Data Sets, Test Strategies

For this project I was tasked with comparing 4 sorting algorithms while selecting the Kth smallest element in each sorted list. The four algorithms tested were Merge Sort, Iterative Quick Sort, Recursive Quick Sort, and Median of Medians Quick Sort. For each algorithm I tested the same list and ran it for k=1, n/4, n/2, 3n/4, and n. All four algorithms were tested 1000 times at each size *n* for each *k*. I was able to complete these tests until n = 10000000; this took my computer about 11 hours. I did not run longer due to my battery getting hot. At the end of my report, I included tables and graphs to display all relevant data collected from the tests.

It is important to note the computer I ran these tests on, as each computer would produce different results. The specs are as follows: 11^{th} Gen Intel Core I7-1185G7 @ 3.00GHz, 2995 MHz, 4 Cores, 16GB of DDR4 RAM and no video card. This is a laptop; the GPU is built into the CPU. I decided to run this project on my laptop because the CPU's single core performance is better than my desktop. During the test, I had all other programs turned off and stopped as many background processes as possible. Overall, the tests took about 11 hours to complete.

Strengths and Constraints of Project

The strength of my work is that the CPU of this laptop is strong. It helped complete the test faster than my desktop. I was surprised by this because I thought my laptop would be slower. However, this did limit me in a sense of time. My laptop cannot run as long as my desktop, but I am unable to run my desktop long enough to be able to finish what my laptop was able to. It helped me complete more tests using my laptop. I programmed the algorithms and tests in Java which is faster compared to other languages like Python. The constraint of my work is that I do not know how to implement these in a multi-threaded way. I researched online and learned that it can be significantly faster when multithreading these algorithms. I believe that one more constraint is that I do not know if my implementations of the algorithms are the most efficient version of them. There could be a better way to code each algorithm.

Theoretical Complexity Comparisons

The Theoretical complexity of Merge Sort is O(nlog(n)) for the best and worth cases. This is because it will partition the collection until it is in groups of 1, therefore it will always do the same amount on the same size collection no matter what. Quick Sort is a little different, the best case is also O(nlog(n)), but the worst case is O(n^2). Quick sort can be made faster by implementing it with the Median of Medians algorithm. This algorithm changes the best and worst case to O(n).

Select 2 Versus Select 3

From my testing and data collection I see that Select 2 (Iterative Quick Sort) starts out slower than Select 3 (Recursive Quick Sort). Select 2 quickly becomes faster than Select 3, it seems that at size less than 50 Select 3 is faster. When the size is 100, Select 3 took about 900ns longer on average than Select 2. It does even out in terms of speed (Select 2 is slightly faster) until sizes larger than 100000. Select 3 starts to get much slower at that size.

Select 4 Versus Select 1

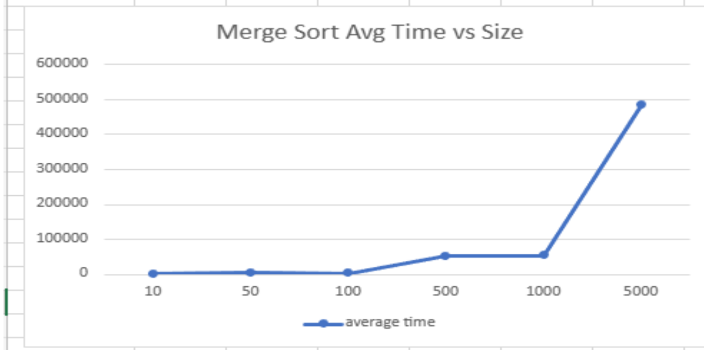
From my testing and data collection I see that Select 4 (Median of Medians Quick Sort) is always faster than Select 1 (Merge Sort). This came as no surprise to me because the worst-case time complexity of Select 4 is better than the best-case time complexity of Select 1.

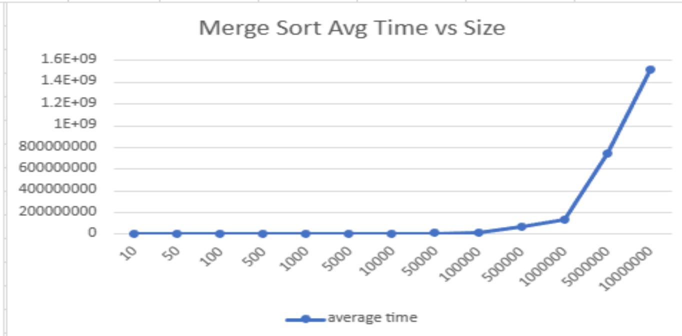
Tables and Graphs of Results

Select 1 (Merge Sort)

n	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
1	1751	4527	3688	53348	54325	485121	985825	5588213	12615953	66925412	129948411	740458416	1518088836
n/4	1682	4327	3544	50646	52310	485301	992287	5574536	12735257	66466881	135459540	739604979	1517172254
n/2	1667	4227	3356	50239	52315	482384	987620	5581399	12687171	64268666	134942818	739453939	1517148074
3n/4	1671	4137	3504	50830	56048	484113	996859	5641518	12685253	63801218	130239937	740740611	1518911496
n	1626	4297	3370	50269	52741	479176	989391	5609781	12740357	66075067	135020421	741643975	1516804695

