Explore – Journal of Research
Peer Reviewed Journal
ISSN 2278–0297 (Print)
ISSN 2278–6414 (Online)
Vol. XIV No. 2, 2022

https://patnawomenscollege.in/explore-journal-of-research/

A Comparative Analysis of Sorting Algorithms under Discrete Uniform Distribution

• **Priyadarshini**

Received : April 2022
Accepted : May 2022
Corresponding Author : Priyadarshini

Abstract: Sorting can be viewed as one of the fundamental aspects of computer applications. The efficiency of a sorting algorithm is a major issue when the amount of data to be sorted is large. Different sorting algorithms are analyzed based on time complexity and space complexity to select the efficient one based on their execution time for varying input sizes. In the case of sorting algorithms, when the array elements to be sorted are randomly generated from some probability distribution; then the true ability of an algorithm can be judged only when it is supported by the study of parametric complexity analysis which includes the study of the behaviour of running times as a function of parameters of the input distribution. To select a particular sorting algorithm from among a set of several algorithms, the behaviour of the parameters of the input distribution plays an important role and it can't be ignored as far as the efficiency of the algorithm is concerned. In this paper, discrete uniform distribution is chosen for the study.

Three sorting algorithms, having similar average-case complexity O (N log N), quick sort, heap sort and merge sort have been chosen here for statistical comparative study of their parametric complexity. The purpose of the study is to investigate the effect of the parameter of the discrete uniform distribution on the sorting time of the three sorting algorithms under study. While working with several algorithms, one way to judge the best algorithm is to find the execution time T(n), as a function of N(input size) and examine the run time complexity using big O notation. However, sometimes it is difficult to find the exact form of the function and it becomes difficult to predict the complexity/ efficiency of an algorithm. Further, it has been tried to find the best sorting algorithm by performing a variance-based analysis of the sorting algorithms.

Keywords: *Sorting, parametric complexity, discrete uniform distribution, variance, the average case, worst case*

Priyadarshini
Assistant Professor, Patna Women's College, P.U.
Email-id : singh.priyadarshini80@gmail.com

Introduction:

Sorting can be viewed as one of the fundamental aspects of computer applications. The efficiency of a sorting algorithm is a major issue when the amount of data to be sorted is large. Different sorting algorithms are analyzed based on time complexity and space complexity to select the efficient one based on their execution time for varying input sizes. The main purpose behind analyzing algorithms is to determine their characteristics to assess their suitability for diverse applications or to judge against the other algorithms for the same application. In the case of sorting algorithms, when the array elements to be sorted are randomly generated from some probability distribution; then the true ability of an algorithm can be judged only when it is supported by the study of parametric complexity analysis which consists of the study of the behaviour of running times as a function of parameters of the input distribution. To select a particular sorting algorithm from among a set of several algorithms, the behaviour of the parameters of the input distribution plays an important role and it can't be ignored as far as the efficiency of the algorithm is concerned. The works done on parameterized complexity have been explored a lot in the recent past.

Mahmoud (2000) may be consulted for literature review with special reference to sorting algorithms. The works of Anchala and Chakraborty (2007), Anchala and Chakraborty (2008), and Prashant, Anchala and Chakraborty (2009) on parameterized complexity of sorting algorithms greatly contribute to the literature review. The theoretical concept of general parameterized complexity can be better understood by referring Flum and Grohe (2006).

A theoretical probability distribution provides a rule according to which various values of the random variable are distributed with specified probabilities based on some fixed rule that can be stated mathematically.

A discrete probability distribution exhibits the probabilities of outcomes with fixed values. The probability distribution of a random variable that takes distinct values is known as a discrete probability distribution.

In this paper, *discrete uniform distribution* is chosen for the study. Three sorting algorithms, having similar average-case complexity O(*N log N*), merge sort, heap sort and quick sort, have been chosen here for statistical comparative study of their parametric complexity. The purpose of the study is to examine the effect of the parameter of the discrete uniform distribution on the sorting times of the three sorting algorithms understudy, for average case and worst case situations as well.

In the quick sort algorithm, the median of the first, middle and last element of the array is taken as the pivot element.

A laptop computer with a configuration (Intel (R) Core ™ i3-4005U Processor @ 1.70GHz, 8GB RAM, Windows 7 operating system) is used to do a series of computer experiments. The execution times of the sorting algorithms are computed by running codes written in Dev C++ using C language and different relative graphs are drawn using Minitab 17 Statistical Package.

