Development of a Universal Running Time Predictor using Multivariate Regression

Farhan Hai Khan[a*], Suvradip Ghosh[b] & Tannistha Pal[b]

[a]Department of Electrical Engineering , Institute of Engineering & Management, Kolkata , India;
[b]Department of Electronics and Communication Engineering , Institute of Engineering & Management, Kolkata , India;

*Corresponding Author:-njrfarhandasilva10@gmail.com

ABSTRACT:

Various types of running times exist for analysis of algorithmic efficiency. This research presents a more empirical approach to the problem for the practical measurements of the actual running time of algorithms by considering a plethora of randomized inputs R_n and then fitting a regression curve in n to the algorithm of practical time complexity υ (n). This will also provide us the productivity factor η which will quantify the universal running time with respect to the asymptotic worst case complexity and evaluate the efficiency of the given algorithm with the help of leading coefficients. This research will also help us compare similar algorithms in a mathematically modeled manner.

KEYWORDS:

Universal Running Time; Efficiency of Algorithms; Practical Time Complexity; Multivariate Regression; Leading Coefficients

Practical Running Time: A Brief Introduction

Statement:

For any algorithm, M, the time it takes to execute a predefined task over a fixed input size n where n belongs to a discrete interval {n∈ [a, b]} and is fed with discrete random inputs R_k {where k=a, a+1, a+2,.....b} when fitted on a regression line will give the practical running time υ (n) [1] for that particular interval.

Generalization:

When the given interval is [1, ∞), the practical running time is the actual or the 'universal' running time of that algorithm M across all inputs.

Caution:

Since infinity is a non-reachable entity, we substitute the upper limit in the interval with a suitably high value. This value may vary suitably for various algorithms, but the key concept for this calculation is that

[1] Pronounced “Upsilon”, the twentieth letter of the Greek Alphabet

the current CPU processor speed standards are limited to $t \cong 10^8$ elementary operations [2] /sec approximately.

Feasibility:

The relation for the operations to be feasible for a computer is if,

$$t(a) + t(a+1) + t(a+2) + \ldots\ldots\ldots\ldots + t(b) \leq t_{max}$$

where $t_{max} = m * 10^8$ operations/sec (CPU Speed) and t (n) is the worst-case asymptotic time complexity for the given algorithm. However, for simplicity purposes, we will only consider the equality case.
Here, m is the time in seconds provided by the user for running the testing algorithm several times.

Simplification:

If we consider the interval [1, b] then the modified relation becomes

$$t(1) + t(2) + t(3) + \ldots\ldots\ldots\ldots + t(b) = t_{max}$$

$$\sum_{n=1}^{b} t(n) \cong \int_{1}^{b} t(n)dn = t_{max} \quad (1)$$

The above sum is approximated to an integral using the Riemann Sum [3][4].

Calculations for the limits of the intervals (a&b):

For QuickSort:

$$\sum_{n=1}^{b} t(n) = t_{max}$$

$$t(n) = \frac{n(n+1)}{2}$$

So, in case of QuickSort, taking m=1s, $t_{max} = 10^8$

$$\sum_{n=1}^{b} \frac{n(n+1)}{2} = 10^8$$

$$\sum_{n=1}^{b} \frac{n2}{2} + \sum_{n=1}^{b} \frac{n}{2} = 10^8$$

$$\frac{b(b+1)(2b+1)}{12} + \frac{b(b+1)}{4} = 10^8 \quad [\because \sum n^2 = \frac{n(n+1)(2n+1)}{6} \ \& \ \sum n = \frac{n(n+1)}{2}$$

[2] An elementary operation may be considered as an arithmetic operation/assignment/comparison.
[3]The detailed derivation with approximations is provided github.com/khanfarhan10/Practical-Running-Time-Predictor/tree/master/Documentations.
[4]For simplicity purposes, you can also use the website www.mathpapa.com/equation-solver/ to solve your equations in b. Also some non-polynomial equations will appear that are much complicated, but are later solved by the Newton Raphson Numerical Method as demonstrated in the above link with documentations.

