Hindawi Publishing Corporation
International Journal of Analysis
Volume 2014, Article ID 840432, 10 pages
http://dx.doi.org/10.1155/2014/840432

Research Article

A New Look at Worst Case Complexity: A Statistical Approach

Niraj Kumar Singh,[1] Soubhik Chakraborty,[2] and Dheeresh Kumar Mallick[1]

[1] *Department of Computer Science & Engineering, B.I.T. Mesra, Ranchi 835215, India*
[2] *Department of Applied Mathematics, B.I.T. Mesra, Ranchi 835215, India*

Correspondence should be addressed to Niraj Kumar Singh; niraj_2027@yahoo.co.in

Received 5 June 2014; Revised 17 September 2014; Accepted 18 September 2014; Published 29 December 2014

Academic Editor: Baruch Cahlon

We present a new and improved worst case complexity model for quick sort as $y_{\text{worst}}(n, t_d) = b_0 + b_1 n^2 + g(n, t_d) + \varepsilon$, where the LHS gives the worst case time complexity, n is the input size, t_d is the frequency of sample elements, and $g(n, t_d)$ is a function of both the input size n and the parameter t_d. The rest of the terms arising due to linear regression have usual meanings. We claim this to be an improvement over the conventional model; namely, $y_{\text{worst}}(n) = b_0 + b_1 n + b_2 n^2 + \varepsilon$, which stems from the worst case $O(n^2)$ complexity for this algorithm.

1. Introduction

Sometimes theoretical results on algorithms are not enough for predicting the algorithm's behavior in real time implementation [1]. From research in parameterized complexity, we already know that for certain algorithms, such as sorting, the parameters of the input distribution must also be taken into account, apart from the input size, for a more precise evaluation of time complexity of the algorithm in question [2, 3]. Based on the results obtained, we present a new and improved worst case complexity model for quick sort as

$$y_{\text{worst}}(n, t_d) = b_0 + b_1 n^2 + g(n, t_d) + \varepsilon, \tag{1}$$

where the LHS gives the worst case time complexity, n is the input size, t_d is the frequency of sample elements, and $g(n, t_d)$ is a function of both the input size n and the parameter t_d. The rest of the terms arising due to linear regression have usual meanings. We claim this to be an improvement over the conventional model; namely, $y_{\text{worst}}(n) = b_0 + b_1 n + b_2 n^2 + \varepsilon$, which stems from the worst case $O(n^2)$ complexity for this algorithm. It is important to note that our results change the order of theoretical $O(n^2)$ complexity of this algorithm as we get $y_{\text{worst}}(n, t_d) = O(n^3)$ complexity in some situations.

This new model in our opinion can be a guiding factor in distinguishing this algorithm from other sorting algorithms of similar order of theoretical average and/or worst case complexities. The dependence of basic operation(s) on the response is more prominent for discrete distributions rather than continuous ones for the probability of a tie is zero in a continuous case. However, presence of ties and their relative positions in the array is crucial for discrete cases. And this is precisely where the parameters, apart from n characterizing the size of the input, of the input distribution come into play.

We make a statistical case study on the robustness of theoretical worst case complexity measures for quick sort [4] over discrete uniform distribution inputs. The uniform distribution input parameters are related as $n = t_d * k$. The runtime complexity of quick sort varies from $O(n\log_2 n)$ to $O(n^2)$ depending on the extent of tied elements present in a sample. For example, complexity is $n\log_2 n$ when all keys are distinct and $O(n^2)$ when all keys are similar [5]. Apart from this result an important observation with respect to average time complexity is made by Singh et al. [6], which claims

quadratic average case complexity of quick sort program under universal data set. This is true especially for certain models where tie-density is a positive linear function over n values. With these observations, it would be interesting to know the behavior of quick sort program when the linear growth of t_d is replaced by some superlinear function. This interest is a major motivation towards this research article.

Just as in this case quick sort is found to be worse than it is; the reverse is also possible in other algorithms. That is to say, an algorithm can perform better than what a worst case mathematical bound says. In this case the bound becomes conservative [7]. A certificate on the level of conservativeness can be provided using a statistical analysis only. *Empirical-O*, the statistical bound estimate, will be pointing to some other bound lower than the mathematical bound obtained by theoretical analysis. The difference provides the desired certificate. For a detailed discussion on *empirical-O* reader is suggested to see [8].

