An Effective Approach to Analyze Algorithms with Linear O(n) Worst-Case Asymptotic Complexity

Qazi Haseeb Yousaf *[1], Muhammad Arif Shah*[2], Rashid Naseem[3], Karzan Wakil[4], Ghufran Ullah[5]
Department of Computer Science, City University of Science and Information Technology, Peshawar, Pakistan[1,2,3,5]
Research Center, Sulaimani Polytechnic University, Sulaimani 46001, Kurdistan Region, Iraq[4]

***Abstract*—A theoretical approach of asymptote analyzes the algorithms for approximate time complexity. The worst-case asymptotic complexity classifies an algorithm to a certain class. The asymptotic complexity for algorithms returns the degree variable of the algorithmic function while ignores the lower terms. In perspective of programming, asymptote only considers the number of iterations in a loop ignoring inside and outside statements. However, every statement must have some execution time. This paper provides an effective approach to analyze the algorithms belonging to the same class of asymptotes. The theoretical analysis of algorithmic functions shows that the difference between theoretical outputs of two algorithmic functions depends upon the difference between their coefficient of 'n' and the constant term. The said difference marks the point for the behavioral change of algorithms. This theoretic analysis approach is applied to algorithms with linear asymptotic complexity. Two algorithms are considered having a different number of statements outside and inside the loop. The results positively indicated the effectiveness of the proposed approach as the tables and graphs validates the results of the derived formula.**

Keywords—Asymptotic complexity; interval analysis; in-depth analysis; Big-Oh; crossover point

I. INTRODUCTION

An algorithm is a set of instructions to specify a solution to the problem in a finite time. The single problem may have more than one algorithm. For instance, algorithms to search maximum value in one-dimensional array or methods to sort elements in an array. Given the number of algorithms for a problem, it is obligatory that every algorithm differs in the order of complexity. Considering the difference in complexity of algorithms, some algorithms perform better than others in the provided environment. The level of complexity effects in execution time, the memory space acquired and lines of code (LOC) for the algorithms. From the given three metrics of complexity, LOC has no direct impact on algorithm complexity and is not considered a good metric for analysis of algorithm [1]. Therefore, execution time and space acquired are metrics for analysis of algorithms. The run-time of the algorithm is measured either by approximate analysis or execution time on the machine. Real execution time requires computational resources to run algorithms in actual time. The computational resources are affected by the environmental factors like temperature, number of resources utilized, dimensions of used resources etc. In contrast, the approximate method follows a virtual computational model. The virtual computational model depends on certain fixed execution times for different programming structures. Random Access Machine (RAM) is one of an adapted model for forming approximate algorithmic functions in terms of input size of the algorithm. RAM has three assumptions:

- Only one processor
- Infinite Memory
- Cost of one unit processor time for each binary operation and each array access.

A mathematical set-theoretic approach of asymptotes gives the time complexity of an algorithm. Asymptotic notations analyze algorithms for different time complexities. Generally, there are five asymptotic notations in mathematics. These are Big-Oh (O), Big-Omega (Ω), Theta (Θ), Small-Oh (o) and Small-Omega (ω). Big-Oh (O), Big-Omega (Ω) and Theta (Θ) are used asymptotes to analyze the algorithms [2].

Big-omega appears for asymptotic lower bound in analysis of algorithm. In computing, it is termed as best-case analysis. Leiserson et al. [1] defines Big-Omega mathematically as:

$$\Omega(f(n)) \geq \{\, g(n) : \text{there exists } c > 0 \text{ and } n0 \text{ such that } g(n) \leq c.f(n) \text{ for all } n > n0. \,\} \tag{1}$$

Big-Oh represents asymptotic upper bound for the analysis, when input size becomes so big that it tends to infinity, which is normally termed as worst-case analysis. Leiserson et al. [1] defines the equation of Big-Oh as follow:

$$O(f(n)) = \{\, g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \,\} \tag{2}$$

Theta shows tight bound for given function. It gives the average case analysis in computing. From mathematical definition stated by Leiserson et al. [1]

$$\theta(f(n)) = \{\, g(n) \text{ if and only if } g(n) = O(f(n)) \text{ and } g(n) = \Omega(f(n)) \text{ for all } n > n_0. \,\} \tag{3}$$

Given in the above defined equations (1), (2) and (3), n is the input size for algorithm. It may be size of array, stack or basic data structure to hold memory. $f(n)$ represents mathematical function of algorithm and $g(n)$ is the comparable function or asymptote.

