A STATISTICAL COMPARATIVE STUDY OF SOME SORTING ALGORITHMS

Anchala Kumari[1], Niraj Kumar Singh[2*] and Soubhik Chakraborty[3]

[1]Department of Statistics, Patna University, Patna
[2]Department of Computer Science & Engineering, BIT Mesra, Ranchi
[3]Department of Mathematics, BIT Mesra, Ranchi

ABSTRACT

This research paper is a statistical comparative study of a few average case asymptotically optimal sorting algorithms namely, Quick sort, Heap sort and K- sort. The three sorting algorithms all with the same average case complexity have been compared by obtaining the corresponding statistical bounds while subjecting these procedures over the randomly generated data from some standard discrete and continuous probability distributions such as Binomial distribution, Uniform discrete and continuous distribution and Poisson distribution. The statistical analysis is well supplemented by the parameterized complexity analysis.

KEYWORDS

Parameterized complexity, Statistical bound, Empirical-O, Computer experiment.

1. INTRODUCTION

Quick sort and *Heap* sort are the two standard sorting techniques used in literature for sorting large data sets. These two methods exhibit the very same average case complexity of $O(N\log_2 N)$, where N is the size of input to be sorted. Quick sort, possibly, is the best choice for sorting random data. The sequential access nature of this algorithm keeps the associated constant terms small and hence resulting in an efficient choice among the algorithms with similar asymptotic class. The worst case complexity of *quick* sort is that of $O(N^2)$ while the *heap* sort in worst case exhibit the same $(N\log_2 N)$ complexity. The $\Theta(N\log_2 N)$ tight complexity of heap sort gives it an edge over much used quick sort for use in stringent real time systems. *New-sort*, an improved version of *Quick* sort which introduced by Sundararajan and Chakarborty [1] also confirms to the same average and worst case complexity as that of Quick sort ,i.e., $O(N\log_2 N)$ and $O(N^2)$ respectively. But this *New sort* technique uses an auxiliary array, increasing the space complexity thereby. A further improvement over the New *sort* was made by removing the concept of auxiliary array from it and this sorting algorithm was named as K-*sort* [2]. Interestingly, for typical inputs, on average the *K-sort* also consumes $O(N\log_2 N)$ time complexity. In a recent research paper Singh et al. [3] explores the quick sort algorithm in detail, where it discusses various interesting patterns obtained for runtime complexity data.

In this paper the three sorting algorithms all with the same average case complexity have been compared by obtaining the corresponding statistical bounds while subjecting these procedures over the randomly generated data from some standard discrete and continuous probability distributions such as Binomial distribution, Uniform discrete and continuous distribution and Poisson distribution.

ASYMPTOTIC ANALYSIS

Asymptotic analysis of Quick sort: The average and the best case recurrence of *quick* sort is $T(N) = 2T(N/2) + \Theta(N)$, $T(1)=0$. This upon solving yields a running time of $O(N\log_2 N)$. The worst case recurrence is $T(N) = T(N-1) + \Theta(N)$, $T(1)=0$, which results in $O(N^2)$ complexity of *quick* sort algorithm.

Asymptotic analysis of Heap sort: The best and worst case complexity of *heap* sort belong to $\Theta(N\log_2 N)$ complexity.

Asymptotic analysis of K-sort: Due to its peculiar similarity the asymptotic time complexities of *K-sort* is similar to that of *quick* sort.

EMPERICAL-O ANALYSIS

Empirical-O is an empirical estimate of the statistical bound over a finite range, obtained by supplying numerical values to the weights which emerge from computer experiment. A computer experiment being defined as a series of runs of a code for various inputs and is called deterministic if it gives identical outputs if the code is re run for the same input.