It is well known that quick sort's performance is dependent on the underlying pivot selection algorithm for a proper pivot selection that greatly reduces the chances of getting the worst case instances. As discussed above, the worst case complexity measures can be conservative. That is, for an arbitrary algorithm with $y(n) = O(n^k)$ worst case complexity, in a finite range setup, we can expect for an $y(n) = O(n^{k-\varepsilon})$ complexity, where $\varepsilon > 0$. However when other input parameter(s) (t_d in our case) are also taken into account we come up with $y(n, t_d) = O(n^{k+\varepsilon})$ worst case complexity, which is a novel finding.

Justifying the Choice of Algorithm. There are many versions of quick sort. Industrial implementations of quick sort typically include heuristics that protect it against $O(n^2)$ performance when keys are similar. With respect to the quick sort, the question of choosing a proper pivot selection algorithm is more relevant in average case complexity measures, as its $O(n\log_2 n)$ average case complexity itself is not robust [5]. The small (but nonzero) probability of getting the worst case instances is often cited as the reason for overall good performance of quick sort. This is true especially for random continuous distribution inputs. This research article is a study on finding the worst case behavior of both the naïve and randomized versions of quick sort algorithm.

The Organization of the paper. The paper is organized as follows. Section 2 gives analysis of quick sort using statistical bound estimate. Under Section 2 Section 2.1 gives analysis for sorted data sequences. Section 2.2 gives the analysis for random data sequences with three case studies. Section 2.3 gives justification for worse than $O(n^2)$ complexity of quick sort. Section 3 gives conclusion.

2. Analysis of Quick Sort Using Statistical Bound Estimate

Our statistical adventure explores the worst case behavior of the well-known standard quick sort algorithm [4] as a case study. The worst case analysis was done by directly working on program run time to estimate the weight based statistical bound over a finite range by running computer experiments [9, 10]. This estimate is called *empirical-O*. Here time of an operation is taken as its weight. Weighing allows collective consideration of all operations, trivial or nontrivial, into a conceptual bound. We call such a bound a statistical bound opposed to the traditional count based mathematical bounds which is operation specific. Since the estimate is obtained by supplying numerical values to the weights obtained by running computer experiments, the credibility of this bound estimate depends on the design and analysis of computer experiments in which time is the response. It is suggested for the interested reader to see [11, 12] to get more insight into statistical bounds and *empirical-O*.

This section includes the empirical results obtained for worst case analysis of quick sort algorithm. The samples are generated randomly, using a random number generating function, to characterize discrete uniform distribution models with k as its parameter. Our sample sizes, for random data sequences, lie in between $5 * 10^5$ and $10 * 10^6$. The discrete uniform distribution depends on the parameter k $[1, \ldots, k]$, which is the key to decide the range of sample obtained.

Most of the mean time entries (in seconds) are averaged over 500 trial readings. These trial counts, however, should be varied depending on the extent of *noise* present at a particular sample size value. As a rule of thumb, the greater the *noise* at each point of n is, the more the numbers of observations should be.

The interpretations made for the various statistical data are guided by [13].

System Specification. All the computer experiments were carried out using *PENTIUM* 1600 MHz processor and 512 MB RAM. Statistical models/results are obtained using *Minitab-16* statistical package. The standard quick sort is implemented using "C" language by the authors themselves. It should be understood that although program run time is system dependent, we are interested in identifying *patterns* in the run time rather than run time itself. It may be emphasized here that statistics is the science of identifying and studying *patterns* in numerical data related to some problem under study.

2.1. Analysis for Sorted Data Sequence. This section includes the empirical results for sorted data sequences. The samples thus generated consist of all distinct elements (at least theoretically). The program runtime data obtained for sorted sequences is fitted for a quadratic model. The regression analysis result is given in Box 1. With a very significant *t-value* (194.92) of quadratic term, the regression analysis statistic strongly supports a quadratic model. The quadratic model goodness is further tested through cubic fit for the same runtime data set.

Next the very same program runtime data is fitted for a cubic model. The regression analysis result is given in Box 2. With a value of 23.75 the *t statistic* for the quadratic term is significantly higher than other terms in the obtained regression model. Remarkably the statistical significance of cubic term is very weak compared to other terms, hence liable to be discarded. The R^2 value is the maximum with a very

```
The regression equation is
y = 0.101 + 0.00151 n + 0.00250 n^2
Predictor         Coef      SE Coef        T       P
Constant       0.10074      0.01636     6.16   0.000
n            0.0015140    0.0009823     1.54   0.158
n^2          0.00250149   0.00001283   194.92   0.000
S = 0.0117210    R-Sq = 100.0%    R-Sq(adj) = 100.0%
PRESS = 0.00186676    R-Sq(pred) = 100.00%
Analysis of Variance
Source           DF        SS      MS           F       P
Regression        2   133.092  66.546   484383.28   0.000
Residual Error    9     0.001   0.000
Total            11   133.093
Source  DF    Seq SS
n        1   127.872
n^2      1     5.220
```

