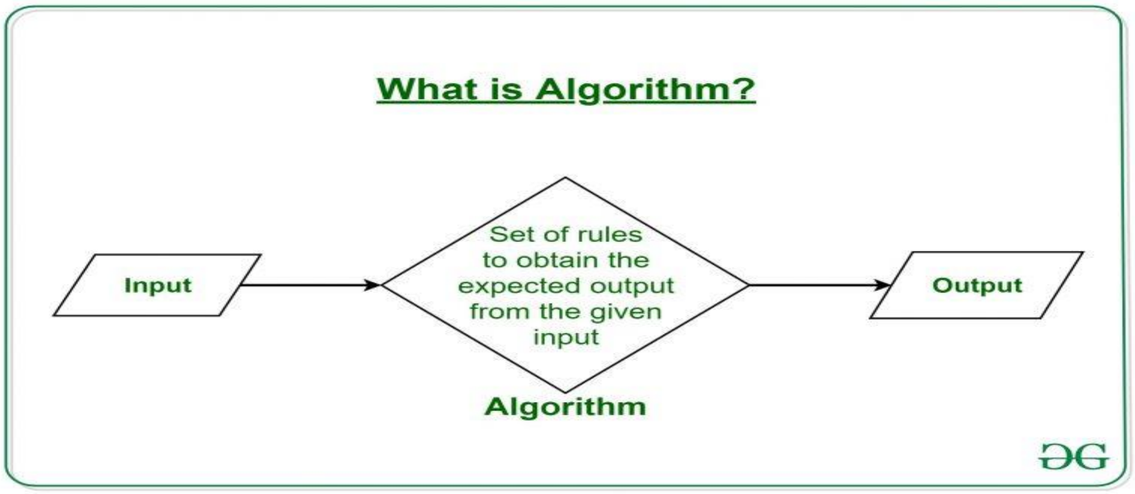
UNIT-1

1) Algorithm

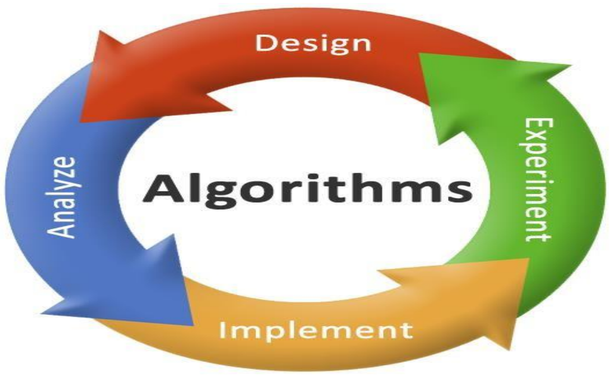
- An algorithm is a set of steps of operations to solve a problem. It performing calculation, data processing, and automated reasoning tasks.

- An algorithm is the best way to represent the solution of a particular problem in a very simple and efficient way.


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

Why study Algorithm?

As the speed of processor increases, performance is frequently said to be less central than other software quality characteristics (e.g. security, extensibility, reusability etc.). However, large problem sizes are commonplace in the area of computational science, which makes performance a very important factor. This is because longer computation time, to name a few mean slower results, less through research and higher cost of computation (if buying CPU Hours from an external party). The study of Algorithm, therefore, gives us a language to express performance as a function of problem size.



<u>An algorithm must have the following properties:</u>

- **Correctness:** It should produce the output according to the requirement of the algorithm
- **Finiteness:** Algorithm must complete after a finite number of instructions have been executed.
- **An Absence of Ambiguity:** Each step must be defined, having only one interpretation.
- **Definition of Sequence:** Each step must have a unique defined preceding and succeeding step. The first step and the last step must be noted.
- **Input/output:** Number and classification of needed inputs and results must be stated.
- **Feasibility:** It must be feasible to execute each instruction.
- **Flexibility:** It should also be possible to make changes in the algorithm without putting so much effort on it.
- **Efficient -** Efficiency is always measured in terms of time and space requires implementing the algorithm, so the algorithm uses a little running time and memory space as possible within the limits of acceptable development time.
- **Independent:** An algorithm should focus on what are inputs, outputs and how to derive output without knowing the language it is defined. Therefore, we can say that the algorithm is independent of language.

<u>Need of Algorithm</u>

1. To understand the basic idea of the problem.
2. To find an approach to solve the problem.
3. To improve the efficiency of existing techniques.
4. To understand the basic principles of designing the algorithms.
5. To compare the performance of the algorithm with respect to other techniques.

