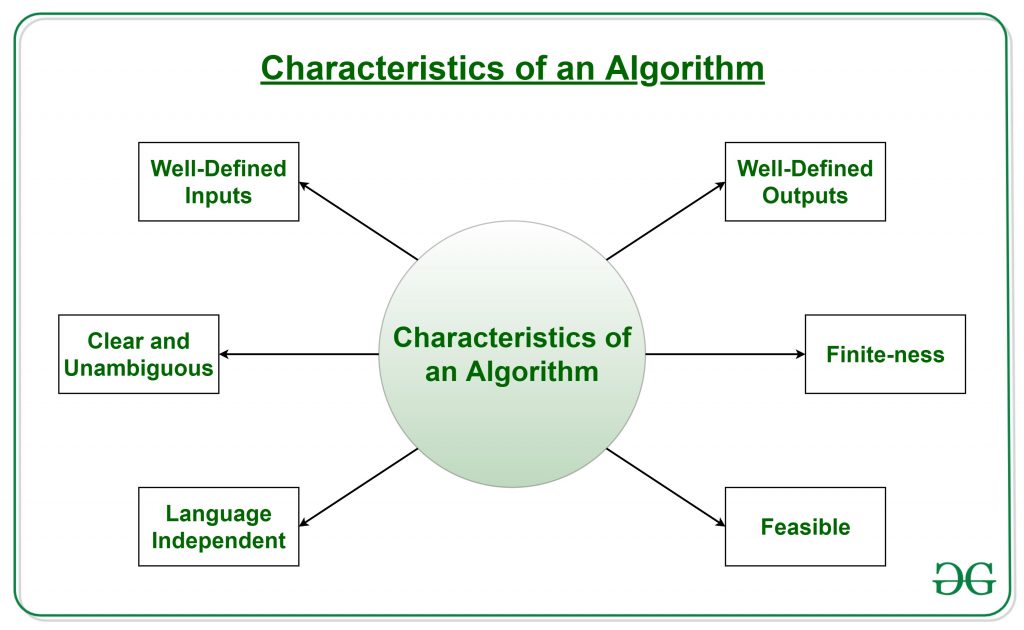
**Algorithm :**

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. 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

**Characteristics of an Algorithm**



* **Clear and Unambiguous**: The 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 terminate after a finite time.
* **Feasible:** The algorithm must be simple,easy and practical, such that it can be executed with 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 the same, as expected

**Advantages of Algorithms:**

* It is easy to understand.
* An algorithm is a step-wise representation of a solution to a given problem.
* In 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

**How to Write an Algorithm?**

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

**OR**

**Step 1** − START ADD

**Step 2** − get values of **a** & **b**

**Step 3** − c ← a + b

**Step 4** − display c

**Step 5** − STOP

**Analysis of algorithm**

Analysis of algorithmprovides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

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.Analysis of algorithms is the determination of the amount of time and space resources required to execute it. However, the main concern of analysis of algorithms is the required time or performance. Generally, we perform the following types of analysis −

**Types of Algorithm Analysis:**

1. Best case
2. Worst case
3. Average case

* **Best case:**Define the input for which algorithm takes less time or minimum time. In the best case calculate the lower bound of an algorithm. Example: In the linear search when search data is present at the first location of large data then the best case occurs.
* **Worst Case**: Define the input for which algorithm takes a long time or maximum time. In the worst calculate the upper bound of an algorithm. Example: In the linear search when search data is not present at all then the worst case occurs.
* **Average case**: In the average case take all random inputs and calculate the computation time for all inputs.  
  And then we divide it by the total number of inputs.

**What Is Time Complexity?**

Time complexity is defined in terms of how many times it takes to run a given algorithm, based on the length of the input. Time complexity is not a measurement of how much time it takes to execute a particular algorithm because such factors as programming language, operating system, and processing power are also considered.

Time complexity is a type of computational complexity that describes the time required to execute an algorithm. The time complexity of an algorithm is the amount of time it takes for each statement to complete. As a result, it is highly dependent on the size of the [processed data](https://www.simplilearn.com/what-is-data-processing-article). It also aids in defining an algorithm's effectiveness and evaluating its performance.

**What Is Space Complexity?**

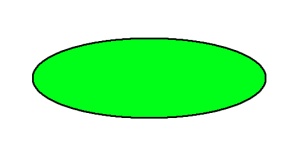
When an algorithm is run on a computer, it necessitates a certain amount of memory space. The amount of memory used by a program to execute it is represented by its space complexity. Because a program requires memory to store input data and temporal values while running, the space complexity is auxiliary and input space.

**Flowchart**

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.   
The process of drawing a flowchart for an algorithm is known as “flowcharting”.

**Basic Symbols used in Flowchart Designs**

1. **Terminal:** The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.



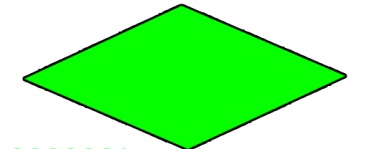
* **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.