Box 1: Regression analysis: y versus n and n^2 (sorted data sequence).

```
The regression equation is
y = 0.0967 + 0.000002 n + 0.000000 n^2 + 0.000000 n^3
Predictor         Coef      SE Coef        T       P
Constant       0.09668      0.03490     2.77   0.024
n           0.00000197   0.00000356     0.55   0.595
n^2         0.00000000   0.00000000    23.75   0.000
n^3         0.00000000   0.00000000     0.13   0.897
S = 0.0124181    R-Sq = 100.0%    R-Sq(adj) = 100.0%
PRESS = 0.00225037    R-Sq(pred) = 100.00%
Analysis of Variance
Source           DF        SS      MS           F       P
Regression        3   133.092  44.364   287685.21   0.000
Residual Error    8     0.001   0.000
Total            11   133.093
Source  DF    Seq SS
n        1   127.872
n^2      1     5.220
n^3      1     0.000
```

Box 2: Regression analysis: y versus n, n^2, and n^3 (sorted data sequence).

small standard error value. With insignificant *t value* (0.13) of the cubic term, this statistic analysis result suggests a *strong quadratic model* for the given data set.

2.2. Analysis for Random Data Sequences. Next the quick sort program runtime data is analyzed for two carefully designed random data sequences whose elements correspond to points on the third degree polynomials shown in Figures 1 and 3, respectively. Due to uniform distribution of sample elements, over increasing n values, a linear growth in k results in a linear growth of t_d values as well and vice versa. In any such case either k or t_d has to be constant. As a change, if growth rate of t_d is made super linear we get random samples in which neither k nor t_d remain constant. The empirical results for these random sequences are given in Boxes 3–7.

2.2.1. Worst Case Analysis of Naïve Quick Sort (Case Study 1). As our first case study with random data sequences we have analyzed the worst case complexity measures for the inputs in the range $5 * 10^5$–$50 * 10^5$.

The points on horizontal axis in Figures 2(a)–2(c) and 2(e) correspond to points on the third degree polynomial shown in Figure 1. It can be seen in Figure 2(a) that the runtime data when fitted to a quadratic model gives an underfit. This fit gets improved significantly when the fitted model is changed to a cubic model of type $y = b_0 + b_1 n + b_2 n^2 + b_3 n^3$. We get a more improved fit when the cubic model is replaced by a fourth degree polynomial. We are not interested in higher order models such as fifth or sixth degree polynomials to avoid the problem of overfitting as we wish to catch the general trend of the population rather than a fit

```
The regression equation is
y = 17.0 − 2.33 n + 0.0608 n^2
Predictor         Coef      SE Coef        T       P
Constant        17.045        9.636     1.77   0.120
n              −2.3324       0.8049    −2.90   0.023
n^2            0.06082      0.01426     4.26   0.004
S = 8.19305    R-Sq = 87.7%    R-Sq(adj) = 84.1%
PRESS = 1809.78    R-Sq(pred) = 52.45%
Analysis of Variance
Source           DF       SS      MS         F       P
Regression        2    3336.0  1668.0     24.85   0.001
Residual Error    7     469.9    67.1
Total             9    3805.9
Source  DF    Seq SS
n        1    2115.3
n^2      1    1220.7
```

Box 3: Regression analysis result for y versus n and n^2 on random data sequences ($5 * 10^5 \leq n \leq 50 * 10^5$).

```
The regression equation is
y = −13.5 + 3.09 n − 0.174 n^2 + 0.00285 n^3
Predictor         Coef      SE Coef        T       P
Constant       −13.518        6.996    −1.93   0.101
n                3.089        1.049     2.95   0.026
n^2            −0.17428      0.04326   −4.03   0.007
n^3           0.0028497    0.0005188     5.49   0.002
S = 3.60446    R-Sq = 98.0%    R-Sq(adj) = 96.9%
PRESS = 684.368    R-Sq(pred) = 82.02%
Analysis of Variance
Source           DF       SS       MS       F       P
Regression        3   3728.0   1242.7   95.65   0.000
Residual Error    6     78.0     13.0
Total             9   3805.9
Source  DF  Seq SS
n        1  2115.3
n^2      1  1220.7
n^3      1   391.9
```

Box 4: Regression analysis result for y versus n, n^2, and n^3 on random data sequences ($5 * 10^5 \leq n \leq 50 * 10^5$).

by forcing a polynomial to pass through all the input points. Our graphical observation is next verified with more rigorous statistical results given with Boxes 3–5.

