**Manasi Patil**

**What is Data Structure -**

A data structure is a particular way of organizing data in a computer so that it can be used effectively.

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 as we have already seen one of the data structures, i.e., array in C language.   
Array is a collection of memory elements in which data is stored sequentially, i.e., one after another  
  
  
When we think of data structure, there are generally four forms:  
1. Linear: arrays, lists  
2.Tree: binary, heaps, space partitioning etc.  
3. Hash: distributed hash table, hash tree etc.  
4. Graphs: decision, directed, acyclic etc.

Arrays: An array is a finite group of data, which is allocated contiguous (i.e., sharing a common border) memory locations, and each element within the array is accessed via an index key (typically numerical, and zero based).

Lists: A linked list is different to an array in that the order of the elements within the list are not determined by a contiguous memory allocation. Instead, the elements of the linked list can be sporadically placed in memory due to its design, which is that each element of the list (also referred to as a ‘node’) consists of two parts:  
1. the data  
2. a pointer  
The data is what you’ve assigned to that element/node, whereas the pointer is a memory address reference to the next node in the list.

Tree: The concept of a ‘tree’ in its simplest terms is to represent a hierarchical tree structure, with a root value and subtrees of children (with a parent node), represented as a set of linked nodes.

A tree contains “nodes” (a node has a value associated with it) and each node is connected by a line called an “edge”. These lines represent the relationship between the nodes.  
  
There are various incarnations of the basic tree structure, each with their own unique characteristics and performance considerations:  
• Binary Tree  
• Binary Search Tree  
• Red-Black Tree  
• B-tree  
• Weight-balanced Tree  
• Heap  
• Abstract Syntax Tree

Hash: A hash table is a data structure which is capable of mapping ‘keys’ to ‘values’, and you’ll typically find this is abstracted and enhanced with additional behaviours by many high-level programming languages such that they behave like an ‘[associative array](https://en.wikipedia.org/wiki/Associative_array)’ abstract data type.

Graphs: A graph is an abstract data type intended to guide the implementation of a data structure following the principles of [graph theory](https://en.wikipedia.org/wiki/Graph_theory)  
  
The data structure itself is non-linear and it consists of:

• nodes: points on the graph (also known as ‘vertices’).  
• edges: lines connecting each node.

**Classification of Data Structure**

Classification of Data Structure are normally divided into two broad categories:

1.Primitive Data Structure

2.Non – Primitive Data Structure

Primitive Data Structure:

The primitive data structures are primitive data types. The int, char, float, double and pointer are the primitive data structures.

These are basic structures and are directly operated upon by the machine instructions.

These to general have different representations on different computers, Integer, Floating point numbers, character-constants, string constants, pointers, etc.

 Non-Primitive Data Structure:

The non-primitive data structure is derived from primitive data structure.

Generally, there is a way of defining our own data types and such data types comes under the non-primitive data structure.

these are the more sophisticated data structure. These are derived from the primitive data structure.

The non-primitive data structures emphasize the structuring of a group of homogeneous or heterogeneous data items.

**Bhavin Patil**

**Algorithm Definition and Complexity**

# Definition

# An algorithm is a step-by-step representation for solving an instance of a problem. Every single procedure that a computer performs is an algorithm. It is a precise procedure for solving a problem in finite number of steps and states the action to be executed and the order in which these actions to be executed. An algorithm is a well-ordered collection of clear and simple instructions of definite and effectively computable operations that when executed produces a result and stops executing at some point in a finite amount of time rather than just going on and on infinitely

# . Not all procedures can be called an algorithm. An algorithm should have the following characteristics

## Unambiguous − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.

## Input − An algorithm should have 0 or more well-defined inputs.

## Output − An algorithm should have 1 or more well-defined outputs, and should match the desired output.

## Finiteness − Algorithms must terminate after a finite number of steps.

## Feasibility − Should be feasible with the available resources.

## Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

# Complexity

# Some algorithms are more efficient than others. We would prefer to choose an efficient algorithm, so it would be nice to have metrics for comparing algorithm efficiency. The *complexity* of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process.

## There are two main complexity measures of the efficiency of an algorithm:

## •Time Complexity

## •Space Complexity

# Time complexity Time complexity of an algorithm represents the amount of time required by the algorithm to complete the procedure Time complexity of a given algorithm can be defined for computation of function as a total number of statements that are executed for computing the value. time complexity of an algorithm is generally classified as three types: worst case : It is the longest time that as algorithm will use over all instances of size n for a given problem to produce a desired result. average case :It is the average time that the algorithm will use to produce a desired result for given problem. it depends on probability distrubution. best case: It is the shortest time that the algorithm will use for given problem to produce the desired result.

