DS Algorithm

What is an Algorithm?

An algorithm is a process, or a set of rules required to perform calculations or some other problem-solving operations especially by a computer. The formal definition of an algorithm is that it contains the finite set of instructions which are being carried in a specific order to perform the specific task. It is not the complete program or code; it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudocode.

Characteristics of an Algorithm

**The following are the characteristics of an algorithm:**

* **Input:** An algorithm has some input values. We can pass 0 or some input value to an algorithm.
* **Output:** We will get 1 or more output at the end of an algorithm.
* **Unambiguity:** An algorithm should be unambiguous which means that the instructions in an algorithm should be clear and simple.
* **Finiteness:** An algorithm should have finiteness. Here, finiteness means that the algorithm should contain a limited number of instructions, i.e., the instructions should be countable.
* **Effectiveness:** An algorithm should be effective as each instruction in an algorithm affects the overall process.
* **Language independent:** An algorithm must be language-independent so that the instructions in an algorithm can be implemented in any of the languages with the same output.

Dataflow of an Algorithm

* **Problem:** A problem can be a real-world problem or any instance from the real-world problem for which we need to create a program or the set of instructions. The set of instructions is known as an algorithm.
* **Algorithm:** An algorithm will be designed for a problem which is a step-by-step procedure.
* **Input:** After designing an algorithm, the required and the desired inputs are provided to the algorithm.
* **Processing unit:** The input will be given to the processing unit, and the processing unit will produce the desired output.
* **Output:** The output is the outcome or the result of the program.

Why do we need Algorithms?

**We need algorithms because of the following reasons:**

* **Scalability:** It helps us to understand the scalability. When we have a big real-world problem, we need to scale it down into small-small steps to easily analyze the problem.
* **Performance:** The real-world is not easily broken down into smaller steps. If the problem, can be easily broken into smaller steps means that the problem is feasible.

Let's understand the algorithm through a real-world example. Suppose we want to make a lemon juice, so following are the steps required to make a lemon juice:

Step 1: First, we will cut the lemon into half.

Step 2: Squeeze the lemon as much you can and take out its juice in a container.

Step 3: Add two tablespoon sugar in it.

Step 4: Stir the container until the sugar gets dissolved.

Step 5: When sugar gets dissolved, add some water and ice in it.

Step 6: Store the juice in a fridge for 5 to minutes.

Step 7: Now, it's ready to drink.

The above real-world can be directly compared to the definition of the algorithm. We cannot perform the step 3 before the step 2, we need to follow the specific order to make lemon juice. An algorithm also says that each and every instruction should be followed in a specific order to perform a specific task.

Now we will look an example of an algorithm in programming.

We will write an algorithm to add two numbers entered by the user.

**The following are the steps required to add two numbers entered by the user:**

Step 1: Start

Step 2: Declare three variables a, b, and sum.

Step 3: Enter the values of a and b.

Step 4: Add the values of a and b and store the result in the sum variable, i.e., sum=a+b.

Step 5: Print sum

Step 6: Stop

Factors of an Algorithm

**The following are the factors that we need to consider for designing an algorithm:**

* **Modularity:** If any problem is given and we can break that problem into small-small modules or small-small steps, which is a basic definition of an algorithm, it means that this feature has been perfectly designed for the algorithm.
* **Correctness:** The correctness of an algorithm is defined as when the given inputs produce the desired output, which means that the algorithm has been designed algorithm. The analysis of an algorithm has been done correctly.
* **Maintainability:** Here, maintainability means that the algorithm should be designed in a very simple structured way so that when we redefine the algorithm, no major change will be done in the algorithm.
* **Functionality:** It considers various logical steps to solve the real-world problem.
* **Robustness:** Robustness means that how an algorithm can clearly define our problem.
* **User-friendly:** If the algorithm is not user-friendly, then the designer will not be able to explain it to the programmer.
* **Simplicity:** If the algorithm is simple then it is easy to understand.
* **Extensibility:** If any other algorithm designer or programmer wants to use your algorithm then it should be extensible.

Importance of Algorithms

1. **Theoretical importance:** When any real-world problem is given to us and we break the problem into small-small modules. To break down the problem, we should know all the theoretical aspects.
2. **Practical importance:** As we know that theory cannot be completed without the practical implementation. So, the importance of algorithm can be considered as both theoretical and practical.

Issues of Algorithms

**The following are the issues that come while designing an algorithm:**

* **How to design algorithms:** As we know that an algorithm is a step-by-step procedure so we must follow some steps to design an algorithm.
* **How to analyze algorithm efficiency**

Approaches of Algorithm

**The following are the approaches used after considering both the theoretical and practical importance of designing an algorithm:**

