## 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.

## Introduction

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are widely used in almost every aspect of Computer Science i.e. Operating System, Compiler Design, Artifical intelligence, Graphics and many more.

Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way. It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible.

### Basic Terminology

Data structures are the building blocks of any program or the software. Choosing the appropriate data structure for a program is the most difficult task for a programmer. Following terminology is used as far as data structures are concerned.

**Data:** Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

**Group Items:** Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

**Record:** Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

**File:** A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

**Attribute and Entity:** An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

**Field:** Field is a single elementary unit of information representing the attribute of an entity.

### Need of Data Structures

As applications are getting complexed and amount of data is increasing day by day, there may arrise the following problems:

**Processor speed:** To handle very large amout of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

**Data Search:** Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

**Multiple requests:** If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process

in order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

### Advantages of Data Structures

**Efficiency:** Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a perticular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

**Reusability:** Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

**Abstraction:** Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

### Data Structure Classification

**Linear Data Structures:** A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element.

Types of Linear Data Structures are given below:

**Arrays:** An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

The elements of array share the same variable name but each one carries a different index number known as subscript. The array can be one dimensional, two dimensional or multidimensional.

The individual elements of the array age are:

age[0], age[1], age[2], age[3],......... age[98], age[99].

**Linked List:** Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.

**Stack:** Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**.

A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

**Queue:** Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**.

It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.

**Non Linear Data Structures:** This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure.

Types of Non Linear Data Structures are given below:

**Trees:** Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the herierchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.

Tree data structure is based on the parent-child relationship among the nodes. Each node in the tree can have more than one children except the leaf nodes whereas each node can have atmost one parent except the root node. Trees can be classfied into many categories which will be discussed later in this tutorial.

**Graphs:** Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree can not have the one.

### Operations on data structure

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

**Example:** If we need to calculate the average of the marks obtained by a student in 6 different subject, we need to traverse the complete array of marks and calculate the total sum, then we will devide that sum by the number of subjects i.e. 6, in order to find the average.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.

If the size of data structure is **n** then we can only insert **n-1** data elements into it.

3) **Deletion:**The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

If we try to delete an element from an empty data structure then **underflow** occurs.

4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search. We will discuss each one of them later in this tutorial.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging

# Primitive vs non-primitive data structure

Data structure means organizing the data in the memory. The data can be organized in two ways either linear or non-linear way.

There are two types of data structure available for the programming purpose:

* Primitive data structure
* Non-primitive data structure

Primitive data structure is a fundamental type of data structure that stores the data of only one type whereas the non-primitive data structure is a type of data structure which is a user-defined that stores the data of different types in a single entity.

In the above image, we can observe the classification of the data structure. The [data structure](https://www.javatpoint.com/data-structure-tutorial)

is classified into two types, i.e., primitive and non-primitive data structure. In the case of primitive data structure, it contains fundamental data types such as integer, float, character, pointer, and these fundamental data types can hold a single type of value. For example, integer variable can hold integer type of value, float variable can hold floating type of value, character variable can hold character type of value whereas the pointer variable can hold pointer type of value.

42.3M

875

Exception Handling in Java - Javatpoint

In the case of non-primitive data structure, it is categorized into two parts such as linear data structure and non-linear data structure. Linear data structure is a sequential type of data structure, and here sequential means that all the elements in the memory are stored in a sequential manner; for example, element stored after the second element would be the third element, the element stored after the third element would be the fourth element and so on. We have different linear data structures holding the sequential values such as [Array](https://www.javatpoint.com/data-structure-array)

, [Linked list](https://www.javatpoint.com/ds-linked-list)

, [Stack](https://www.javatpoint.com/data-structure-stack)

, [Queue](https://www.javatpoint.com/data-structure-queue)

.

Non-linear data structure is a kind of random type of data structure. The non-linear data structures are Tree and Graph.

### Primitive data structure

Primitive data structure is a data structure that can hold a single value in a specific location whereas the non-linear data structure can hold multiple values either in a contiguous location or random locations

The examples of primitive data structure are float, character, integer and pointer. The value to the primitive data structure is provided by the programmer. The following are the four primitive data structures:

* **Integer:** The integer data type contains the numeric values. It contains the whole numbers that can be either negative or positive. When the range of integer data type is not large enough then in that case, we can use long.
* **Float:** The float is a data type that can hold decimal values. When the precision of decimal value increases then the Double data type is used.
* **Boolean:** It is a data type that can hold either a True or a False value. It is mainly used for checking the condition.
* **Character:** It is a data type that can hold a single character value both uppercase and lowercase such as 'A' or 'a'.

### Non-primitive data structure

The non-primitive data structure is a kind of data structure that can hold multiple values either in a contiguous or random location. The non-primitive data types are defined by the programmer. The non-primitive data structure is further classified into two categories, i.e., linear and non-linear data structure.

In case of linear data structure, the data is stored in a sequence, i.e., one data after another data. When we access the data from the linear data structure, we just need to start from one place and will find other data in a sequence.

**The following are the types of linear data structure:**

* **Array:** An array is a data structure that can hold the elements of same type. It cannot contain the elements of different types like integer with character. The commonly used operation in an array is insertion, deletion, traversing, searching.

**For example:**

int a[6] = {1,2,3,4,5,6};

The above example is an array that contains the integer type elements stored in a contiguous manner.

* **String:** String is defined as an array of characters. The difference between the character array and string is that the string data structure terminates with a 'NULL' character, and it is denoted as a '\0'.

