Introduction to Analysis of Algorithms:-

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Most algorithms are designed to work with inputs of arbitrary length. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as **time complexity**, or volume of memory, known as **space complexity**.

Types of analysis:-

- **Worst-case** − The maximum number of steps taken on any instance of size **a**.
- **Best-case** − The minimum number of steps taken on any instance of size **a**.
- **Average case** − An average number of steps taken on any instance of size **a**.
- **Amortized** − A sequence of operations applied to the input of size **a** averaged over time.

Importance of Algorithm Analysis:-

In the analysis of the algorithm, it generally focused on CPU (time) usage, Memory usage, Disk usage, and Network usage. All are important, but the most concern is about the CPU time. Be careful to differentiate between:

- **Performance:** How much time/memory/disk/etc. is used when a program is run. This depends on the machine, compiler, etc. as well as the code we write.
- **Complexity:** How do the resource requirements of a program or algorithm scale, i.e. what happens as the size of the problem being solved by the code gets larger.

Role of Algorithms in Computing:-

Algorithms play a crucial role in computing by providing a set of instructions for a computer to perform a specific task. They are used to solve problems and carry out tasks in computer systems, such as sorting data, searching for information, image processing, and much more. An algorithm defines the steps necessary to produce the desired outcome, and the computer follows the instructions to complete the task efficiently and accurately. The development of efficient algorithms is a central area of computer science and has significant impacts in various fields, from cryptography and finance to machine learning and robotics.

Algorithms are widely used in various industrial areas to improve efficiency, accuracy, and decision-making. Some of the key applications include:

1.Manufacturing: Algorithms are used to optimize production processes and supply chain management, reducing waste and increasing efficiency.

2.Finance: Algorithms are used to analyze financial data and make predictions, enabling traders and investors to make informed decisions.

3.Healthcare: Algorithms are used to process and analyze medical images, assist in diagnosing diseases, and optimize treatment plans.4Retail: Algorithms are used for customer relationship management, personalized product recommendations, and pricing optimization.

4.Transportation: Algorithms are used to optimize routes for delivery and transportation, reducing fuel consumption and increasing delivery speed.

5.Energy: Algorithms are used to optimize energy generation, distribution, and consumption, reducing waste and increasing efficiency.

6.Security: Algorithms are used to detect and prevent security threats, such as hacking, fraud, and cyber-attacks.

In these and many other industries, algorithms play a crucial role in automating tasks, improving decision-making, and enhancing overall performance and efficiency. Algorithms are fundamental to computing and play a crucial role in many aspects of the field. Some of the key needs and applications of algorithms in computing include:

1.Data processing: Algorithms are used to process and analyze large amounts of data, such as sorting and searching algorithms.

2.Problem solving: Algorithms are used to solve computational problems, such as mathematical problems, optimization problems, and decision-making problems.

3.Computer graphics: Algorithms are used to create and process images and graphics, such as image compression algorithms and computer-generated graphics algorithms.

4.Artificial Intelligence: Algorithms are used to develop intelligent systems, such as machine learning algorithms, natural language processing algorithms, and computer vision algorithms.

5.Database management: Algorithms are used to manage and organize large amounts of data in databases, such as indexing algorithms and query optimization algorithms.

6.Network communication: Algorithms are used for efficient communication and data transfer in networks, such as routing algorithms and error correction algorithms.

7.Operating systems: Algorithms are used in operating systems for tasks such as process scheduling, memory management, and disk management.

Algorithm of Efficiency:-

1. Space efficiency - the memory required, also called, space complexity
2. Time efficiency - the time required, also called time complexity

Space Efficiency:-

There are some circumstances where the space/memory used must be analyzed. For example, for large quantities of data or for embedded systems programming.

Components of space/memory use:

1. instruction space	Affected by: the compiler, compiler options, target computer (cpu)
2. data space	Affected by: the data size/dynamically allocated memory, static program variables,
3. run-time stack space	Affected by: the compiler, run-time function calls and recursion, local variables, parameters

The space requirement has fixed/static/compile time and a variable/dynamic/runtime components. Fixed components are the machine language instructions and static variables. Variable components are the runtime stack usage and dynamically allocated memory usage.

Time Efficiency:-

Clearly the more quickly a program/function accomplishes its task the better.

The actual running time depends on many factors:

- The speed of the computer: cpu (not just clock speed), I/O, etc.
- The compiler, compiler options .
- The quantity of data - ex. search a long list or short.
- The actual data - ex. in the sequential search if the name is first or last.

Analysis of Recursive Algorithms:-

1. Decide on a parameter (or parameters) indicating an input's size.
2. Identify the algorithm's basic operation.
3. Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.

4. Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.
5. Solve the recurrence or, at least, ascertain the order of growth of its solution.

Recursive Evaluation of n!

