King Abdul-Aziz University
Faculty of Computing and Information Technology
Computer Science Department
Analysis & Design of Algorithms – CPCS223 Spring 2022

Empirical Analysis Using Quick Select With Sedgewick Partition And Lomuto Partition To Find The Median

Dear Student name: Razan Arif Alamri

Section: DAR

Instructor: Dr.Muhammed Al-hashimi

Table of Contents:

ILLUSTRATIONS: **3**

1. INTRODUCTION **4**

2. EMPIRICAL ANALYSIS OF ALGORITHMS **5**

2.1 PURPOSE 5

2.1 CHOICE OF EFFICIENCY METRIC 5

3. DESIGN & PROCEDURE **6**

3.1 CHARACTERISTICS OF THE INPUT SIZE 6

3.2 THE ALGORITHMS 6

3.3 THE TOOLS 7

4. DISCUSSION **7**

4.1 THE RESULTS: 7

4.1.1 THE RESULTS TABLES: 8

4.2 THE GRAPHS: 9

4.3 THE ORDER OF GROWTH 10

4.3.1 QUICK SELECT WITH SEDGEWICK PARTITION: 10

4.3.2 QUICK SELECT WITH LOMUTO PARTITION: 11

5. CONCLUSION **12**

6. REFERENCES **13**

7. APPENDEX **14**

Illustrations:

Table 1: Quickselect with Sedgewick partition (Basic Operation Counter) Data 8

Table 2: Quickselect with Lomuto partition (Basic Operation Counter) Data 8

Table 3: Summary of efficiency 12

Figure 1: Quickselect with Sedgewick partition (Basic Operation Counter) for all cases 9

Figure 2: Quickselect with Lomuto partition (Basic Operation Counter) for all cases 9

Figure 3: Quick select with Sedgewick partition with famous classes 10

Figure 4: Quickselect with Lomuto partition with famous classes 11

1. Introduction

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time. There are many important things that must be taken care of when writing algorithms, including security, ease of use, ability to develop, and others, but most important of all is the efficiency of the algorithm.

Algorithm efficiency is a measure of the average execution time necessary for an algorithm to complete work on a set of data. We use algorithm analysis to measure the efficiency of the algorithm. Algorithms are usually analyzed and their efficiency calculated mathematically based on theories, but in some algorithms it is difficult to calculate the efficiency mathematically so another analysis method is used which is empirical analysis.

Empirical analysis of an algorithm is the inference of an algorithm's efficiency by experiment and data collection. In this report we will use empirical analysis to compare two different algorithms to find the median of a set of numbers, and determine which one is more efficient than the other.

2. Empirical Analysis of Algorithms

2.1 Purpose

The purpose of the empirical analysis is to compare the efficiency and study the behavior of Quick select using two different partitions Sedgewick Partition and Lomuto Partition to find the median for a set of random numbers in the range from 0 to 100. We aim to empirically analyze the two partitions and compare the results to determine the best algorithm.

2.1 Choice of efficiency metric

There are two ways to measure efficiency in the first empirical analysis by using physical time or by using a basic operation counter.

In this report, we will choose the basic operation counter to measure the efficiency by counting the number of times the basic operation is executed, because the physical time of the algorithm is not the actual time required to execute a particular code, since that depends on other factors like programming language, operating software, other programs on the background of the computer, processing power, etc. Each machine and CPU may operate at different speeds. The CPU in my computer may be faster than the CPU in your computer. So the algorithm will run faster on my computer.

Therefore basic operation counter is the best way to analyze algorithms empirically because it does not depend on other factors in the measurement.

3. Design & Procedure

3.1 Characteristics of The Input size

To make a correct comparing, the input size must be the same for both algorithms. we chose to start with sizes from 100 to 1.5 million (due to restrictions of the machine it was tested on, it couldn't process any larger numbers), and to generalize the results we must make sure that both even and odd numbers are used as input size. the point of taking a big range of sizes is to test the efficiency more accurately on a wide variety of inputs. As for the range, the range of values was pre-specified as random integers between 0 and 100.