The algorithms are compared on their worst-case analysis [3]. If the results of the worst-case analysis for two algorithms are the same, then the second metric to analyze the algorithms is best-case analysis. The problem arises when the worst-case and best-case analysis gives the same time complexity. It becomes difficult with asymptotes to analyze the better algorithm from the given pool [4]. The problem of analyzing the algorithm with the same time-complexity is discussed by [2,4,5] in their respective papers. Leiserson et al. [1] and Sedgewick et al. [3] in their books quoted a generalized statement for the above-discussed problem that "*Asymptotic big-Oh approximations are for the functions tend towards infinite large values*". This means with asymptotes, analysis for small input values is not feasible. So this traditional approach is not enough to analyze and choose the best algorithm from the given set [2–4]. Moreover, the real-world problems work with a specific interval of inputs. While asymptotes give us peak value analysis. It ignores the in-between behavior of an algorithmic function. The alternative method to differentiate the same asymptotic time complexity algorithms is real-time analysis on RAM. The RAM technique is a theoretical analysis requires less time and is not affected by certain environmental factors. Rahman [4] proposed two theoretical approaches (i) graph analysis and (ii) interval analysis [4]. Interval analysis works on a mathematical point of intersections for two or more functions. Full mathematical implementation of point of intersection contains a number of calculating steps with an overload of undesirable results in many cases like imaginary values, negative integer values, and continuous values. To some extent, the point of intersection approach throws an idea to check for the positive integer where the *crossover* occurs between the two algorithmic functions. This paper suggests a solution, which will analyze algorithms with the same asymptotic complexity, but different algorithmic functions. The proposed method finds approximate number 'n_o', where n_o is the point from which the algorithmic functions start changing their behaviors.

II. Literature Review

Algorithms are analyzed and validated from the very start of programming, for example, Garey et al [6], discussed the memory allocation algorithms for complexity effecting running time. Similarly, Mackeorth and Freuder [7], discussed network consistency algorithms related to artificial intelligence and analyzed them for worst-case analysis resulting in the linear time for binary constraints. Vitter and

Flajolet [8] introduced average-case analysis: this technique worked well for small algorithms and data structures. The study was on statistical measurements and formulations. Analysis of algorithms is still a need of the day. The statement is supported by the recent studies of [9–15] on analysis of algorithms applied in different fields via different techniques. Adams and Aschheim (2016) compared algorithms for dental coding and ranking problems. The paper concluded with setting optimized (OPT) algorithm as best for large inputs with simple coding [13]. Schubert and Zimek [14] also discussed the analysis of algorithms technique separating implementation and evaluation. Most recently a study is conducted for the max-min filter on random inputs. The study focused on asymptotic analysis for the said problem on dynamic algorithms [15].

Analysis of algorithm is done considering different aspects of real-time evaluation, asymptotic analysis, non-asymptotic analysis, and sensitive analysis etc. Among the above asymptotic analysis is one of the easiest ways to get approximated results theoretically but with accuracy payoff. Tarek [2] wrote on the set-theoretic approach of asymptotes i.e. asymptotic functions are relative functions than individual bare ones. This means asymptotic functions always depend on coefficient and exponents of input size *n*. The extended work of Decelle et al. [16] was an exact analysis of the stochastic block model. The author used an asymptotic approximation for analysis. A research study [17] on non-asymptotic analysis of machine learning algorithms may be in principle costly to generate results beyond a certain number of equations. Teng [18] discussed linear time algorithms for preconditioning and solving dominant linear systems. Xu et al. [19] explore the algorithms for Virtual Machine (VM) deployment and load balancing. The research included certain new metrics to conclude for the best performing algorithm in the deployment and load balancing of VM. The proposed and existed all metrics are environmental variables dependent on the applied platform. Pietri et al. [20] give the survey of VM deployment on Physical Machines (PM). The adopted techniques were proven correct but research was inconclusive about the best technique. Thi and Thi [21] published work on the same problem of approximated the complexity of algorithms using asymptotes. The work suggested statistical and probability formulas resulting in equivalent values that of asymptotes. The formula calculates sorting algorithms realistic complexity. The method is partially automated and most calculations are done manually. In addition, the method depends on the type and nature of input [21].