Simplifying we get,

$$\frac{b^3}{6} + \frac{b^2}{2} + \frac{b}{3} = 10^8$$

By solving this cubic equation, we get,

$$b=842.432 \cong 10^3$$

Hence we chose the interval $[1,10^3]$ for QuickSort.

For Merge Sort:

$$\sum_{n=1}^{b} t(n) \cong \int_1^b t(n)dn = t_{max}$$

$$t(n) = 7n\log_2 n + n$$

$$\int_1^b (7n\log_2 n + n)\, dn = t_{max}$$

$$\int_1^b \left(7n\frac{\ln n}{\ln 2} + n\right) dn = t_{max} \quad [\because \log_2 n = \frac{\ln n}{\ln 2}]$$

Now,

$$\int nlnn = lnn \int ndn - \int [\frac{d(lnn)}{dn} \int ndn]dn = \frac{n^2}{2} lnn - \int \frac{1}{n} \times \frac{n^2}{2}\, dn = \frac{n^2}{2} lnn - \int \frac{n}{2}\, dn = \frac{n^2}{2} lnn - \frac{n^2}{4}$$

Using the above result $[\int nlnndn = \frac{n^2}{2} lnn - \frac{n^2}{4}]$ in the MergeSort equation we get,

$$\frac{7}{ln2}[\frac{n^2}{2} lnn - \frac{n^2}{4}]_1^b + [\frac{n^2}{2}]_1^b = t_{max}$$

$$\frac{14b^2 lnb}{4ln2} - \frac{7b^2}{4ln2} + \frac{7}{4ln2} + \frac{b^2}{2} - \frac{1}{2} = t_{max}$$

$$5.049433b^2 lnb - 2.024716b^2 + 2.024716b = t_{max}$$

Taking m=2 min. =120s, $t_{max}=120x10^8$,

To solve this equation in b, we used the Newton Raphson Numerical Method. [5] Solving, we get,

$$b=16003.153 \cong 10^4$$

Hence we chose the interval $[1,10^4]$ for MergeSort.

[5]The code for the same is demonstrated at : github.com/khanfarhan10/Practical-Running-Time-Predictor/tree/master/Documentations.You can try and experiment for different values of t_{max} as suitable.

Productivity Factor: A Measurable Entity

Let α(g) denotes the coefficient of the highest order(growth) term also known as the leading coefficient found in the function g(n) .We are interested to find the α (g) when g =

I) the asymptotic worst case complexity , i.e., leading coefficient of t(n) where t(n) is the worst case running time. Mathematically, $c_0=\alpha$ (t (n)).

II) the practical running time complexity, i.e., leading coefficient of υ(n). Mathematically $c_p=\alpha$ (υ (n)).

The productivity factor (η) is defined as the ratio of the practical coefficient by worst case coefficient.

In layman terms, it quantifies how good/bad is the measure of the running time of an algorithm's worst case asymptotic complexity.

Mathematically,

$\eta= c_p/c_0 =\alpha$ (υ (n))/ α (t (n)).

Generalizing:

If η<=0.5, the worst case asymptotic complexity is a bad approximation of the algorithm's practical running time and the worst case occurs rarely.

If η>0.5, the worst case asymptotic complexity is a good approximation of the algorithm's practical running time and the worst case occurs more frequently.