We will run the program ten times for the same size with different values of the array and test the efficiency in each time, these steps will be repeated on all sizes. Then we collect and save the data and calculate the average of the data to ensure the accuracy of the results.

3.2 The Algorithms

In this project we have used the following algorithms:

- **Quick Select:**

  From Dr.Muhammad Al-Hashimi Website-lecture 21.

- **Lomuto Partition:**

  Obtained from textbook, chapter 4.5 page 159.

- **Sedgewick Partition:**

  From Dr.Muhammad Al-Hashimi Website-lecture 22.

3.3 The Tools

- **Visual Studio:**

  Used to write and edit the code in JavaScript language.

- **Firefox browser:**

  Used to run the code and show the output on the console.

- **Excel:**

  Used to analyze data, draw graphs and make calculations.

4. Discussion

4.1 The results:

The results were taken from the web browser and imported into an excel sheet, to determine the average, best and worst cases of efficiency (by basic operation counter), excel functions average, min, and max were used, respectively. Physical runtime results can be found in the **Appendix** on page 12.

4.1.1 The results tables:

Quickselect with Sedgewick partition (Basic Operation Counter)

Input Size	1	2	3	4	5	6	7	8	9	10	Avrege	Best	Worst
100	589	305	241	193	333	216	225	395	323	291	311.1	193	589
699	3370	2473	1629	2274	1573	1460	1625	2176	1709	2133	2042.2	1460	3370
1000	3299	2601	3877	3576	3885	4123	1629	2519	3235	2612	3135.6	1629	4123
6899	34737	16323	25504	41687	30871	19210	16722	45940	25380	28290	28466.4	16323	45940
10000	27500	19175	28492	24108	46388	20371	22306	39303	35841	24867	28835.1	19175	46388
57799	273058	249047	156383	152241	169423	140489	197221	178181	220334	257848	199423	140489	273058
100000	377456	309172	328894	252690	295575	507887	306217	190384	369232	272487	320999	190384	507887
457777	1767584	906660	1440953	1316577	2174917	1335425	1572412	1889173	1076064	1535940	1501571	906660	2174917
1000000	2501036	2703754	5279143	4821817	3410845	1924325	2268611	3460521	2775414	5878291	3502376	1924325	5878291
1377777	7273368	4213661	2457310	2827587	3300457	3013665	4841339	3039830	4366705	4246087	3958001	2457310	7273368
1500000	3716987	5381921	3350111	4511434	3215431	5070929	5079101	7608481	3338437	4278235	4555107	3215431	7608481

Table 1: Quickselect with Sedgewick partition (Basic Operation Counter) Data

Quickselect with Lomuto partition (Basic Operation Counter)

Input Size	1	2	3	4	5	6	7	8	9	10	Avrege	Best	Worst
100	355	365	365	249	310	351	306	392	259	451	340.3	249	451
699	2410	2835	3617	2922	1557	1411	2834	2540	2092	2277	2449.5	1411	3617
1000	4889	2406	2406	3273	3840	2862	3057	2248	3863	5306	3415	2248	5306
6899	16499	18467	25881	18448	59687	16588	16588	40265	39114	27135	27867.2	16499	59687
10000	36719	38538	25601	26599	43740	30079	30108	25844	34924	45073	33722.5	25601	45073
57799	329258	302151	277034	344664	245995	309956	322687	191157	355116	284131	296215	191157	355116
100000	845508	813939	852135	688186	777102	901349	712601	643462	590464	594463	741921	590464	901349
457777	1E+07	7959949	1.1E+07	8634427	1E+07	1E+07	1E+07	8699238	9134367	8088166	9485568	7959949	1.1E+07
1000000	4.3E+07	3.8E+07	4.1E+07	3.8E+07	4.1E+07	3.6E+07	4.4E+07	4.1E+07	3.8E+07	4.1E+07	4E+07	3.6E+07	4.4E+07
1377777	7.2E+07	7E+07	7.1E+07	7.3E+07	7.8E+07	8.1E+07	7.6E+07	7.2E+07	8.1E+07	7.8E+07	7.5E+07	7E+07	8.1E+07
1500000	9E+07	8.6E+07	7.7E+07	9E+07	8.6E+07	8.1E+07	8.7E+07	9E+07	8.6E+07	8.5E+07	8.6E+07	7.7E+07	9E+07