The above studies show that authors are inconclusive to decide between algorithms. For the same degree of two different algorithmic functions asymptotes failed to analyze them for small or large input sizes. Asymptotes treat them equally good for all input sizes, which is not exactly the case. The inference is to replace asymptotic analysis by realistic analysis specifically for same worst-case complexity algorithms, but the realistic analysis also bears the load of environmental factors and availability of physical resources. The problem of analyzing with the same time-complexity is discussed by [5] and [4] in their respective papers. Rahman et al. [4] suggested the analysis of algorithmic functions by graphs through drawing them. The author also gave an interval analysis to mark the point where the big value starts with the creation of threshold value (k) for *n* tends to infinity. The study considers the insertion sort as an example. Insertion sort is quadratic in nature. The worst-case analysis for insertion sort is $O(n^2)$. It was claimed algorithmic functions for two different implementations of insertion sort. The proposed methodology was to suppose a threshold point and then calculate values of 'k' for all inputs and mark the input point breaking a specific threshold. The problem in the study was the selection of a threshold point and then to find that input where the big value starts. Alternative for interval analysis is the graph method. It is very difficult to draw and analyze the graphs for infinite input values. Also, the input size is discrete in nature [3] while graphs of polynomials are continuous in nature [21,22]. Ferreira et al. [5] conducted a survey on memory allocation techniques. The article concluded that the malloc family has linear time complexity, but it is difficult to choose the best algorithm of the family because of the same time complexity. However, sensitivity analysis is suggested for overcoming this problem. The sensitive analysis is not an efficient metric because the hardware and software vary too much from point to point and doing an analysis every time before implementation may not be easy. Most recently Schubert and Zimek [14] discussed the hurdles in algorithm evaluation and implementation on environment-sensitive metrics like machine execution time and platform dependent testing. The author argues with different techniques used to validate the efficiency of algorithms presented in research papers. Most of the time the current algorithm proved less efficient than presented in the previous research papers. The affecting factor is a run-time evaluation of algorithms, the most common technique used for analysis of research algorithms. Therefore, there must be a standard technique to check the algorithm and its implementation separately. The data mining algorithms are tested on different platforms with different datasets giving the changed result on every platform [14]. On the other side of the discussion, there are many sorting algorithms available. Merge sort, heap sort, insertion sort, bubble sort, selection sort, and many others. These algorithms are passing through advancements to reduce the overall complexity. Small improvements in algorithms are totally negligible due to discrete classes of asymptotes. Some of the recent work in advancement and analysis of sorting algorithms is the study of quadratic algorithms for sorting evolving data [24] and analysis of algorithms on the multi-threaded and multi-core environment. As the algorithms are advancing it is obligatory to make changes in the algorithm evaluation techniques. A review is conducted on sorting algorithms with respect to the size of the array. The review included all the used sorting algorithms of various complexities. The results proved some of the quadratic algorithms better over merge and quicksort in the provided conditions [25]. The time complexity of insertion sort, bubble sort, and selection sort is the same $O(n^2)$. It is difficult to analyze best among these three discussed. However, their algorithmic functions may be helpful to conclude some results.