# Space complexity

## Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. Space complexity of a problem is the amount of memory consumed by the algorithm until it completes its execution. the amount of storage space required by the algorithm varies with the size of the problem to be solved. The space required by an algorithm is equal to the sum of the following two components –

## • A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc. • A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc. Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I

**Operations on Data Structure**

Data structure is defined as a mathematical or logical model to store data. The operations are data structure functions which is used to process the stored data. All the data structure has some common operations used to manipulate and process the stored data for the user.

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 divide 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

**Uma Thakur**

**Data Structure for Languages**

Most software systems are built on the top of predefined components. In this way, softwarearchitects can concentrate on the high-level design aspects and, as a result, save development costs and time. Data structures libraries are an important kind of software components that define interfaces and implement fundamental data structures and algorithms. Nowadays, data structures libraries are very accessible because they are included as part of most programming languages

In order to ease the specialized algorithm and data structure, they can be included in data structure libraries.

Library is the place where the actual functionality is implemented i.e., they contain function body. Libraries have mainly two categories:

• Static

• Shared or Dynamic

Static: Static libraries contains object code linked with an end user application and then they become the part of the executable. These libraries are specifically used at *compile time* which means the library should be present in correct location when user wants to compile his/her C or C++ program. In windows they end with .lib extension and with. a

for MacOS.

Shared or Dynamic: These libraries are only required at run*-time* i.e.; user can compile his/her code without using these libraries. In short, these libraries are linked against at compile time to resolve undefined references and then its distributed to the application so that the application can load it at run time. For example, when we open our game folders, we can find many .dll (dynamic link libraries) files. As these libraries can be shared by multiple programs, they are also called as shared libraries.These files end with .dll or .lib extensions. In windows they end with .dll extension.

Example: Math.h is a header file which includes the prototype for function calls like sqrt (), pow () etc, whereas  libm.lib, libmmd.lib, libmmd.dll are some of the math libraries. In simple terms a header file is like a visiting card and libraries are like a real person, so we use visiting card (Header file) to reach to the actual person (Library).

**Languages for Data Structure**

Data structures and algorithms in the context of computer science are essential because of the key role that these concepts play in programming. Most software programs will use these notions to manipulate data, and therefore, the study of these methods is crucial for the proper understanding of code.

A data structure is a way of organizing computer memory space in a mathematical fashion, enabling efficient access to the values therein

we implement this method to achieve the most basic tasks for a data structure.  
  
In order to complete these tasks, we manage memory using pointers and dynamic memory allocation. In object-oriented programming, the management of memory is often done at a class level through our constructors and destructors. This general idea of freeing used space when we don’t need it is called garbage collection. Ideally, we try to form efficient algorithms that only change what’s necessary to complete the tasks  
  
It’s important to note that different programming languages have their own advantages and disadvantages.

We implements methods to complete the most basic tasks like insertion, deletion, searching for data structure.

Some most common programming language:  
C requires us to be explicit with our coding, which makes it the perfect language for someone that wants to learn how to implement things from scratch. However, it’s not object-oriented, and it requires us to manually de-allocate the memory we used up later in the code*;* there’s no garbage collection here. Also note that we have to include libraries to work with strings, amongst other things.  
  
  
C++   
This language is widely used as part of many operating systems, as well as in the making of efficient video games and machine learning tools. It can be seen as the object-oriented version of C. This means that C++ will support abstraction, polymorphism, encapsulation, and inheritance.  
  
Java

Java is a high-level language based on C++. It’s used in all the same fields as C++, and we can also see that the syntax is very similar. Note that this language includes a lot more functions from the start, without needing to include any external libraries. In this language, we also don’t have to include any libraries to work with strings. We can begin our code by defining a class, just like in C++.  
  
Python   
python is another high-level language that’s used in a wide variety of fields, from machine learning to web applications. There’s no need to specify that we’re using a pointer for this and a string for that, as the language is loosely type. Just like with Java, we don’t need to import anything to work with strings.  
  
JavaScript

JavaScript is a programming language that’s mostly used in web development. Attention to detail matters a little bit more with this language than with Python.

all of these languages are able to perform basic data structure tasks. In order to pick the best language, it’s helpful to discuss what and how exactly we plan to program.

