Empirical Analysis of Radix Sort using Curve Fitting Technique in Personal Computer

Arijit Chakraborty[1], Sanchari Banerjee[2], Avik Mitra[3], Dipankar Das[4]

*[1,3,4]Assistant Professor, Department of BCA(H), The Heritage Academy
Chowbaga, Anandapur, East Kolkata Township, Kolkata-700107, India
[2]Student, Department of Information Technology, Heritage Institute of Technology
Chowbaga, Anandapur, East Kolkata Township, Kolkata-700107, India*

---***---

Abstract - *The proposed research article aims at analyzing empirically a non comparative integer sorting algorithm such as radix sort using known curve fitting technique(s) in a domestic computing machine (laptop) through various known curve fitting models using time performance as a metric. We have used eleven best known models to observe the behavioral pattern of radix sort on the fly and concluded that power model is the candidate model for best fit.*

Keywords: Curve fitting, Empirical Analysis, Power Model, Radix Sort, Performance Analysis

1. INTRODUCTION

Sorting is an art of arranging items and almost all computing machines can sort data items, many are available readily and many are yet to be explored, it is a well known fact that no key comparison based sorting algorithms can sort N no. of keys lesser than O(NlogN) operations with some require O(N2) operations in worst case. We did picked radix sort amongst many due to a beautiful feature being N of keys can be sorted in O(N) operations. History of Radix sort dates back as far as 1887 credit goes to the work of Herman Hollerith [16] in tabulating machines. This paper aims at finding the most suitable curve that can be fitted on time generated data in computing machines used in common households on day to day basis, curve fitting technique gives us a platform to analyze and visualize experimental data which may give further insight on the behavioral pattern of radix sort. There is and always will be a scope of data refinement as we have not considered many effects of hardware architecture that plays a pivotal role in generating such data, we just want it to keep it simple. In this paper we found that out of many models power model is an ideal model to fit the time data in personal computer(s).

2. Related Work

Intermediate step of Radix sort [1] uses the value of a digit at a given position to determine the position of the number in intermediate array; this array, in its final iteration, becomes the sorted array. The scanning of the digits can either start from left to right or right to left. Left to right scanning during radix sort is termed as Most Significant Digit (MSD) radix sort, whereas, right to left scanning during radix sort is termed as Least Significant Digit (LSD) radix sort. LSD radix sorts [2] use queue to store the numbers where the position in the queue is based on the present digit being scanned. MSD radix sorts uses bins or buckets to store the numbers where the bucket in which the number is to be stored is determined by the digit presently being scanned; for example, 412 and 032 in list should be stored in bucket numbers 4 and 0 respectively; in the next iteration, sub-buckets are allocated for each bucket and the allocation of the number in the sub-bucket is determined by the digit in the next scanned position; the procedure recursively continues to get a set of sub-buckets in order each containing a number, and hence the numbers gets sorted when numbers are extracted in order of the sub-buckets if the buckets. Since the scanning is a sequential process, it slows down the actual running time, although the run-time order remains O(kn), where k is the number of digits in a number and n is the size of data to be sorted. Moreover, the allocation of sub-buckets in each recursive stages of MSD radix sorts can result exponential growth of the allocated space. To reduce the actual running-time in radix sorts in scanning of digits and allocation of space [3] proposed parallel radix sort where buckets are allocated at each processor of a multi-core processing system, the numbers are then moved between the buckets in subsequent iterations. Though the parallel version of the radix sort reduces the run-time, it suffers from time lag in exchange of the numbers between the buckets. Moreover, if the buckets contents vary, there will be further waste of CPU cycles to deal with asymmetric inter-processor transfer of numbers, called load imbalance. To reduce the load imbalance, load balanced parallel radix sort [4] proposed to split the buckets into multiple processors so that the count of the numbers in each processor remains equal to each other thus improving the run-time significantly. Partitioned parallel radix sort [2] is proposed where the communication overhead is reduced by parallelizing the MSD radix sort. Non-recursive MSD radix sort [5] reduces the space overhead in radix sort by using two sets of identical buckets for each digit, and sorting the numbers in each bucket using Quicksort [6], and then transferring the contents of each bucket to another in alternative fashion. Since space overhead is reduced, the communication overhead in transferring the numbers between the buckets is also reduced, thus reducing the run-time. The dependency of run-time of LSD radix sort in vector multi-processor environment is analyzed in [7] where empirical formulation results also revealed dependency on the number of processors actually used in the sorting process. Quicksort to sort numbers in MSD radix sort in each buckets is again employed in [8], called Matesort, thus eliminating the need of allocation of sub-buckets. To reduce the space overhead in LSD radix sort in GPUs, [9] proposed to maintain a global count of the numbers for a set of processors, so that the count can be used to determine to position of a number in the final sorted array, resulting 20% speed up of the algorithm.

