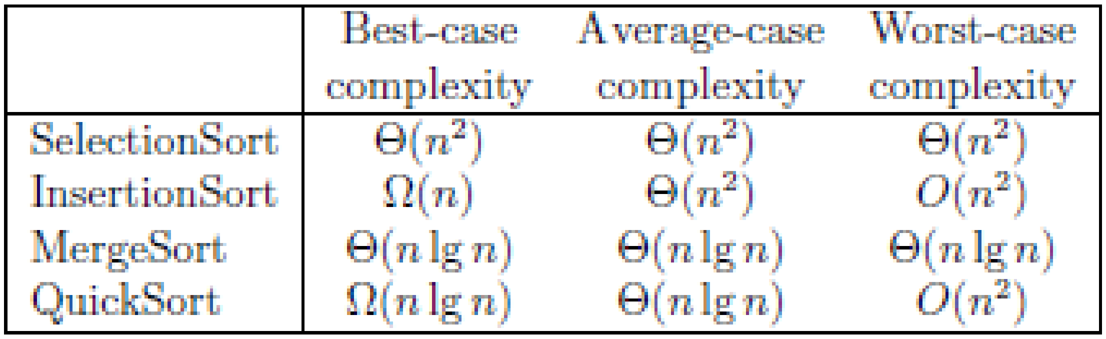
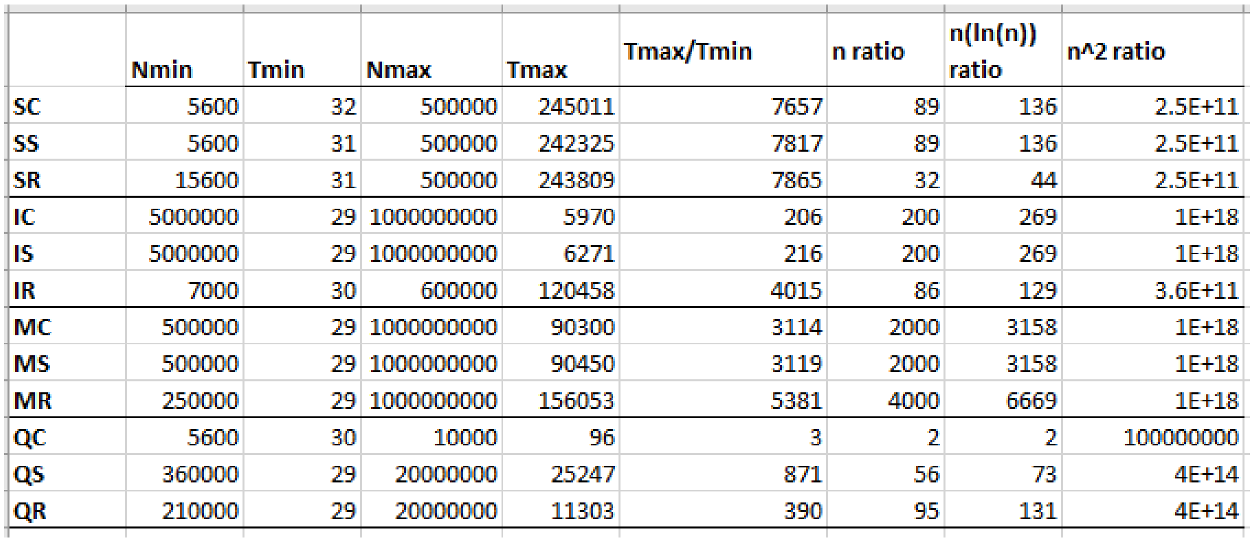
	Tmax/Tmin	**n ratio**	**n(ln(n)) ratio**	**n^2 ratio**	**Behavior**
SC	7657	89	136	2.5E+11	Quadratic
SS	7817	89	136	2.5E+11	Quadratic
SR	7865	32	44	2.5E+11	Quadratic
IC	206	200	269	1E+18	Linear
IS	216	200	269	1E+18	Linear
IR	4015	86	129	3.6E+11	Quadratic
MC	3114	2000	3158	1E+18	nln(n)
MS	3119	2000	3158	1E+18	nln(n)
MR	5381	4000	6669	1E+18	nln(n)
QC	3	2	2	100000000	Quadratic
QS	871	56	73	4E+14	nln(n)

