Experimental Setup

While I was making implementing the experiment, I determine N value in between 100.000 and 1.000.000. There are 10 different experiment scenarios that are experimented for 4 times each. I implemented each experiment for multiple times since run time varies.

Table 1: Experiment of AlgorithmSortAll

AlgorithmSortAll						
Input Value			Experiment Time (seconds)			
K	N	Range	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment
50000	100000	200000	5.486	4.627	5.381	4.525
100000	200000	400000	23.727	22.793	21.911	22.333
150000	300000	600000	48.191	47.769	49.809	44.459
200000	400000	800000	107.871	91.868	83.5	86.431
250000	500000	1000000	128.083	135.298	135.906	146.76
300000	600000	1200000	190.421	212.105	178.592	198.723
350000	700000	1400000	273.846	280.307	258.431	267.873
400000	800000	1600000	348.452	374.194	314.831	359.741
450000	900000	1800000	474.649	467.555	434.024	441.87
500000	1000000	2000000	566.077	579.449	574.29	568.755

Table 2: Experiment of AlgorithmSortK

AlgorithmSortK						
Input Value			Experiment Time (seconds)			
K	N	Range	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment
50000	100000	200000	2.903	3.055	3.074	3.002
100000	200000	400000	12.033	12.721	11.701	10.99
150000	300000	600000	27.311	27.496	30.539	28.813
200000	400000	800000	51.382	56.734	46.237	48.278
250000	500000	1000000	79.868	75.272	87.238	80.313
300000	600000	1200000	118.252	120.298	124.064	121.733
350000	700000	1400000	165.155	164.461	163.221	164.013
400000	800000	1600000	230.073	224.317	227.035	225.476
450000	900000	1800000	277.979	277.918	278.128	277.512
500000	1000000	2000000	347.643	335.943	337.191	343.484

Table 3: Experiment of AlgorithmSortHeap

AlgorithmSortHeap						
Input Value			Experiment Time (seconds)			
K	N	Range	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment
50000	100000	200000	0.38	0.388	0.35	0.365
100000	200000	400000	0.717	0.767	0.728	0.757
150000	300000	600000	1.101	1.106	1.13	1.104
200000	400000	800000	1.443	1.45	1.566	1.548
250000	500000	1000000	1.823	1.928	1.828	1.93
300000	600000	1200000	2.266	2.419	2.321	2.258
350000	700000	1400000	2.617	2.674	2.696	2.694
400000	800000	1600000	2.98	3.021	3.056	3.108
450000	900000	1800000	3.368	3.509	3.449	3.391
500000	1000000	2000000	3.744	3.833	3.812	3.825

Table 4: Experiment of AlgorithmSortQuick

AlgorithmSortQuick						
Input Value			Experiment Time (seconds)			
K	N	Range	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment
50000	100000	200000	0.359	0.365	0.37	0.366
100000	200000	400000	0.706	0.747	0.744	0.722
150000	300000	600000	1.149	1.13	1.114	1.148
200000	400000	800000	1.458	1.475	1.55	1.463
250000	500000	1000000	1.816	1.861	1.806	1.903
300000	600000	1200000	2.214	2.236	2.284	2.225
350000	700000	1400000	2.57	2.684	2.686	2.698
400000	800000	1600000	2.944	3.057	3.02	3.053
450000	900000	1800000	3.327	3.456	3.428	3.436
500000	1000000	2000000	3.725	3.798	3.847	3.791

Results

Table 5: Average Time Taken for 4 different Trying

Average Time Taken for each Algorithm (seconds)				
N	AlgorithmSortAll	AlgorithmSortK	AlgorithmSortHeap	AlgorithmSortQuick
100000	5.005	3.009	0.371	0.365
200000	22.691	11.861	0.742	0.730
300000	47.557	28.540	1.110	1.135
400000	92.418	50.658	1.502	1.487
500000	136.512	80.673	1.877	1.847
600000	194.960	121.087	2.316	2.240
700000	270.114	164.213	2.670	2.660
800000	349.305	226.725	3.041	3.019
900000	454.525	277.884	3.429	3.412
1000000	572.143	341.065	3.804	3.790

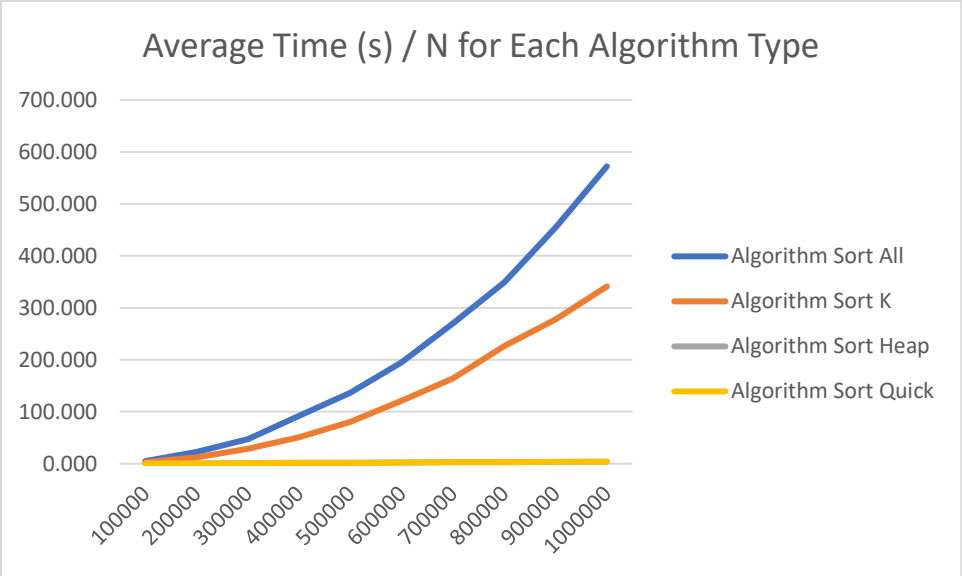
In the Table 5, you can see the average time taken for different algorithm types. Bear in mind that they are the average value of 4 different testing. Since average value of AlgorithmSortHeap and AlgorithmSortQuick are so close to each other, I implement new experiment by using greater input value.