6. It is the best method of description without describing the implementation detail.

7. The Algorithm gives a clear description of requirements and goal of the problem to the designer.

8. A good design can produce a good solution.

9. To understand the flow of the problem.

10. To measure the behavior (or performance) of the methods in all cases (best cases, worst cases, average cases)

11. With the help of an algorithm, we can also identify the resources (memory, input-output) cycles required by the algorithm.

12. With the help of algorithm, we convert art into a science.

13. To understand the principle of designing.

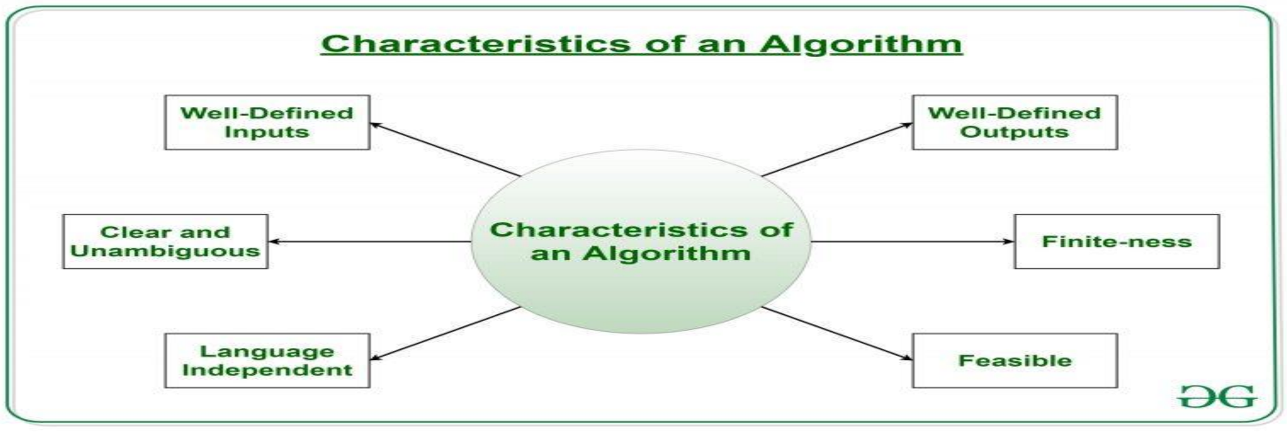
14. We can measure and analyze the complexity (time and space) of the problems concerning input size without implementing and running it; it will reduce the cost of design.

Characteristics of Algorithms

The main characteristics of algorithms are as follows:

- **Clear and Unambiguous**: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs**: The algorithm must clearly define what output will be yielded and it should be well-defined as well.
- **Finite-ness:** The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
- **Feasible:** The algorithm must be simple, generic and practical; such that it can be executed upon will the available resources. It must not contain some future technology, or anything.

- **Language Independent**: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.



