Machine Configuration:

Processor: AMD Ryzen 5 3600 6-Core Processor 3.60GHz

RAM: 16GM

Operating System: Windows 10 64-bit Operating System, x64-based processor

Complexity Analysis:

Merge Sort:

In merge sort, whether an array is sorted or not doesn't affect the running time,

Let the sorting time be T(n)

And for n = 1 , T(n) = O(1)

Otherwise,

T(n) = 2*T(n/2) + k*n ; because every time the array is divided into two part and then they are inserted in the main array. Here k is a constant.

And base case is T(n) = b;

Now, the worst case occurs when n = 2^p

- ⇨ P = log n

So, T(n) = kn+k*n* log n

Thus T(n) = n log n

So, time complexity for merge sort in all case is O(n log n)

And during sorting we only create two temporary arrays to store the sorted results. So, the space complexity is O(n)

Quick Sort:

Ascending and Descending:

For quick sort the worst case occurs when the array is sorted in ascending or descending order.

Let the sorting time be T(n)

And for n = 1 , T(n) = O(1)

Otherwise,

T(n) = T(n-1) + k*n ; because every time the pivot is the extreme element and the recursion is called for the left or right elements.

So, the time complexity is O(n^2)

Quicksort is an inplace sort, so the space complexity is O(n)

Random:

And for random numbers, we can assume that the pivot is set to the middle element at average,

So, T(n) = 2*T(n/2) + k*n ; because everytime the array is divided into two part and then they are inserted in the main array. Here k is a constant.

And base case is T(n) = b;

Now , the worst case occurs when n = 2^p

⇨ P = log n

So, T(n) = kn+k*n* log n

Thus T(n) = n log n

Quicksort is an in-place sort, so the space complexity is O(n)

Summary:

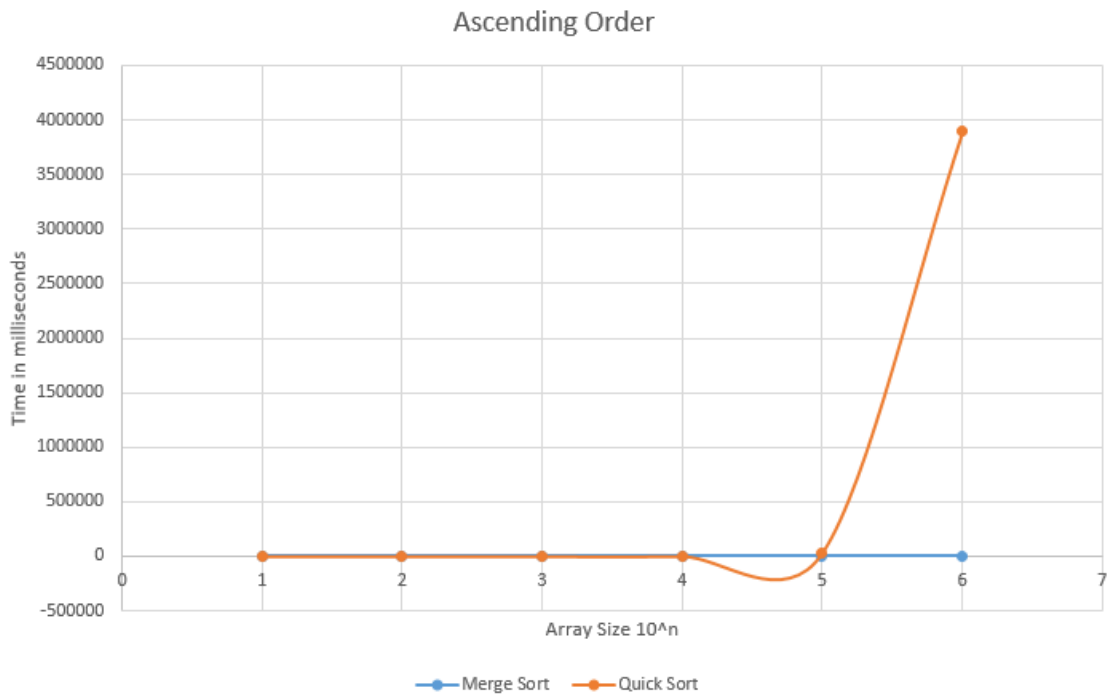
Input Order	Sorting Algorithm	Time Complexity	Space Complexity
Ascending	Merge	O(nlogn)	O(n)
	Quick	O(n^2)	O(n)
Descending	Merge	O(nlogn)	O(n)
	Quick	O(n^2)	O(n)
Random	Merge	O(nlogn)	O(n)
	Quick	O(nlogn)	O(n)