All discussed implementations has used specialized computing environment like CRAY [10] to report the run-time. However, with the growth of computing needs, it is expected the sorting is not the only computation to be performed and any application employing sorting has to be executed with the other algorithms. Therefore, it is intuitive to use common computation environment like the personal computing system for analyzing the run-time of radix sort. In the next section, we analyze the actual run-time of the radix sort algorithm in personal computing system.

3. OBJECTIVES OF THE STUDY

To identify the best curve that can be fitted to the experimental data points (Run time versus Data size) obtained by running Radix sort in the worst case in personal computer and to propose a mathematical model of the best fitted curve.

4. RESEARCH METHODOLOGY

The steps of the research methodology are given below:

Step 1: The Radix sort algorithm is implemented as a C programme with data size varying from 10000 to 27000 with interval of 500. For each data size, the programme is run 100 times and their average run-time is taken.

Step 2: We have used curve fitting technique to find the best curve that can be fitted to the data points i.e. Run time versus Data size. In the present study we have opted R square, Adjusted R square and Root Mean Square Error (RMSE) as the 'Goodness of fit' statistics [11][12][15]. The model which has highest R square value, highest Adjusted R square value and lowest RMSE has been selected as the candidate model for the best curve for the dataset [11][12][15].

Step 3: The normality tests of the residuals of the candidate model are carried out in this step. We have considered both graphical methods (Histogram analysis of the residuals & Q-Q plot analysis of the residuals) [11][12] and quantitative method (Shapiro – Wilk test statistics of the residuals) [13][14] for this purpose. We should observe a symmetric bell shaped curve around the histogram, a linear pattern of the points on the Q-Q plot and the significance of Shapiro – Wilk statistics higher than .05 to meet the assumption of normality of error distribution.

Software used: We have used GCC compiler of Dev-C++ 4.0 under Windows XP to generate the experimental data SPSS have been used for data analysis.

Hardware used: Intel Core 2 Duo CPU T6570 with frequency of 2.1 GHz with 3 GB RAM (having frequency of 1.19 GHz).

5. DATA ANALYSIS & FINDINGS

The Sample dataset is given in the following table:

Table -1: Data Table

Sl No.	Data Size	Run Time (milliseconds)
1	10000	1292.49
2	10500	1319.71
3	11000	1463.44
4	11500	1654.06
5	12000	1772.01
6	12500	1921.54
7	13000	2103.61
8	13500	2326.9
9	14000	2529.23
10	14500	2691.85
11	15000	2859.07
12	15500	3116.67
13	16000	3339.06
14	16500	3326.1
15	17000	3601.71
16	17500	3805.93
17	18000	4131.39
18	18500	4367.04
19	19000	3846.41
20	19500	3988.74
21	20000	4198.6
22	20500	4581.72
23	21000	4517.81
24	21500	5006.43
25	22000	5272.06
26	22500	5667.37
27	23000	6187.86
28	23500	6220
29	24000	6636.72
30	24500	6721.21
31	25000	6877.31
32	25500	7288.3
33	26000	7447.19
34	26500	7436.41
35	27000	7836.4

Identification of the best curve that can be fitted to the data points:

The 'Goodness of fit' statistics of the Run time versus Data size is given below:

Table -2: Goodness of Fit Statistics Table

Model Name	R Square	Adjusted R Square	RMSE
Linear	.9797	.9791	288.6211
Logarithmic	.9412	.9394	491.338
Inverse	.8733	.8695	721.1321
Quadratic	.9886	.9878	220.0855
Cubic	.9886	.9878	220.0855
Compound	.9661	.9651	.0999
Power	.9897	.9894	.055
S	.981	.9804	.0747
Growth	.9661	.9651	.0999
Exponential	.9661	.9651	.0999
Logistic	.8051	.7992	.729

Findings: From the above table we found that out of eleven (11) tried models, ten (10) models are having very high R square and very high Adjusted R square. Out of these ten (10) models, five (5) models are having low RMSE. We observe that the Power model is having highest R square value (.9897), highest Adjusted R square value (.9894) and lowest RMSE value (.055). Therefore, we have selected the Power model as the candidate model for the best curve for this dataset.