Table 2: Quickselect with Lomuto partition (Basic Operation Counter) Data

4.2 The graphs:

Plotting the data to get visual results:

Sedgewick partition's (average, best and worst) efficiency and Lomuto partition's (average, best and worst) efficiency, visualized as a graphs:

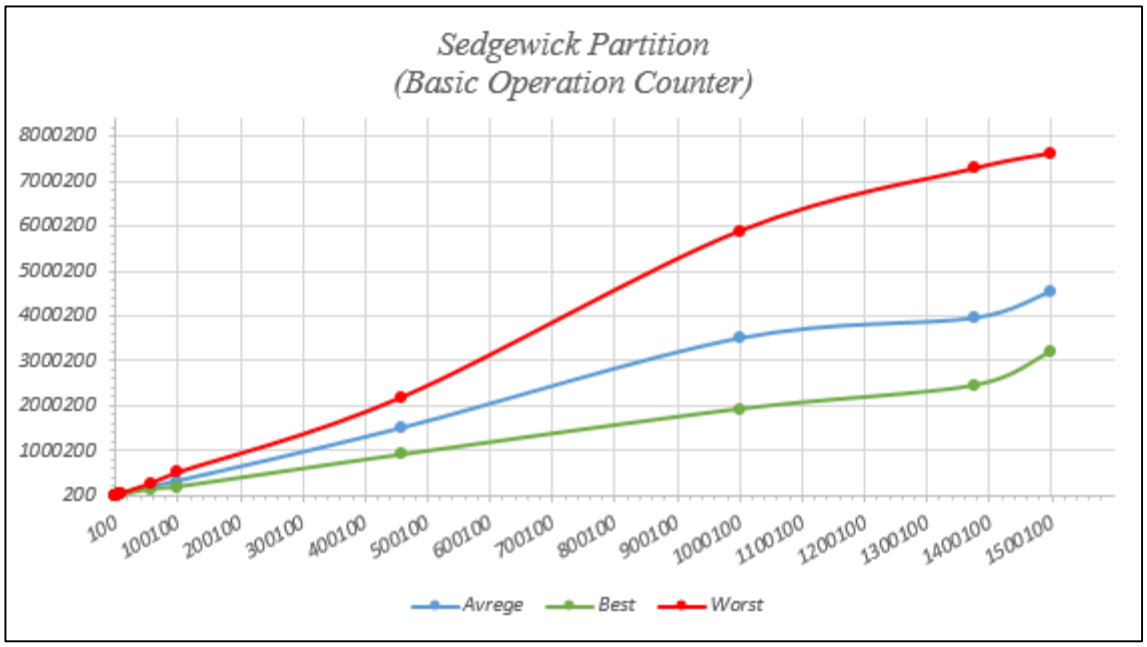


Figure 1: Quickselect with Sedgewick partition (Basic Operation Counter) for all cases