Discrete Uniform Distribution

Definition: "If a random variable X follows discrete uniform distribution over the range [1,θ] if its probability mass function is expressed as follows":

$$P(X = x) = \begin{cases} \frac{1}{\theta}; x = 1,2,3,4,...\theta \\ 0, otherwise \end{cases}$$

Here, θ is called the parameter of the distribution and lies in the set of all positive integers.

The discrete uniform distribution is a symmetric distribution in which there is an equal chance of occurrence of a finite number of values. Each one of the θ values has an equal probability of occurrence $1/\theta$.

Random number generation

For the generation of random numbers from the discrete uniform distribution
UD (1, θ):

Step 1: A random number u~U(0,1) is generated and [1 + θ*u] is returned as a uniform variate.

Relative performance of various sorting algorithms for discrete uniform input:

Heap sort in average and worst cases

Table 1. Shows the average case runtime of heap sort for varying parameter (θ)			
N	θ = 100	θ = 500	θ = 1000
10000	0.009400	0.009200	0.009200
20000	0.022000	0.012600	0.009200
30000	0.022200	0.019200	0.024600
40000	0.025000	0.024800	0.037400
50000	0.034400	0.037400	0.050200
60000	0.043600	0.040600	0.056600
70000	0.056200	0.056400	0.053200
80000	0.075000	0.066000	0.065800
90000	0.081200	0.081400	0.081000
100000	0.071600	0.094000	0.093800

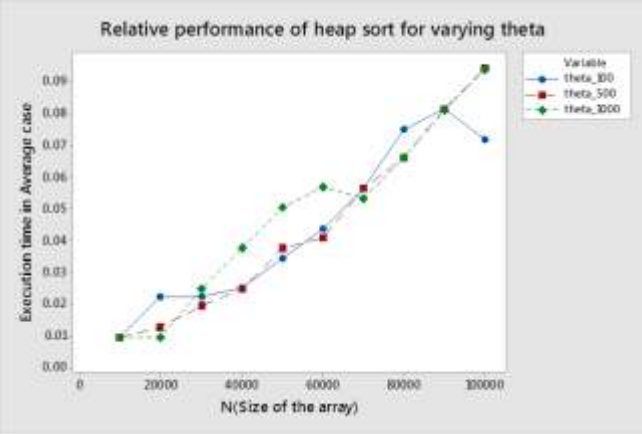

Fig. 1

Table 2. Shows the worst-case runtime of heap sort for varying parameter (θ)			
N	θ = 100	θ = 500	θ = 1000
10000	0.012600	0.006400	0.009400
20000	0.015400	0.009200	0.009600
30000	0.018600	0.021600	0.022200
40000	0.022200	0.028200	0.034600
50000	0.024800	0.028200	0.046800
60000	0.031200	0.034600	0.040200
70000	0.050000	0.052800	0.047000
80000	0.056200	0.059000	0.050000
90000	0.062600	0.062400	0.053200
100000	0.072000	0.062400	0.071600

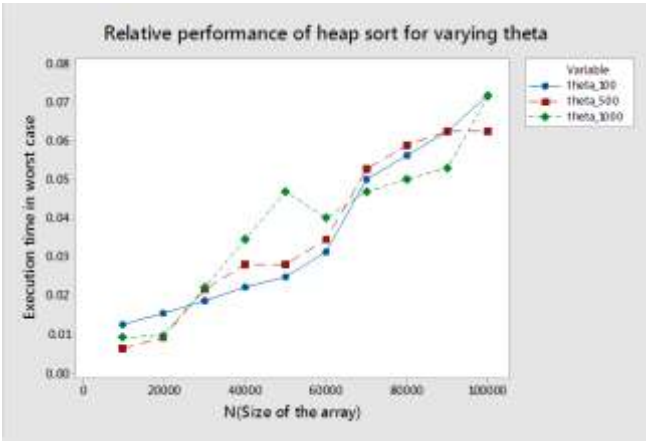

Fig. 2

The execution time of heap sort for different values of θ (parameter of the distribution) when applied to an array of random elements generated from U(θ) distribution in both average and worst-case has been recorded in Tables 1 and 2 respectively and corresponding plots (execution time vs array size) for average and worst cases are shown in Fig. 1 and 2 respectively. If average-case complexity is compared with its worst-case counterpart, it reveals a very irregular pattern for small values of θ in the average case whereas a similar irregular complexity pattern is observed for high values of θ in the worst case. However, for arrays of moderate size (N between 20000 to 70000), smaller values of the parameter in both average and worst cases are preferable.