Table 6: Average Time Taken for SortHeap and SortQuick with Greater Value

AlgorithmSortHeap vs AlgorithmSortQuick (Average Time Comparison)				
Input Value			Experiment Time (seconds)	
K	N	Range	AlgorithmSortHeap	AlgorithmSortQuick
500000	1000000	2000000	4.034	3.81
1000000	2000000	4000000	7.74	7.601
1500000	3000000	6000000	11.706	11.433
2000000	4000000	8000000	15.465	15.59
2500000	5000000	10000000	19.54	19.38

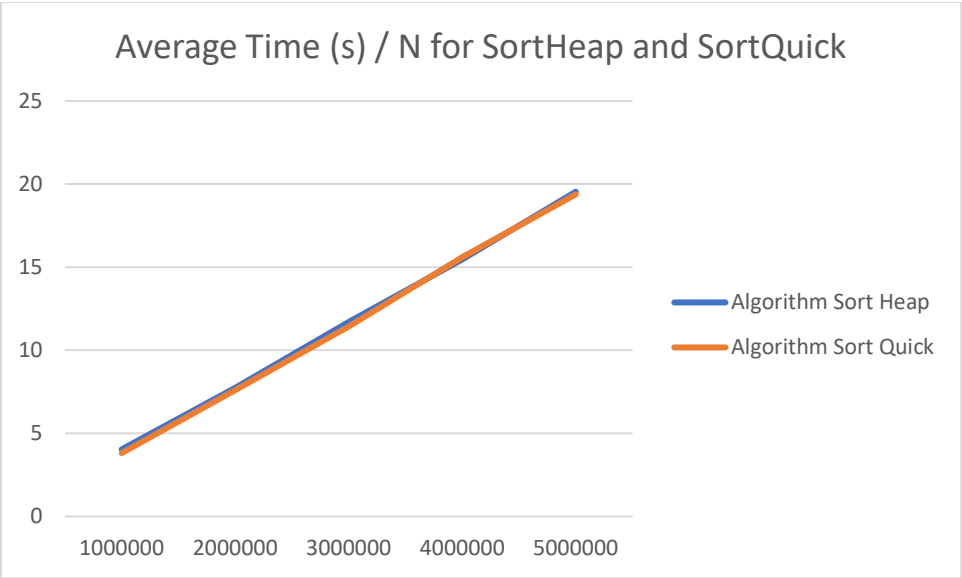
Bear in mind that results in Table 6 are average value of 4 different trying of each segment like previous experiment.

Table 7: Line Chart for Average Time Taken



As it can be seen in Table 7, since the graph does not give any insight about SortHeap and SortQuick, I needed to implement new experiment having higher input value.

Table 8: Average Time Taken for SortHeap and SortQuick



Discussion

I created 10 files in the beginning to compare 4 different algorithms. There are 10 different file type varying between 100.000 to 1.000.000 (100k, 200k, … 1m). These files were built to compare run time of each algorithm regarding the input size. As expected, While SortAll and SortK algorithms formed quadratic line in Table 7, SortHeap and SortQuick formed linear line in Table 8. The reason of this formation (line structure of each algorithm) is their Big-O (worst case scenario) notation. While the average time complexity of SortAll and SortK is N^2, SortHeap's and SortQuick's average time complexity is N*log(N).

Since the results of SortHeap and SortQuick respecting input size is too close to each other, I created another 5 file varies between 1.000.000 – 5.000.000 (1m, 2m, …, 5m). I edited each file's algorithm type to measure the run time. Ultimately, I worked with 50 file variations in total.

My observations:

- When Table 7 is examined, we can see that there is a quadratic line for SortAll and SortK because both algorithm's average time complexity is N^2. Therefore, result of the experiment satisfied my theoretical expectations.
- When two different algorithms SortAll and SortK is compared, we can see that SortK significantly performed better. While the gap between k'th (half of N) number and N increasing, the run time difference between SortAll and SortK increases. For the 10^{th} experiment, SortK accomplished to sort the file almost half of the time than SortAll's took.
- To see the comparison of SortHeap and SortQuick better, I implemented the algorithm by using greater input sizes like 5 million. Although the increment of input size, run time difference between SortHeap and SortQuick is almost negligible (can be shown in Table 8). For instance, in the input size of 5 million, while run time of SortHeap is 19.54, the run time of SortQuick is 19.38. In general, SortQuick have slightly better performance than SortHeap.
- By looking the theoretical perspective, I expected for SortQuick to have better performance than SortHeap, but they performed almost equally. The reason can be insufficiently optimized operation that requires more time than usual in the SortQuick algorithm code.
- I have not encountered any problems related with crash or uncompleted sort operations.
- Ultimately, the experiment shows that the speed of the algorithms can be sorted such as: SortQuick > SortHeap > SortK > SortAll.

S014670

Cemil Şahin