III. Methodology

Interval analysis is the feasible theoretical technique to analyze the algorithms with the same worst-case analysis [2,3]. Observation of mathematical graphs and functions clarifies that there may be a point of intersection for two same degree mathematical curves [21–24]. Asymptote always considers a mathematical function of an algorithm termed as algorithmic function. It takes the highest power term or degree of algorithmic function. The highest terms show the worst-case scenario for the algorithm. However, in the real run-time, every line of code has an effect on time of execution. So, the conclusive statement may come as that every term and related coefficient of the algorithmic function is important and must be considered for theoretical analysis. The impact will be theoretical analysis close to real-time analysis of algorithms. For the implementation of the proposed method, few linear expressions are considered as algorithmic functions performing the same task in a different manner.

$$3n+7 \tag{4}$$

$$n+29 \tag{5}$$

$$2n+145 \tag{6}$$

The *n* in the above equations represents input size for the algorithm. The Big-Oh for the above defined algorithmic functions is *O(n)*. According to asymptotic analysis, algorithms are under the same curve of *O(n)* but graph analysis shows the changing behavior of a function from interval to interval. Table I compares the input size to the output time of the algorithms. As the input size increases, the algorithm will consume more time.

From Table I it is seen that for input size *n=1* to *n=10* in (5) gives higher output values while the case reverses after input size *n=11* and values start diverging. The value for (4) gets higher and higher due to the larger coefficient than (5). Consider some other examples from Table II and III.

Table II shows the comparison between (5) and (6). The output values show that the difference is constantly increasing due to the fact of the higher coefficient and the constant term of (6) in comparison with (5).

Table III compares the mathematical equation of (6) with (4) for input size *n*. It shows that the coefficient of *n* in (4) is greater than the coefficient of *n* in (6), while constant in (6) is greater than constant in (4). As a result, for input size *n=1 to 137* the difference starts converging and after input size *138* it again starts diverging due to the higher coefficient of *n* for (4). The above results lead to the conclusion that output also depends on the coefficient of *n* and constant as it depends on *n* specifically, for small inputs. Further analysis shows that there must be a *crossover* point if coefficient for first and constant for second is larger and vice versa. This *crossover* is dependent on the difference or gap between the respective coefficient and constants. The formula to find the *crossover* point is proposed on the basis of these findings. The formula gives the *crossover* point where algorithmic functions change their behaviors. The functions in Tables I, II and III are all linear. Therefore, generalized forms of two linear algorithmic functions are assumed as shown in (7) and (8):

$$f(n)=a_1\, n+b_1 \tag{7}$$

$$g(n)=a_2\, n+b_2 \tag{8}$$

Where n_o is the *crossover* point, a_1 coefficient of *n* in *f(n)*, b_1 constant term in *f(n)*, a_2 coefficient of *n* in *g(n)*, b_2 constant term in *g(n)*

The following are the possible cases for relating the coefficients and constants of (7) and (8):

Case 1: If the coefficient of *n* and the constant term of the first function (*f(n)*) is greater than the coefficient of *n* and the constant term of the second function (*g(n)*) then second function (*g(n)*) will take less execution time throughout from 1 to infinity. Mathematically it can be represented as,

$$\text{if } a_1>a_2 \text{ and } b_1>b_2 \rightarrow g(n) \text{ is better than } f(n) \text{ for interval } [1,\infty) \tag{9}$$

Case 2: If coefficient of *n* of first function (*f(n)*) is greater than coefficient of *n* of second function (*g(n)*) while constant term of first function (*f(n)*) is smaller than constant term of the second function (*g (n)*) then *crossover* point n_o will decide the intervals. The first function (*f(n)*) will take less execution time before n_o and second function (*g(n)*) will take less execution time after n_o. Mathematically it can be represented as:

if $a_1 > a_2$ and $b_1 < b_2$ then

$n_o = | b_2 - b_1 | / | a_2 - a_1 |$ (10)

So *f(n)* is better than *g(n)* for interval $[1, n_o)$ and for interval (n_o, ∞) *g(n)* is better than *f(n)*

Case 3: If the coefficient of *n* and the constant term of the first function (*f (n)*) is smaller than the coefficient of *n* and the constant term of the second function (*g(n)*) then first function (*f(n)*) will take less execution time throughout from 1 to infinity. Mathematically it can be represented as:

if $a_1 < a_2$ and $b_1 < b_2 \rightarrow$ f(n) is better than f(n) for interval $[1, \infty)$ (11)