* **Brute force algorithm:** The general logic structure is applied to design an algorithm. It is also known as an exhaustive search algorithm that searches all the possibilities to provide the required solution. Such algorithms are of two types:
  1. **Optimizing:** Finding all the solutions of a problem and then take out the best solution or if the value of the best solution is known then it will terminate if the best solution is known.
  2. **Sacrificing:** As soon as the best solution is found, then it will stop.
* **Divide and conquer:** It is a very implementation of an algorithm. It allows you to design an algorithm in a step-by-step variation. It breaks down the algorithm to solve the problem in different methods. It allows you to break down the problem into different methods, and valid output is produced for the valid input. This valid output is passed to some other function.
* **Greedy algorithm:** It is an algorithm paradigm that makes an optimal choice on each iteration with the hope of getting the best solution. It is easy to implement and has a faster execution time. But, there are very rare cases in which it provides the optimal solution.
* **Dynamic programming:** It makes the algorithm more efficient by storing the intermediate results. It follows five different steps to find the optimal solution for the problem:
  1. It breaks down the problem into a subproblem to find the optimal solution.
  2. After breaking down the problem, it finds the optimal solution out of these subproblems.
  3. Stores the result of the subproblems is known as memorization.
  4. Reuse the result so that it cannot be recomputed for the same subproblems.
  5. Finally, it computes the result of the complex program.
* **Branch and Bound Algorithm:** The branch and bound algorithm can be applied to only integer programming problems. This approach divides all the sets of feasible solutions into smaller subsets. These subsets are further evaluated to find the best solution.
* **Randomized Algorithm:** As we have seen in a regular algorithm, we have predefined input and required output. Those algorithms that have some defined set of inputs and required output, and follow some described steps are known as deterministic algorithms. What happens that when the random variable is introduced in the randomized algorithm?. In a randomized algorithm, some random bits are introduced by the algorithm and added in the input to produce the output, which is random in nature. Randomized algorithms are simpler and efficient than the deterministic algorithm.
* **Backtracking:** Backtracking is an algorithmic technique that solves the problem recursively and removes the solution if it does not satisfy the constraints of a problem.

The major categories of algorithms are given below:

* **Sort:** Algorithm developed for sorting the items in a certain order.
* **Search:** Algorithm developed for searching the items inside a data structure.
* **Delete:** Algorithm developed for deleting the existing element from the data structure.
* **Insert:** Algorithm developed for inserting an item inside a data structure.
* **Update:** Algorithm developed for updating the existing element inside a data structure.

Algorithm Analysis

The algorithm can be analyzed in two levels, i.e., first is before creating the algorithm, and second is after creating the algorithm. The following are the two analysis of an algorithm:

* Priori Analysis: Here, priori analysis is the theoretical analysis of an algorithm which is done before implementing the algorithm. Various factors can be considered before implementing the algorithm like processor speed, which has no effect on the implementation part.
* Posterior Analysis: Here, posterior analysis is a practical analysis of an algorithm. The practical analysis is achieved by implementing the algorithm using any programming language. This analysis basically evaluate that how much running time and space taken by the algorithm.

Algorithm Complexity

The performance of the algorithm can be measured in two factors:

* **Time complexity:** The time complexity of an algorithm is the amount of time required to complete the execution. The time complexity of an algorithm is denoted by the big O notation. Here, big O notation is the asymptotic notation to represent the time complexity. The time complexity is mainly calculated by counting the number of steps to finish the execution. Let's understand the time complexity through an example.

1. sum=0;
2. // Suppose we have to calculate the sum of n numbers.
3. **for** i=1 to n
4. sum=sum+i;
5. // when the loop ends then sum holds the sum of the n numbers
6. **return** sum;

In the above code, the time complexity of the loop statement will be atleast n, and if the value of n increases, then the time complexity also increases. While the complexity of the code, i.e., return sum will be constant as its value is not dependent on the value of n and will provide the result in one step only. We generally consider the worst-time complexity as it is the maximum time taken for any given input size.

* **Space complexity:** An algorithm's space complexity is the amount of space required to solve a problem and produce an output. Similar to the time complexity, space complexity is also expressed in big O notation.

For an algorithm, the space is required for the following purposes:

1. To store program instructions
2. To store constant values
3. To store variable values
4. To track the function calls, jumping statements, etc.

Auxiliary space: The extra space required by the algorithm, excluding the input size, is known as an auxiliary space. The space complexity considers both the spaces, i.e., auxiliary space, and space used by the input.

So,

**Space complexity = Auxiliary space + Input size.**

Types of Algorithms

**The following are the types of algorithm:**

* **Search Algorithm**
* **Sort Algorithm**

**Search Algorithm**

On each day, we search for something in our day to day life. Similarly, with the case of computer, huge data is stored in a computer that whenever the user asks for any data then the computer searches for that data in the memory and provides that data to the user. There are mainly two techniques available to search the data in an array:

* **Linear search**
* **Binary search**

**Linear Search**

Linear search is a very simple algorithm that starts searching for an element or a value from the beginning of an array until the required element is not found. It compares the element to be searched with all the elements in an array, if the match is found, then it returns the index of the element else it returns -1. This algorithm can be implemented on the unsorted list.

**Binary Search**

A Binary algorithm is the simplest algorithm that searches the element very quickly. It is used to search the element from the sorted list. The elements must be stored in sequential order or the sorted manner to implement the binary algorithm. Binary search cannot be implemented if the elements are stored in a random manner. It is used to find the middle element of the list.

Sorting Algorithms

Sorting algorithms are used to rearrange the elements in an array or a given data structure either in an ascending or descending order. The comparison operator decides the new order of the elements.

Why do we need a sorting algorithm?

* An efficient sorting algorithm is required for optimizing the efficiency of other algorithms like binary search algorithm as a binary search algorithm requires an array to be sorted in a particular order, mainly in ascending order.
* It produces information in a sorted order, which is a human-readable format.
* Searching a particular element in a sorted list is faster than the unsorted list.