A REVIEW DATA STRUCTURE , ALGORITHMS & ANALYSIS

[1]Mir Omranudin Abhar, [2]Nisha Gatuam

[1]M.Tech Student, [2]Assistant Professor

[1,2]Department of Computer Science & Engineering

[1,2]AP Goyal Shimla University, Shimla, India

Abstract: In this review paper, we will discuss about data structure & its algorithms and their complexity. First we begin with the introduction of data structure and the algorithm and then we discuss the relationship between them. Then we will discuss about complexities of the algorithm in data structure. A data structure is a specialized format for organizing and storing data. An algorithm is a step by step method of solving a problem. It is commonly used for data processing, calculation and other related computer and mathematical operations. To write any program we have to select proper algorithm and the data structure. If we choose improper data structure, algorithm cannot work effectively. Similarly, if we choose improper algorithm we cannot utilize the data structure effectively. Thus there is a strong relationship between data structure and algorithm. The complexity of an algorithm is a measure of the time and/or space required by an algorithm for an input of a given size (n).

***Key Word*- Algorithm, Complexity, Asymptotic Notation, Analysis.**

I. INTRODUCTION

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structure is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's name \Virat" and age 26. Here \Virat" is of String data type and 26 is of integer data type.

We can organize this data as a record like Player record, which will have both player's name and age on it. Now we can collect and store player's records in a le or database as a data structure. For example: \Dhoni" 30, \Gambhir" 31, \Sehwag" 33.
Data Structures are structures, programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency [15].

Data structure help in storing, organizing, and analyzing the data in a logical manner. The following points highlight the need of data structures in computer science [8]:

- It depicts the logical representation of data in computer memory.
- It represents the logical relationship between the various data elements.
- It helps in efficient manipulation of stored data elements.
- It allows the programs to process the data in an efficient manner.

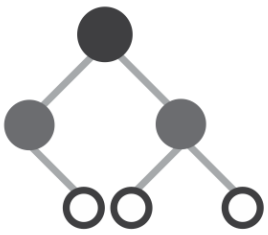

Fig. 1. Data Structure.

Primitive data structures these data structures are basic structures and are manipulated/ operated directly by Machine instructions. We will present storage representations for these data structures for a variety of machines. The integer's floating-point numbers (real numbers), character constants, string constants, pointers etc. are some of the primitive data structures. In C language, these primitive data structures are defined using data types such as int, oat char and double. But you are already aware of these representations in computer main memory [7].

Non-Primitive data structures these data structures are more sophisticated and are derived from the primitive structures. But, these cannot be manipulated/operated directly by machine instructions. A non-primitive data structure emphasizes on structuring of a set of homogeneous (same data type) or heterogeneous (different data types) data elements. Arrays, structures, stacks queues linked lists, les etc. are some of the non-primitive data structures [7].

II. DATA STRUCTURES

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some arc highly specialized to special c tasks. Data structures are used in almost every program or software system. Data structures provide a means to manage huge

amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in soft-ware design [12, 11].

Data structure deals with the study of how data is organized in the computer's main memory and how it maintains logical relationship between individual elements of data and also, how efficiently the data can be retrieved and manipulated. A data structure is a study of organizing all the data elements that consider not only the elements stored but also their relationship with each other. Data structures are the building blocks of a program and hence, the selection of a particular data structure addresses the following two things: [7]

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address - a bit string that can be itself stored in memory and manipulated by the program. Thus the record and array data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing ad-dresses of data items within the structure itself. Many data structures use both principles, sometimes combined in non-trivial ways. The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost) [12].

III. Relation Between Data Structure and Algorithm

Implementation of data structures can be done with the help of programs. To write any program we need an algorithm. Algorithm is nothing but collection of instructions which has to be executed in step by step manner. And data structure tells us the way to organize data. Algorithm and data structure together give the implementation of data structures. To write any program we have to select proper algorithm and the data structure. If we choose improper data structure, algorithm cannot work effectively. Similarly, if we choose improper algorithm we cannot utilize the data structure effectively. Thus there is a strong relationship between data structure and algorithm. As data structure can be very well understood with the help of a programming language [2].