The eleven (11) tried models are depicted in the following figure:

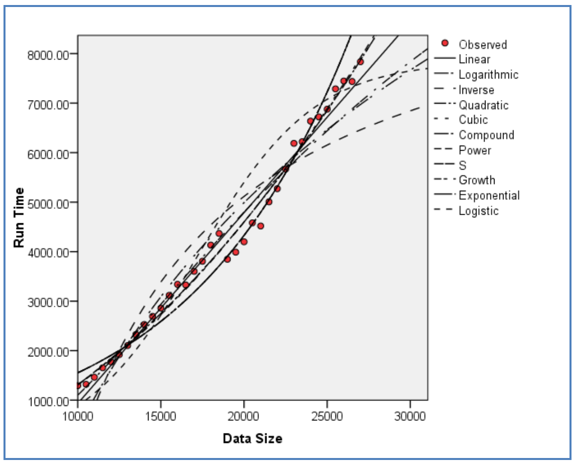

Chart -1: Chart of Eleven Models

The normality test of the residuals of the candidate model is given below:

(a) Histogram of the residuals –

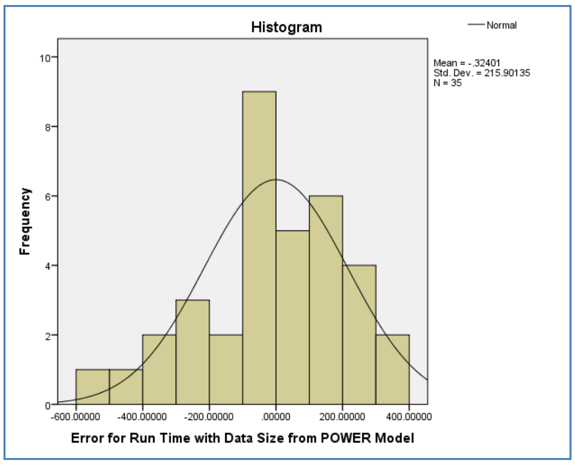

Chart -2: Histogram of the residuals

Observations: From the above figure we have observed a symmetric bell shaped curve around the histogram which is approximately evenly distributed around zero.

(b) Q-Q plot of the residuals –

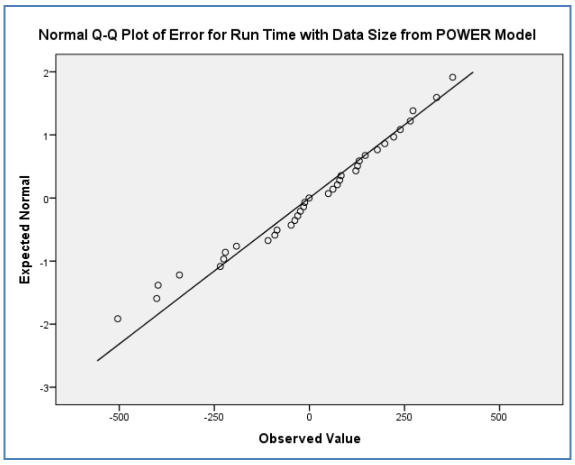

Chart -3: Q-Q plot of the residuals

Observations: From the above figure we have observed that the points on the Q-Q plot are approximately linear.

(c) Shapiro – Wilk (SW) test statistics of the residuals –

Table -3: SW Test Statistics Table

Model	Shapiro-Wilk		
	Statistic	df	Sig.
Power	.971	35	.463

Observations: From the above table we have observed that the significance of SW statistics is .463 (higher than .05).

Findings of the normality test of the residuals of the candidate model: From the above observations i.e. (a) Histogram of the residuals, (b) Q-Q plot of the residuals and (c) Shapiro – Wilk (SW) test statistics of the residuals, we have found that the residuals are approximately normally distributed.

