**Design and Analysis of Algorithm**

**UNIT – 1 : Analysis of Algorithm**

**What is Algorithm?**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

**Search** − Algorithm to search an item in a data structure.

**Sort** − Algorithm to sort items in a certain order.

**Insert** − Algorithm to insert item in a data structure.

**Update** − Algorithm to update an existing item in a data structure.

**Delete** − Algorithm to delete an existing item from a data structure.

**How to Write an Algorithm?**

There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.

We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

**Example:**

Let's try to learn algorithm-writing by using an example.

Problem − Design an algorithm to add two numbers and display the result.

**Step 1** – START

**Step 2** − declare three integers a, b & c

**Step 3** − define values of a & b

**Step 4** − add values of a & b

**Step 5** − store output of step 4 to c

**Step 6** − print c

**Step 7** – STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

**Step 1** − START ADD

**Step 2** − get values of a & b

**Step 3** − c ← a + b

**Step 4** − display c

**Step 5** – STOP

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Writing step numbers, is optional.

## **Use of the Algorithms:**

Algorithms play an important role in various fields and have many applications. Some of the key areas where algorithms are used include:

**Computer Science:**

Algorithms form the basis of computer programming and are used to solve problems ranging from simple sorting and searching to complex tasks such as artificial intelligence and machine learning.

**Mathematics:**

Algorithms are used to solve mathematical problems, such as finding the optimal solution to a system of equations or finding the shortest path in a graph.

**Operations Research:**

Algorithms are used to optimize and make decisions in fields such as transportation, Information Technology etc.

**Artificial Intelligence:**

Algorithms are the foundation of artificial intelligence and machine learning, and are used to develop intelligent systems that can perform tasks such as image recognition, natural language processing, and decision-making.

**Data Science:**

Algorithms are used to analyze, process, and extract insights from large amounts of data in fields such as marketing, finance, and healthcare.

## **What is the need for algorithms?**

* Algorithms are necessary for solving complex problems effectively.
* They help to automate processes and make them more reliable, faster, and easier to perform.
* Algorithms also enable computers to perform tasks that would be difficult or impossible for humans to do manually.
* They are used in various fields such as mathematics, computer science, engineering, finance, and many others to optimize processes, analyze data, make predictions, and provide solutions to problems.

## **Types of Algorithms:**

There are several types of algorithms available. Some important algorithms are:

### [**Brute Force Algorithm**](https://www.geeksforgeeks.org/brute-force-approach-and-its-pros-and-cons/)**:**

It is the simplest approach to a problem. A brute force algorithm is the first approach that comes to finding when we see a problem.

### [**Recursive Algorithm**](https://www.geeksforgeeks.org/recursion/)**:**

A recursive algorithm is based on [recursion](http://www.geeksforgeeks.org/recursion/). In this case, a problem is broken into several sub-parts and called the same function again and again.

### [**Backtracking Algorithm**](https://www.geeksforgeeks.org/backtracking-algorithms/)**:**

The backtracking algorithm builds the solution by searching among all possible solutions. Using this algorithm, we keep on building the solution following criteria. Whenever a solution fails we trace back to the failure point build on the next solution and continue this process till we find the solution or all possible solutions are looked after.

### [**Searching Algorithm**](https://www.geeksforgeeks.org/searching-algorithms/)**:**

Searching algorithms are the ones that are used for searching elements or groups of elements from a particular data structure. They can be of different types based on their approach or the data structure in which the element should be found.

### [**Sorting Algorithm**](https://www.geeksforgeeks.org/sorting-algorithms/)**:**

Sorting is arranging a group of data in a particular manner according to the requirement. The algorithms which help in performing this function are called sorting algorithms. Generally sorting algorithms are used to sort groups of data in an increasing or decreasing manner.

### [**Hashing Algorithm**](https://www.geeksforgeeks.org/hashing-set-1-introduction/)**:**

Hashing algorithms work similarly to the searching algorithm. But they contain an index with a key ID. In hashing, a key is assigned to specific data.