Case 4: If coefficient of *n* of first function (*f(n)*) is smaller than coefficient of *n* of second function (*g(n)*) while constant term of first function (*f(n)*) is greater than constant term of the second function (g (n)) then *crossover* point n_o will decide the intervals. The second function (*g(n)*) will take less execution time before n_o and first function (*f(n)*) will take less execution time after n_o. Mathematically it can be represented as:

if $a_1 < a_2$ and $b_1 > b_2$ then

$n_o = | b_2 - b_1 | / | a_2 - a_1 |$ (12)

So *g(n)* is better than *f(n)* for interval $[1, n_o)$ and for interval (n_o, ∞) *f(n)* is better than *g(n)*.

For two algorithmic functions with same worst-case analysis Big-Oh, the better will be "whose coefficient 'a' and constant 'b' is lesser"

If one of the two ('a' or 'b') for an algorithmic function is lesser while the second is higher than finding the *crossover* point by the given formula:

$n_o =$ (higher *b* – smaller *b*)/(higher *a* – smaller *a*) (13)

The algorithmic function with a greater coefficient of *n* will take less time before n_o and algorithmic function with a smaller coefficient of *n* will take less time after n_o.

The same technique can be helpful for higher order polynomial algorithmic functions, for example, quadratic and cubical.

IV. Results

The proposed formula is validated on graphs and tables. The first step is to calculate intervals for (4), (5) and (6) by applying the proposed formula. The graphs for (4), (5) and (6) are drawn for small random intervals. The intersection point in the graph is compared with values in Tables I, II and III. Lastly, values of the intersection point of formula are validated through graph and table.

(4)→3n+7

(5)→n+29

(6)→2n+145

TABLE I. Crossover Point at Input Size 11

Input Size (n)	Output Time Eq 4 (3n+7)	Output Time Eq 5 (n+29)
1	10	30
2	13	31
10	37	39
.	.	..
11	40	40
12	43	41
.	.	.
71	220	100
72	223	101

TABLE II. No Crossover Point

Input Size (n)	Output Time Eq 5 (n+29)	Output Time Eq 6 (2n+145)
1	30	147
2	31	149
.	.	.
98	127	341
99	128	343
.	.	.
174	203	493
175	204	495

TABLE III. Crossover Point at Input Size 138

Input Size (n)	Output Time Eq 4 (3n+7)	Output Time Eq 6 (2n+145)
1	10	147
2	13	149
.	.	.
106	325	357
.	.	.
137	418	419
138	421	421
139	424	423
.	.	.
177	538	499
178	541	501

As from (4) and (5), the coefficient of *n* in (4) is higher than (5) but on contrary constant of (5) is greater. So, applying the proposed formula:

$n_o =$ (higher *b* – smaller *b*)/(higher *a* – smaller *a*)

$n_o = (29-7)/(3-1)$

$n_o = 11$

The *crossover* point n_o is similar to that given in Table I. This shows that (4) is better before input size *11* and (5) performs better after input size *11*. The graph shows the same results in Fig. 1.

Similarly, the graphical representation of (4) and (6) are given in Fig. 2. The graph changes behavior between 135 and 140 as shown in Fig. 2 (Graph with *input size (n)* 100-500). The value in Table III for the given equations is input size 138. To validate it with proposed formula, consider (4) and (6), the coefficient of *n* for (6) is smaller to (4) while vice versa is the case if constants of both equations are observed. By applying the formula:

n_o= (higher *b* – smaller *b*)/(higher *a* – smaller *a*)

n_o= (145-7)/(3-2)

n_o=138

It is concluded from the results that (4) has less execution time on RAM on the interval [1,138) as compare to (6). On the other hand, (6) performs better in the interval (138, ∞).