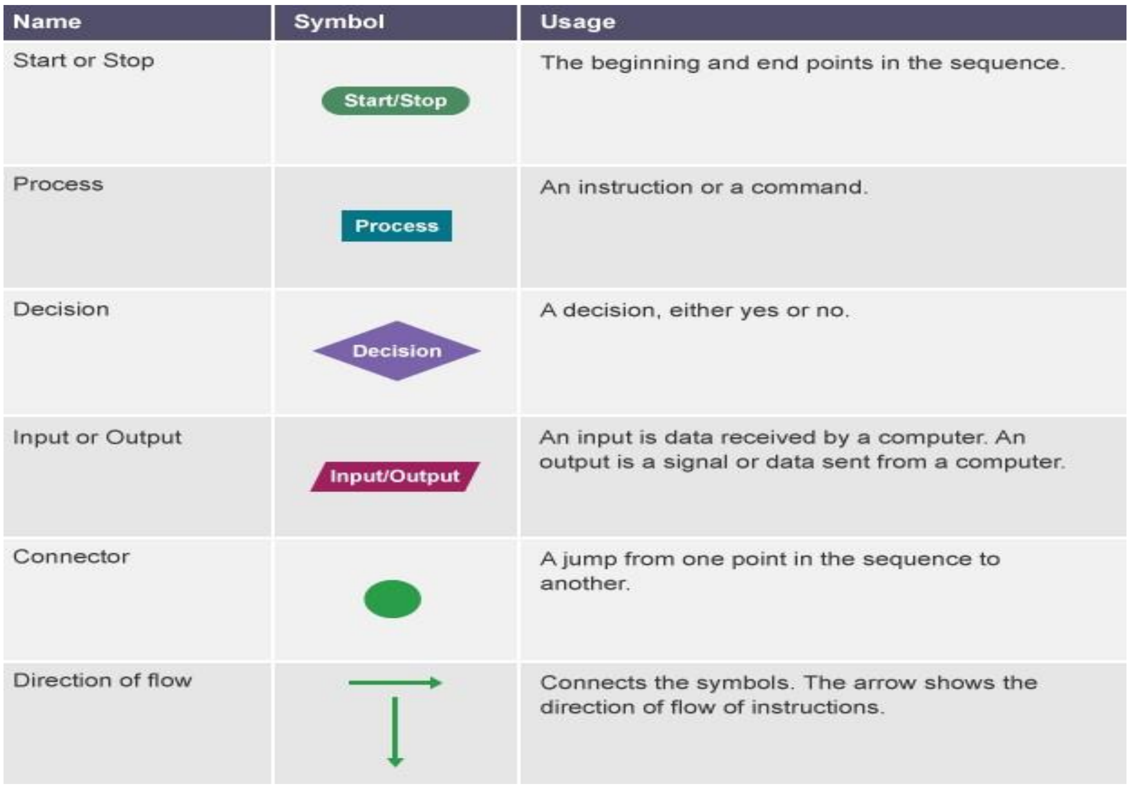
<u>2) Algorithm Specification:</u>

Algorithm can be described in three ways

(i) Natural language: use a natural language like English

(ii) Flow charts: Graphic representations called flowcharts, only if the algorithm is small and simple.

Common flowchart symbols



Name	Symbol	Usage
Start or Stop	Start/Stop	The beginning and end points in the sequence.
Process	Process	An instruction or a command.
Decision	Decision	A decision, either yes or no.
Input or Output	Input/Output	An input is data received by a computer. An output is a signal or data sent from a computer.
Connector	●	A jump from one point in the sequence to another.
Direction of flow	→ ↓	Connects the symbols. The arrow shows the direction of flow of instructions.

(iii) Pseudo-code: also avoids most issues of ambiguity; no particularity on syntax programming language

- Pseudo code is a kind of structure English for describing the algorithm.
- It allows the designer to focus on a logic of the program without being distraction (diverted) by detailed of language.
- Pseudo code is a detailed yet readable description of what a computer program (or) algorithm must do, expressed in a formally styled natural language rather than in a programming language.
- Comments begin with // and continue until the end of the line.
- Blocks are indicated with the matching braces {and}

- An identifier begins with a letter
- The data type of variables are not explicitly declared
- Assignment of values to variables is done using assignment statement

  <Variable> :=< expression>;
- There are two Boolean values TRUE and FALSE.
- Logical operators: AND, OR, NOT
- Relational operators: <, <=,>,>=, =
- The following looping statements are employed:

  while, do-while, for
- Input and Output are done using the instructions read & write

3) Performance Analysis:

Performance analysis of an algorithm depends upon two factors i.e. amount of memory used and amount of compute time consumed on any CPU. Formally they are notified as complexities in terms of:

(i) **Space Complexity**

(ii) **Time Complexity**

Space Complexity of an algorithm is the amount of memory it needs to run to completion i.e. from start of execution to its termination. Space need by any algorithm is the sum of following components:

1. **Fixed Component**: This is independent of the characteristics of the inputs and outputs. This part includes: Instruction Space, Space of simple variables, fixed size component variables, and constants variables.

2. **Variable Component**: This consist of the space needed by component variables whose size is dependent on the particular problems instances(Inputs/Outputs) being solved, the space needed by referenced variables and the recursion stack space is one of the most prominent components. Also this included the data structure components like Linked list, heap, trees, graphs etc.

Therefore the total space requirement of any algorithm 'A' can be provided as

Space (A) = Fixed Components (A) + Variable Components (A)

Among both fixed and variable component the variable part is important to be determined accurately, so that the actual space requirement can be identified for an algorithm 'A'. To identify the space complexity of any algorithm following steps can be followed:

1. Determine the variables which are instantiated by some default values.
2. Determine which instance characteristics should be used to measure the space requirement and this is will be problem specific.
3. Generally the choices are limited to quantities related to the number and magnitudes of the inputs to and outputs from the algorithms.
4. Sometimes more complex measures of the interrelationships among the data items can used.

<u>Example: Space Complexity</u>

```
Algorithm Sum (number, size)\\ procedure will produce sum of all numbers provided in 'number' list
 {
      result=0.0;
      for count = 1 to size do                \\will repeat from 1,2,3,4,. ..size times
           result= result + number[count]; return
      result;
 }
```

In above example, when calculating the space complexity we will be looking for both fixed and variable components. here we have

<u>Fixed components</u> as 'result','count' and 'size' variable there for total space required is three(3) words.

<u>Variable components</u> is characterized as the value stored in 'size' variable (suppose value store in variable 'size 'is 'n'). because this will decide the size of 'number' list and will also drive the for loop. therefore if the space used by size is one word then the total space required by 'number' variable will be 'n'(value stored in variable 'size').