Data Structure + Algorithm = Programs

IV. Algorithm

Algorithm is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word Algebra as well [13, 17].

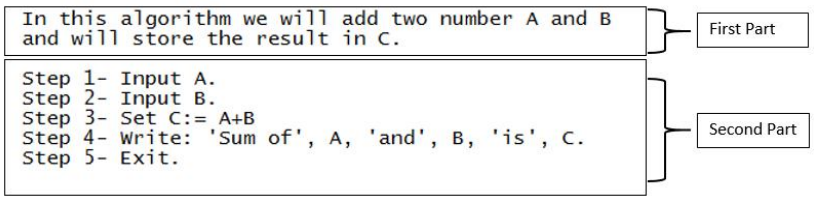

Figure. 2. Representation Algorithms.

An algorithm is a step by step method of solving a problem [16]. It is commonly used for data processing, calculation and other related computer and mathematical operations [4]. Once an algorithm is given for a problem and decided (somehow) to be correct, an important step is to determine how much in the way of resources, such as time or space, the algorithm will require. An algorithm that solves a problem but requires a year is hardly of any use [22].

V. Implementation of Algorithm

To develop any program we should first select a proper data structure, and then we should develop an algorithm for implementing the given problem with the help of a data structure which we have chosen. Before actual implementation of the program, designing a program is very important step. Suppose, if we want to build a house we do not directly start constructing the house. Instead we consult an architect, we put our ideas and suggestions, accordingly he draws a plan of the house, and he discusses it with us. If we have some suggestion, the architect notes it down and makes the necessary changes accordingly in the plan. This process continues till we are satisfied. Finally, the blue print of house gets ready. Once design process is over actual construction activity starts. Now it becomes very easy and systematic for construction of desired house. In this example, you will find that all designing is just a paper work and at that in-stance if we want some changes to be done then those can be easily carried out on the paper. After a satisfactory de-sign the construction activities start. Same is a program development process.

If we could follow same kind of approach while developing the program then we can call it as Program development cycle which involves several steps as - Feasibility study, requirement analysis and problem specification, Design, Coding, Debugging, Testing and Maintenance [2].

VI. Analysis of Algorithm

To analyze an algorithm is to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity) [10, 20].

Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem. These estimates provide an insight into reasonable directions of search for efficient algorithms. In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input [10, 6].

VII. ANALYSIS OF PROGRAM

The analysis of the program does not mean simply working of the program but to check whether for all possible situations program works or not. The analysis also involves working of the program efficiently. Efficiency in the sense,

1. The program requires less amount of storage space.
2. The program gets executed in very less amount of time.

The time and space are the factors which determine the efficiency of the program. Time required for execution of the program cannot be computed in terms of seconds because of the following factors –

1. The hardware of the machine.
2. The amount of time required by each machine instruction.
3. The amount of time required by the compilers to execute the instruction.
4. The instruction set.

Hence we will assume that time required by the program to execute means the total number of times the statements get executed. The number of times the statement executing is known as frequency count [2].

VIII. COMPLEXITY OF ALGORITHM

The complexity of an algorithm is a function f (n) which measures the time and space used by an algorithm in terms of input size n. In computer science, the complexity of an algorithm is a way to classify how efficient an algorithm is, compared to alternative ones. The focus is on how execution time increases with the data set to be processed. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures [23, 3]

If we have two algorithms that perform same task, and the first one has a computing time of O (n) and the second of O (n2), then we will usually prefer the first one. The reason for this is that as n increases the time required for the execution of second algorithm will get far more than the time required for the execution of first. We will study various values for computing function for the constant values. The graph given below will indicate the rate of growth of common computing time functions [2].