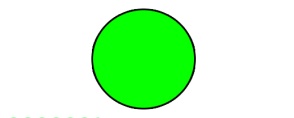
* **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



* **Decision** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.



* **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.



* **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

**Rules For Creating Flowchart :**

A flowchart is a graphical representation of an algorithm. it should follow some rules while creating a flowchart  
Rule 1: Flowchart opening statement must be ‘start’ keyword.  
Rule 2:  Flowchart ending statement must be ‘end’ keyword.  
Rule 3: All symbols in the flowchart must be connected with an arrow line.  
Rule 4: The decision symbol in the flowchart cannot be associated with the arrow line.

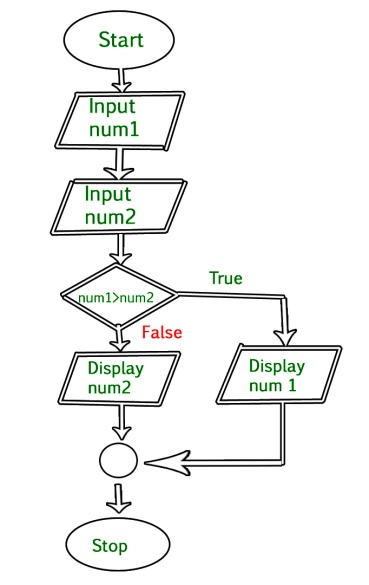
**Advantages of Flowchart:**

* Flowcharts are a better way of communicating the logic of the system.
* Flowcharts act as a guide for blueprint during program designed.
* Flowcharts help in debugging process.
* With the help of flowcharts programs can be easily analyzed.
* It provides better documentation.
* Flowcharts serve as a good proper documentation.
* Easy to trace errors in the software.
* Easy to understand.
* The flowchart can be reused for inconvenience in the future.
* It helps to provide correct logic.

**Disadvantages of Flowchart:**

* It is difficult to draw flowcharts for large and complex programs.
* There is no standard to determine the amount of detail.
* Difficult to reproduce the flowcharts.
* It is very difficult to modify the Flowchart.
* Making a flowchart is costly.
* Some developer thinks that it is waste of time.
* It makes software processes low.
* If changes are done in software, then the flowchart must be redrawn.

**Example :** **Draw a flowchart to input two numbers from the user and display the largest of two numbers** 



The data structure name indicates itself that organizing the data in memory. We can say that Data Structure is a way to store and organize data so that it can be used efficiently. The data structure is not any programming language like C, C++, java, etc. It is a set of algorithms that we can use in any programming language to structure the data in the memory.To structure the data in memory, 'n' number of algorithms were proposed, and all these algorithms are known as Abstract data types. These abstract data types are the set of rules.

For example, we can store a list of items having the same data-type using the *array* data structure.



There are many ways of organizing the data in the memory but one of the data structures, i.e., array in C language.

### Advantages of Data structures

**The following are the advantages of a data structure:**

* **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
* **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
* **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

**Types of Data Structure**

Basically, data structures are divided into two categories:

* Linear data structure
* Non-linear data structure

**Linear data structures**

The arrangement of data in a sequential manner is known as a linear data structure. The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues. In these data structures, one element is connected to only one another element in a linear form.Since elements are arranged in particular order, they are easy to implement.

However, when the complexity of the program increases, the linear data structures might not be the best choice because of operational complexities.

**Popular linear data structures are:**

### 1. Array Data Structure

In an array, elements in memory are arranged in continuous memory. All the elements of an array are of the same type. And, the type of elements that can be stored in the form of arrays is determined by the programming language.

### 2. Stack Data Structure

In stack data structure, elements are stored in the LIFO principle. That is, the last element stored in a stack will be removed first.

It works just like a pile of plates where the last plate kept on the pile will be removed first.

### 3. Queue Data Structure

Unlike stack, the queue data structure works in the FIFO principle where first element stored in the queue will be removed first.

It works just like a queue of people in the ticket counter where first person on the queue will get the ticket first.

### 4. Linked List Data Structure

In linked list data structure, data elements are connected through a series of nodes. And, each node contains the data items and address to the next node.

**Non linear data structures**

Unlike linear data structures, elements in non-linear data structures are not in any sequence. Instead they are arranged in a hierarchical manner where one element will be connected to one or more elements.

Non-linear data structures are further divided into graph and tree based data structures.

Non-linear data structures are further divided into graph and tree based data structures.

### 1. Graph Data Structure

In graph data structure, each node is called vertex and each vertex is connected to other vertices through edges.

To learn more, visit [Graph Data Structure](https://www.programiz.com/dsa/graph).

### 2. Trees Data Structure

Similar to a graph, a tree is also a collection of vertices and edges. However, in tree data structure, there can only be one edge between two vertices.