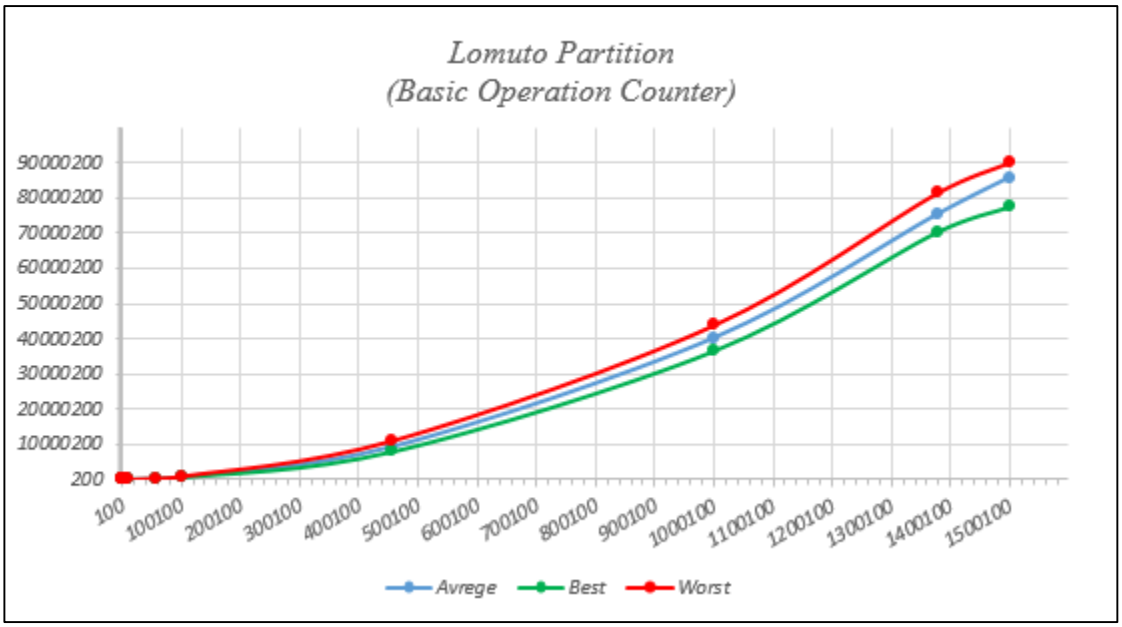


Figure 2: Quickselect with Lomuto partition (Basic Operation Counter) for all cases

4.3 The Order of Growth

Now that we have showed the graph of the average, best and the worst cases of each approach, we can analyze the algorithms by using those graphs.

4.3.1 Quick Select with Sedgewick Partition:

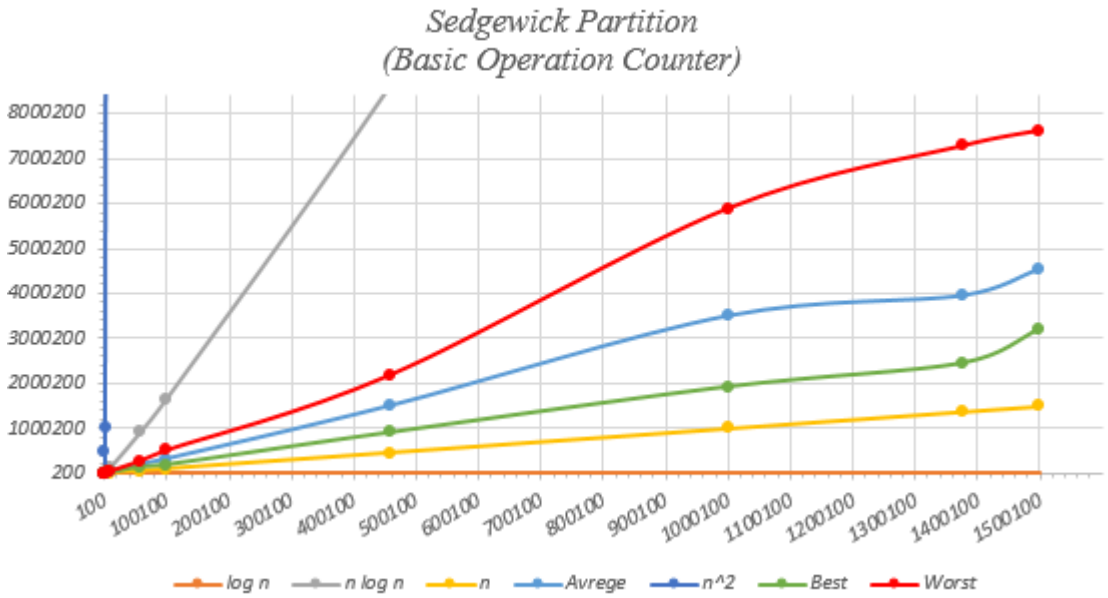


Figure 3: Quick select with Sedgewick partition with famous classes

Figure 3 shows the Quick Select with Sedgewick Partition approach, the data collected in the best, worst and average cases are close to the (n) class. Theoretically the quick select with sedgewick partition efficiency classes are:

- $C_{best}(n) = C_{average}(n) \in \Theta(n \log n)$
- $C_{worst}(n) \in \Theta(n^2)$

The data approaches the (best & average) case side where the efficiency is g(n) = n, and we have not encountered data that approaches the worst case side of our data.