In case of failure to find some systematic complexity pattern of heap sort in case of uniform distribution, complexity pattern was further investigated with very high values of θ(>1000) and it was found that there is almost no change in the performance of heap sort with change in the parameter of the input distribution i.e. θ for both average and worst-case situations. The fact that the parameter of the discrete uniform distribution does not affect the heap sort complexity is further supported by the parametric complexity in section 4. The size of the array (N) to be sorted is the only factor affecting the execution time of the sorting algorithm in average and worst cases.

Quick sort in Average and Worst cases

Table 3. Average case runtime of quick sort for varying parameter (θ)			
N	θ = 100	θ = 500	θ = 1000
10000	0.003200	0.000000	0.003000
20000	0.009200	0.003200	0.009600
30000	0.018600	0.009200	0.006200
40000	0.028000	0.015800	0.012400
50000	0.037400	0.015800	0.015600
60000	0.056600	0.022000	0.015800
70000	0.065600	0.022000	0.021600
80000	0.093800	0.031200	0.028000
90000	0.112400	0.043600	0.028200
100000	0.137600	0.043800	0.031200

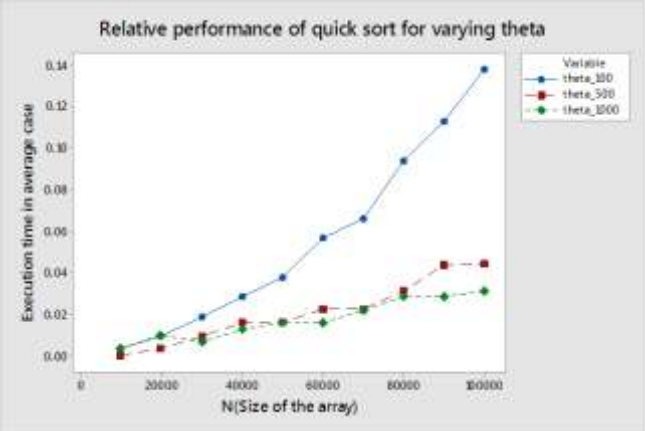

Fig. 3

Table 4. Shows the worst-case runtime of quicksort for varying parameter (θ)			
N	θ = 100	θ = 500	θ = 1000
10000	0.021800	0.018600	0.018800
20000	0.053400	0.040800	0.043400
30000	0.109400	0.084600	0.075200
40000	0.231600	0.121600	0.137600
50000	0.319000	0.177800	0.209400
60000	0.390200	0.318600	0.240800
70000	0.578000	0.390800	0.231600
80000	0.818800	0.537400	0.456200
90000	0.640600	0.500000	0.537600
100000	1.121600	0.756200	0.831200

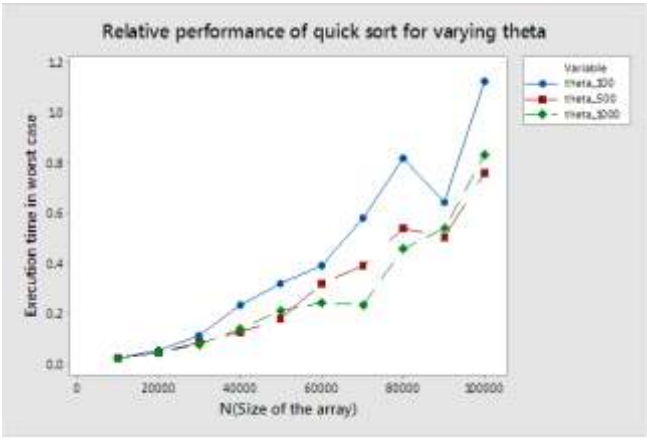

Fig. 4

The results of the computer experiments done in the case of quick sort, for uniform input in average and worst cases are depicted in tables 3 and 4 respectively. Figures 3 and 4 are the scatter plots based on these tabular data.