<https://www.baeldung.com/cs/languages-learn-data-structures>

**Chaitanya**

**Importance of Data Structure**

Data structure

provides the right way to organize information in the digital space. The data structure is a key component of Computer Science and is largely used in the areas of Artificial Intelligence, operating systems, graphics, etc.

In computer science, the Importance of data structure is everywhere. Data structure provides basic stuff to resolve problems. I   
Data Structure is the combination of data objects which gives a way of collecting and handling data in the computer so that it can be used. Their various types are arrays, Linked List, Stack, Queue, etc.

Its importance can be understood by the following:  
  
1. Handling complexity:  
Increase in complexities in computer algorithms, the volume of data usage is rising; this can affect the execution of the application and can create remarkable areas of concern like processing speed, data search, and multiple requests. To counter these data structures are used.   
  
2. Systematic memory use:  
Systematic application of data structure memory usage can be optimized, e.g., we can use linked list vs. arrays when we are not particular about the data size. When there is no longer use of memory, it can be cleared.   
  
3. Ability to reuse:  
Once we have executed a particular data structure, we can reuse it in any distinct position. Implementation of data structures can be assembled into libraries that can be utilized by various clients.   
  
4. Abstraction:  
Data structure acts as the foundation of abstract data types; the data structure describes the physical form of Abstract Data Type. In ADT, the set of operations is supposed to be understood, and the data structure provides physicality to them.

<https://www.youtube.com/watch?v=jc1t0KFsOcs>

**Application of Data Structure**

Data structure is a particular way of organizing data in computer so that it can used effectively. Data structure offers many advantages to IT-related processes,

Different types of Data structure are used for different kinds of purpose. Some of them are highly specialized in specific task and is a specialized format for organizing and storing the data in.

Arrays

Arrays are the simplest data structures that stores items of the same data type. A basic application of Arrays can be storing data in tabular format. For example, if we wish to store the contacts on our phone, then the software will simply place all our contacts in an array. • Similarly, songs playlist in our music player. • 2-Dimensional Arrays also called Matrix, are used in the processing of an image. • These are also used in the Online ticket booking system – if a user wants to book a seat in A-9, the array becomes seat[A][9] or seat [1][9].

Linked List

A Linked List is a sequence data structure, which connects elements, called nodes, through links.  1. Images are linked with each other. So, an image viewer software uses a linked list to view the previous and the next images using the previous and next buttons. 2. Web pages can be accessed using the previous and the next URL links which are linked using linked list.3. The music players also use the same technique to switch between music.4. To keep the track of turns in a multi-player game, a [circular linked list](https://www.geeksforgeeks.org/circular-linked-list/) is used. 

Stack  
A stack is defined as a linear list in which insertions and deletions take place at the same end based on the Last-In-First-Out (LIFO) strategy. This end is called the top of the stack and the other end is called the bottom of the stack.

Applications Of Stack:  
Stacks are useful for solving many problems in computer science. One of the most important is to store the return address in a function call-return structure of a compiler.  
To check for left and right parenthesis match in an expression.  
To evaluate a postfix expression.  
To convert an infix expression to postfix or suffix expression.

Some Applications of a stack are:   
1. Converting infix to postfix expressions.  
2. Undo operation is also carried out through stacks.  
3. Syntaxes in languages are parsed using stacks.  
4. It is used in many virtual machines like [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/).  
5. Forward – backward surfing in browser  
6. History of visited websites  
7. Message logs and all messages you get are arranged in stack  
8. Call logs, E-mails, Google photos’ any gallery, YouTube downloads, Notifications (latest appears first)  
9. Scratch cards earned after Google pay transaction

Queues  
A queue is a linear list in which additions and deletions take place at two different ends. Insertions take place at the rear end and deletions take place in the front end.

Applications Of Queues:  
• Priority queues are used in browsers while downloading multiple files.  
• Used in call centers where calls of people are put on hold while the person is in another call.  
• Used by printer software.  
• CPU for task scheduling.

Some applications of a queue are:   
1. [Operating System](https://www.geeksforgeeks.org/types-of-operating-systems/) uses queue for job scheduling.  
2. To handle congestion in networking queue can be used.  
3. Data packets in communication are arranged in queue format.  
4. Sending an E-mail, it will be queued  
5. server while responding request  
6. Uploading and downloading photo’s, first kept for uploading/downloading will completed first (Not if there is threading)  
7. Most of internet requests and processes uses queue  
8. While switching multiple applications, windows uses circular queue.