Therefore the space complexity can be written as **Space (Sum) = 3 + n;**

<u>Time Complexity</u> of an algorithm (basically when converted to program) is the amount of computer time it needs to run to completion. The time taken by a program is the sum of the compile time and the run/execution time. The compile time is independent of the instance (problem specific) characteristics. following factors affect the time complexity:

1. Characteristics of compiler used to compile the program.
2. Computer Machine on which the program is executed and physically clocked.
3. Multiuser execution system.
4. Number of program steps.

Therefore the again the time complexity consist of two components fixed(factor 1 only) and variable/instance(factor 2,3 & 4), so for any algorithm 'A' it is provided as:

Time (A) = Fixed Time (A) + Instance Time (A)

Here the number of steps is the most prominent instance characteristics and The number of steps any program statement is assigned depends on the kind of statement like

- comments count as zero steps,
- an assignment statement which does not involve any calls to other algorithm is counted as one step,
- for iterative statements we consider the steps count only for the control part of the statement etc.

Therefore to calculate total number program of program steps we use following procedure. For this we build a table in which we list the total number of steps contributed by each statement. This is often arrived at by first determining the number of steps per execution of the statement and the frequency of each statement executed. This procedure is explained using an example.

<u>Example: Time Complexity</u>

In above example if you analyze carefully frequency of "for count = 1 to size do" it is 'size +1' this is because the statement will be executed one time more die to condition check for false situation of condition provided in for statement. Now once the total steps are calculated they will resemble the instance characteristics in time complexity of algorithm. Also the repeated compile time of an algorithm will also be constant every time we compile the same set of instructions so we can consider this time as constant 'C'.

Therefore the time complexity can be expressed as: **Time (Sum) = C + (2size +3)**

So in this way both the Space complexity and Time complexity can be calculated. Combination of both complexities comprises the Performance analysis of any algorithm and cannot be used independently. Both these complexities also help in defining parameters on basis of which we optimize algorithms.

<u>Asymptotic Notations</u>

Execution time of an algorithm depends on the instruction set, processor speed, disk I/O speed, etc. Hence, we estimate the efficiency of an algorithm asymptotically.

Time function of an algorithm is represented by **T (n)**, where **n** is the input size.

Different types of asymptotic notations are used to represent the complexity of an algorithm. Following asymptotic notations are used to calculate the running time complexity of an algorithm.

O − Big Oh

Ω − Big omega

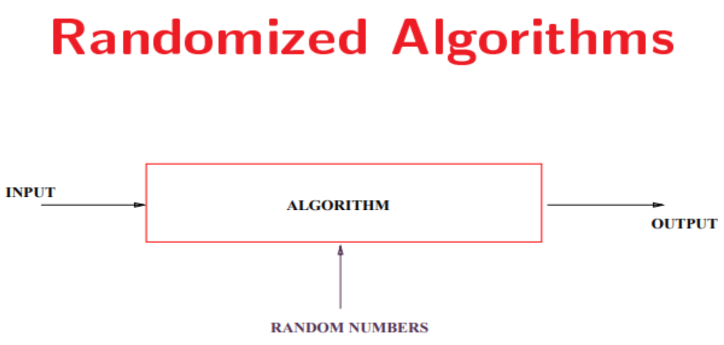
θ − Big theta

o – Little Oh

ω – Little omega

<u>**Randomized Algorithm:**</u>

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm.. For example, in Randomized Quick Sort, we use random number to pick the next pivot (or we randomly shuffle the array). And in Karger's algorithm, we randomly pick an edge.



In addition to input, algorithm takes a source of random numbers and makes random choices during execution . Behavior can vary even on a fixed input.

Simple randomized algorithm contains.

(i) Execution time may vary from run to run

(ii) Different inputs there may be different out comes on each execution

(iii) For same input there may be different out comes on each execution.

Randomized algorithm can be classified into two classes.

1. Las Vegas Algorithm.
2. Monte Carlo Algorithm.

1. Las Vegas Algorithm.

A Las Vegas algorithm runs within a specified amount of time.

If it finds a solution within the time frame the solution will be exactly correct however it is possible that ir runs out of time and does not find any solution.

2. Monte Carlo Algorithm.

A Monte Carlo algorithm is a probabilistic algorithm which depending on the input ha a slight probability of producing an incorrect result or failing to produce a result altogether performance analysis.

REFERENCE:

- Fundamentals of Computer Algorithms, Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, Galgotia Publications, 2015
- Introduction to the Design and Analysis of Algorithms, Anany Levitin, Pearson Education, 2nd Edition.