By inspection, we see that in average case the performance of quick sort is quite similar to that of worst-case performance. But for an array of any size and the same value of θ, the execution time in the worst case is much higher than that in the average case. However, high values of θ(>=500) is preferable, both in the average and worst cases.

Merge sort in average and worst cases

Table 5. Shows the average case runtime of merge sort for varying parameter (θ)			
N	θ = 100	θ = 500	θ = 1000
10000	0.006200	0.009200	0.009000
20000	0.012600	0.015600	0.016000
30000	0.018600	0.019000	0.019000
40000	0.024600	0.028400	0.025000
50000	0.031400	0.031400	0.031400
60000	0.034600	0.037400	0.037400
70000	0.050200	0.040600	0.046600
80000	0.053000	0.050000	0.050000
90000	0.049800	0.059400	0.056200
100000	0.059400	0.062400	0.062800

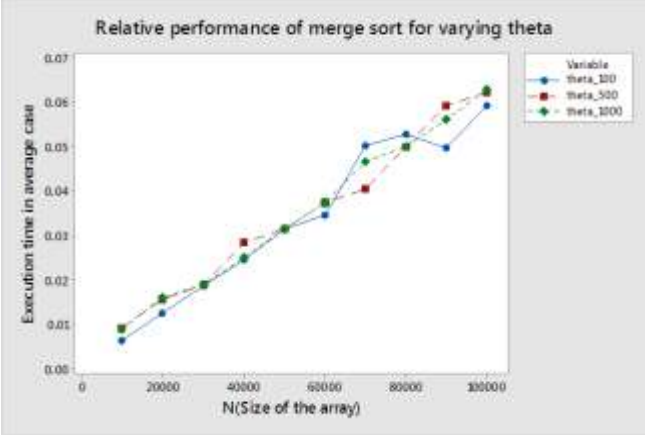

Fig. 5

Table 6. Shows the worst-case runtime of merge sort for varying parameter (θ)

N	θ = 100	θ = 500	θ = 1000
10000	0.003200	0.003200	0.006400
20000	0.012200	0.009600	0.009000
30000	0.015800	0.021600	0.015600
40000	0.022200	0.018400	0.015400
50000	0.021800	0.028000	0.025000
60000	0.027800	0.028200	0.034600
70000	0.034200	0.037400	0.034600
80000	0.040600	0.040600	0.031400
90000	0.046800	0.043800	0.047000
100000	0.050000	0.053200	0.046600

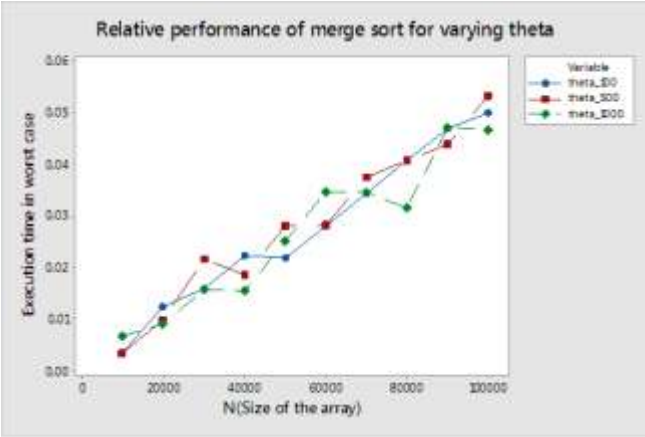

Fig. 6

In both average and worst cases when the data is simulated from uniform distribution and sorted using merge sort, a non-symmetric complexity pattern is observed. This may be because while generating data randomly, though keeping at a fixed value, some arrays (especially when the array size is large) may have a large number of tied elements which may cause irregularity in the complexity pattern.