4.3.2 Quick Select with Lomuto Partition:

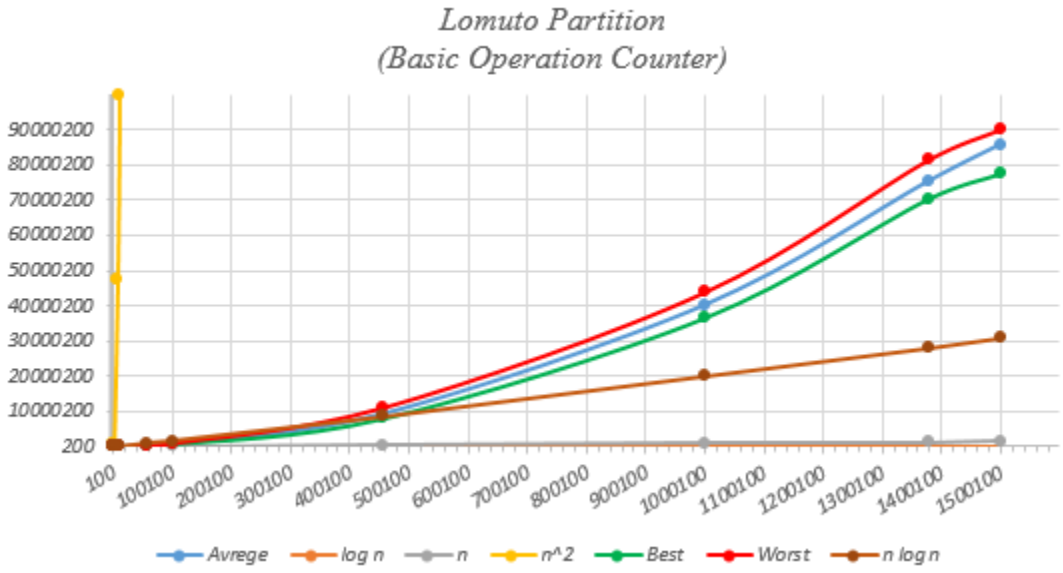


Figure 4: Quickselect with Lomuto partition with famous classes

Figure 4 shows the Quick Select with Lomuto Partition approach, the data collected in the best, worst and average cases are close to the (n log n) class. Theoretically the quick select with lomuto partition efficiency classes are:

- $C_{best}(n) = C_{average}(n) \in \Theta(n)$
- $C_{worst}(n) \in \Theta(n^2)$

The data approaches the efficiency g(n) = n log n between the (best & average) case side where the efficiency is g(n) = n and the worst side where the efficiency is g(n) = n^2, but we have rarely encountered data that approaches the worst case side of our data.

5. Conclusion

The efficiency of Quick select using Sedgewick and Lomuto partitions on an application that's expected to process datasets of integers in the range 0 -100 of unspecified sizes, according to our experiment follows:

Case	***Sedgewick partition***	***Lomuto partition***
C best (n)	$\epsilon\ \Theta$ *(n)*	$\epsilon\ \Theta$ *(n log n)*
C average (n)	$\epsilon\ \Theta$ *(n)*	$\epsilon\ \Theta$ *(n log n)*
C worst (n)	$\epsilon\ \Theta$ *(n)*	$\epsilon\ \Theta$ *(n log n)*

Table 3: Summary of efficiency

We observe that Quick Select with Sedgewick Partition is the most efficient way to find the median of a set of numbers theoretically and through experiment. Whenever the input size increase, the Sedgewick partition still gives better results, unlike the Lomuto partition.

In conclusion, these results that appeared cannot be generalized to all cases of the two algorithms because they were experimented with on a random sample and in a certain range of numbers, so more accurate results can be obtained for efficiency on the way more samples are used and in a larger range of numbers.