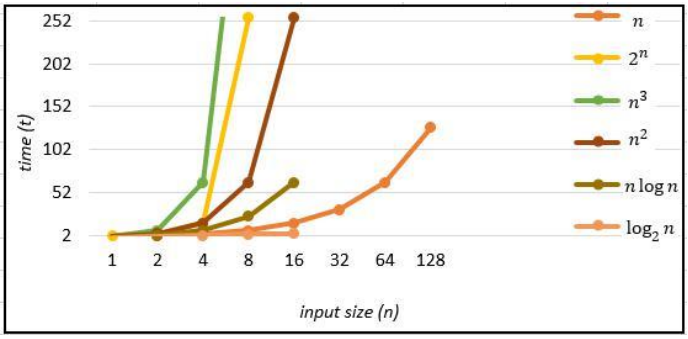

Fig. 3. Rate of growth of common computing time function

n	$\log_2 n$	$n \log_2 n$	n^2	n^3	2^n
1	0	0	1	1	2
2	1	9	4	8	4
3	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	2147483648

Notice how the times O (n) and O (n log n) grow much more slowly than the others. For large data sets algorithms with a complexity greater than O (n log n) are often impractical. The very slow algorithm will be the one having time complexity 2n [2].

IX. TIME SPACE COMPLEXITY (ASYMPTOTIC NOTATION)

Asymptotic notation describes the behavior of the time or space complexity for large instance characteristics [19]. To choose the best algorithm, we need to check efficiency of each algorithm. The efficiency can be measured by computing time complexity of each algorithm. Asymptotic notation is a shorthand way to represent the time complexity. Using asymptotic notations, we can

give time complexity as "fastest possible", "slowest possible" or "average time". Various notations such as; and O used are called asymptotic notions [1].

Theta Notation: Theta notation denoted as ' ' is a method of representing running time between upper bound and lower bound.

Definition: Let, f (n) and g (n) be two non-negative functions. There exists positive constants C_1 and C_2 such that C_1 g (n) ≤ f (n) ≤ C_2 g (n) and f(n) is theta of g of n. [1, 2]

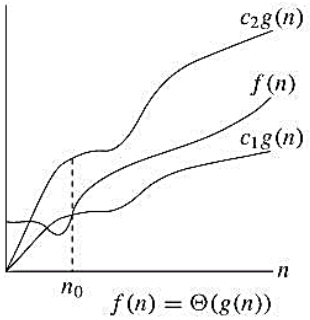

Fig. 4. Theta Notation

Big Oh Notation: Big Oh notation denoted by 'O' is a method of representing the upper bound of algorithm's running time. Using big oh notation we can give longest amount of time taken by the algorithm to complete.

Definition Let, f (n) and g (n) are two non-negative functions. And if there exists an integer no. and constant C such that C > 0 and for all integers n > n_0, f (n) ≤ c∗ g (n); then f (n) is big oh of g (n). It is also denoted as "f (n) = O (g (n))". [1, 2]

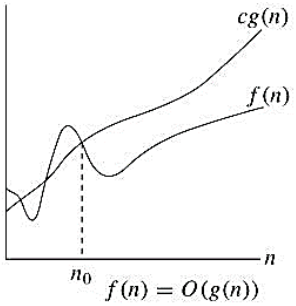

Fig. 5. Big Oh Notation

Omega Notation: Omega notation denoted as is a method of representing the lower bound of algorithm's running time. Using omega notation, we can denote shortest amount of time taken by algorithm to complete.

Definition Let, f(n) and g(n) are two non-negative functions and if there exists constant C and integer no. such that C > 0 and n > n_o then f(n) > C ∗ g(n) i.e. f(n) is omega of g of n. This is denoted as f(n) =Ω(g(n)). [1, 2]

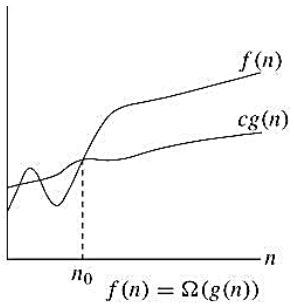

Fig. 6. Omega Notation

X. Space Complexity

Another useful measure of an algorithm is the amount of storage space it needs. The space complexity of an algorithm can be computed by considering the data and their sizes. Again we are concerned with those data items which demand for maximum storage space. A similar notation 'O' is used to denote the space complexity of an algorithm. When computing for storage requirement we assume each data element needs one unit of storage space. While as the aggregate data items such as arrays will need n units of storage space n is the number of elements in an array. This assumption again is independent of the machines on which the algorithms are to be executed [2].

