King Abdelaziz university

Faculty of computer and information technology
Computer Science Department

Empirical Analysis of Comparing Two Approaches to Partition Using Quick Select

Wajd Alharbi ID: 2007057 Section: GAR

Table of Contents

1.INTRODUCTION

The efficiency of algorithms is a key concept in the field of computer science. We studied mathematical procedures to analyze the efficiency of recursive and non-recursive algorithms by specifying the basic operation and calculating the efficiency mathematically in this course, but some algorithms, despite their apparent simplicity, require complex mathematical techniques to analyze. Empirical analysis, which is the subject of our project, is another way to assess the effectiveness of algorithms.

In this report, we will examine and compare empirically two methods for determining the median in a set of numbers: Quick Select approach using **Sedgewick Partitioning** and Quick Select approach using **Lumoto Partitioning**. We will collect data and determine the efficiency of both algorithms, as well as their order of growth classes, to determine which of them may be the most appropriate.

2. Empirical Analysis of Algorithms

2.1 Purpose

The purpose of this report is, using empirical analysis to compare the efficiency of two different techniques for finding the median of a set of numbers with random values ranging of between 0 - 100, Quick Select approach using Sedgewick Partitioning and Quick Select approach using Lumoto Partitioning, both of which are designed to solve the same problem. We want to look at the outcomes practically and compare them to the algorithm's theoretical assertions before determining which method is the best for obtaining the median.

2.2 Choice of efficiency metric

There are two measurements in empirical analysis of an algorithm:

1. Physical Time
2. Basic operation Counter

For many reasons, we chose to utilize the basic operation counter in this report. The first is that the physical runtime is inaccurate since it may contain other applications operating in the background of the computer. Second, hardware has an impact on runtime; if the hardware is advanced, the runtime will be drastically different from a computer with basic hardware, resulting in significant variances in calculating the algorithm's run time. Also, because today's computers have high-speed processors, we may encounter cases where the run time appears to be zero when it actually took a fraction of a second to complete, especially when dealing with small sets of numbers; not including those outputs may have an impact on how this algorithm handles small instances. Third, certain operating systems, such as UNIX, incorporate the time spent by the CPU with the application, which reduces the output accuracy and ruins the analytic idea.

We shall count the fundamental operation in both techniques of analysis since mathematically assessing the algorithm focuses on the basic operation. The fundamental operation will remove any external manipulations such as the CPU runtime and other processes that operate, and it will not be influenced by the computer hardware set, making it the best technique for experimentally analyzing the methodologies. We simply need to know where to position the counters to count the fundamental operation, which is especially important if the algorithm is used by many functions. Because of the above considerations, we will only focus on examining the basic operation of each algorithm; however, the analysis of the physical run time may be found in the appendix on page 8.

3.Design & Procedure

3.1Characteristics of The Input

We cannot deal with a set of size zero because we are dealing with a median problem; however, a set of 3 elements or more may be appropriate in trying to find the median problem since we will start to have an element in the center of the set which represents the median; nevertheless, a set of 3 elements is way too small; we may start with a set of small sizes such as 100 elements and continue increasing the input size until we reach a big set where the clear distinction between both algorimths is discovered. Beginning with a small set will help to approximate the efficiency of the algorithm because we are including most of the possibilities which the algorithm may face, including small sets, and combining them with the analysis, so we decided to start with an input size of 100 and increase it in each until we reach a large input size for both algorithms, trying to make sure to have both even and odd numbers as input sizes to generalize the results.

For the purpose of the study, we must run the very same input size more than once, with every set having distinct instances, ranging from 0-100. To implement that, we'll send the set to the Quick Select approach using Sedgewick Partitioning, and Quick Select approach using Lumoto Partitioning to find the median. Finally, we will collect and store the data obtained. To avoid data variation, it is useful to measure the average of the values. To do that, I ran the same code 10 times in this project. Keeping a distinct instance every time we produce a different set with the same size seeks to guarantee the reliability of the results because the data will not be confined to those few trails, minimising the chances of a deceptive conclusion that could result to the algorithm being classified incorrectly.

3.2 Algorithms

In this project we have used the following algorithms:

- Sedgewick Partitioning:

  Obtained from Dr.Muhammad Al-Hashimi's class slides (Lecture 22, pg6.

- Quick Select (recursive):

  Obtained from Dr.Muhammad Al-Hashimi's class slides(Lecture 21).

- Lumoto Partitioning:

  Obtained from textbook, chapter 4.5 page 159.

- Random values:
  Obtained from Dr.Muhammad Al-Hashimi’s Website + (found in chapter 2.6 p87).

- Median index:

  ALGORITHM *medianInx(Arr[0 « n])*
  //Find the median index of a given array
  1. Lower <- 0, upper <- Arr.length-1
  2. m <- [lower + upper]/2
  3. Return m

3.3. Tools of the analysis

This study was done with the help of the listed tools below:

1. **Visual Studio:** This program is used to create and run JavaScript code.
2. **Excel:** To record the data from every algorithm output and to make the graphs, I utilized an Excel spreadsheet.
3. **Firefox Browser:** was used to execute the code and find out how many basic operations there are.

4. Empirical Analysis of The Algorithms

4.1. Run Program and Collect Data

The tables below contain records of each algorithm's data. Each size was tried ten times, and the results were then compiled and utilized to calculate the average of each size. My input ranged from 100-25,000