Empirical complexity of various sorting algorithms under discrete uniform input

Table 7. Shows the average case runtime of sorting algorithms for θ = 1000

N	Quicksort	Heapsort	Merge sort
10000	0.003000	0.009200	0.009000
20000	0.009600	0.009200	0.016000
30000	0.006200	0.024600	0.019000
40000	0.012400	0.037400	0.025000
50000	0.015600	0.050200	0.031400
60000	0.015800	0.056600	0.037400
70000	0.021600	0.053200	0.046600
80000	0.028000	0.065800	0.050000
90000	0.028200	0.081000	0.056200
100000	0.031200	0.093800	0.062800

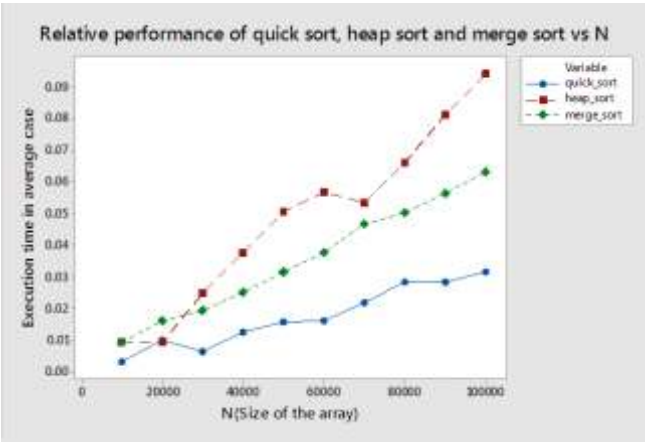

Fig. 7

Table 8 : Shows the worst case runtime of sorting algorithms for θ = 1000			
N	Quicksort	Heapsort	Merge sort
10000	0.018800	0.009400	0.006400
20000	0.043400	0.009600	0.009000
30000	0.075200	0.022200	0.015600
40000	0.137600	0.034600	0.015400
50000	0.209400	0.046800	0.025000
60000	0.240800	0.040200	0.034600
70000	0.231600	0.047000	0.034600
80000	0.456200	0.050000	0.031400
90000	0.537600	0.053200	0.047000
100000	0.831200	0.071600	0.046600

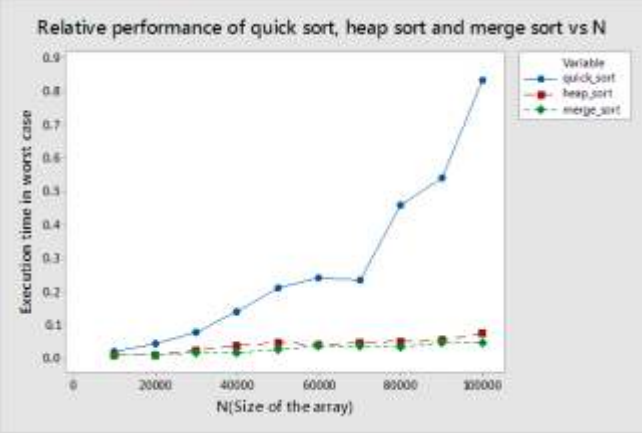

Fig. 8

As far as the relative performance of three algorithms for the uniform input is concerned, the average-case performance of quicksort is best whereas heap sort gave the worst performance. On the other hand, in the worst case, quick sort did not perform well but the other two algorithms performed almost similarly, having the same complexity level for an array of the same size. Regression analysis was performed on the data given in tables 7 and 8 to find the empirical complexity of various algorithms as a function of N(array size) which is displayed in Table 9 below.

Table 9. Empirical complexity of sorting algorithms for discrete uniform input		
Name	Average	Worst
QuicksortO	($N\log_2 N$)	$O(N^2)$
Heapsort	$O(N\log_2 N)$	O(N)
Merge sort	$O(N\log_2 N)$	$O(N\log_2 N)$

From Table 9, it is observed that for discrete uniform input, only heap sort worst case empirical complexity is different from the theoretical one i.e. $O(N\log_2 N)$. All the other sorting algorithms have shown similar empirical complexities as their theoretical complexities.

Parametric complexity:

The discussion on the performance analysis of heap sort, quick sort and merge sort under uniform input is further extended by parametric complexity analysis using a factorial experiment. The factorial experiment presents a clear picture of the significance or insignificance of the parameters of the input distribution on the response time of sorting algorithms.