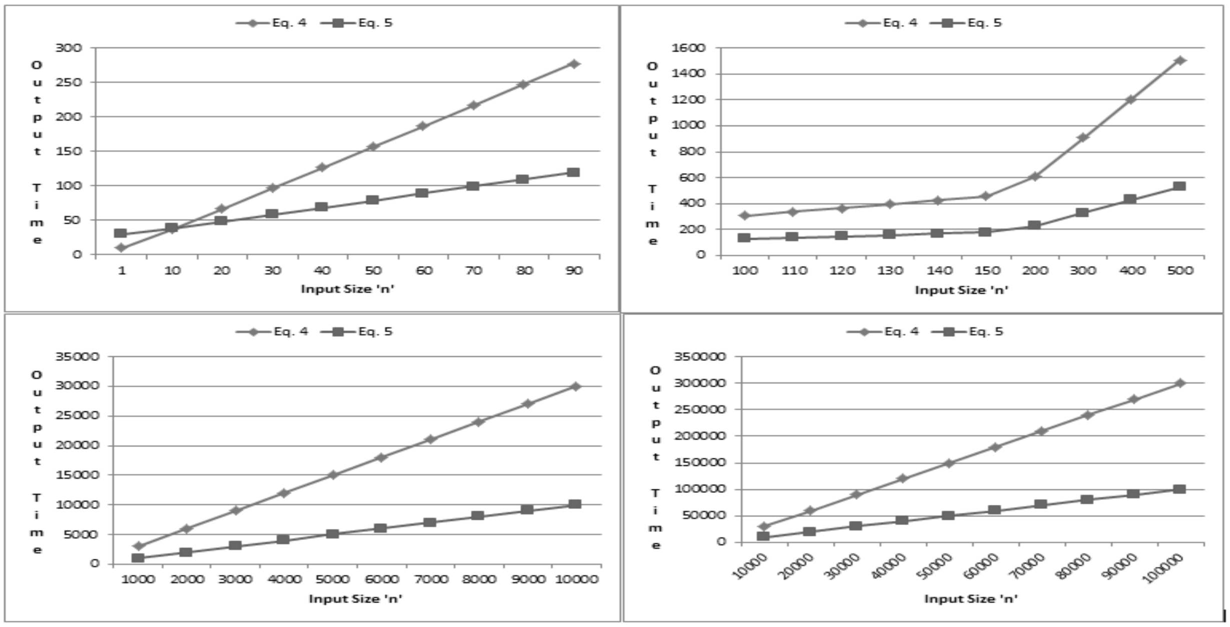

Fig. 1. Graphs Representing Eqs. (4) and (5) for Various Ranges of Input (n).