The proposed mathematical model is given below:

Y = 7.274680973837771e-005 * X**1.813662220055146

Here,
Y = Run Time
X = Data Size

The plot of the above model is given below:

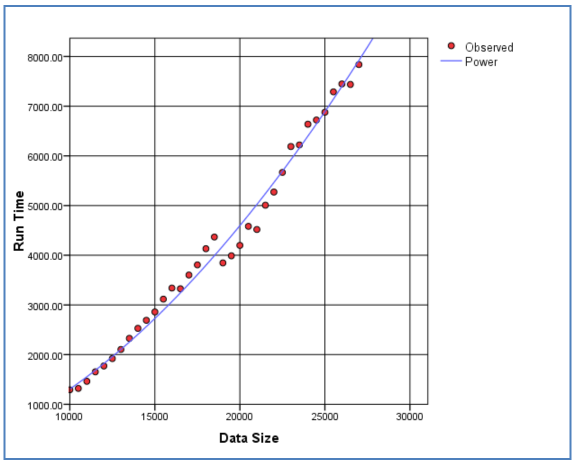

Chart -4: Final Concluded Model (Power Model)

6. CONCLUSION:

We have analyzed run-time behavior of Radix sort in its average case. Where we found that power model (~ C*x**1.81) fits the data. Our analysis has considered everyday computing scenario where executing the sorting algorithm will not be the only task performed by a computing device, that is, we have not taken into the factors of cache misses, OS context switch etc. In future, we would like to explore the effects of OS context switches on the performance of Radix sort and shall try to propose an empirical model that will give account of such events.

REFERENCES

[1] radix sort, National Institute of Standards and Technology, [online], https://xlinux.nist.gov/dads//HTML/radixsort.html
[2] S.J. Lee, M. Jeon, D. Kim, "Partitioned Parallel Radix Sort", vol 190, pp 160-171, LNCS High Performance Computing, April, 2001.
[3] A. Maus, "A Full Parallel Radix Sorting Algorithm for Multicore Processors", NIK 2011.
[4] A. Sohn, Y. Kodama, "Load Balanced Parallel Radix Sort", pp 3015-312, Proceedings of the 12th International Conference of Supercomputing, 1998.
[5] A.A.Aydin, G. Alaghband, "Sequential and Hybrid Approach for non-recursive Most Significant Digit Radix Sort", International Conference on Applied Computing 2013.
[6] C.A.R.Hoare, "Quicksort", vol. 5, issue 1, pp 10-16, The Computer Journal, 1962.
[7] M. Zagha, G.E. Blelloch, "Radix Sort for Vector Multiprocessors", pp 712-721, Proceedings of the ACM/IEEE conference of Supercomputing, 1991.
[8] N.A.Darwish, "Formulation and Analysis of in-place MSD Radix sort Algorithms", vol. 31, no. 6, pp 467-481, December 2005.
[9] L. Ha, J. Kruger, C.T. Silva, "Fast Four-way Parallel Radix Sorting on GPUs", vol. 28, no. 8, pp 2368-2378, December 2009.
[10] CRAY-1, COMPUTER SYSTEM, HARDWARE REFERENCE MANNUAL, 2240004, [online], http://ed-thelen.org/comp-hist/CRAY-1-HardRefMan/CRAY-1-HRM.html
[11] D. Das, A. Chakraborty, A. Mitra, "Sample Based Curve Fitting Computation on the Performance of Quicksort in Personal Computer", vol. 5, issue 2, pp 885-891, February 2014.
[12] A. Chakraborty, A. Mitra, D. Das, "Empirical Analysis of Merge sort in Personal Computer by Curve Fitting Technique", vol IV, issue IV, pp 1-6, April 2015.
[13] D. Das, P. Chakraborti, "Performance Measurement and Management Model of Data Generation and Writing Time in Personal Computer", vol. 5, issue 6, pp 1218-1226, June 2014.
[14] Testing for Normality using SPSS Statistics, Laerd statistics, (n.d.). Retrieved May 17, 2014, from ttps://statistics.laerd.com/spss-tutorials/testing-for-normality-using-spss-statistics.php
[15] Evaluating Goodness of Fit – MATLAB & Simulink – MathWorks India, MathWorks, [online], http://www.mathworks.in/help/curvefit/evaluating-goodness-of-fit.html
[16] The Art of Computer Programming by D. Knuth