Time taken in milliseconds:

Input Order	N = / Sorting Algorithm	10	100	1000	10000	100000	1000000
Ascending	Merge	0.000000	0.000000	0.020000	0.400000	5.800000	72.000000
	Quick	0.000000	0.020000	2.740000	343.420000	34197.800000	3894732.000
Descending	Merge	0.000000	0.000000	0.000000	0.300000	5.900000	74.500000
	Quick	0.000000	0.100000	1.900000	194.200000	22160.300000	2331852.000
Random	Merge	0.000000	0.000000	0.100000	1.200000	12.300000	142.800000
	Quick	0.000000	0.050000	0.100000	1.000000	13.100000	163.300000

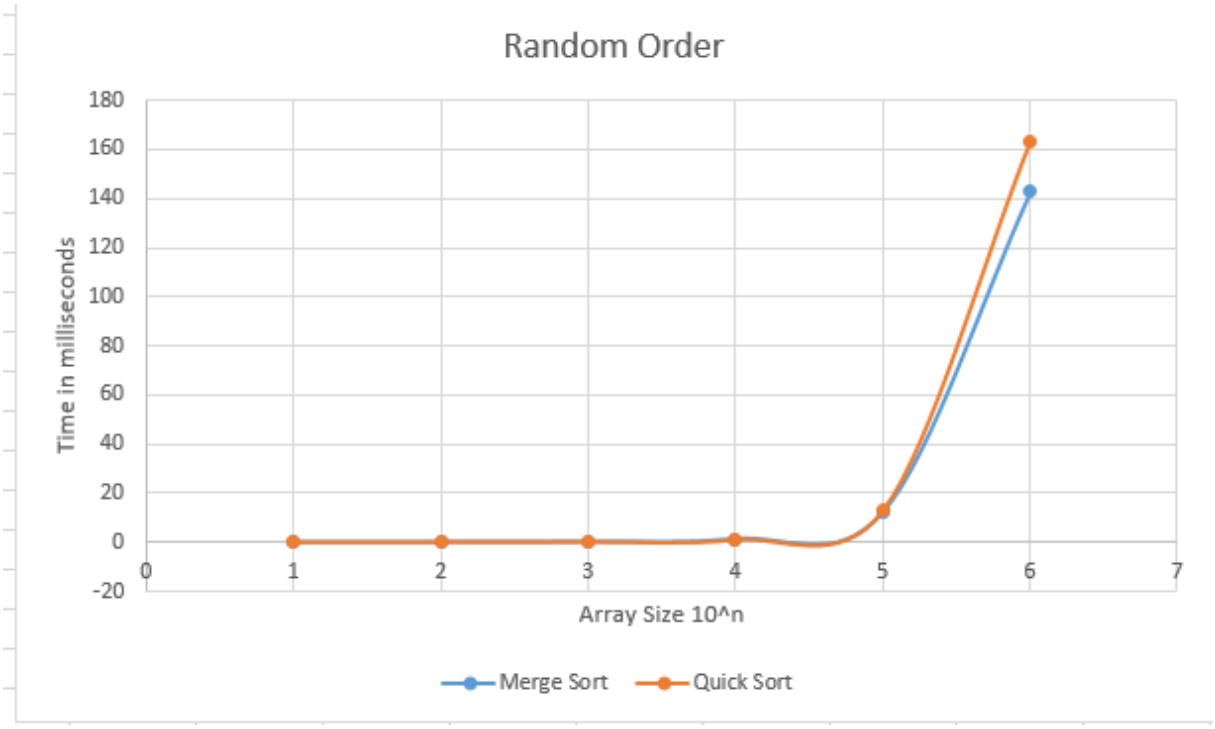
Graphs:

Ascending Order:



Descending Order:

Random Order:



Observation:

It is seen that merge sort is always providing consistent result, while quicksort provides some inconsistency. This is mainly because of the choice of pivot point. This time consumption can be omitted if the pivot s taken at the middle. But even so, an anti-case may be generated based on fixed pivot. This can be omitted by using randomized pivot.

And again, for fixed pivot at last position, ascending order takes the maximum time. This is because the number of swap it needs is the maximum. But if we observe we will see that the swap is always occurring with itself. So, a precheck or a better performance swap may increase the efficiency by many folds.