### [**Divide and Conquer Algorithm**](http://www.geeksforgeeks.org/divide-and-conquer-introduction/)**:**

This algorithm breaks a problem into sub-problems, solves a single sub-problem, and merges the solutions to get the final solution. It consists of the following three steps: Divide, Solve, Combine

### [**Greedy Algorithm**](http://www.geeksforgeeks.org/greedy-algorithms/)**:**

In this type of algorithm, the solution is built part by part. The solution for the next part is built based on the immediate benefit of the next part. The one solution that gives the most benefit will be chosen as the solution for the next part.

## **Advantages of Algorithms:**

* It is easy to understand.
* An algorithm is a step-wise representation of a solution to a given problem.
* In an Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

## **Disadvantages of Algorithms:**

* Writing an algorithm takes a long time so it is time-consuming.
* Understanding complex logic through algorithms can be very difficult.
* Branching and Looping statements are difficult to show in Algorithms**(imp)**.

**Efficiency of an algorithm:**

Computer resources are limited that should be utilized efficiently. The efficiency of an algorithm is defined as the number of computational resources used by the algorithm.

An algorithm must be analysed to determine its resource usage. The efficiency of an algorithm can be measured based on the usage of different resources.

For maximum efficiency of algorithm, we wish to minimize resource usage. The important resources such as time and space complexity cannot be compared directly, so time and space complexity could be considered for an algorithmic efficiency.

**Average, Best and Worst case analysis:**

When analysing the performance of algorithms, we often consider three different cases: average case, best case, and worst case. Each case provides insights into how an algorithm is expected to behave under different conditions:

**Best Case Analysis:**

The best case analysis considers the scenario in which the algorithm performs at its optimal level. It assumes that the input is structured or behaves in a way that is most favourable for the algorithm.

Best-case time complexity represents the minimum amount of time or resources required to solve a problem of a given size.

Best-case analysis is useful for understanding the lower bounds of an algorithm's performance and for identifying situations in which the algorithm excels.

In practice, best-case scenarios are often theoretical and rarely encountered in real-world situations.

**Worst Case Analysis:**

The worst case analysis considers the scenario in which the algorithm performs at its least favourable level. It assumes that the input is structured or behaves in a way that is most challenging for the algorithm.

Worst-case time complexity represents the maximum amount of time or resources required to solve a problem of a given size.

Worst-case analysis is essential because it guarantees that the algorithm will perform no worse than this level under any input conditions.

In practice, worst-case scenarios help in determining the upper bounds of an algorithm's performance, ensuring that it does not exceed these bounds regardless of input.

**Average Case Analysis:**

The average case analysis considers the expected performance of the algorithm when inputs are randomly distributed or follow a specific probability distribution.

Average-case time complexity represents the expected amount of time or resources required to solve a problem of a given size when considering all possible inputs.

Average-case analysis provides a more realistic assessment of an algorithm's performance under typical or random input conditions.

However, conducting average-case analysis can be challenging and may require knowledge of probability theory and statistics.

In summary, best-case analysis gives us insights into the lower bounds of an algorithm's performance, worst-case analysis provides upper bounds, and average-case analysis offers a more practical view of expected performance under typical input conditions.

Depending on the specific problem and the algorithm's characteristics, one or more of these analyses may be used to evaluate and compare different algorithms.

**Example of Average, Best and worst case analysis:**

**Problem:** Given an unsorted array of integers, find the index of a specific

target element if it exists in the array.

**Algorithm:** Linear Search

Here's how we can analyze the time complexity for each of the three cases:

**Best Case (Target element is the first element of the array):**

Best-case scenario occurs when the element we are searching for is found at the very beginning of the array.

In this case, the algorithm will find the target in the first comparison, and it will take only one comparison.

**Worst Case (Target element is not in the array or is at the end of the array):**

Worst-case scenario occurs when the element we are searching for is not present in the array, or it is at the very end of the array.

In this case, the algorithm needs to compare the target element with all elements in the array until reaching the last element.

**Average Case (Target element is equally likely to be anywhere in the array):**

In average-case analysis, we assume that the target element is equally likely to be at any position in the array.

On average, the algorithm will need to examine half of the elements in the array before finding the target element.