Implementation: Application of the Practical Running Time Complexity on Sample Standard Examples

Multivariate Regression: Effective Modifications

The standard equation for Multivariate Linear Regression is:

$$\upsilon(n) = \theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_3..... +\theta_mx_m \quad (2)$$

where m = number of features

x_i= feature values [i= 0,1, 2...m]

θ_i=coefficient of features (to be determined) [i= 0,1, 2...m]

υ (n)= Practical Running Time Complexity (Hypothesis Function) [6]

[6] Also to measure the execution time [individual υ (n)] of algorithms, in python, the following code snippet is often useful though not unique:pythonhow.com/measure-execution-time-python-code/. (We can use other quantitative measures, such as explicitly counting the number of elementary operations /steps we take to execute an algorithm or use the time it library codes provided as before. Our Research says that the time it code may vary from processor to processor and may give a different execution time every time, hence we used the other approach. The actual implementation of the ideas and their working is drafted on : github.com/khanfarhan10/Practical-Running-Time-Predictor) .

since we want to fit a curve in n to υ(n) hence we will assume that υ(n) is a predefined function of n and will attempt to find out all the θ_i [i= 0,1, 2...m]. Hence we would replace the x_i with suitable values such as x_1=n, x_2=logn, x_3=n^2, x_4=nlogn, x_5=√n...etc. A sample equation would be:

$$\upsilon(n) = \theta_0+\theta_1 n+\theta_2 \log n+\theta_3 n^2+\theta_4 n\log n+\theta_5\sqrt{n}..... +\theta_m x_m$$

Note that this implementation is completely problem dependent and may vary from algorithm to algorithm. We then use any optimization technique for minimizing the cost function for regression fits.

QuickSort:

QuickSort is a Divide and Conquer algorithm; it picks an element as pivot and partitions the given array around the picked pivot.There are many different versions of QuickSort that pick pivot in different ways. We considered choosing the last element of the partitioned list as our fixed pivot.

The recurrence relation for Worst Case of QuickSort is:

$$t(n) = n+t(n-1)$$

$$t(n) = n+(n-1)+(n-2)+.... 3+2+1=\sum n = \frac{n(n+1)}{2} = \frac{n^2}{2}+\frac{n}{2}$$

Hence, t (n) =O (n^2) and c0= α (t (n)) =0.5

The acquired relation [7] by our model is striking as it completely ignores the O (n^2) term:

$$\upsilon(n) = -464.3136289+27.3296038n+0.0048230n^2$$

We will draw attention to the coefficient of the highest order, the leading coefficient of this practical running time, c_p=α (υ (n)) =0.0048230. This is a very low value and will be discussed more in a later section.

Therefore the Productivity Factor,

$$\eta = c_p/c_0 = 0.0048230/0.5 = 0.009646 \cong 10^{-2}$$

So the worst case estimation is a disastrously bad estimation!

MergeSort:

MergeSort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

[7]The detailed draft and Octave/MATLAB implementation of regression for QuickSort can be found out on: github.com/khanfarhan10/Practical-Running-Time-Predictor/tree/master/Regression%20Program(OCTAVE-MATLAB)

The recurrence relation is therefore,

$$t(n) = 2*t(n/2) \text{ [Divide step]} + 7n \text{ [Merge Step]}$$

On solving this recurrence [8] with t (1) =1, we get,

$$t(n) = 7\, n*\log_2 n + n$$

The recurrence relation obtained by regression is,

$$\upsilon(n) = 2421.28748 + 0.66023n + 4.57608n\log_2 n$$

Productivity Factor $\eta = c_p/c_0 = \alpha(\upsilon(n))/\alpha(t(n)) = 4.57608/7 = 0.652944 \cong 0.65$

Hence, the worst case approximation is a rather good estimation in MergeSort.

Comparison of Algorithms:

We can compare the efficiency of two or more similar algorithms on the basis of their practical running times and leading coefficients.

For e.g., let us consider the two algorithms discussed earlier, viz., MergeSort (M) and QuickSort (Q).

The corresponding relations are:

$$\upsilon'_Q(n) = 464.3136289 + 27.3296038n + 0.0048230n^2$$

$$\upsilon_M(n) = 2421.24620 + 0.66024n + 4.57608n\log_2 n$$

Our Research tells us that QuickSort almost never reaches the worst case of O (n^2). Not surprisingly, we tried to fit a linearithmic (nlogn) fitting to the QuickSort data and the values obtained were truly remarkable:

$$\upsilon_Q(n) = 0.59037 + 5.22448n + 2.63849n\log_2 n$$

This clearly displays that the QuickSort algorithm, in practice, is actually O (nlogn).