**String data structure:**

1. **char** name[100] = "Hello javaTpoint";

In the above example, the length of the string is 17 as the last character is the NULL character which denotes the termination of the string.

**Char Representation:**

1. **char** name[100] = {'H', 'e', 'l','l','o',' ', 'j', 'a', 'v', 'a', 't','p', 'o', 'i', 'n', 't' }

In the above example, the length of the string is 16 as it does not have any NULL character as the last character to denote the termination.

* **Stack:** Stack is a data structure that follows the principle **LIFO** (Last In First Out). All the operations on the stack are performed from the top of the stack such as PUSH and POP operation. The push operation is the process of inserting element into the stack while the pop operation is the process of removing element from the stack. The stack data structure can be implemented by using either array or linked list.
* **Queue:** Queue is a data structure that can be implemented by using array. The difference between the stack and queue data structure is that the elements in the queue are inserted from the rear end while the elements in the queue are removed from the front end.

# Array

## Definition

* Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
* Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
* Array is the simplest data structure where each data element can be randomly accessed by using its index number.
* For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

Advantages of Arrays

* Arrays represent multiple data items of the same type using a single name.
* In arrays, the elements can be accessed randomly by using the index number.
* Arrays allocate memory in contiguous memory locations for all its elements. Hence there is no chance of extra memory being allocated in case of arrays. This avoids memory overflow or shortage of memory in arrays.
* Using arrays, other data structures like linked lists, stacks, queues, trees, graphs etc can be implemented.
* Two-dimensional arrays are used to represent matrices.

Disadvantages of Arrays

* The number of elements to be stored in an array should be known in advance.
* An array is a static structure (which means the array is of fixed size). Once declared the size of the array cannot be modified. The memory which is allocated to it cannot be increased or decreased.
* Insertion and deletion are quite difficult in an array as the elements are stored in consecutive memory locations and the shifting operation is costly.
* Allocating more memory than the requirement leads to wastage of memory space and less allocation of memory also leads to a problem.

Applications of Arrays

1) Array stores data elements of the same data type.

2) Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoid the confusion of using multiple variables.

3) Arrays can be used for sorting data elements. Different sorting techniques like Bubble sort, Insertion sort, Selection sort etc use arrays to store and sort elements easily.

4) Arrays can be used for performing matrix operations. Many databases, small and large, consist of one-dimensional and two-dimensional arrays whose elements are records.

5) Arrays can be used for CPU scheduling.

6) Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps, Hash tables etc.

**Example:**

You use arrays all the time in programming. Whenever you have to keep track of an ordered list of items, you will end up using an array. Be it a list of songs, a list of books, or list of anything for that matter. Even in the JSON data format, you will often use an array to hold a list of objects.

Databases generally provide a different functionality, as in they let you keep data over time. You can retrieve that data and update it and view it from other computers using other web browsers. You will often retrieve information from a database and store it for local processing in an array. So arrays is an important concept in programming languages.

Advantages and Disadvantages of Linked List

Difficulty Level : Easy

Last Updated : 28 Mar, 2022

There are many data structures like arrays, linked lists, etc. Each sort of arrangement has its strengths and weaknesses. For these reasons, it’s important to know the benefits and drawbacks of different data structures when it comes to designing, optimizing, and scaling programs. In this article, we will discuss the advantages and disadvantages of the linked list.

Linked List:

A Linked list is a dynamic arrangement that contains a “link” to the structure containing the subsequent items. It’s a set of structures ordered not by their physical placement in memory (like an array) but by logical links that are stored as a part of the info within the structure itself.

A linked list is another way to collect similar data. However, unlike an array, elements during a linked list aren’t in consecutive memory locations. A linked list consists of nodes that are connected with one another using pointers. The figure illustrates a linked list.

Types Of Linked List:

Singly Linked List: It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows the traversal of data only in one way.

Doubly or Two Way Linked List: A doubly linked list or a two-way linked list is a more complex type of linked list that contains a pointer to the next as well as the previous node in sequence, Therefore, it contains three parts are data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well.

Circular Linked List: A circular linked list is that in which the last node contains the pointer to the first node of the list. While traversing a circular linked list, one can begin at any node and traverse the list in any direction forward and backward until reaching the same node where started. Thus, a circular linked list has no beginning and no end.

Circular Doubly Linked List: A Doubly Circular linked list or a circular two-way linked list is a more complex type of linked-list that contains a pointer to the next as well as the previous node in the sequence. The difference between the doubly linked and circular doubly list is the same as that between a singly linked list and a circular linked list. The circular doubly linked list does not contain null in the previous field of the first node.

Advantages Of Linked List:

Dynamic data structure: A linked list is a dynamic arrangement so it can grow and shrink at runtime by allocating and deallocating memory. So there is no need to give the initial size of the linked list.

No memory wastage: In the Linked list, efficient memory utilization can be achieved since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre-allocate the memory.

Implementation: Linear data structures like stack and queues are often easily implemented using a linked list.

Insertion and Deletion Operations: Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.

Disadvantages Of Linked List:

Memory usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.

Traversal: In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index. For example, for accessing a node at position n, one has to traverse all the nodes before it.

Reverse Traversing: In a singly linked list reverse traversing is not possible, but in the case of a doubly-linked list, it can be possible as it contains a pointer to the previously connected nodes with each node. For performing this extra memory is required for the back pointer hence, there is a wastage of memory.

Random Access: Random access is not possible in a linked list due to its dynamic memory allocation.