Regression Analysis and ANOVA (Analysis of Variance) Results. The program runtime data obtained for random data sequences corresponding to curve in Figure 1 is fitted for a quadratic model. The corresponding regression analysis result is given in Box 3. The significant *t-value* of quadratic term suggests for a quadratic complexity. However the R^2 value is relatively low and the standard error is high.

The Cubic Model as a Test Of Quadratic Goodness of Fit. A test of quadratic goodness of fit is performed by fitting a cubic model to the same program runtime data. From the regression and ANOVA table (Box 4), this cubic model looks much better than the earlier quadratic model. The R^2 is much higher (98.0 against 87.7), the standard error is smaller (3.60446 against 8.19305), but, more importantly, the coefficient of the cubic term is highly significant. Interestingly the cubic term is statistically more significant than the quadratic term (5.49 against −4.03)!

Verifying the Cubic Complexity through "Box-Cox Transformation". In order to get a better fit of the model we prefer the response to be transformed. In general, transformations are used for three purposes: stabilizing response variance, making the distribution of the response variable closer to the normal distribution, and improving the fit of the model to the data [14]. We perform transformation to simultaneously accomplish more than one of these objectives. The power family of transformations $y^* = y^\lambda$ is very useful, where λ is the parameter of the transformation to be determined.

General Regression Analysis: y versus $n, n^{1.5}$

Box-Cox transformation of the response with specified lambda = 0.5

Regression Equation

$y^{\wedge}0.5 = 2.39481 - 4.2737e - 006\ n + 2.34136e - 009\ n^{1.5}$

Coefficients

Term	Coef	SE Coef	T	P
Constant	2.39481	0.833261	2.87402	0.024
n	−0.00000	0.000001	−3.95514	0.005
$n^{1.5}$	0.00000	0.000000	5.30621	0.001

Summary of Model

S = 0.606531 R-Sq = 95.26% R-Sq(adj) = 93.90%

PRESS = 9.71668 R-Sq(pred) = 82.10%

Analysis of Variance

Source	DF	Seq SS	Adj SS	Adj MS	F	P
Regression	2	51.7072	51.7072	25.8536	70.2774	0.0000233
n	1	41.3493	5.7548	5.7548	15.6432	0.0054957
$n^{1.5}$	1	10.3579	10.3579	10.3579	28.1558	0.0011158
Error	7	2.5752	2.5752	0.3679		
Total	9	54.2824				

Fits and Diagnostics for Unusual Observations for Transformed Response

Obs	$y^{\wedge}0.5$	Fit	SE Fit	Residual	St Resid	
10	8.05841	7.20352	0.462707	0.854895	2.18002	R

Fits for Unusual Observations for Original Response

Obs	y	Fit	
10	64.938	51.8907	R

R denotes an observation with a large standardized residual.

Box 5: Regression analysis result for y versus n and $n^{1.5}$ (Box-Cox transformation) on random data sequences ($5 * 10^5 \le n \le 50 * 10^5$).

The regression equation is

$y = -480 + 22.0\ n - 0.328\ n^2 + 0.00161\ n^3$

Predictor	Coef	SE Coef	T	P
Constant	−479.8	119.2	−4.03	0.005
n	22.001	4.991	4.41	0.003
n^2	−0.32799	0.06795	−4.83	0.002
n^3	0.0016093	0.0003015	5.34	0.001

S = 2.96191 R-Sq = 97.2% R-Sq(adj) = 96.0%

PRESS = 434.729 R-Sq(pred) = 80.14%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	3	2127.97	709.32	80.85	0.000
Residual Error	7	61.41	8.77		
Total	10	2189.38			

Source	DF	Seq SS
n	1	1254.24
n^2	1	623.73
n^3	1	249.99

Box 6: Regression analysis result for y versus n, n^2, and n^3 on random data sequences ($50 * 10^5 \le n \le 100 * 10^5$).

Below we provide statistical results for *Box-Cox* transformation of the response variable for λ (lambda) = 0.5. The detailed result of *Box-Cox* transformation is provided in Box 5.