Statistical bound (non probabilistic): If $W_{ij}$is the weight of (a computing) operation of type i in the j^{th} repetition (generally time is taken as a weight) and y is a "stochastic realization of the deterministic $T= \sum 1.W_{ij}$ where we count one for each operation repetition irrespective of the type, the statistical bound of the algorithm is the asymptotic least upper bound of y expressed as a function of N, N being the algorithm's input size. T is realised for a fixed input while y is expressed for a fixed size of the input. It is important to know y becomes stochastic for those algorithms where fixing size does not fix all the computing operations. Sorting algorithm fall in this category

Now we perform the empirical analysis of the results obtained by applying the specified algorithms over the input data generated from the probability distributions mentioned earlier. The codes were written in Dev C++ 5.8.2 and analysis was performed using Minitab statistical Package.

The response (CPU time to run the code), the mean time in seconds is given in the tables 1-4 and relative performance plots are presented in figures 1-4. Average case analysis is performed directly on program run time to estimate the weight based statistical bound over a finite range by running computer experiments [4][5]. This estimate is called *empirical-O*[6]][7]. Time of an operation is taken as weight. Weighing permits collective consideration of all operations into a conceptual bound which we call a statistical bound in order to distinguish it from the count based mathematical bounds that are operation specific.

Sample size: The various inputs vary from a size of 1-10 lac which may be considered as reasonably large for practical data set.

2. RELATIVE PERFORMANCE ANALYSIS OF DIFFERENT ALGORITHMS

2.1 Discrete Uniform Distribution

Discrete Uniform distribution is characterised by single parameter k. In this experiment k has been fixed at 1000.

Table 1. Data for discrete uniform distribution

N	*Heap sort (HS)*	*K- Sort (KS)*	*Quick Sort (QS)*
100000	.0528	.0526	.03040
200000	.0998	.1842	.0966
300000	.1778	.4032	.1904
400000	.2436	.6340	.3096
500000	.2878	.9424	.4750
600000	.3472	1.3376	.6594
700000	.4028	1.775	.860
800000	.4622	2.886	1.092
900000	.5216	2.9168	1.301
1000000	.5828	3.5912	1.744

Some interesting results are seemed to emerge from the above table. For *N*< 200000, *quick* sort gives better performance as *heap* sort compared to other two algorithms but for *N*>=*30000*, the *heap* sort outperforms the *quick* sort. The reason for ill performance of *Quick* sort may is due to increase in number of ties in data set as *N* increases. As for as the *K-sort* is considered it gives the worst performance at this particular value of ***k***.

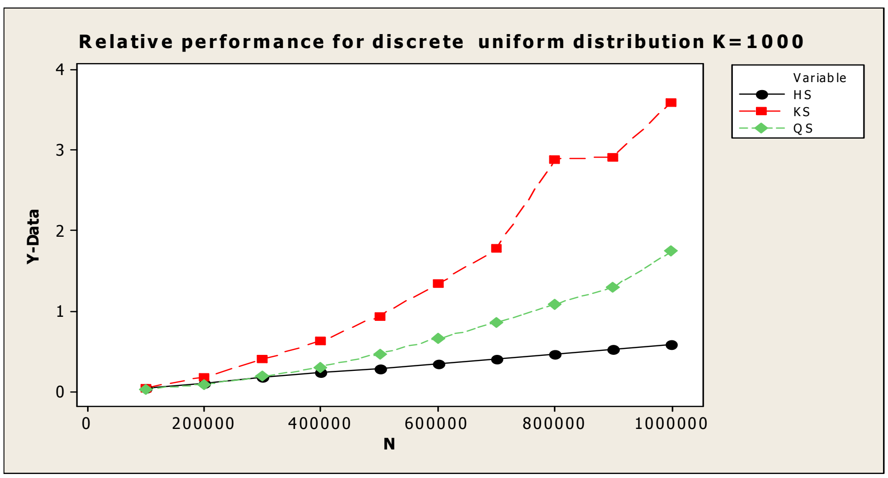

Figure 1. Relative plots of quick heap and K-sort algorithms (k=1000)

For Discrete Uniform inputs in average case *Quick sort*, and *K-Sort* exhibit a time complexity of $O_{emp}(N^2)$ whereas that of heap sort it is $O_{emp}(N\log_2 N)$ [2].