		Sedgewick											
input	1st Trail	2nd Trail	3rd Trail	4th Trail	5th Trail	6th Trail	7th Trail	8th Trail	9th Trail	10th Trail	Average	Best	Worst
100	97	70	61	50	118	159	251	96	74	229	120.5	50	251
250	140	359	256	422	234	162	260	205	376	483	289.7	140	483
500	804	472	420	473	151	270	299	388	666	563	450.6	151	804
999	1131	348	1229	885	1632	1148	1777	874	1293	1278	1159.5	874	1632
5000	12375	6054	5999	7021	1977	5812	6603	5147	6864	4403	6225.5	1977	12375
8000	8121	12582	3844	6543	7925	4659	7134	5857	6796	2796	6625.7	2796	12582
12000	27809	10873	16596	13569	3612	21950	7911	12397	4280	9416	12841.3	3612	27809
15000	19761	11502	13519	10359	27226	5793	14451	14351	5963	30113	15303.8	5793	30113
17999	25863	9469	34331	19113	14177	34210	9481	6214	13372	27317	19354.7	6214	34331
25000	12690	26230	28495	45927	10867	27907	16339	16433	24648	14749	22428.5	10867	45927
50000	71813	39982	36950	53348	30827	73275	88610	54412	106012	49705	60493.4	30827	106012
100000	88031	67452	133394	122823	132022	102470	117935	99385	20397	154417	103832.6	20397	154417
1000000	756846	1041124	531519	897807	1813412	1530062	1167564	397908	436841	1528488	1010157.1	397908	1813412

Sedgewick Partitioning 1

			Lumoto										
input	1st Trail	2nd Trail	3rd Trail	4th Trail	5th Trail	6th Trail	7th Trail	8th Trail	9th Trail	10th Trail	Average	Best	Worst
100	1139	939	1146	819	579	1437	1437	1375	779	1332	1098.2	579	1437
250	4322	3654	2840	2109	4140	3181	2779	4422	4608	3367	3542.2	2109	4608
500	7130	3414	10358	6557	8380	7389	4236	8498	1172	7517	6465.1	1172	10358
999	15633	15962	11416	23957	27611	20806	10943	15299	16399	9114	16714	9114	23957
5000	115804	802219	101958	223758	85477	152399	159862	189763	174056	83287	208858.3	83287	802219
8000	126843	178602	228230	259791	181794	278808	139743	233982	215137	186634	202956.4	126843	278808
12000	524766	331882	192651	386325	522417	344484	765750	256573	337327	587969	425014.4	192651	765750
15000	729754	285340	416813	620874	476569	938398	685776	471707	324294	591119	554064.4	285340	938398
17999	746044	504392	882786	589851	740764	469130	457155	590822	558218	557997	609715.9	457155	882786
25000	1391602	217654	960892	815908	827969	815208	758773	417400	1210042	1207756	862320.4	217654	1391602
50000	2612257	1654164	820341	1791837	2414688	1677088	2481274	1621826	1028570	1622140	1772418.5	820341	2612257
100000	2375352	2303215	2680289	3078950	3110630	3624364	4715797	4009532	3225829	2738504	3186246.2	2303215	4715797
1000000	83727578	43148704	49266016	50140212	85392464	83264465	58209218	48936472	38345326	86416761	62684721.6	38345326	86416761

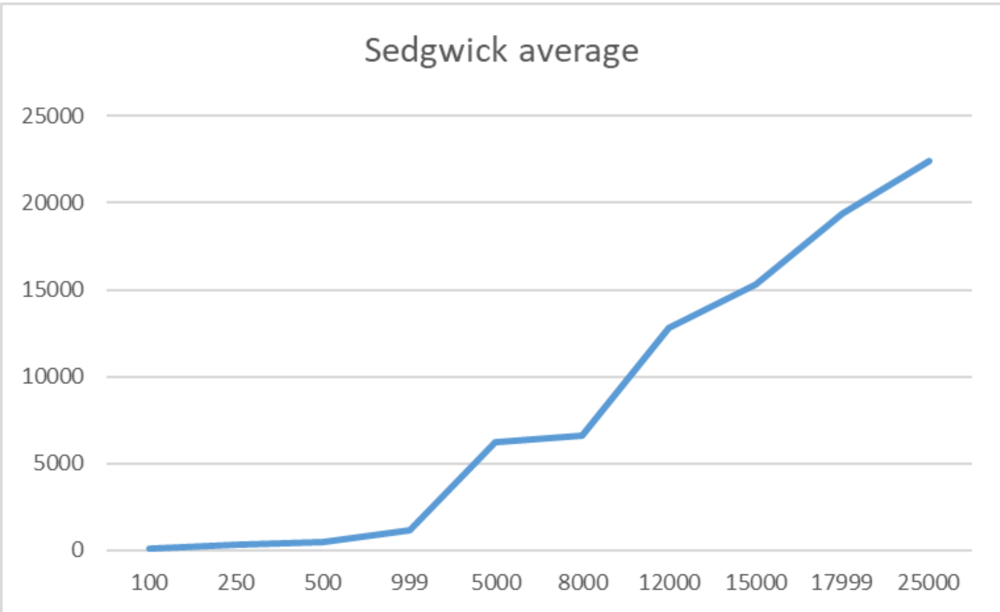
Lumoto Partitioning 1

We'll use those numbers to create a graph that shows the average of the basic operation, with the x-axis representing the input size and the y-axis representing the average case value.

I had to divide my data into more than one graph to examine the data more precisely, as it is very large; however, I do present the final graph with all the data inserted.

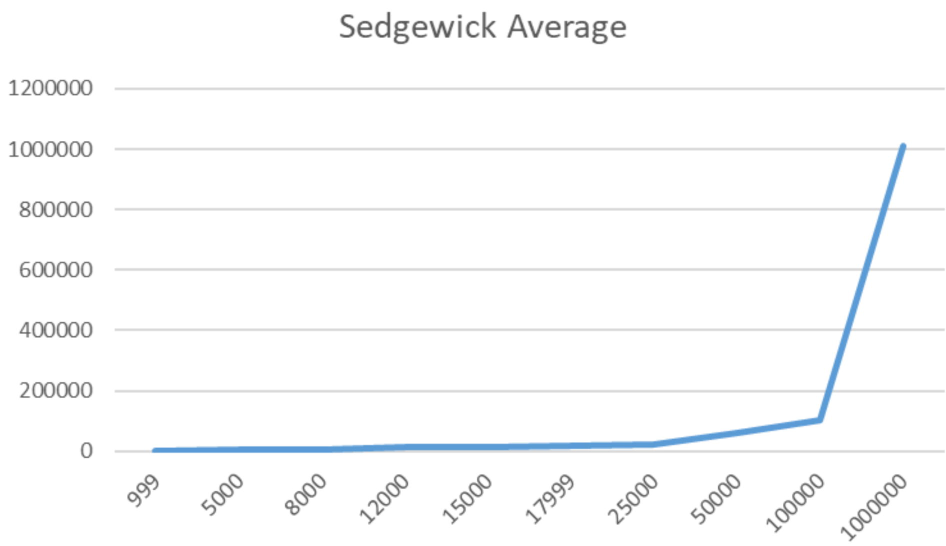
Sedgewick:

The first 10 inputs:



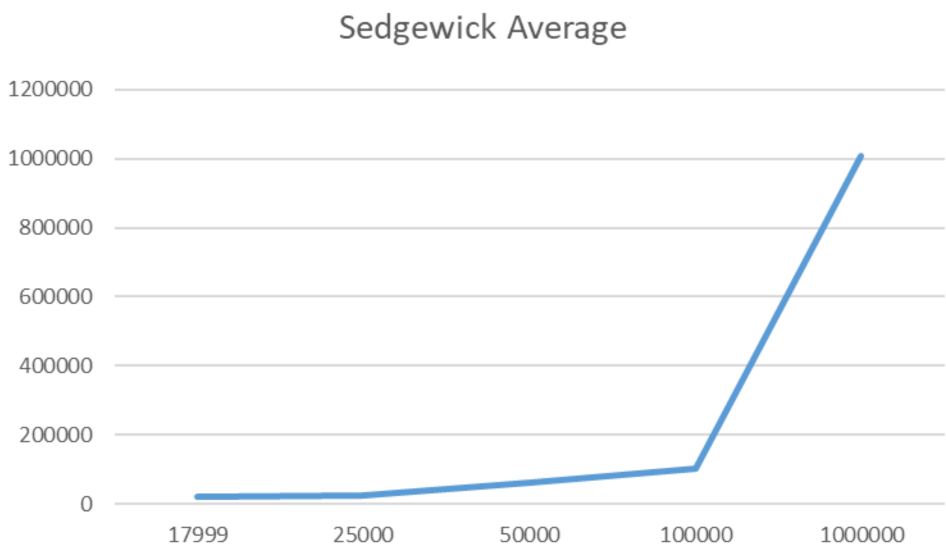
Average Sedgewick 1

From 999 till 1,000,000



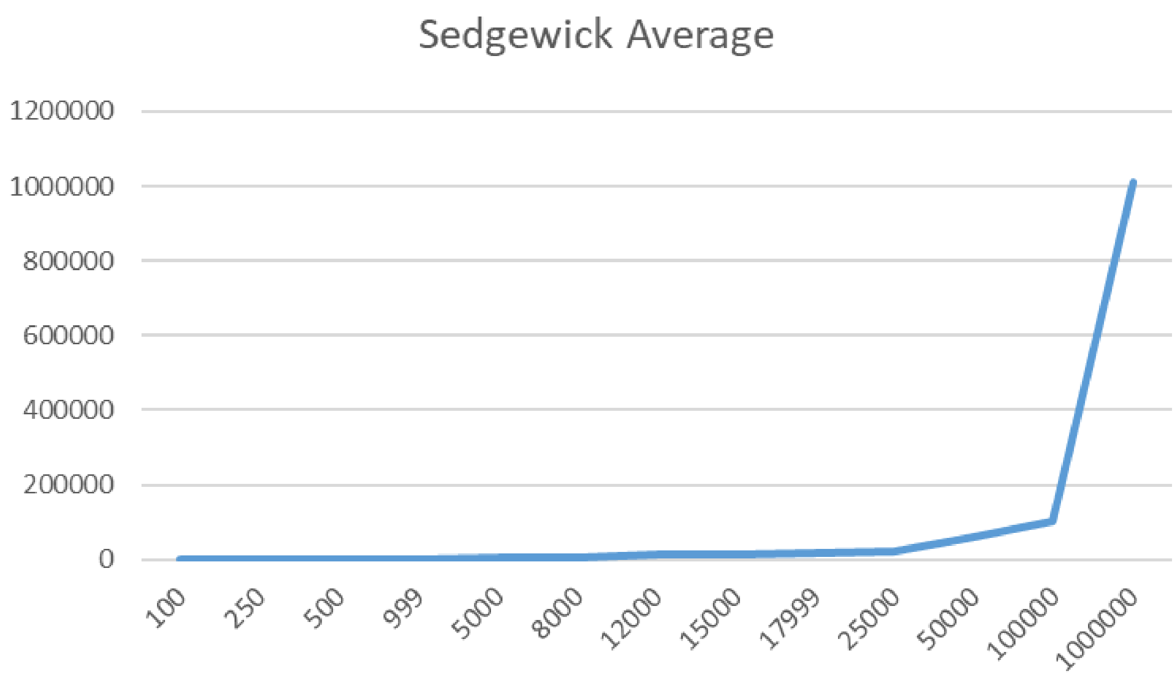
Average Sedgewick 2

Last 5 inputs:



Average Sedgewick 3

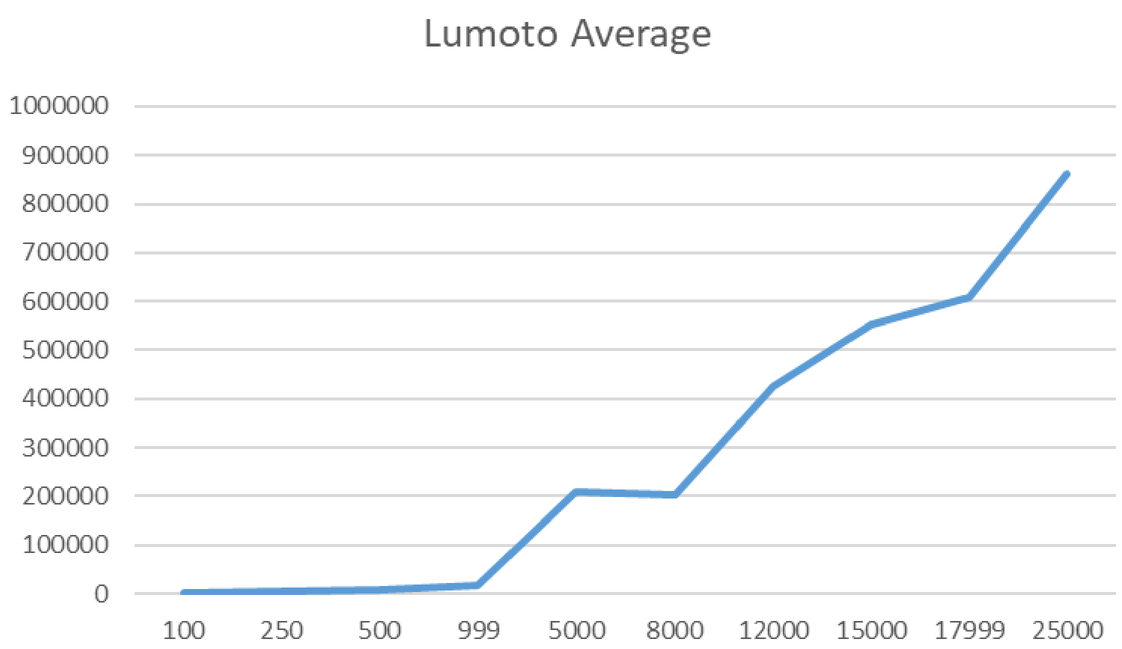
Full graph of the Sedgewick average (all inputs):



Average Sedgewick 4

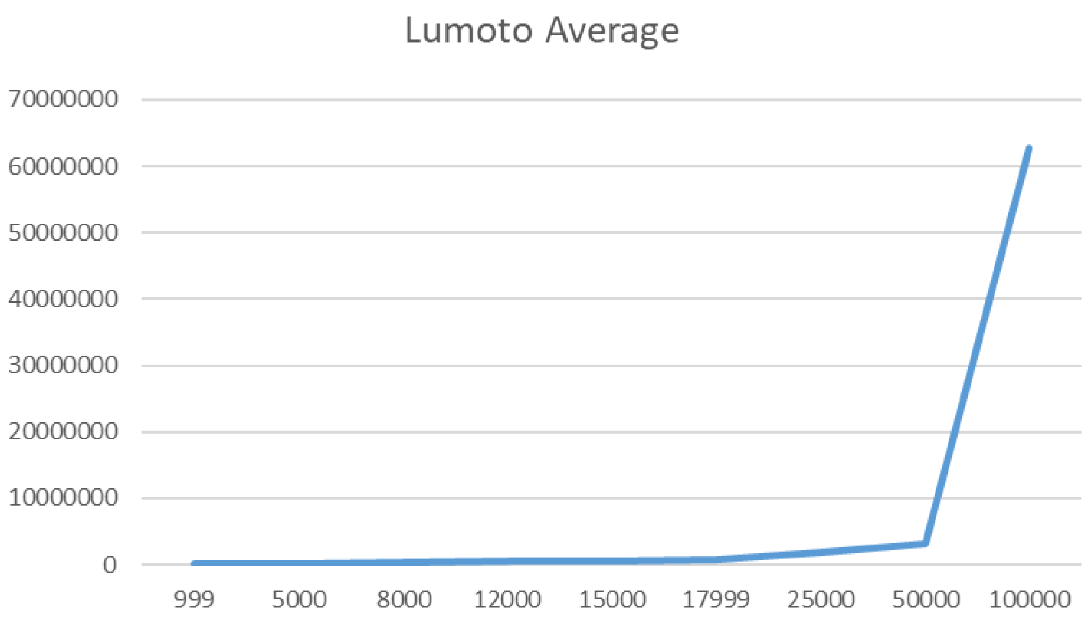
Lumoto

The first 10 inputs:



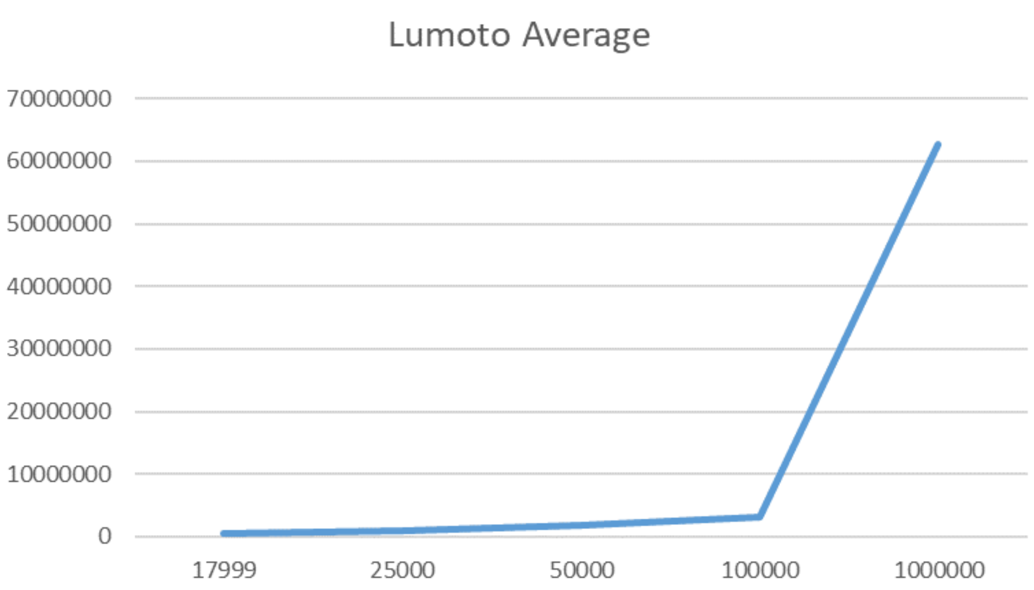
Average Lumoto 2

From 999-1,000,000

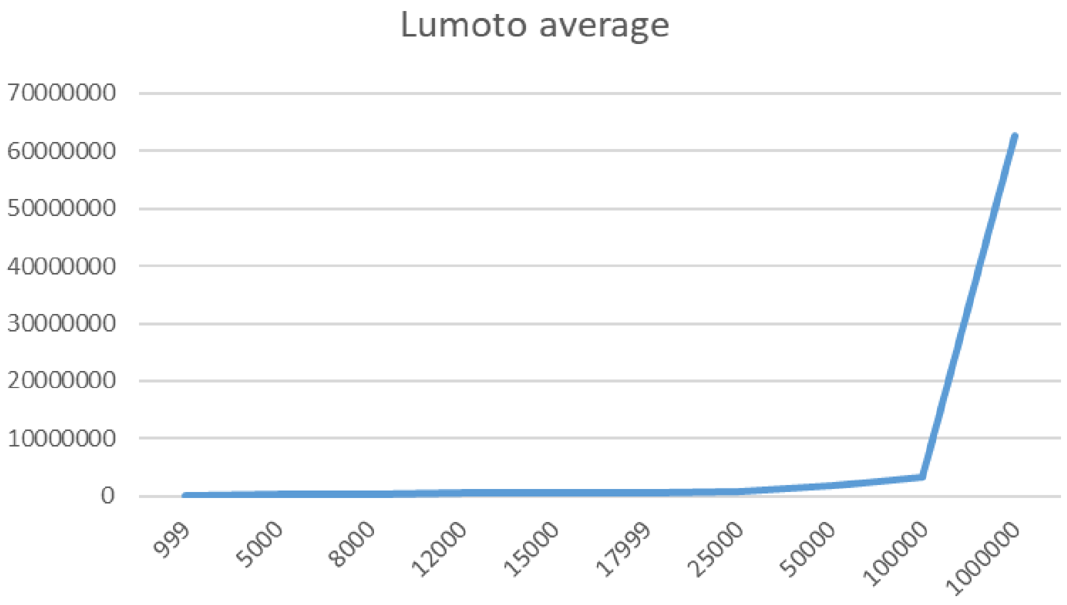


Average Lumoto 3

Last 5 inputs:



Full graph of the Lumoto average (all inputs):

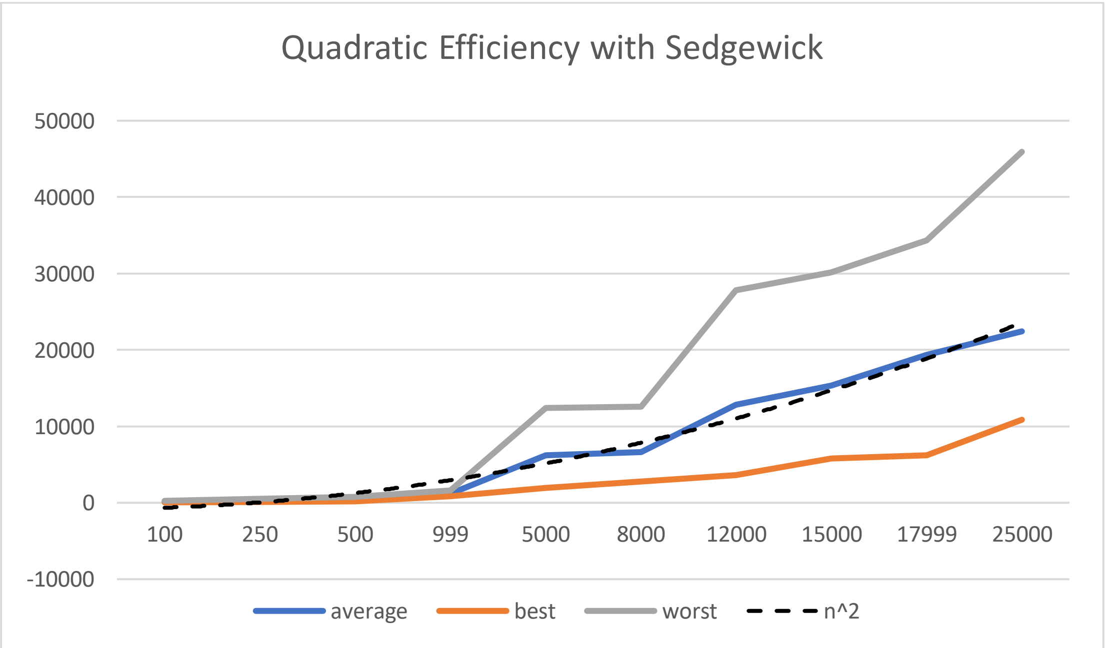


Average Lumoto 4

4.2. The Order of Growth

The graphs from the previous give us an overview on the efficiency of the algorithms, the average efficiency of the Quick Select with Lumoto partition seems to be linear on the first 10 instances; however, as the inputs got more spaced out it took a sharp curve. The average of Quick select with Sedgewick approach, on the other hand is quadratic. The following graphs show the average alongside with the complexity classes we determined from analyzing the previous graphs:

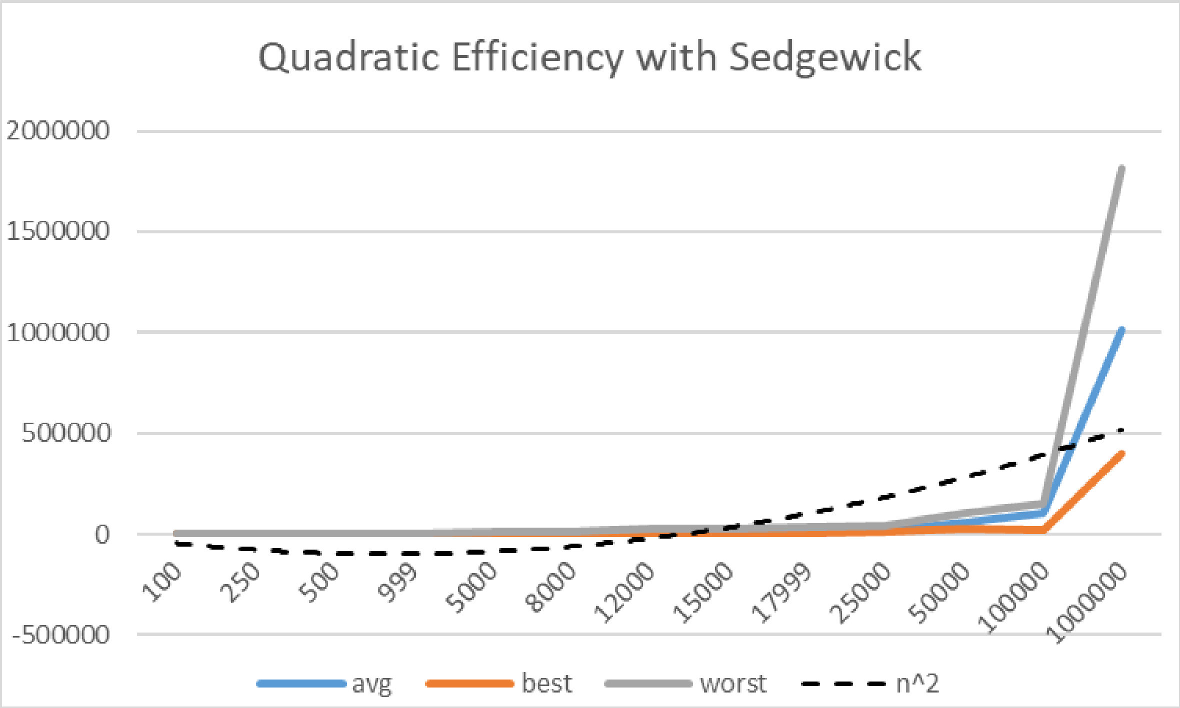
Sedgewick



Quick select using Sedgewick partitioning, with quadratic efficiency class

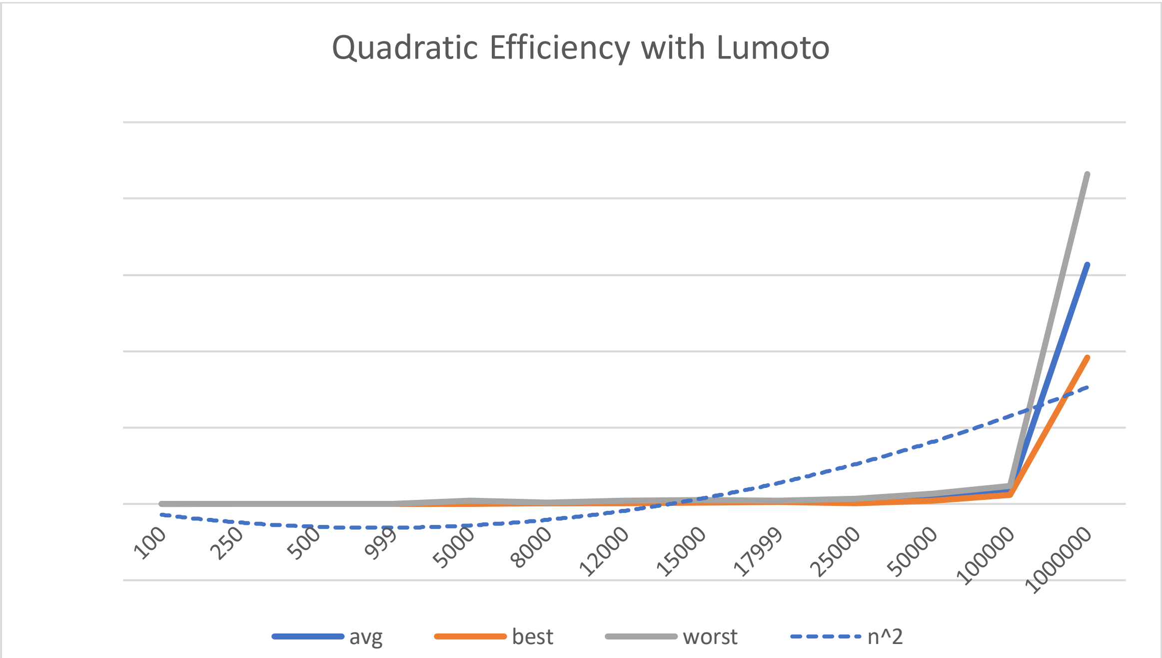
(In the above graph, I used the first 10 inputs because my data is extremely large and does not clearly show the curve when using all inputs, as seen below. The last 3 inputs show a big jump in the values; still however, it resembles the quadratic class n^2)

Graph of All inputs with quadratic efficiency:

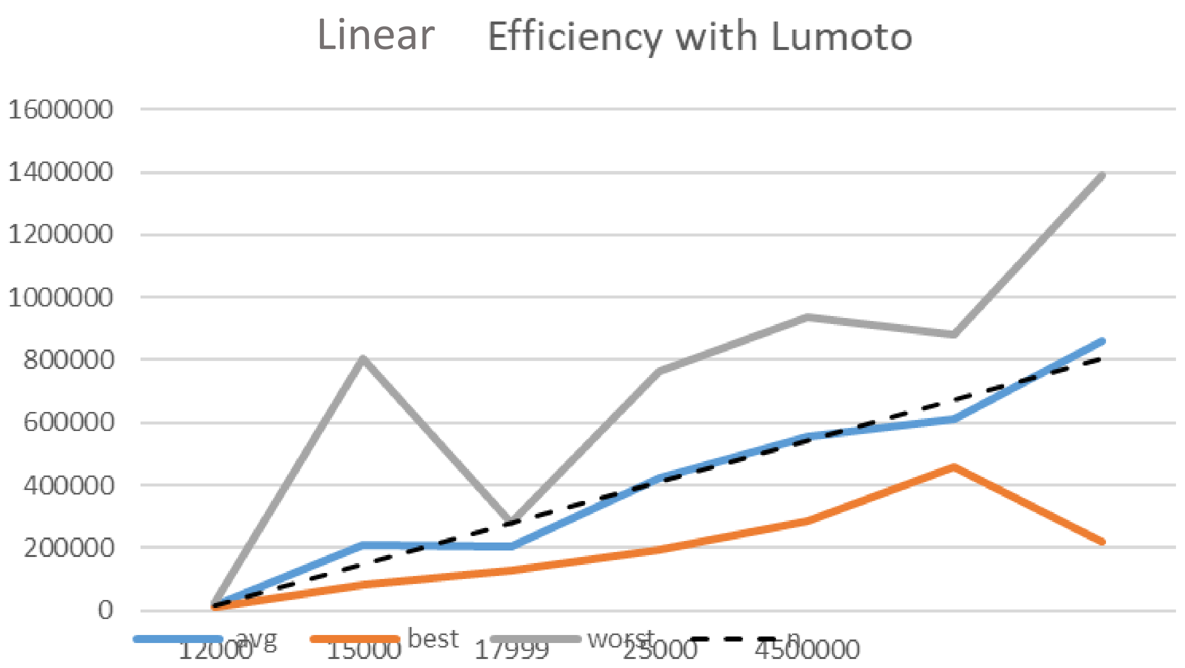


Lumoto

Graph of All inputs with quadratic efficiency:



Graph of half the inputs with linear efficiency class:



Quick select using Lumoto partitioning, with linear efficiency class

We now can examine the algorithms using the graphs of the best, worst, and average cases of each approach. Remember that both our quick select approaches of returning the median are accomplished by arranging the set first and then locating the median in the middle index, simply disregarding the even and odd factor. We have used Quick select algorithm for the sorting.

Quick Select using Lumoto Paritioning:

Cbest() $\in \Theta$ (nlogn)

CAverage(n) $\in \Theta$(n)

Cworst = $\in \Theta(n^2)$

Quick Select using Sedgewick Paritioning:

Cbest() $\in \Theta$ (logn)

CAverage(n) $\in \Theta(n^2)$

Our data fell into the linear class in the average cases, and nlogn in the best case while the worst case was indifferent to the quadratic class but did not demonstrate a quadratic order of growth, indicating that the data we gathered fell into the best and average case categories.

Even though the data we collected was transmitted to both techniques, to run for every single trial, we can see that there is a significant difference in the efficiency with which both algorithms addressed the problem.

5. Conclusion

At the end of analysis we concluded that Sedgwick is running faster than Lumoto, Best and average case for Sedgewick is Θ(log n) Θ(n^2) respectively, and for Lumoto it will be Θ(nlogn) for the best case and Θ(n) for the average case.

As an aside, since the data from this project's empirical analysis fell into a specific class for all of the cases (average, best, and worst), it might be a smart option to use more data to ensure that we cover all of the possible cases and properly define the efficiency classes. This way, we'll have a more precise results and be able to articulate the algorithm in the most effective way; this is why I had my inputs range from 100-1,000,000.

6. Appendix

We will focus on showing the analysis of both algorithms this time using the physical runtime.

The physical run time for both the Quick Select Sedgwick and Lumoto approaches is shown in the tables below:

the following tables show the physical run time for both.

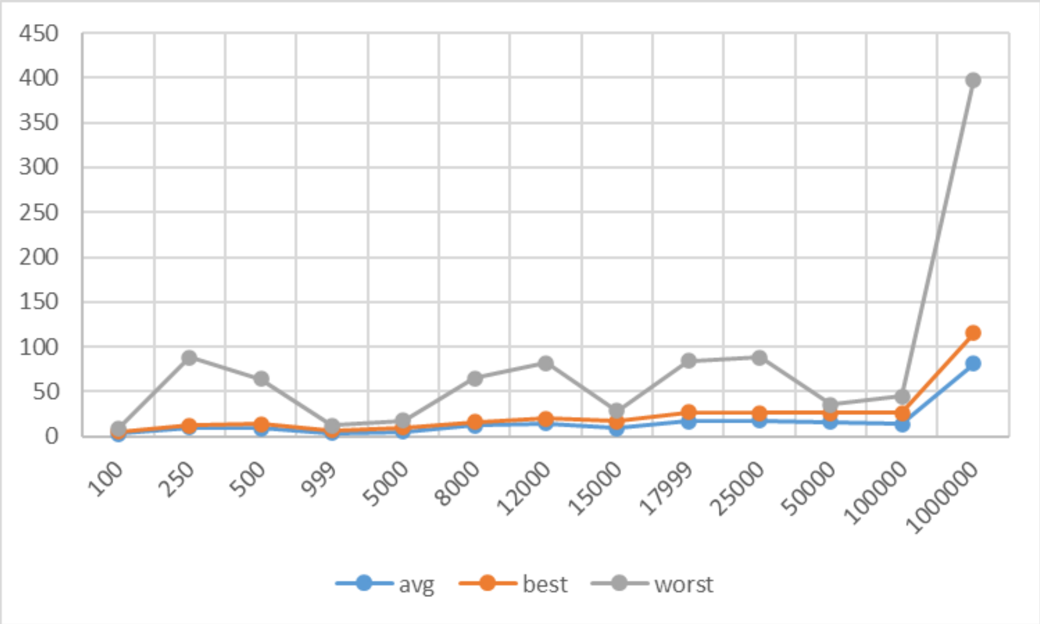
		Sedgewick											
input	1st Trail	2nd Trail	3rd Trail	4th Trail	5th Trail	6th Trail	7th Trail	8th Trail	9th Trail	10th Trail	Average	Best	Worst
100	2.71	2.882	2.908	2.491	3.261	3.044	2.458	2.797	2.932	2.339	2.7822	2.339	3.261
250	3.669	2.989	2.789	75.619	2.457	2.586	5.01	2.553	3.106	2.339	10.3117	2.339	75.619
500	5.192	4.279	4.905	4.707	5.056	4.011	6.739	4.136	5.859	50.876	9.576	4.011	50.876
999	3.311	4.878	3.308	3.589	3.248	4.477	3.369	3.462	3.411	5.109	3.8162	3.308	5.109
5000	7.648	4.879	4.487	6.889	6.634	4.407	5.017	6.762	4.743	6.306	5.7772	4.407	7.648
8000	6.903	6.509	7.799	6.629	6.425	12.041	48.654	7.112	15.291	4.08	12.1443	4.08	48.654
12000	10.487	6.142	62.459	12.8	9.131	5.353	12.263	13.701	6.682	5.716	14.4734	5.353	62.459
15000	7.548	11.701	10.438	7.688	10.556	10.196	8.353	8.173	10.809	7.734	9.3196	7.548	11.701
17999	20.465	11.107	57.008	12.29	12.655	16.314	10.214	9.988	10.481	12.263	17.2785	9.988	57.008
25000	10.862	11.084	12.179	8.152	10.794	10.642	62.338	24.937	15.764	12.538	17.929	8.152	62.338
50000	10.815	10.836	10.438	13.034	12.551	10.59	10.842	61.681	9.842	10.675	16.1304	9.842	9.842
100000	14.507	11.725	18.807	13.858	14.824	16.411	14.631	13.18	11.903	12.556	14.2402	11.725	18.807
1000000	77.274	54.663	84.228	69.81	69.157	46.051	64.632	34.383	33.995	281.783	81.5976	33.995	281.783