Regression Equation is as follows:

$$y^{0.5} = 2.39481 - 4.2737e - 006n + 2.34136e - 009n^{1.5}. \quad (2)$$

From the above regression equation we have $y^{0.5} = O_{\text{emp}}(n^{1.5})$, as $n^{1.5}$ term is statistically the most significant. This implies $y = O_{\text{emp}}(n^{1.5})^{1/0.5} = O_{\text{emp}}(n^3)$ complexity model. This result once again refutes the theoretical $O(n^2)$ worst case complexity of quick sort algorithm for certain data patterns.

```
The regression equation is
Y = −8.38 + 1.90 n − 0.107 n^2 + 0.00174 n^3
Predictor          Coef      SE Coef        T       P
Constant         −8.378        4.456    −1.88   0.109
n                1.9025       0.6679     2.85   0.029
n^2            −0.10675      0.02756    −3.87   0.008
n^3          0.0017411    0.0003305     5.27   0.002
S = 2.29581    R-Sq = 97.8%    R-Sq(adj) = 96.7%
PRESS = 278.092    R-Sq(pred) = 80.37%
Analysis of Variance
Source           DF        SS       MS       F       P
Regression        3   1385.32   461.77   87.61   0.000
Residual Error    6     31.62     5.27
Total             9   1416.95
Source  DF    Seq SS
n        1    789.97
n^2      1    449.04
n^3      1    146.31
R denotes an observation with a large standardized residual.
Durbin-Watson statistic = 1.75322
```

Box 7: Regression analysis for y versus n, n^2, and n^3 (randomized quicksort) on random data sequences ($5 * 10^5 \leq n \leq 50 * 10^5$).

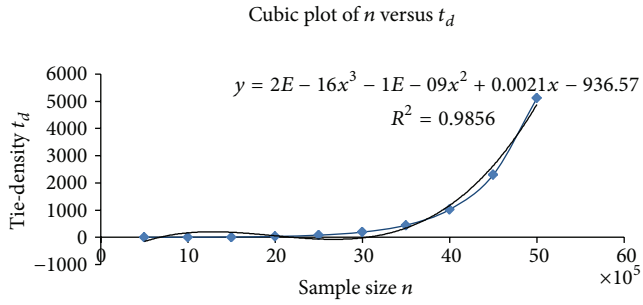

Figure 1: Cubic plot of n versus t_d ($5 * 10^5 \leq n \leq 50 * 10^5$).

2.2.2. Worst Case Analysis of Naïve Quick Sort (Case Study 2). We investigate the behavior of quick sort program for yet larger data sets whose horizontal axis elements correspond to points on Figure 3. In this experimental setup the samples vary from 5 to 10 million of data in terms of their size. From the regression and ANOVA table (Box 6), this new cubic model looks much better than the quadratic model in Box 3. The R^2 is much higher (97.2 against 87.7), the standard error is still smaller (2.96191 against 8.19305), but, more importantly, the coefficient of the cubic term is highly significant. In fact the cubic term is statistically more significant than the quadratic term. These statistics indicate that a cubic model better describes the sample experimental data, as it does not impart overfit to the complexity data. The smaller *PRESS* statistic for the cubic models in Boxes 4 and 6 is also favorable. See Figure 4 to get an insight into super quadratic nature of n versus y curve. These observations lead to a conjecture that the quick sort worst case complexity is $y_{\text{worst}}(n, t_d) = b_0 + b_1 n^2 + g(n, t_d) + \varepsilon = O_{\text{emp}}(n^3)$. It is important to note that our results do change the order of theoretical $O(n^2)$ complexity of this algorithm.

The statistical analysis in both case studies suggests a super quadratic worst case complexity for quick sort algorithm provided y is a function of both n and t_d.

2.2.3. Worst Case Analysis of Randomized Quick Sort (Case Study 3). As our third case study with random data sequences we have analyzed the worst case complexity measures of randomized quick sort for the inputs in the range $5 * 10^5$–$50 * 10^5$. Apart from the version of quick sort used in this case study other input requirements are similar to that in Section 2.2.1. The response variable y in this case corresponds to points in Figure 5.