2.2 Binomial Distribution

The Binomial distribution has two parameters *m* and *p*, *m* being the number of independent Bernoulli trials and *p* the probability of success in each trial. The mean time given in table 2 was obtained by varying N between 100000 to 1000000 and fixing *m* and *p* at 1000 and 0.5 respectively and making several runs for the same input.

Table 2. Data for Binomial distribution

N	*Heap sort (HS)*	*K-sort (KS)*	*Quick sort (QS)*
100000	.4998	.6096	.2872
200000	1.0450	4.0198	1.1166
300000	1.6344	5.3716	2.46360
400000	2.24000	9.9072	4.3850
500000	2.8534	15.02379	6.8992
600000	3.4602	22.8832	9.8526
700000	4.0792	30.51688	13.5338
800000	4.74800	41.0869	17.66879
900000	5.3766	49.628720	22.0784
1000000	5.9798	62.4471	22.2676

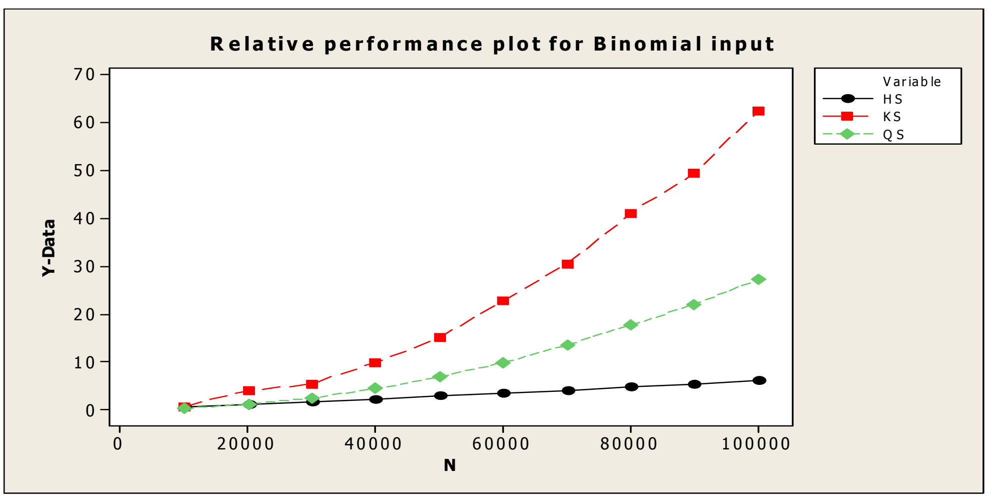

Figure 2. Relative plots of quick heap and K-sort algorithms (m=1000 and p=0.5)

The input data table (2) and relative performance plot (figure 2) supports the fact that on the average *K-sort* consumes more time as compared to other two algorithms for sorting array of same size. However for Binomial inputs *quick* sort and *K-sort* both confirms to $O_{emp}(N^2)$, whereas *heap* sort has $O(Nlog_2N)$ complexity.

2.3 Poisson Distribution

The Poisson distribution depends on the parameter λ. Lambda (which is both the mean and the variance) should not be large as this is the distribution of rare events. While performing the empirical analysis with Poisson inputs, its parameter, λ is fixed at 5.0

Table 3. Data for Poisson Input

N	*Heap sort (HS)*	*K-sort (KS)*	*Quick sort (QS)*
20000	.015	.1778	.0964
40000	.019	.7158	.3190
60000	.0342	1.7450	.7250
80000	.047	2.8082	1.2696
100000	.0636	4.5356	1.9696
120000	.0716	6.320	2.8502
140000	.0818	8.5972	3.8226
160000	.094	11.2604	4.979
18000	.110	14.736	6.3406
200000	.125	18.0549	7.8044