But both QuickSort and MergeSort are then having practical running time υ (nlogn). This induces a conflict! **How do we know which algorithm is better?** We compare them using the leading coefficients.

As $\alpha(\upsilon_Q(n)) = 2.63849$ and $\alpha(\upsilon_M(n)) = 4.57608$

Clearly, $\alpha(\upsilon_Q(n)) < \alpha(\upsilon_M(n))$ and hence, QuickSort is a better practical algorithm than MergeSort although the former has asymptotic worst case complexity $O(n^2)$ and the latter O(nlogn)!

[8]The detailed derivation for the worst case time complexity of MergeSort and Octave/MATLAB implementation of regression can be found out on:github.com/khanfarhan10/Practical-Running-Time-Predictor

This is actually seen in practical reality, as python, spreadsheets, etc., use QuickSort as an inbuilt sorting function.

Data Visualization: Plotting the Running Time υ (n) vs. Features & Regression Line[9]

2-Dimensional Plots (x, y) ≡ (n, υ (n))

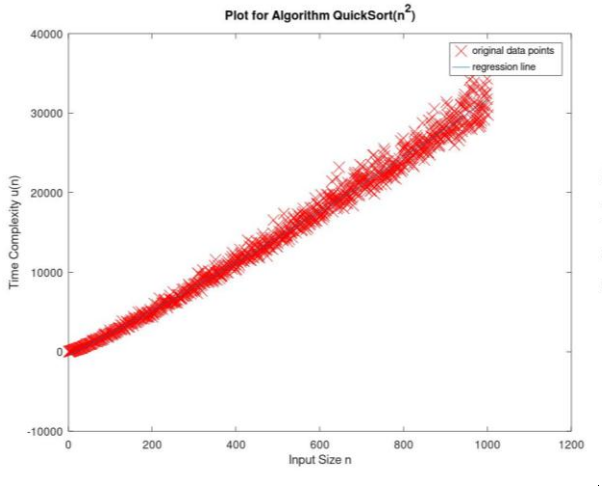

Fig. 1: QuickSort Algorithm (O (n^2)),υ(n) vs. .n

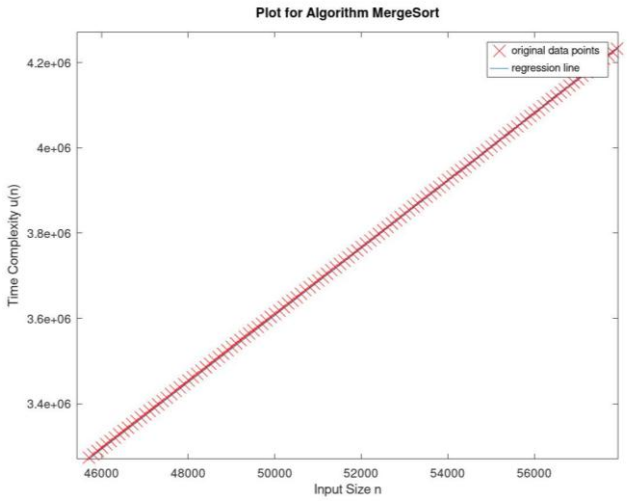

Fig.2: MergeSort Algorithm υ(n) vs. n

3-Dimensional Plots (x, y, z) ≡ (n_1, n_2, υ (n))