Theoretical Time Complexities:



	Best-case complexity	Average-case complexity	Worst-case complexity
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
InsertionSort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
MergeSort	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n \lg n)$
QuickSort	$\Omega(n \lg n)$	$\Theta(n \lg n)$	$O(n^2)$

Chart with all computations:



	Nmin	**Tmin**	**Nmax**	**Tmax**	**Tmax/Tmin**	**n ratio**	**n(ln(n)) ratio**	**n^2 ratio**
SC	5600	32	500000	245011	7657	89	136	2.5E+11
SS	5600	31	500000	242325	7817	89	136	2.5E+11
SR	15600	31	500000	243809	7865	32	44	2.5E+11
IC	5000000	29	1000000000	5970	206	200	269	1E+18
IS	5000000	29	1000000000	6271	216	200	269	1E+18
IR	7000	30	600000	120458	4015	86	129	3.6E+11
MC	500000	29	1000000000	90300	3114	2000	3158	1E+18
MS	500000	29	1000000000	90450	3119	2000	3158	1E+18
MR	250000	29	1000000000	156053	5381	4000	6669	1E+18
QC	5600	30	10000	96	3	2	2	100000000
QS	360000	29	20000000	25247	871	56	73	4E+14
QR	210000	29	20000000	11303	390	95	131	4E+14

Selection Sort (constant):

The algorithm of selection sort the time ratio was closest to the computed n^2 ratio. This is what would be expected when sorting a constant array with this algorithm. As it still loops through the n^2 times regardless of the pre-sorted array's contents.

Selection Sort (sorted):

The timing ratio is closest to n^2, as expected. The algorithm still goes through the whole array O(n^2) times, a sorted array does not make a difference to the behavior. The algorithm executed very closely to the constant array surprisingly.

Selection Sort (random):

Time ratio is closest to the n^2 ratio again for the algorithm. The time ratio again is very close to the previous arrays' variations, sorted and constant. The algorithm sorts in its casual behavior looping through the array n^2 times. The sorting of this specific array I would say is the average case of this algorithm.

Insertion Sort (constant):

When the array has identical values, this algorithm has linear time complexity. The time ratio observed in the chart reflects that. The median time ratio is closes to the n ratio. Giving the best case scenario for this algorithm, where both loops iterate through the array in a total of O(n) time.

Insertion Sort (sorted):

The time ratio this time is fairly similar to the constant array, suggesting a linear behavior. This again, is the best case scenario for insertion sort. The complex algorithm manages to skip elements that have already been sorted, this makes the array be passed through once.

Insertion Sort (random):

The time ratio is higher significantly for this array variation. With more elements out of place, thus having to sort more, the algorithm executes in O(n^2) time. The data reflects that as there is a huge increase in the time ratio, and when compared to the previous variations. This leans towards the worst time complexity for this algorithm.

Merge Sort (constant):

The time complexity here is closest to the nln(n) ratio. This is expected since the algorithm behaves normally regardless of the way the elements are sorted. The algorithm uses the divide and conquer method to give the correct output, performance will not vary.

Merge Sort (sorted):

Merge sort performs normally, and time ratio is very similar to when the array had identical elements. Suggesting the average case for sorting this array nln(n).

Merge Sort (random):

Observing the data of the median time complexity we can notice that there is a huge leap when the array is random. This however is still closest to the same average case ratio even with the maximum array to be sorted being 1 billion. The time ratio is closest to the nln(n) ratio. Merge sort still performs in nln(n) time, as expected.

Quick Sort (constant):

When quick sort performs on a constant array it behaves in best case time which is nln(n). This is also observed in the data as the time ratio is closest to the n(ln(n)) ratio. The algorithm here performs as expected.

Quick Sort (sorted):

Even though there is a huge increase in the time ratio for this variation compare to the constant array. The algorithm still should perform in average time which is nln(n). This is reflected in the data from the chart as it is the closest to the nln(n) ratio computed.

Quick Sort (random):

For a random array quick sort is expected to perform with a time complexity close to the average case. This is reflected in the data as the time ratio is closest to the nln(n) ratio. Which happens to be the time complexity for the average case. The algorithm works in the expected manner in this case as well.