6. References

1. *Textbook: Levitin, A. (2012). Introduction to Design and Analysis of Algorithms, 3/E. Pearson Education India.*
2. *Dr. M Hashimi website: https://www.hashimi.ws/cs223/index.ph*

7. Appendex

Quickselect with Sedgewick partition (Runtime) data

Quickselect with Sedgewick partition (Runtime in ms)

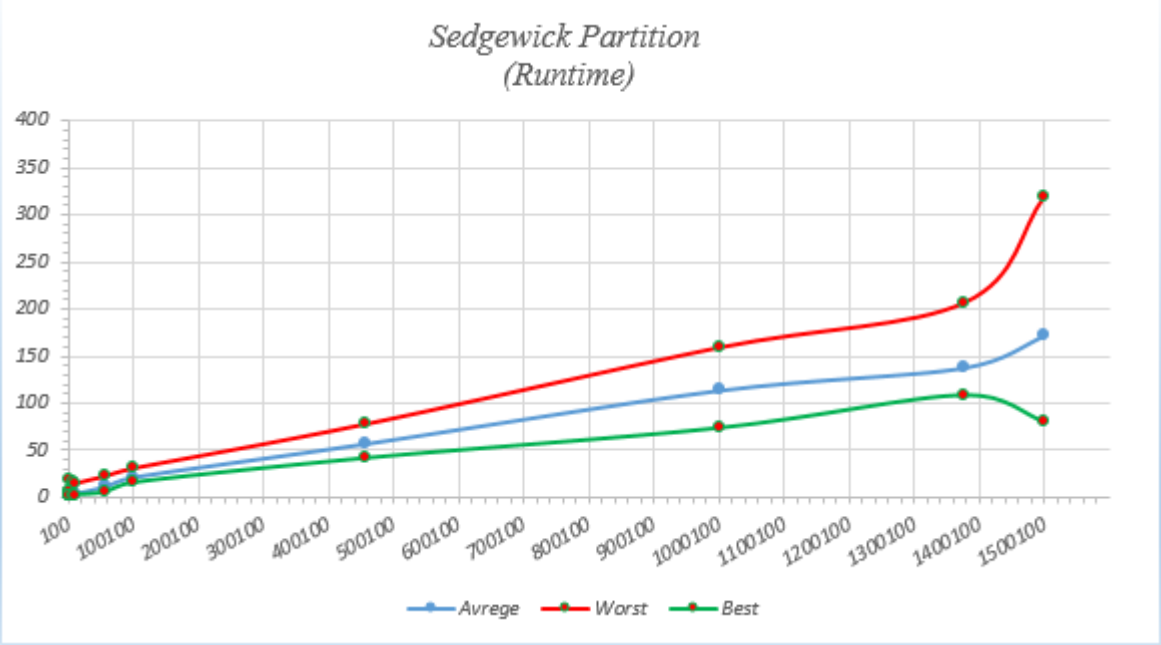
Input Size	1	2	3	4	5	6	7	8	9	10	Avrege	Best	Worst
100	7	3	4	6	1	3	2	4	2	2	3.4	1	7
699	6	2	1	1	3	2	2	1	1	3	2.2	1	6
1000	20	7	2	5	2	3	2	2	4	3	5	2	20
6899	16	5	4	4	3	3	5	5	3	4	5.2	3	16
10000	15	3	5	3	4	3	3	4	4	3	4.7	3	15
57799	23	11	9	7	8	6	19	9	8	22	12.2	6	23
100000	23	18	26	16	22	31	21	19	26	19	22.1	16	31
457777	66	42	58	59	64	55	59	78	43	52	57.6	42	78
1000000	107	87	157	135	114	74	92	119	96	159	114	74	159
1377777	207	142	111	113	117	118	171	109	146	148	138.2	109	207
1500000	147	80	156	320	98	212	192	249	108	159	172.1	80	320