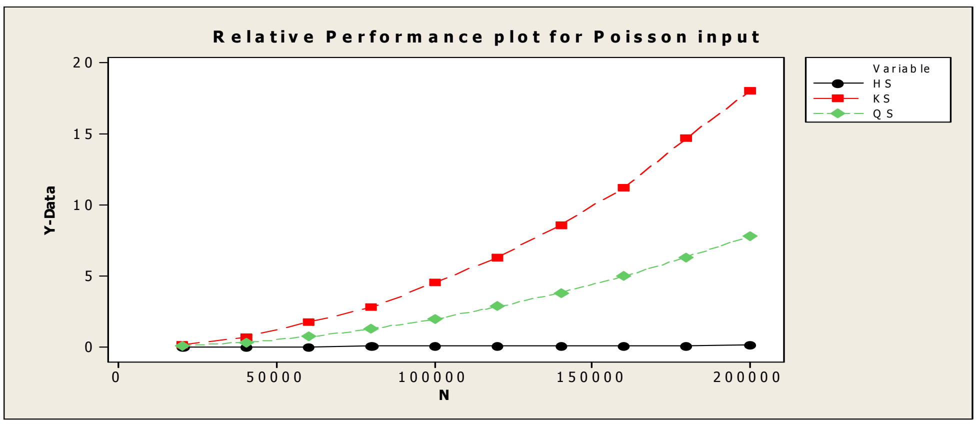

Figure 3. Relative plots of quick heap and K-sort algorithms (λ=5.0)

For Poisson inputs, *heap* sort gives the good performance as compared to other two algorithms. *Quick* sort and *K-sort* both have $O_{emp}(N^2)$ complexity whereas if we have a look on the three graph (figure 3), it is obvious that heap sort again confirms to $O_{emp}(Nlog_2N)$ complexity,

2.4 Uniform Continuous Distribution

Table 4 gives the mean execution time for the data simulated from uniform continuous distribution [0, θ].

Table 4. Data for Uniform Continuous Distribution

N	*Heap sort (HS)*	*K-sort (KS)*	*Quick sort (QS)*
100000	.055	.0310	.0340
200000	.11277	.0748	.062
300000	.1780	.1092	.0872

400000	.2400	.1468	.1064
500000	.3150	.2002	.1312
600000	.3842	.2624	.159
700000	.4533	.3092	.1908
800000	.5188	.3672	.219
900000	.902	.425	.2534
10000	1.0154	.4750	.2842

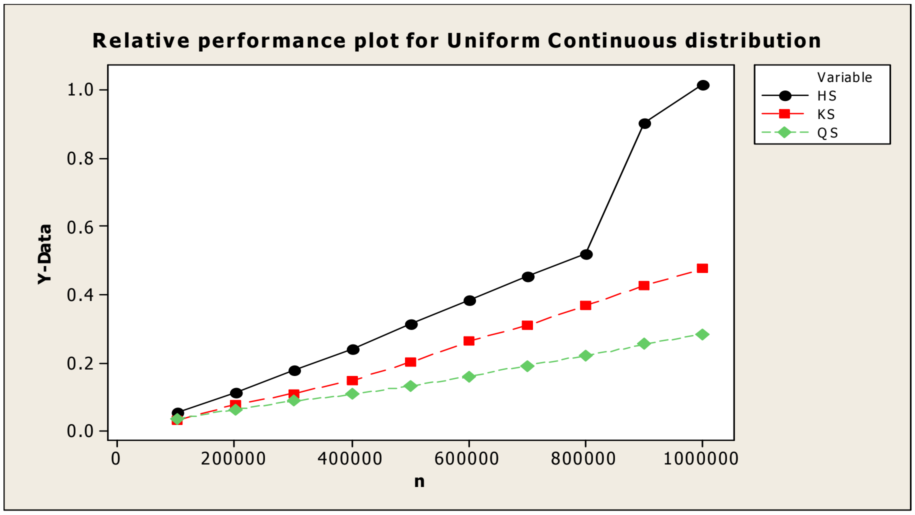

Figure 4. Relative plots of quick heap and K-sort algorithms [0-1]

Here the scenario has just changed. Quick sort outperforms the two algorithms in its performance. The *K-sort* for all values of *N* in selected range out performance the heap algorithm . However all the three algorithms suggest $O_{emp}(Nlog_2N)$ complexity.