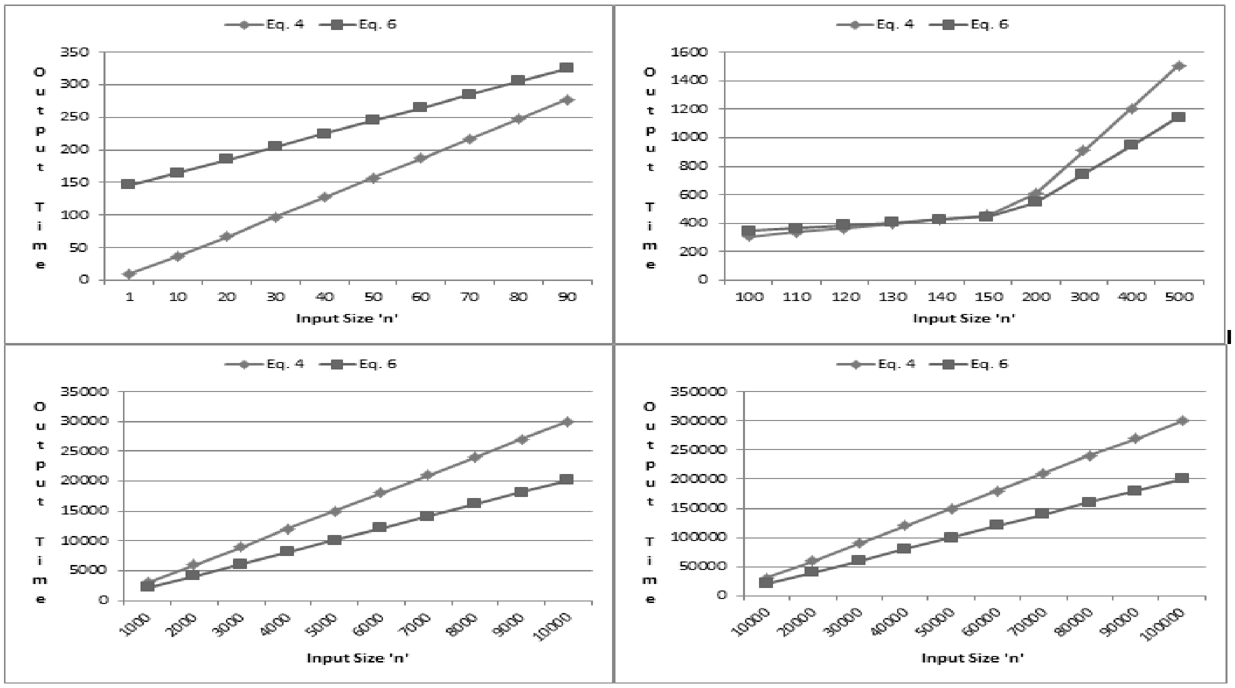

Fig. 2. Graphs Representing Eqs. (4) and (6) for Various Ranges of Input (n).

V. Discussion

As suggested from the literature set-theoretic approach of asymptote is commonly used for analysis of algorithms. The approach gives an easier way to analyze algorithms in less time and effort as compared to run-time analysis. However, the technique of asymptotic complexity does not separate the algorithms with the same worst-case analysis. In reality, asymptotes assign a generic class of Big-Oh to each algorithmic function by focusing degree term thus ignoring lower terms. However, according to RAM assumptions each statement has some execution time. It is also to be noted that more and more statements inside and outside repetition structure increase the value of coefficient and constants respectively. Hence the impact falls as increased execution time. It is observed from the study that algorithmic functions of different algorithms with the same worst-case analysis relate to each other in terms of the corresponding coefficient and constant terms. Lesser the coefficient and constant term lower will be the execution time and vice versa. To calculate the relation between coefficient and constants of two algorithms a formula is proposed and validated in this study. As the results demonstrated that difference of coefficients of n and difference of *constant terms* of two algorithmic functions is nearer so a simpler formula generates the approximate *crossover* point for algorithmic functions. The *crossover* point forms the intervals for each given algorithm. Hence the method narrows the gap between theoretical and exact analysis for the algorithms.

VI. Conclusion

Analysis of algorithms is important to categorize algorithms on the basis of time and space. Algorithms are analyzed approximately by calculating their asymptotic complexity. The mathematical functions are discussed to relate two algorithms in order of statements inside and outside the repetition structures. The relation between terms of algorithmic functions leads to the formula for intersecting point. The intersecting point eventually forms intervals for performance behavior of algorithms. The analysis of linear algorithmic functions follows quadratic algorithmic functions as part of future work. The same formula works efficiently well for quadratic algorithmic function if constant terms c_1 and c_2 have a very small difference i.e. $|c_1 - c_2|$ is approximately equal to zero. For the large difference between constant terms of quadratic algorithmic functions, the formula requires slightest of changes. Besides this for future work, it is in the pipeline to generalize the formula for all the polynomial algorithmic functions. It is also set as a milestone to design a computational algorithm for the defined formula.

References

[1] C. C. E. Leiserson, R. R. L. Rivest, C. Stein, and T. H. Cormen, Introduction to Algorithms, Third Edition, vol. 7. 2009.

[2] A. Tarek, “A generalized set-theoretic approach for time and space complexity analysis of algorithms and functions,” 2006, vol. 2, pp. 316–324.

[3] R. Sedgewick and P. Flajolet, An introduction to the analysis of algorithms. 2013.

[4] L. Rahman, M. S. A. Khan, A. Kabir, and M. S. Miah, “A model for set-theoretic analysis of algorithm with asymptotic perspectives,” in 2013 International Conference on Informatics, Electronics and Vision (ICIEV), 2013, no. 3, pp. 1–6.