Factor Information

Factor	Levels	Values
θ	3	100, 500, 1000
N	3	10000, 50000, 100000

Table 10 (a). Discrete uniform distribution (average case)							
		Heap Sort		Merge sort		Quick sort	
Sources	D.f.	F	p-value	F	p-value	F	p-value
N	2	64.30	0.000	255.19	0.000	612.7	0.000
θ	2	1.76	0.189	0.49	0.617	279.45	0.000
Nθ	4	0.80	0.333	0.13	0.972	150.76	0.000

Table 10 (b): Discrete uniform distribution (worst case)							
		Heap Sort		Merge sort		Quicksort	
Sources	D.f.	F	p-value	F	p-value	F	p-value
N	2	74.07	0.000	128.01	0.000	41.27	0.000
θ	2	2.23	0.124	0.63	0.541	1.56	0.225
Nθ	4	1.38	0.262	0.66	0.623	0.60	0.662

The results of the factorial experiments have been summarized in Table 10(a) and 10(b) for average and worst-case respectively. As seen in Tables 10(a) and 10(b), the factor N (size of the array) is found to be significant for both the average and worst cases of all the three sorting algorithms. The parameter (θ) is found to be significant for the quick sort algorithm only in the average case. The parameter θ does not affect merge sort complexity as well as heap sort complexity both in average and worst situations. It implies that while sorting array elements using merge sort and heap sort, data can be simulated from a uniform distribution with any value of θ.

Since θ is found to be significant in the case of quicksort for the average case, it is tried to obtain the optimal value for θ. The execution times obtained are given below for varying θ from 100 to 1000. At the same time, it is also observed that the interaction Nθ has a significant effect on the execution time of quicksort. Thus, the data for the optimal value of θ wereobtained for various values of array size i.e. N=10000, N=50000 and N=100000 in the tables to follow:

Table 11(a). Optimal value of θ for average-case quick sort for N=10000

θ	Quicksort
100	0.000
200	0.0062
300	0.003
400	0.0008
500	0.003
600	0.000
700	0.0032
800	0.0032
900	0.0092
1000	0.008

Table 11(b). Optimal value of θ for average-case quick sort for N=50000

θ	Quicksort
100	0.0512
200	0.0254
300	0.0152
400	0.0158
500	0.0192
600	0.0222
700	0.0156
800	0.0122
900	0.0202
1000	0.0126

From table 11(a), it is found that for array size, N=10000, the execution time of quicksort is zero (minimum) at the points θ = 100 and θ = 600. On the other hand from table 11(b) for array size N = 50000, it is seen that the minimum execution time is observed at the point θ = 800.

Table 11(c). Optimal value of θ for average-case quick sort for N=100000

θ	Quicksort
100	0.2182
200	0.1106
300	0.0846
400	0.0812
500	0.0438
600	0.0494
700	0.0374
800	0.0374
900	0.0408
1000	0.0540

Table 11(c) confirms that for array size, N=100000, the execution time of quicksort is minimum at the points θ=700 and θ=800. It can be depicted from the above investigations that for an array of large size, a high value of the parameter is preferable. While in the worst case any value of the parameter is preferable.

Comparision of sorting algorithms using the variance of execution times:

A lot of research has been done for judging the efficiency of sorting algorithms by analyzing the execution times for changing the array size. In this paper, the efficiency of a sorting algorithm is found, when the data is generated from some probability distribution. It is of paramount importance to examine the robustness of the algorithms in different situations. Variability in execution times for a given problem size is usually ignored. Efficiency is based on variance. An algorithm having a low variance of execution times will be considered more efficient than the others.

In this section, the efficiency of the three sorting algorithms understudy was compared using the variance of their execution times over the range of array size (N) from 10000 to 100000. The array is generated from uniform distribution keeping the value of the parameter θ=1000. In this procedure for the same input, the data is generated for an array of size N and the average execution time was found for 20 runs. This procedure is repeated for different values of N (10000 to 100000). Next, the variance for the data so generated

over the values of N was found and this procedure was repeated for all the three algorithms. The algorithm for sorting data simulated from a probability distribution is selected as the best one which has minimum variance.