3. STATISTICAL ANALYSIS OF DIFFERENT ALGORITHMS FOR DISCRETE DISTRIBUTIONS

In this section we test the hypothesis whether the average performance of the algorithms for different input distributions is same or not. For this we apply two way analysis of variance. A value of p = 0.358 greater than 0.05 as shown in table 5, is indicative of the fact that as for as their average performance is concerned there is no reason to differentiate between the three.

Table 5. Results of Two Way ANOVA

Source	DF	SS	MS	F	P
PRODIS	3	163.354	54.4514	2.52	0.155
SORTALG	2	52.955	26.4776	1.23	0.358
Error	6	129.602	21.6003		
Total		11345.911			

3.1 Parametric complexity

Parametric complexity is one of the important criterions for selection of an algorithm among the several algorithms, since besides the size of input, parameters of input distribution has direct effect on execution time of an algorithm. We have examined the parameterized complexity of the three algorithms for binomial inputs, the reason being that distribution has two parameters . A 3^2 factorial experiment with three repeated set of data elements at same combination of level of factors *m* and *p* has been employed. The results given in table 6 below reveal some interesting findings:

Table 6. Parameterized Complexity of heap sort, k-sort and quick sort

sources	df	Heap sort (HS)		K-sort (KS)		Quick sort (QS)	
		F	P	F	P	F	P
M	2	0.13	.876	3.85	.04	1.11	.351
P	2	0.15	.866	37.37	0.00	26.18	.000
MP	4	0.18	.948	1.62	.213	0.31	.867

The *F* and *P* values revealed that in case of *Heap* Sort the two parameters neither singularly nor jointly has any effect on sorting time while in case of *K-sort* though both the factors have independent effects on the complexity , probability of success being highly significant. While applying the *quick* sort algorithm, the number of trials (*m*) shows highly non significant effect while the probability of success *p* delivers significantly high effect on complexity .Thus the proper selection of input parameters can have rewarding effect on reducing the complexity of an algorithm.. For different values of *p*, the average execution time is given in the table 7 below

Table 7. optimal value of p m=5000, n=50000

P	K-sort	Quick sort
0.1	.188	.056
0.2	.144	.047
0.3	.126	.040
0.4	.127	.044
0.5	.115	.0438
06	.117	.0566
0.7	.125	.063
o.8	.1406	..072
0.9	.183	..072

Here we find that in both the cases initially the execution time decreases as the value of *p* is increased, at *p*=.5 , execution time is minimum and then again it goes on increasing. Thus the optimal value of *p* is .5. But as for the optimal value of m is concerned in case of k sort, as shown in the following table 8, the execution time decreases with increase in the value of *m* Thus in this case a high value of *m* is preferable.

Table 8. optimal value of m for K sort (p=.5, n=50000)

M	Ksort
2000	.174
3000	,1456
4000	.1275
5000	.0782
6000	.0704
7000	.0666
8000	.0644
9000	.061
10000	.056

4. CONCLUSIONS

The three sorting algorithms, *heap* sort, *K-sort* and *quick* sort though theoretically deliberating to same complexity $O(Nlog_2N)$ has been supported by the statistical analysis in section 3. We have no evidence against rejection of the hypothesis of homogeneity of algorithms as far as their average performance is considered. But unfortunately in worst case quick sort and K-sorts have complexity $O(N^2)$ than *heap* sort which exhibits a complexity of $O(Nlog_2N)$ since we have the relation that $O(Nlog_2N)<O(N^2)$.