	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
average time	1679.4	4303	3492.4	51066.4	53547.8	483219	990396.4	5599089	12692798	65507449	133122225	740380384	1517625071

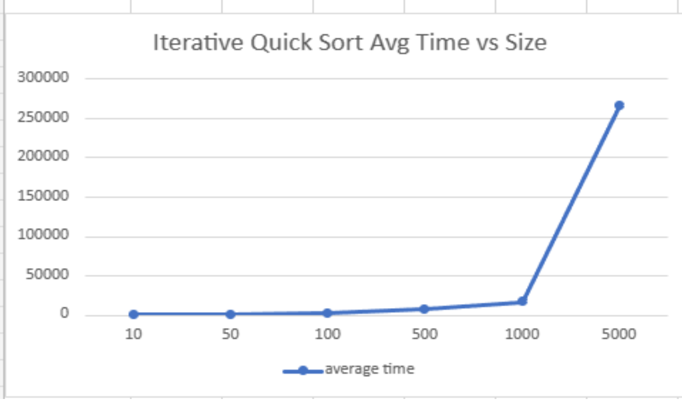


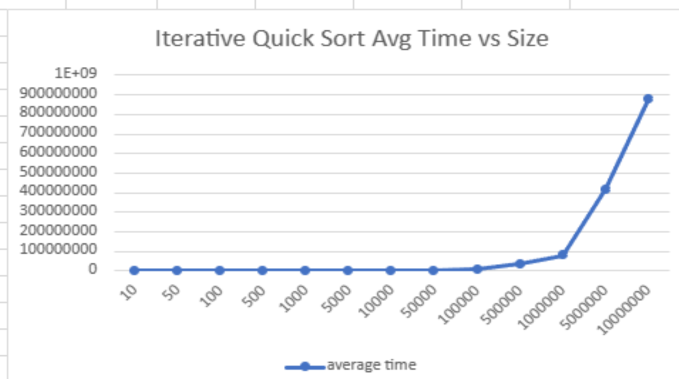


Select 2 (Iterative Quick Sort)

n	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
1	783	697	2054	7275	16846	265255	565890	3036077	6719074	36417234	76693907	415036513	874303474
n/4	745	707	2062	7401	16376	266536	573604	3032237	6746322	36377569	76413398	415312872	874699995
n/2	742	716	2055	7355	16405	265505	565649	3042138	6771138	36232126	80178209	414691354	873115833
3n/4	727	708	2028	7263	16488	264707	567911	3036805	6749834	36636827	84965008	415044338	876264900
n	725	692	2001	7329	16325	264750	566019	3036372	6698225	36358913	76413163	413701999	874455841

	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
average time	744.4	704	2040	7324.6	16488	265350.6	567814.6	3036726	6736918.6	36404534	78932737	414757415	874568008.6

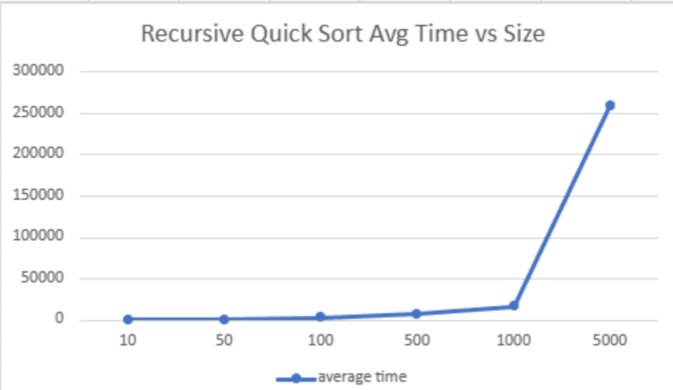


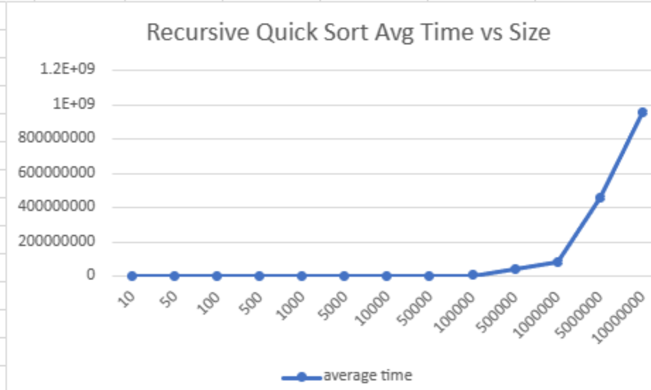


Select 3 (Recursive Quick Sort)