Table 12. Shows the average case runtime of different algorithms for varying N

N	Quicksort	Heapsort	Merge sort
10000	0.001550	0.006950	0.004700
20000	0.001550	0.007000	0.010900
30000	0.007850	0.015650	0.015700
40000	0.008500	0.017950	0.021300
50000	0.012650	0.025000	0.029000
60000	0.014100	0.028300	0.034300
70000	0.016650	0.031950	0.039900
80000	0.021050	0.040650	0.044550
90000	0.025050	0.049050	0.050050
100000	0.030450	0.053100	0.056300
Variance	**0.000092**	**0.000266**	**0.0003**

Table 13. Shows the worst-case runtime of different algorithms for varying N

N	Quicksort	Heapsort	Merge sort
10000	0.014050	0.001600	0.006250
20000	0.038250	0.008600	0.007850
30000	0.067950	0.011700	0.012450
40000	0.118100	0.014850	0.015400
50000	0.173250	0.022650	0.018600
60000	0.250800	0.022650	0.030550
70000	0.327050	0.032100	0.032000
80000	0.451600	0.029650	0.033600
90000	0.449300	0.039250	0.038250
100000	0.592200	0.036700	0.041450
Variance	**0.040035**	**0.000159**	**0.000169**

As indicated by the results in Table 12, quicksort has the least variability in the execution times for sorting arrays of different sizes. Consequently for the average case, quick is the best sorting algorithm among the three. On the other hand, for the worst-case situation, the results shown in Table 13 exhibit that heap sort has the least variance. As a result, it can be said that in the worst-case situation, heap sort is the most efficient algorithm for sorting the array.

Conclusion:

The factor N (size of the array) is found to be significant for both the average and worst cases of all the three sorting algorithms. On the other hand, the other factor theta (θ) is found to be significant for the quick sort algorithm only both in the average and worst case. In both average and worst cases the performance of heap sort and merge sort are not affected by the parameter θ.

It is observed that for discrete uniform input, only heap sort worst case empirical complexity is different from the theoretical one i.e. $O(N\log_2 N)$. All the other sorting algorithms have shown similar empirical complexities as their theoretical complexities.

As far as the variance-based analysis is concerned, quicksort was the efficient algorithm in the average-case analysis, whereas heap sort was found to be the best algorithm for the worst-case situation.

The above work can be further improvised by using different statistical techniques for deciding the best sorting algorithm. This approach can be applied to arrays generated from newly developed statistical distributions that represent the real-time data more efficiently.

This research method on sorting algorithms can be further coded in some different programming languages that are implemented using compilers. Also, the variability in the execution times can be examined when the array data is constant or non-random.

References :

1. Chakraborty S. and Sourabh S. K.(2010). A Computer Experiment Oriented Approach to Algorithmic Complexity, Lambert Academic Publishing
2. Chakraborty S., Wahi C, Sourabh S. K.(2007). "On the Philosophy of Statistical Bounds: A Case Study on a Determinant Algorithm", Journal of Modern Mathematics and Statistics1(1-4), pp: 15-23
3. Knuth D. E., "The Art of Computer Programming Vol 3: Sorting and Searching", Pearson Edu. Reprint 2000.

4. Kumari, Anchala& Kumar Singh, Niraj & Chakraborty, Soubhik. (2015). “A Statistical Comparative Study of Some Sorting Algorithms” International Journal in Foundations of Computer Science & Technology. 5. 21-29. 10.5121/ijfcst.2015.5403.

5. Mahmoud, H. “Sorting: A distribution Theory”, John Wiley and Sons, 2000

6. Priyadarshini, & Kumari, Anchala. (2020). A Comparative Performance of Sorting Algorithms: Statistical Investigation. © Springer Nature Singapore Pte Ltd. 2020 M. Pant et al. (eds.), Soft Computing: Theories and Applications, Advances in Intelligent Systems and Computing 1154, https://doi.org/10.1007/978-981-15-4032-5_34

7. Priyadarshini, Kumari Anchala "Parameterized Complexity Analysis of Heapsort, K-sort, and Quicksort for Binomial Input with Varying Probability of Success", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and ISSN Approved), ISSN:2349-5162, Vol.5, Issue 7, page no. pp491-495, July-2018, Available at: http://www.jetir.org/papers/JETIR1807424.pdf8. Priyadarshini, Soubhik Chakraborty, Anchala Kumari (2018). “Parameterised Complexity of quick sort, heap sort and k-sort Algorithms with Quasi Binomial Input”, International Journal on Future Revolution in Computer Science & Communication Engineering (IJFRSCE), January 18 Volume 4 Issue 1, pp:117-123

9. Jain R., “Commonly used distributions”, (1994) www.cse.wustl.edu/~jain/ books/ftp/ch5f_slides.pdf10. Priyadarshini (2020) A Statistical Comparative Study of Parametric Complexity of Sorting Algorithms [PhD Thesis], Patna University.