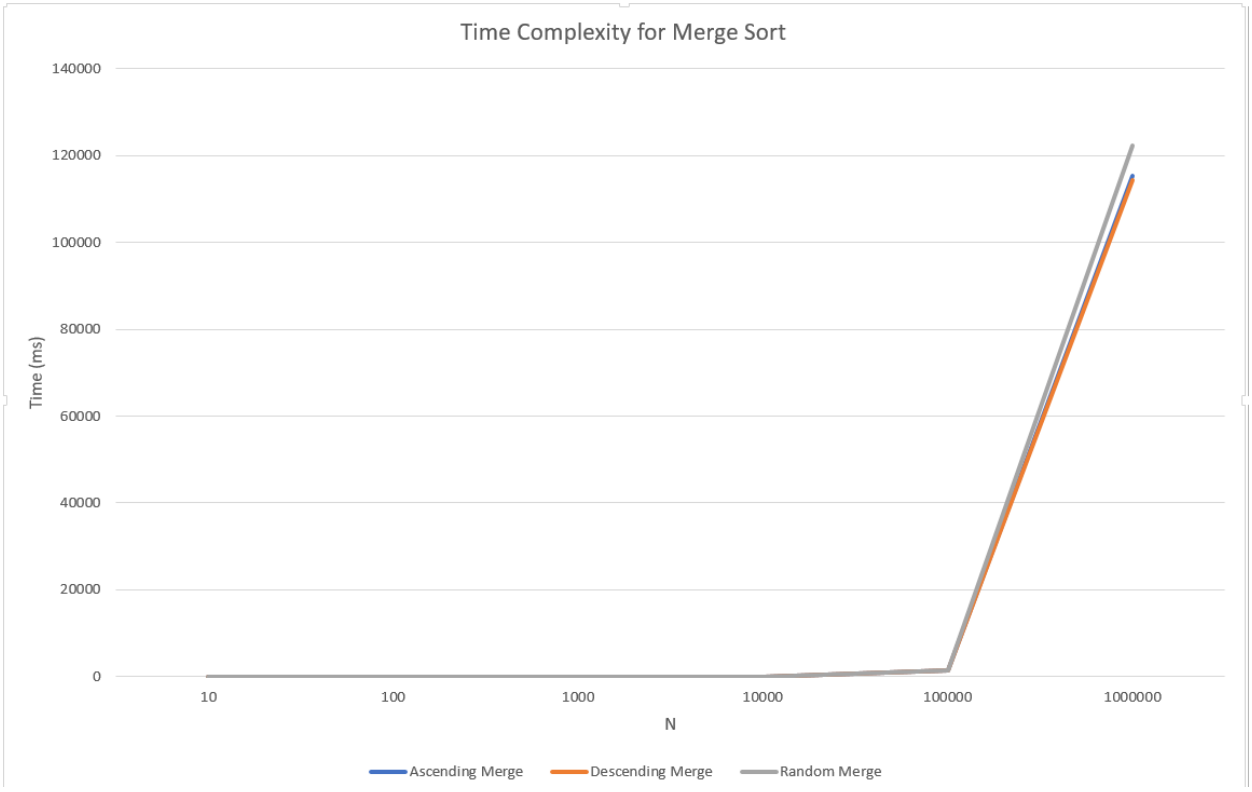


`

*Time vs N graph for **QuickSort in Different** input order*



*Time vs N graph for **MergeSort in Different** input order*



`

Analysis Table:

Basis for comparison	Quick Sort	Merge Sort
Ascending	In our algorithm the pivot element was the last element. As the algorithm was implemented on the ascending order, same order as input was the worst case for it. Normally quicksort takes less time than merge sort. However it is not stable and varies from machine to machine. Here as the input data increases, the time difference between the random and ascending is noticeable. For random the complexity was O(nlogn), but in case of ascending, it became $O(n^2)$. In case of 10^6 data as inputs, as the recursion goes so deep, stack overflow error occurs sometimes.	In the merge sort, the array is always parted into just 2 halves (i.e. n/2). So the complexity usually remains consistent for any size of data and any sort of order. The complexity for merge sort is O(nlogn), and since it takes more space than quicksort, usually it is less sufficient than quicksort. Since it remains consistent, the execution time doesn't change much for ascending, descending or random.
Descending	Input as descending order is considered as worst case for quicksort when pivot element is fixed. However, in here for large amount data even though descending order takes more time than random order, but it doesn't take as much time as ascending. However, in case of	As explained earlier, merge sort is more consistent regarding execution time. It doesn't vary much whether the input array is in descending or ascending. As the input size

`

	10^6 data, it still causes stack over flow error sometimes.	increases, it follows it's normal complexity form.
Random	Random order is considered as the average case for quicksort. In case of random order, pivot element randomly divides the array into different parts when in case of ascending or descending, usually array is not divided into small parts at the beginning, so execution time is very high. But for random case, the complexity is O(nlogn). Even for large data size, it is seen that it is more efficient than ascending and descending.	Random order doesn't change the working procedure of merge sort. From the table it is seen that whether it is random or ascending or descending, the behavior is same.

From the analysis, it is clear that quicksort is more unstable than merge sort and mainly depends on the configuration of the machine. Here the machine configuration that was used to implement these algorithms is given below.

Machine Configuration:

- Ram: **8 GB**
- Processor: **Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz**
- System Type: **64-bit operating system**
- OS: **Windows 10**
- Storage: **512GB SSD**
- IDE Used: **IntelliJ**
- Approx. Heap Memory Used for IntelliJ: **1.5GB**