To compute the space complexity, we use two factors: constant and instance characteristics. The space requirement S(p) can be given as

$$S(p) = C + Sp$$

Where C is a constant i.e. fixed part and it denotes the space of inputs and outputs. This space is an amount of space taken by instruction, variables and identifiers.

Sp is a space dependent upon instance characteristics. This is a variable part whose space requirement depends on particular problem instance [1].

1. **Constant Complexity: $O(1)$**

A constant tasks run time won't change no matter what the input value is. Consider a function that prints a value in an array. No matter which elements value you're asking the function to print, only one step is required. So we can say the function runs in O (1) time; its run-time does not increase. Its order of magnitude is always 1. [18, 5]

2. **Linear Complexity: $O(n)$**

A linear tasks run time will vary depending on its input value. If you ask a function to print all the items in a 10-element array, it will require less steps to complete than it would a 10,000 element array. This is said to run at O (n); its run time increases at an order of magnitude proportional to n. [18, 5]

3. **Quadratic Complexity: $O(n^2)$**

A quadratic task requires a number of steps equal to the square of its input value. Let's look at a function that takes an array and N as its input values where N is the number of values in the array. If I use a nested loop both of which use N as its limit condition, and I ask the function to print the arrays contents, the function will perform N rounds, each round printing N lines for a total of O (N^2) print steps.

Let's look at that practically. Assume the index length N of an array is 10. If the function prints the contents of its array in a nested-loop, it will perform 10 rounds, each round printing 10 lines for a total of 100 print steps. This is said to run in O (N^2) time; its total run time increases at an order of magnitude proportional to N^2. [18, 5]

4. **Exponential: $O(2^n)$**

O (2^n) is just one example of exponential growth (among O (3^n); O (4^n), etc.). Time complexity at an exponential rate means that with each step the function performs, its subsequent step will take longer by an order of magnitude equivalent to a factor of N. For instance, with a function whose step-time doubles with each subsequent step, it is said to have a complexity of O (2^n). A function whose step-time triples with each iteration is said to have a complexity of O (3^n) and so on. [18, 5]

5. **Logarithmic Complexity: $O(log\ n)$**

This is the type of algorithm that makes computation blazingly fast. Instead of increasing the time it takes to perform each subsequent step, the time is decreased at magnitude inversely proportional to N.

Let's say we want to search a database for a particular number. In the data set below, we want to search 20 numbers for the number 100. In this example, searching through 20 numbers is a non-issue. But imagine were dealing with data sets that store millions of users pro le information. Searching through each index value from beginning to end would be ridiculously inefficient. Especially if it had to be done multiple times.

A logarithmic algorithm that performs a binary search looks through only half of an increasingly smaller data set per step.

Assume we have an ascending ordered set of numbers. The algorithm starts by searching half of the entire data set. If it doesn't and the number, it discards the set just checked and then searches half of the remaining set of numbers. Round printing 10 lines for a total of 100 print steps. This is said to run in O (N2) time; its total run time increases at an order of magnitude proportional to N2.

As illustrated above, each round of searching consists of a smaller data set than the previous, decreasing the time each subsequent round is performed. This makes long algorithms very scalable.

While I've only touched on the basics of time complexity and Big O notation, this should get you off to a good start. Besides the benefit of being more adept at scaling programs efficiently, understanding the concept of time complexity is HUGE benefit as I've been told by a few people that it comes up in interviews a lot. [18, 5]

XI. CONCLUSION

In this paper, we discussed the data structure, Algorithm, relationship between data structure with Algorithm and Analyzing complexity of Algorithm.

A data structure is a specialized format for organizing and storing data. An algorithm is a step by step method of solving a problem. It is commonly used for data processing, calculation and other related computer and mathematical operations. Implementation of data structures can be done with the help of programs. To write any program we need an algorithm. Thus there is a strong relationship between data structure and algorithm. As data structure can be very well understood with the help of a programming language.