Sedgewick time 1

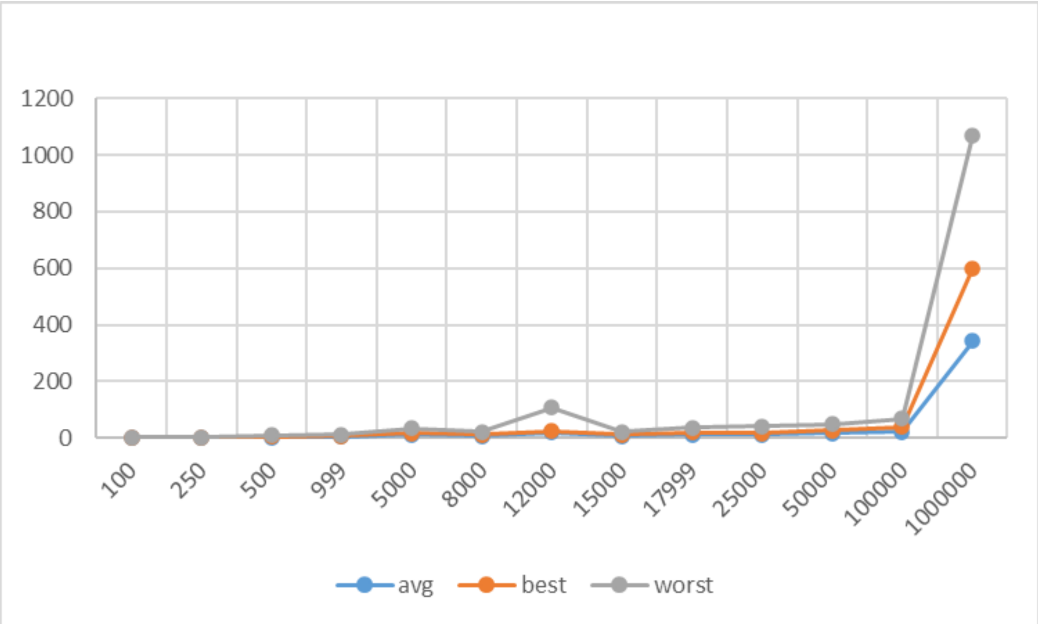
		Lumoto											
input	1st Trail	2nd Trail	3rd Trail	4th Trail	5th Trail	6th Trail	7th Trail	8th Trail	9th Trail	10th Trail	Average	Best	Worst
100	0.363	0.599	0.554	0.472	0.425	0.574	0.446	0.628	0.687	0.25	0.4998	0.25	0.687
250	0.489	1.273	0.822	0.468	0.399	0.786	0.782	0.462	1.001	0.446	0.6928	0.399	1.273
500	1.097	5.64	1.1	1.561	4.394	1.125	3.204	3.285	1.223	0.934	2.3563	0.934	5.64
999	3.104	6.131	5.558	2.399	3.873	4.313	2.852	3.843	2.953	3.475	3.8501	2.399	6.131
5000	16.766	10.087	8.905	12.573	13.783	8.944	9.245	6.292	6.113	14.315	10.7023	6.113	16.766
8000	5.196	4.922	4.955	7.509	5.817	8.188	5.71	6.087	10.824	4.563	6.3771	4.563	10.824
12000	9.191	4.814	8.041	81.347	5.679	5.882	63.571	10.636	5.118	5.595	19.9874	4.814	81.347
15000	5.817	8.555	7.799	7.304	5.451	8.491	6.109	5.517	7.771	5.451	6.8265	5.451	8.555
17999	12.507	8.099	17.712	9.868	9.848	12.722	13.917	8.534	9.754	11.746	11.4707	8.099	17.712
25000	7.591	10.192	9.501	6.766	10.438	10.091	12.079	22.868	10.383	10.694	11.0603	6.766	22.868
50000	23.515	19.282	9.934	14.75	16.929	18.444	16.735	13.79	10.467	13.465	15.7311	9.934	23.515
100000	17.554	17.304	22.311	20.632	24.62	24.061	28.136	25.282	21.472	20.102	22.1474	17.304	28.136
1000000	394.46	260.741	259.027	274.955	406.847	433.671	335.52	318.76	262.587	468.92	341.5488	259.027	468.92

Lumtot time 1

In these graphs, we'll be using the scatter plot to display the data from the preceding tables to examine if they follow a structure or fluctuate arbitrarily:



Sedgewick time complexity 1



Lumoto time complexity 1

The preceding graphs clearly reveal no structure to the generally recognized classes. However, as mentioned before, run time is an extremely poor way to analyze the efficiency of an algorithm. When the algorithms run on small instances, the run times, when compared, show that Lumoto is faster. As the input size increased to 1,000,000, Sedgewick appeared to be the faster algorithm. Those chart shows how well the physical run time growth seems to be horribly inaccurate in comparison to the results of the principle or the basic operation count.