Quickselect with Lomuto partition (Runtime) data

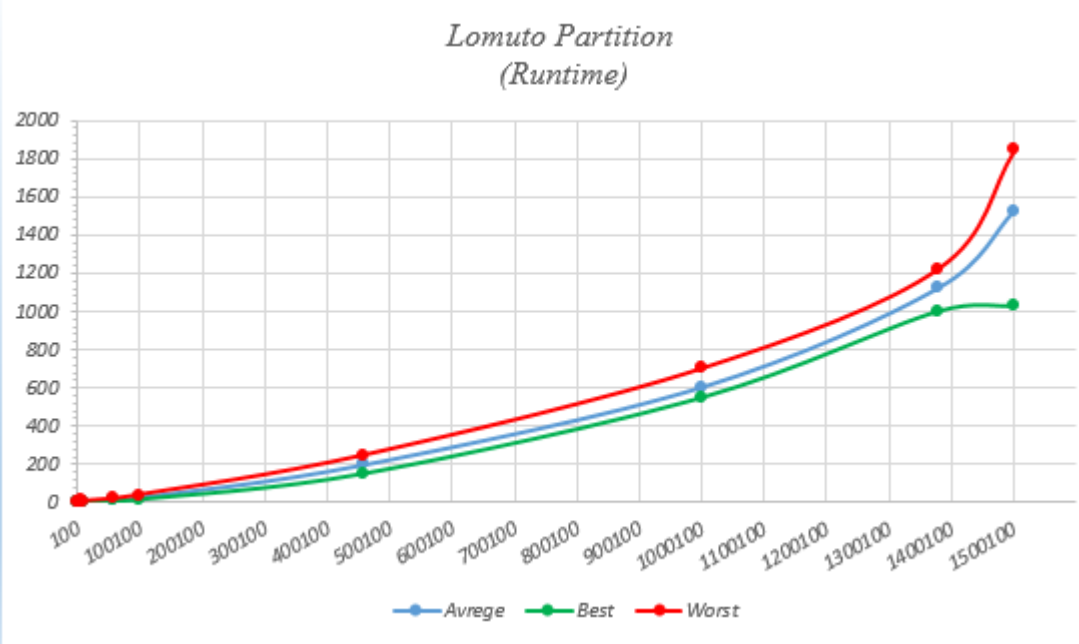
Quickselect with Lomuto partition (Runtime in ms)

Input Size	1	2	3	4	5	6	7	8	9	10	Avrege	Best	Wort
100	7	7	6	1	3	2	1	1	1	3	3.2	1	7
699	8	4	2	2	3	2	4	1	2	3	3.1	1	8
1000	6	4	4	1	4	3	7	5	3	4	4.1	1	7
6899	11	3	4	3	5	3	3	6	4	4	4.6	3	11
10000	9	3	3	2	4	4	4	3	4	4	4	2	9
57799	20	12	10	13	14	18	13	11	12	21	14.4	10	21
100000	31	22	40	35	35	41	26	22	19	21	29.2	19	41
457777	248	154	237	156	205	197	233	154	181	234	199.9	154	248
1000000	703	585	604	584	640	553	629	627	563	621	605.5	553	703
1377777	1042	1224	1175	1004	1100	1201	1106	1056	1157	1164	1122.9	1004	1224
1500000	1286	1031	1847	1718	1808	1602	1707	1541	1295	1446	1528.1	1031	1847

Quickselect with Sedgewick partition (Runtime) graph



Quickselect with Lomuto partition (Runtime) graph



Efficiency classes for chosen input sizes

Input Size	log n	n log n	n	n ^2	n ^3
100	6.64386	664.386	100	10000	1000000
699	9.44915	6604.95	699	488601	3.4E+08
1000	9.96578	9965.78	1000	1000000	1E+09
6899	12.7522	87977.2	6899	4.8E+07	3.3E+11
10000	13.2877	132877	10000	1E+08	1E+12
57799	15.8188	914308	57799	3.3E+09	1.9E+14
100000	16.6096	1660964	100000	1E+10	1E+15
457777	18.8043	8608169	457777	2.1E+11	9.6E+16
1000000	19.9316	2E+07	1000000	1E+12	1E+18
1377777	20.3939	2.8E+07	1377777	1.9E+12	2.6E+18
1500000	20.5165	3.1E+07	1500000	2.3E+12	3.4E+18