To analyze an algorithm is to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity).

The future work will have a review on sorting algorithms and will introduce a new sorting algorithm.

XII. ACKNOWLEDGEMENTS

First of all, I would like to thank the most merciful and the most gracious Allah who teaches and shows us the way of happiness of this and next word by the agency of his messenger the owner of honor Mohammad (peace be open him) and give command for learning the knowledge. I am grateful from the all lectures especially my supervisor Assistant Professor Ms NISHA GAUTAM for

his valuable contributions, patience and guidance through this study. I am grateful to my parents for their hidden help in terms of prayers and faith in me, which actually empowered me to fulfill this task. And last, I am grateful to all my entourage, classmates and companions who helped me on the journey of university career and made it easier. I would also like to thank my entourage, classmates and companions especially from my partner Eng. Ehsan Bayat who helped me on the journey of university career and made it easier.

And last but not least, I am grateful to my parents for their hidden help in terms of prayers and faith in me, which actually empowered me to fulfill this task.

REFERENCES

[1] A.A.Puntambekar. Advance Data Structures. 2007.
[2] A.A.Puntambekar. Data Structures And Algorithms. 2009.
[3] Saleh Abdel-hafeez and Ann Gordon-Ross. \A Comparison-Free Sorting Algorithm, IEEE Journals". In: (2014).
[4] Algorithm. https : / / www . techopedia . com /definition/3739/algorithm. Accessed: 2018-10-01.
[5] Algorithm Time Complexity and Big O Notation. https://medium.com/StueyGK/algorithm- time-complexity- and- big- o-notation- 51502e612b4d. Accessed: 2018-10-15.
[6] Analysis of Algorithms. https : / / www . tutorialspoint . com / design _ and _ analysis _ of _ algorithms / analysis _ of _ algorithms . htm. Ac-cessed: 2018-10-08.
[7] Venkatesh N. Baitipuli. Introduction to Data Struc-tures Using C. 2009.
[8] E Balagurusamy. Data Structures Using C. Tata McGraw-Hill Education, 2013.
[9] Thenmozhi S Bremananth R Radhika V. \Visualizing Sequence of Algorithms for Searching and Sorting , IEEE Journal". In: (2009).
[10] Computer Science An Overview. PediaPress GmbH, Boppstrasse 64 , Mainz, Germany, 2011.
[11] Tata McGraw-Hill Education. DATA STRUTURES THROUGH C++. 2011.
[12] Mario Eguiluz Alebicto Erik Azar. Swift Data Struc-ture and Algorithms. 2016.
[13] Urs E. Gattiker. The Information Security Dictionary.2006
[14] Viliam Ge ert Gianni Franceschini. \An In-Place Sorting with O (nlogn) Comparisons and O(n) Moves, IEEE Journals". In: (2017).
[15] Introduction-to-data-structures. https: / / www . studytonight . com / data - structures / introduction - to - data - structures. Accessed: 2018-10-03.
[16] Johnsonbugh. Discrete mathematicx,6/E. 2007.
[17] Dheeraj Mehrotra. ISC Computer Science for Class 12. 2008.
[18] Arjun Sawhney Rugved Deolekar Rayner Vaz Vi-raj Shah. \Automated Big-O Analysis of Algorithms, IEEE Journals". In: (2017).
[19] Sartaj Sahni. Data Structures, Algorithms Applica-tions In C++. 2005.
[20] IEEE Saleh Abdel-Hafeez Member and IEEE Ann Gordon-Ross Member\An Efcient O(N) Comparison-Free Sorting Algorithm , IEEE Jour-nals"In: (2017).
[21] Mudasser A. Khan H. Akbar Sultan Ullah Muham-mad A. Khan and SyedS. Hassan. \Optimized Se-lection Sort Algorithm for Two Dimensional. Array , IEEE Journal". In: (2015).
[22] Mark Allen Weiss. Data structures and algorithms analysis in java.2012
[23] What is complexity of algorithm? https : / / www . quora . com /What - is - complexity - of - algorithm. Accessed: 2018-10-02