However as far as *Empirical-O* estimates are considered , *quick* sort for N less than 200000 gives better performance for some discrete distributions such as Binomial and Uniform distribution while for N>300000 *heap* sort is best, *K-sort* for these distributions does not work good. But sorting an array generated randomly (not generated from any standard probability distribution, *K-sort* works good. It can sort an array of size never greater than 7000000 in less time than heap sort [2].

For continuous uniform distribution, *quick* sort gives the good performance as compared to other two. This result is quite expected as in case of a continuous distribution the probability of getting similar valued elements (ties) is theoretically zero. It is well known through various results [8] that *quick* sort behaves exceptionally good in such cases. Whereas in the very same scenario *heap* sort is expected to perform relatively poor (however the time complexity of *heap* sort remains of the order of $Nlog_2N$) [9].

The different behaviour of the algorithms to input data can be supplemented by the parameterised complexity analysis since the true potential of an algorithm is related to parameters of the input distribution. As far as the parameterised complexity of *heap* sort is considered, it is in favour of its worst case complexity which is less than the other two algorithms. The reason being obvious as execution time does not depend on the binomial parameters. But in case of quick sort the parameter *p* has highly significant effect on execution time. Though the two parameters m and p have independent effects on complexity in case of K sort, but we find that parameter m has a very little effect where as p has highly significant . In both the cases of K sort and Quick sort, the complexity is minimum at p=.5 but a high value of p is preferable to reduce the complexity while using the K-sort algorithm. Thus proper selection of input parameters can have rewarding effect on reducing complexity of an algorithm.

REFERENCES

[1] Sundararajan, KK & Chakraborty S, (2007) "A New sorting algorithm" Journal of Applied Ma1thematics and Computation 188, pp 1037-1041.

[2] Sundararajan, KK, Pal, M, Chakarborty ,S., & Mahanti, NC, (2013) "K-Sort: A New Sorting Algorithm that beats Heap Sort for n ≤ 70 lakhs".

[3] Singh, N K, Chakraborty, S, and Mallick, DK, (2014) "A Statistical Peek into Average Case Complexity", IJEDPO, Vol. 4, No 2, Int. J. on Recent Trends in Engineering and Technology 8(1), 64-67.

[4] Fang, KT, Li, R, and Sudjianto, (2000) Design and Modelling of computer Experiments, Taylor & Francis Group, Boca Raton.

[5] Sacks, J, Welch,W, Mitchel, T, and Wynn, (1989) "Design and Analysis of Computer Experiments", Statistical Science 4.

[6] Chakraborty , S, & Sourabh, SK, (2010) A Computer Oriented Approach to Algorithmic Complexity, Lambert Academic Publishing.

[7] Chakraborty, S, Modi, N and Panigrahi, S, (2009) "Will the weight based Statistical Bounds Revolutionize the It?", International Journal of Computational Cognition 7(3).

[8] Singh, NK & Chakraborty, S, (2012) "Smart Sort: A Fast, Efficient and Robust Sorting Algorithm", IJACMS, ISSN 2230-9624, Vol 3, Issue 4, pp 487-493.

[9] Sourabh, SK ,and Chakraborty, S, (2009) "Empirical Study on the Robustness of Average Complexity and Parameterized Complexity Measure for Heap sort Algorithm", International Journal of Computational Cognition.Vol.7, No.4.

Authors

Anchala Kumari is a Professor in the department of statistics, Patna University, India. Her research area includes: Operations research, Design of Experiments and Computer Programming.

Niraj Kumar Singh is a Teaching Cum Research Fellow at department of CSE, BIT Mesra, India. His research interest includes: Design and Analysis of Algorithms and Algorithmic Analysis through Statistical bounds.

Soubhik Chakraborty is a Professor in the department of Mathematics, BIT Mesra, India. His research interest includes: Music Analysis, Algorithmic Complexity and Statistical Computing.