Definition: n! = 1 * 2 * ... *(n-1) * n for n 1 and 0! = 1

Recursive definition of **n! :** F(n) = F(n-1)*n for n >= 1 and F(0) = 1

****

ALGORITHM F(n)

//Computes n! recursively

//Input: A nonnegative integer n

//Output: The value of n!

if n = 0 return 1

else return F(n - 1)*n

****

M(n)= M(n − 1) + 1 for n > 0
to compute F(n-1) to multiply F(n-1) by n

M(n)= M(n − 1) +1 for n>0
M(0) = 0

The calls stop when n=0 no multiplications when n = 0

M(n-1) = M(n-2) + 1;

M(n-2) = M(n-3)+1

M(n) = [M(n-2)+1] + 1
= M(n-2) + 2
= [M(n-3)+1+2]
= M(n-3) + 3

=M(n-n) + n

= n

Overall time Complexity: O(n)

Example: Tower Hanoi

Explain the problem using figure

Demo and show recursion

1. Problem size is *n*, the number of discs

2. The basic operation is moving a disc from rod to another

3. There is no worst or best case

4. Recursive relation for moving n discs

$M(n) = M(n\text{-}1) + 1 + M(n\text{-}1) = 2M(n\text{-}1) + 1$

IC: $M(1) = 1$

5. Solve using backward substitution

$M(n) = 2M(n\text{-}1) + 1$

$= 2[2M(n\text{-}2) + 1] + 1 = 2^2M(n\text{-}2) + 2 + 1$

$= 2^2[2M(n\text{-}3) + 1] + 2 + 1 = 2^3M(n\text{-}3) + 2^2 + 2 + 1$

...

$M(n) = 2^iM(n\text{-}i) + \sum_{j=0}^{i-1} 2^j = 2^iM(n\text{-}i) + 2^i\text{-}1$

...

$M(n) = 2^{n-1}M(n\text{-}(n\text{-}1)) + 2^{n-1}\text{-}1 = 2^{n-1}M(1) + 2^{n-1}\text{-}1 = 2^{n-1} + 2^{n-1}\text{-}1 = 2^n\text{-}1$

$M(n) \in \Theta(2^n)$

Analyzing the Time Efficiency of Non recursive Algorithms:-

1. Decide on a parameter (or parameters) indicating an input's size.

2. Identify the algorithm's basic operation (in the inner most oop).

3. Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately.

4. Set up a sum expressing the number of times the algorithm's basic operation is executed.

5. Using standard formulas and rules of sum manipulation either find a closed form formula for the count or at the least, establish its order of growth.

EXAMPLE 1: Consider the problem of finding the value of the largest element in a list of n numbers. Assume that the list is implemented as an array for simplicity.

ALGORITHM Max Element(A[0..n − 1]) //Determines the value of the largest element in a given array //Input: An array A[0..n − 1] of real numbers

//Output: The value of the largest element in A

Max val ←A[0]

for i ←1 to n − 1 do

if A[i]>maxval

maxval←A[i]

return maxval

Algorithm analysis

• The measure of an input's size here is the number of elements in the array, i.e., n.

• There are two operations in the for loop's body:

- The comparison A[i]> maxval and
- The assignment max val←A[i].

• The comparison operation is considered as the algorithm's basic operation, because the comparison is executed on each repetition of the loop and not the assignment.

• The number of comparisons will be the same for all arrays of size n;Therefore, there is no need to distinguish among the worst, average, and best cases here.

• Let C(n) denotes the number of times this comparison is executed. The algorithm makes one comparison on each execution of the loop, which is repeated for each value of the loop's variable i within the bounds 1 and n − 1, inclusive. Therefore, the sum for C(n) is calculated as follows: − () = ∑ = i.e., Sum up 1 in repeated n-1 times − () = ∑ = − ∈ ()

Empirical Analysis of Algorithm:-

Empirical analysis is an evidence-based approach to the study and interpretation of information. Empirical evidence is information that can be gathered from experience or by the five senses. In a scientific context, it is called empirical research.

Empirical analysis requires evidence to prove any theory. An empirical approach gathers observable data and sets out a repeatable process to produce verifiable results. Empirical analysis often requires statistical analysis to support a claim.

1. **Observation.** Initial observations of a phenomena are made. This sparks an idea or a line of inquiry. Initial empirical data and research into existing information can be done.
2. **Induction.** A probable explanation of the observed phenomenon is proposed. Inductive reasoning is used to take the specific example from step one and infer a generalized explanation for it.
3. **Deduction.** A testable hypothesis is proposed that can support the explanation. Deductive reasoning is used to take the generalized explanation and make a specific prediction that can be tested and observed.
4. **Testing.** Quantitative and qualitative empirical data are gathered. The data is examined, often with statistical analysis. The results can support, refute or be neutral to the hypothesis. Because of the limits of empirical data and human perception, it is not said that the results prove or disapprove the hypothesis, only that they support or don't support it.
5. **Evaluation.** The reasoning, methodology and findings of the experiment are written down, and the conclusions of the researcher are presented. Information relating to any difficulties, challenges and limits of the test are also included. It may also include further possible avenues of research.