The runtime data obtained for randomized quick sort is fitted to a cubic model. The corresponding result is given in Box 7. It can be seen that with a *t statistic* of 5.27 the cubic term is statistically more significant than the quadratic term. The standard error ($S = 2.29581$) of the model thus obtained is relatively small. Also the *PRESS* statistic (278.092) strongly supports a cubic complexity for the observed data. Compare these values against the corresponding values in Box 3. The worst case complexity thus can be expressed as

$$y_{\text{worst}}(n, t_d) = b_0 + b_1 n^2 + g(n, t_d) + \varepsilon = O_{\text{emp}}(n^3). \quad (3)$$

2.3. Justification for Worse than $O(n^2)$ Complexity. It is well known that runtime of quick sort depends on the number of equal keys present in the sample [5]. With fixed tie-density value, in a finite but reasonably wide range setup, k is a linear function over input parameter n. Similarly for fixed k, the value t_d grows linearly (see Figures 6(a) and 6(b) for an insight) with respect to input parameter n. For a fixed sample size the runtime of quick sort is an increasing function

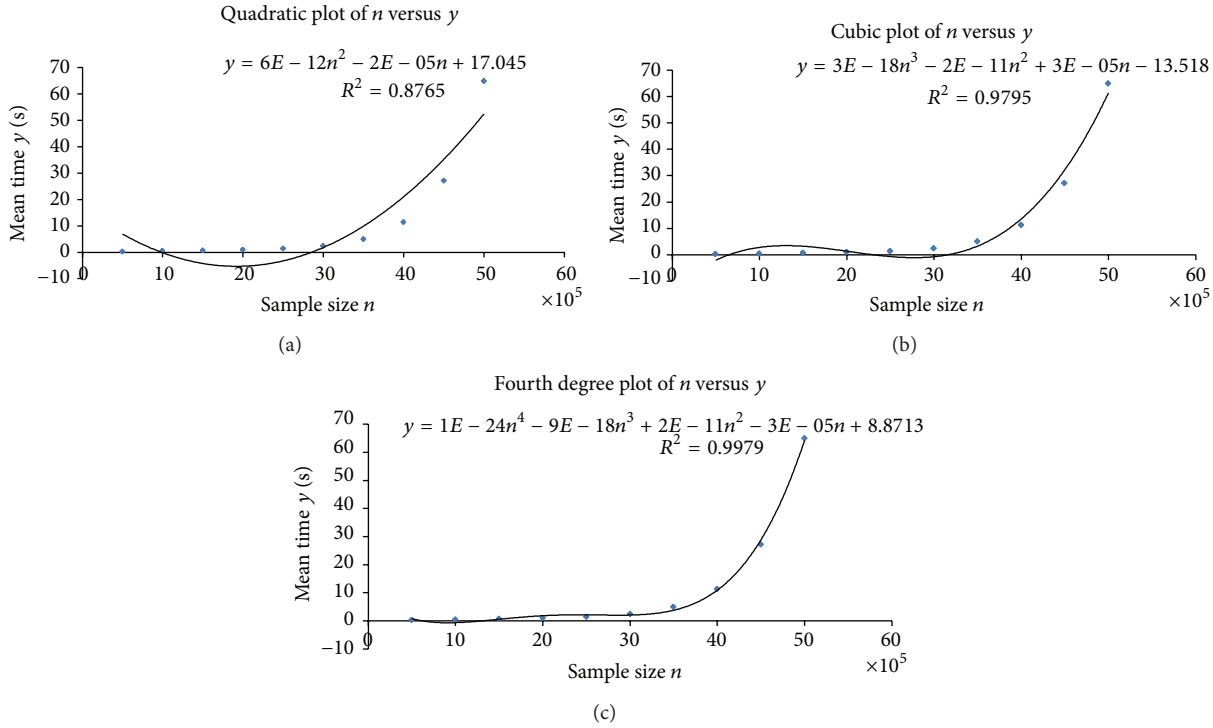

FIGURE 2: (a) Second degree polynomial curve of n versus y ($5 * 10^5 \leq n \leq 50 * 10^5$). (b) Third degree polynomial curve of n versus y ($5 * 10^5 \leq n \leq 50 * 10^5$). (c) Fourth degree polynomial curve of n versus y ($5 * 10^5 \leq n \leq 50 * 10^5$).

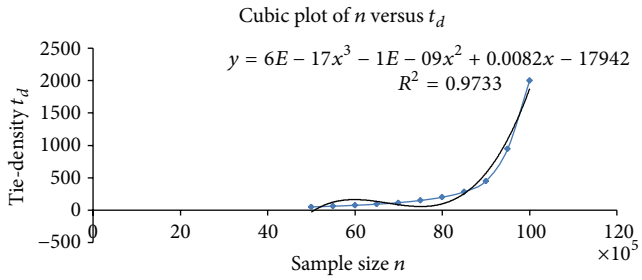

FIGURE 3: Cubic plot of n versus t_d ($5 * 10^6 \leq n \leq 1 * 10^7$).