n	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
1	583	515	2940	7340	16945	259696	601205	3319015	7397050	40528624	85243346	458807608	954176027
n/4	589	509	3028	7265	16865	260115	608370	3315777	7395482	41281737	85441927	458699229	955362286
n/2	548	539	2953	7428	17102	260075	609567	3325845	7320187	40815011	84935028	458732011	955325148
3n/4	537	507	2956	7326	17018	259285	601464	3323698	7363832	40964008	85066820	458576192	954028176
n	530	509	2990	7357	17078	260766	602232	3311629	7378022	40962956	85289378	458122068	955327728

	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
average time	557.4	515.8	2973.4	7343.2	17001.6	259987.4	604567.6	3319193	7370914.6	40910467	85195299.8	458587422	954843873

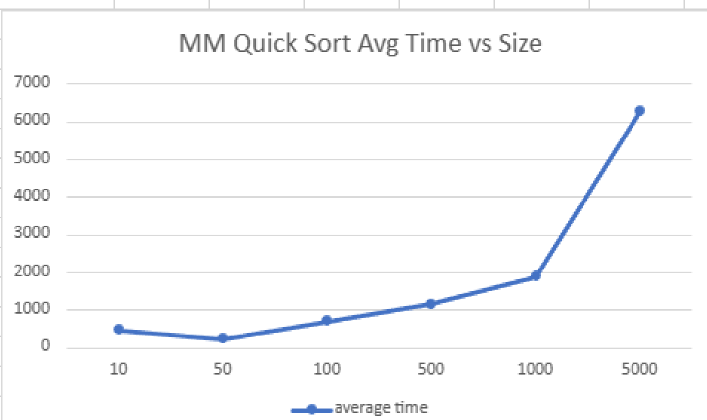


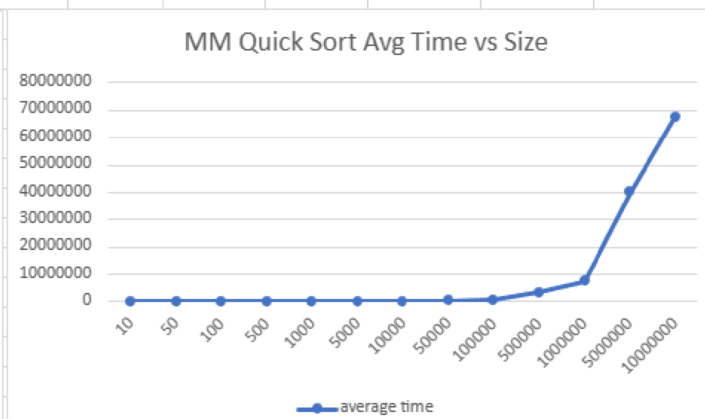


Select 4 (MM Quick Sort)

n	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
1	553	166	476	310	1620	3576	12926	97206	863211	1112618	8056277	35588890	17127397
n/4	616	148	833	1138	2660	5569	34168	272499	505726	4714058	8690563	34722883	79581550
n/2	390	278	595	1201	1183	6236	44956	389370	880112	2791618	10807318	48293352	101977174
3n/4	496	265	894	1510	1404	8214	71078	477484	750014	4965623	6795156	45395736	67695805
n	326	302	762	1659	2609	7761	60194	412544	661699	3566300	2297492	36411464	71490476

	10	50	100	500	1000	5000	10000	50000	100000	500000	1000000	5000000	10000000
average time	476.2	231.8	712	1163.6	1895.2	6271.2	44664.4	329820.6	732152.4	3430043.4	7329361.2	40082465	67574480.4





Conclusion

In conclusion, I learned that time complexity can vary between the classes. The difference in algorithm that is O(n) and O(nlog(n)) is significant, especially on larger size collections. I was not expecting MM to be so fast. I was extremely surprised to see that the size of 10000000 test for MM took 12 minutes, while the Recursive took over an hour. This exercise really shows how selecting a good pivot can really make a difference in sorting algorithms.