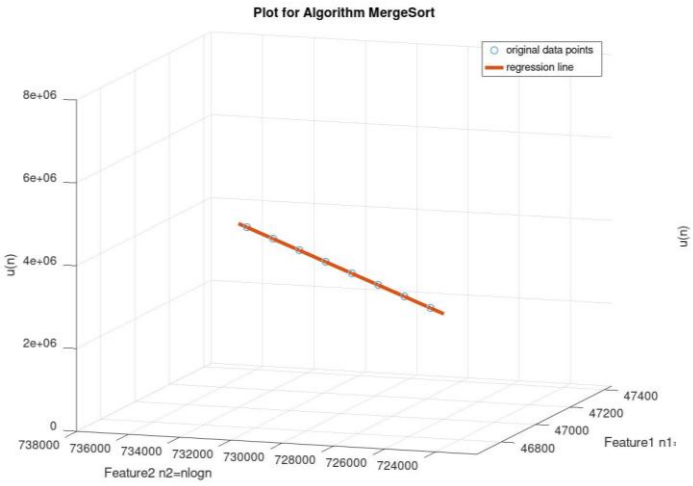

Fig. 3: MergeSort Algorithm (n,nlogn,υ(n))

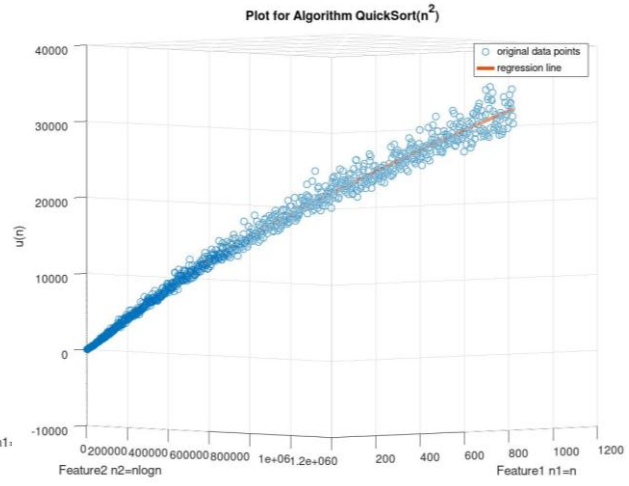

Fig.4: QuickSort Algorithm (O (n^2)) (n,n^2,υ(n))

[9]Since presenting all facets of the 3d plot and showing a plot of all the algorithms is cumbersome, we have taken screenshots for the same which are documented on: [github.com/khanfarhan10/Practical-Running-Time-Predictor/tree/master/Plots.](github.com/khanfarhan10/Practical-Running-Time-Predictor/tree/master/Plots)

Conclusion:

Actual running time of an algorithm is an empirical analysis that anticipates the running time of an algorithm based on various different inputs with variable input size. In this paper we present the practical measurement of this actual running time using multivariate regression curve to evaluate the efficiency of the algorithm and quantified it considering the asymptotic worst case time complexity. We applied this approach to compare the practical running time of similar type of algorithms, such as Quick Sort and Merge Sort, where we obtained a relation between the practical running time υ (n) vs. the input size (n) of these two algorithms. Using the productivity factor of the leading coefficients of these two equations we have proved that QuickSort is a better sorting algorithm compared to MergeSort. We can also use this concept in various mathematical-computational models to quantify running time on the basis of repeated application with randomized inputs.

References and Useful Links:

1. Algorithmic Analysis & Recurrence Relations, in depth, Design and Analysis of Algorithms, NPTEL Course, by Prof. Madhavan Mukund, Chennai Mathematical Institute.

2. Algorithms, by Christos Papadimitriou, Sanjoy Dasgupta, and Umesh Vazirani for effective approaches used in text.

3. Geek for Geeks, for QuickSort & MergeSort implementation codes.www.geeksforgeeks.org

4. Multivariate Linear Regression, Machine Learning, Stanford University, Coursera, by Prof. Andrew Ng. www.coursera.com/learn/machine-learning

5. Wikipedia, the free encyclopedia, for standard definitions. www.wikipedia.org

6. Stackoverflow, for simplifications and error corrections. www.stackoverflow.com