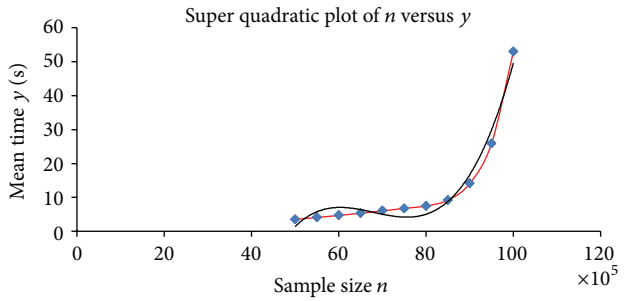

FIGURE 4: Super quadratic plot of n versus y ($5 * 10^6 \leq n \leq 1 * 10^7$).

over the input parameter t_d. An important observation as that made in [6] is that, for a linear growth in t_d (of course k is to be kept constant to satisfy $n = t_d * k$), the time complexity

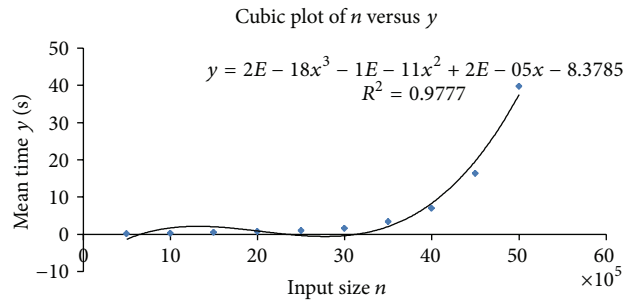

FIGURE 5: Cubic plot of n versus y ($5 * 10^5 \leq n \leq 50 * 10^5$) for randomized quick sort.

is quadratic. With this observation, it is interesting to know the behavior of quick sort program when this linear growth is replaced by some steeper function. This interest is a major motivation towards this research article. In this requirement, due to nonlinear nature of t_d, even k is not a constant but rather a decreasing function over input parameter n.

3. Conclusions

We conclude this paper with the following remarks.

Worst case analysis is termed as a useful science, since the worst case bounds give a sense of guarantee against the nonfavorable cases. But is the science as useful as it is projected by the computer scientists? Far from it. Worst case

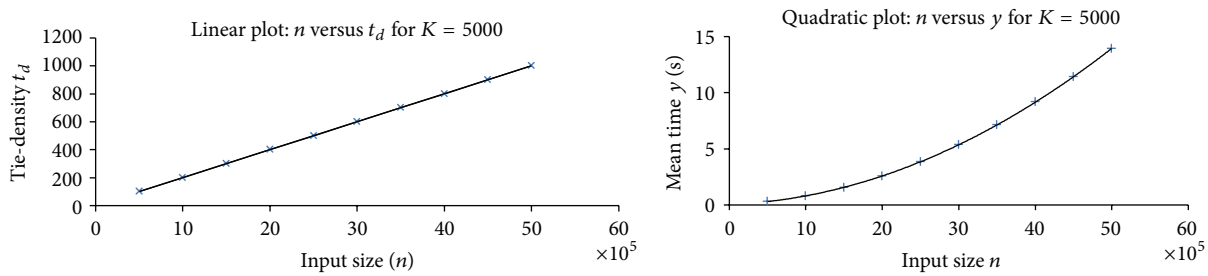

Figure 6: (a) Linear growth of t_d value. (b) Quadratic plot for linear growth of t_d value.

```
unsigned int iseed = (unsigned int)time(NULL);
   srand (iseed);
   A=(int *)malloc(n*sizeof(int));
   printf("Get the elements in random order\n");
   for( i=1; i<= n; i++)
      {
         r = (float) rand() / RAND_MAX;
         *(A+i)=(int)(k*r)+1;
      }
```

Algorithm 1: The "C" code snippet for simulating discrete uniform distribution inputs.

Table 1

Term →	Constant	Linear (n)	Quadratic (n^2)	Cubic (n^3)
Multiplier →	1	10^{-5}	10^{-10}	10^{-15}

bounds are often conservative in the sense that algorithms can and do perform better than what the bound says. No certificate is given on the level of conservativeness [8]. The present finding adds a new chapter to this criticism in that even the guarantee provided against the so-called *worst situations* can fail as we have discovered in the case of quick sort.

We hope that our findings will encourage other researchers to investigate and discover similar contradictions between theory and practice in algorithmic complexity so that ultimately the correct message is conveyed to the scientific community.