[5] T. B. Ferreira, M. A. Fernandes, and R. Matias Jr., “A comprehensive complexity analysis of user-level memory allocator algorithms,” in 2012 Brazilian Symposium on Computing System Engineering, 2012, pp. 99–104.

[6] M. Garey and R. Graham, “Worst-case analysis of memory allocation algorithms,” STOC ’72 Proc. fourth Annu. ACM Symp. Theory Comput., pp. 143–150, 1972.

[7] A. K. Mackworth and E. C. Freuder, “The complexity of come polynomial network consistency algorithms for constraint satisfaction Problems,” vol. 25, no. 1985, pp. 65–74.

[8] J. S. Vitter and P. Flajolet, “Average-Case analysis of algorithms and data structures L’analyse en Moyenne des algorithms et des structures de donn ees,” no. page i, 1990.

[9] R. Goyal and D. K. Srivastava, “A study on cluster analysis technique - Hierarchical Algorithms,” no. 9, pp. 1274–1279, 2016.

[10] M. Wolfram, A. Marten, and D. Westermann, “A comparative study of evolutionary algorithms for phase shifting transformer setting optimization,” in 2016 IEEE International Energy Conference (ENERGYCON), 2016, pp. 1–6.

[11] K. S. Prado, N. T. R. B, V. F. Silva, L. B. Jr, L. A. Digiampietri, and E. M. Ortega, Human-Computer Interaction. Interaction Platforms and Techniques, vol. 4551. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[12] C. Meshram, S. Gajbhiye, and D. Gupta, “Statistical analysis of an algorithm’s complexity for Linear Equation 2 . statistical analysis on experimental results,” vol. 8, no. 2, pp. 191–198, 2015.

[13] B. J. Adams and K. W. Aschheim, “Computerized dental comparison : A critical review of dental coding and ranking algorithms used in victim identification,” vol. 61, no. 1, pp. 76–86, 2016.

[14] H. K. E. Schubert and A. Zimek, “The (black) art of runtime evaluation : Are we comparing algorithms or implementations ?,” Knowl. Inf. Syst., 2016.

[15] M. Li, H. Liang, S. Liu, C. K. Poon, and H. Yuan, “Asymptotically optimal algorithms for running Max and Min Filters on random inputs,” IEEE Trans. Signal Process., vol. 66, no. 13, pp. 3421–3435, 2018.

[16] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborov??, “Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications,” Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys., vol. 84, no. 6, p. 066106, Dec. 2011.

[17] F. R. Bach and E. Moulines, “Non-asymptotic analysis of stochastic approximation algorithms for machine learning,” Neural Inf. Process. Syst., no. 2, pp. 1–9, 2011.

[18] S. Teng, “Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear,” vol. 35, no. 3, pp. 835–885, 2014.

[19] M. Xu, W. Tian, and R. Buyya, “A survey on load balancing algorithms for virtual machines placement in cloud computing,” pp. 1–20, 2016.

[20] I. Pietri and R. Sakellariou, “Mapping virtual machines onto physical machines in cloud computing : A Survey,” vol. 49, no. 3, 2016.

[21] J. Clément, T. H. Nguyen Thi, and B. Vallée, “Towards a realistic analysis of some popular sorting algorithms,” Comb. Probab. Comput., vol. 24, no. 01, pp. 104–144, Jan. 2015.

[22] I. These, “Introduction to graph theory,” Discrete Math., vol. 37, no. 1, p. 133, 1981.

[23] J. L. Gross, J. Yellen, and P. Zhang, Handbook of graph theory. Second Edition. 2013.

[24] J. J. Besa, W. E. Devanny, D. Eppstein, M. Goodrich, and T. Johnson, “Quadratic time algorithms appear to be optimal for sorting evolving data,” pp. 87–96, 2018.

[25] J. Totla, “Review on execution time of sorting algorithms- A Comparative Study,” vol. 5, no. 11, pp. 158–166, 2016.