Introduction to an Algorithm

What is an Algorithm?

In computer science, whenever we want to solve some computational problem then we define a set of steps that need to be followed to solve that problem. These steps are collectively known as an **algorithm**.

For example, you have two integers "a" and "b" and you want to find the sum of those two number. How will you solve this? One possible solution for the above problem can be:

- Take two integers as input
- create a variable "*sum*" to store the sum of two integers
- put the sum of those two variables in the "*sum*" variable
- return the "*sum*" variable

```
//taking two integers as input
int findSum(int a, int b)
{
   int sum; // creating the sum variable
   sum = a + b; // storing the sum of a and b
   return sum; // returning the sum variable
}
```

In the above example, you will find three things i.e. input, algorithm, and output:

- **Input:** Input is something for which you need to write an algorithm and transform it into the desired output. Just like in machines where you give some raw product and the machine transforms the raw product into some desirable product. In our example, the input is the two numbers i.e. "a" and "b". Before writing an algorithm, you should find the data type of input, distribution or

range of input and other relevant details related to it. So critically analyze your input before writing the solution.

- **Algorithm:** An algorithm is well-defined steps by step procedure that take some value or set of values as input and produce some value or set of values as output. In the above example, we are having three steps to find the sum of two numbers. So, all three steps are collectively called an algorithm to find the sum of two numbers.
- **Output:** Output is the desired result in the problem. For example, if we are finding the sum of two integers a and b then for every value of a and b it must produce the correct sum as an output.

Properties of an algorithm:

What do you mean by a good Algorithm?

There can be many algorithms for a particular problem. So, how will you classify an algorithm to be good and others to be bad? Let's understand the properties of a good algorithm:

- **Correctness:** An algorithm is said to be correct if for every set of input it halts with the correct output. If you are not getting the correct output for any particular set of input, then your algorithm is wrong.
- **Finiteness:** Generally, people ignore this but it is one of the important factors in algorithm evaluation. The algorithm must always terminate after a finite number of steps. For example, in the case of recursion and loop, your algorithm should terminate otherwise you will end up having a stack overflow and infinite loop scenario respectively.
- **Efficiency:** An efficient algorithm is always used. By the term efficiency, we mean to say that:

1. The algorithm should efficiently use the resources available to the system.
2. The computational time (the time taken to generate an output corresponding to a particular input) should be as less as possible.
3. The memory used by the algorithm should also be as less as possible. Generally, there is a trade-off between computational time and memory. So, we need to find if the time is more important than space or vice-versa and then write the algorithm accordingly.

So, we have seen the three factors that can be used to evaluate an algorithm. Out of these three factors, the most important one is the efficiency of algorithms. So let's dive deeper into the efficiency of the algorithm.

Characteristics of an Algorithm

- An algorithm should be unambiguous.
- It should take well-defined inputs.
- The algorithm should take well-defined outputs.
- It should be simple, generic, and can be executed using the available resources.
- The algorithm must be finite.
- It must be language-independent.

Advantages and Disadvantage of Algorithms

The advantages and disadvantages are mentioned below:

Advantages:

- The algorithms are very easy to understand and can be written in simple language, which anyone can understand.
- Algorithms can be broken down into different pieces, which will be easy to implement practically.
- By using algorithms, we can easily understand the sequence to be followed in processing.

Disadvantages:

- It isn’t easy to convert a complex task into proper algorithms.
- It is a time-consuming process because we need to spend proper time writing the algorithm, and later, we need to implement it in a programming language.
- It’s complicated to show functionalities for each step of introduction to algorithms, and it’s hard to understand each flow in the term for loop and branch.

Algorithm vs Program: Difference between Algorithm and Program

Computer algorithms solve the problem while computer programs implement them in a form that a computer can execute. Here are the main differences between algorithms and programs:

Algorithm	Program
It is a well-defined, step-by-step, logical procedure for solving a given problem.	It refers to a set of instructions for a computer to follow. A program can be an implementation of many algorithms, or a program can even contain no algorithms.
An algorithm provides abstract steps for processing one sequence of related information into a different sequence of derived information.	The constituents of a program may not be conceptually related.
It is written using plain English language and can be understood by those from a non-programming background.	It could be written in any programming language such as C, Python, Java, C++, JavaScript, or any other language, depending on the particular task the program is being designed for.
It can be expressed in natural language, flow charts, pseudocode, and in a variety of programming languages.	We write computer programs in a computer language. Then a compiler or interpreter translates it into a language that is understandable by any computer system.
An algorithm can be executed by a person.	A program is always executed by a computer.

Algorithm Design Techniques

The following is a list of several popular design approaches:

1. Divide and Conquer Approach: It is a top-down approach. The algorithms which follow the divide & conquer techniques involve three steps:

- Divide the original problem into a set of subproblems.
- Solve every sub-problem individually, recursively.
- Combine the solution of the sub-problems (top level) into a solution of the whole original problem.

2. Greedy Technique: Greedy method is used to solve the optimization problem. An optimization problem is one in which we are given a set of input values, which are required either to be maximized or minimized (known as objective), i.e. some constraints or conditions.

- Greedy Algorithm always makes the choice (greedy criteria) looks best at the moment, to optimize a given objective.
- The greedy algorithm doesn't always guarantee the optimal solution however it generally produces a solution that is very close in value to the optimal.

3. Dynamic Programming: Dynamic Programming is a bottom-up approach we solve all possible small problems and then combine them to obtain solutions for bigger problems. This is particularly helpful when the number of copying sub-problems is exponentially large. Dynamic Programming is frequently related to **Optimization Problems**.

4. Branch and Bound: In Branch & Bound algorithm a given subproblem, which cannot be bounded, has to be divided into at least two new restricted subproblems. Branch and Bound algorithm are methods for global optimization in non-convex problems. Branch and Bound algorithms can be slow, however in the worst case they require effort that grows exponentially with problem size, but in some cases we are lucky, and the method coverage with much less effort.

5. Backtracking Algorithm: Backtracking Algorithm tries each possibility until they find the right one. It is a depth-first search of the set of possible solution. During the search, if an alternative doesn't work, then backtrack to the choice point, the place which presented different alternatives, and tries the next alternative.

6. Randomized Algorithm: A randomized algorithm uses a random number at least once during the computation make a decision.

Example 1: In Quick Sort, using a random number to choose a pivot.

Example 2: Trying to factor a large number by choosing a random number as possible divisors