Note. Due to the limitations of the underlying software we have done mapping while analyzing the various runtime data using MINITAB software. Hence in order to obtain the correct coefficients use Table 1 multipliers for the various terms in the models obtained in Box 1 through Box 7.

Appendix

The "C" Code Implementations

See Algorithms 1, 2, and 3.

```
// System Specification: as given in main text
int main (int argc, char *argv[])
{
.....................
   quicksort( A, low, high);
.....................
}
quicksort(int A[], int low, int high)
{
   int p;
   if(low>=high)
   return;
   p=partition(A,low,high);
   quicksort(A,low,p-1);
   quicksort(A,p+1,high);
}
int partition(int A[], int low, int high)
{
   int temp,key,i,j;
   key = A[low];
   i=low+1;
   j=high;
   while(i <= j)
   {
            while(A[i] <= key && i<=j)
            {
             i++;
            }
            while(A[j] > key && i <= j)
            {
             j- -;
            }
            if(i <= j)
            {
                 temp=A[i];
                 A[i]=A[j];
                 A[j]=temp;
            }
    }
    A[low]=A[j];
    A[j]=key;
    return(j);
}
```

Algorithm 2: The "C" code implementation of the naive (first element pivot) quick sort.

```
// System specification:
//Processor: Intel(R) Core (TM) i5-3210 CPU @ 2.50 GHz 2.50 GHz
// Installed Memory (RAM): 4.00 GB
// System Type: 64-bitoperating system
int main (int argc, char *argv[])
{
   ......................
   quicksort( A, low, high);
   ......................
}
void quicksort(int A[], int low, int high)
{
   int p;
   if(low>=high)
   return;
   p=Random_partition(A,low,high);
   quicksort(A,low,p-1);
   quicksort(A,p+1,high);
}
int Random_partition(int A[], int low, int high)
{
   int temp,key,i,j;
   key = A[rand()%(high-low+1)+low];
   i=low+1;
   j=high;
   while(i <= j)
   {
            while(A[i] <= key && i<=j)
            {
             i++;
            }
            while(A[j] > key && i <= j)
            {
             j- -;
            }
            if(i <= j)
            {
                temp=A[i];
                A[i]=A[j];
                A[j]=temp;
            }
   }
     A[low]=A[j];
     A[j]=key;
     return(j);
}
```

Algorithm 3: The "C" code implementation of the randomized quick sort.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] D. S. Johnson, *A Theoretician's Guide to the Experimental Analysis of Algorithms*, 2001, http://www.researchatt.com/~dsj/.

[2] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer, New York, NY, USA, 1999.

[3] H. M. Mahmoud, *Sorting: A Distribution Theory*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, New York, NY, USA, 2000.

[4] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, pp. 10–15, 1962.

[5] R. Sedgewick, "Quicksort with equal keys," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 240–267, 1977.

[6] N. K. Singh, S. Chakraborty, and D. K. Mallick, "A statistical peek into average case complexity," *International Journal of Experimental Design and Process Optimisation*. In press.

[7] S. Chakraborty and S. K. Sourabh, "On why an algorithmic time complexity measure can be system invariant rather than system independent," *Applied Mathematics and Computation*, vol. 190, no. 1, pp. 195–204, 2007.

[8] S. Chakraborty, D. N. Modi, and S. K. Panigrahi, "Will the weight-based statistical bounds revolutionize the IT," *International Journal of Computational Cognition*, vol. 7, no. 3, pp. 16–22, 2009.

[9] K.-T. Fang, R. Li, and A. Sudjianto, *Design and Modeling for Computer Experiments*, Chapman & Hall, New York, NY, USA, 2006.

[10] J. Sacks, W. Weltch, T. Mitchel, and H. Wynn, "Design and analysis of experiments," *Statistical Science*, vol. 4, no. 4, pp. 409–423, 1989.

[11] S. Chakraborty and S. K. Sourabh, *A Computer Experiment Oriented Approach to Algorithmic Complexity*, Lambert Academic, 2010.

[12] N. K. Singh and S. Chakraborty, "Partition sort and its empirical analysis," in *Proceedings of the International Conference on Computational Intelligence and Information Technology (CIIT '11)*, vol. 250 of *Communications in Computer and Information Science*, pp. 340–346, 2011.

[13] P. Mathews, *Design of Experiments with MINITAB*, New Age International, New Delhi, India, 2010.

[14] D. C. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons, New York, NY, USA, 8th edition, 2013.