**What is Data Structure?**

* A data structure is defined as a particular way of storing and organizing data in our devices to use the data efficiently and effectively.
* The main idea behind using data structures is to minimize the time and space complexities.
* An efficient data structure takes minimum memory space and requires minimum time to execute the data.

**What is Algorithm?**

* Algorithm is defined as a process or set of well-defined instructions that are typically used to solve a particular group of problems or perform a specific type of calculation.
* To explain in simpler terms, it is a set of operations performed in a step-by-step manner to execute a task.

**[How to start learning DSA?](https://www.geeksforgeeks.org/how-to-start-learning-dsa/" \l "startdsa)**

The complete process to learn DSA from scratch can be broken into 4 parts:

1. Learn about Time and Space complexities
2. Learn the basics of individual Data Structures
3. Learn the basics of Algorithms
4. Practice Problems on DSA

**1. Learn about Complexities**

The primary motive to use DSA is to solve a problem effectively and efficiently. How can you decide if a program written by you is efficient or not? This is measured by complexities. Complexity is of two types:

[**Time** **Complexity**](https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/): Time complexity is used to measure the amount of time required to execute the code.

[**Space** **Complexity**](https://www.geeksforgeeks.org/g-fact-86/): Space complexity means the amount of space required to execute successfully the functionalities of the code.

 **Auxiliary Space** (very commonly in DSA), which refers to the extra space used in the program other than the input data structure.

Both of the above complexities are measured with respect to the input parameters.

The time required for executing a code depends on several factors, such as:

* The number of operations performed in the program,
* The speed of the device, and also
* The speed of data transfer if being executed on an online platform.

So how can we determine which one is efficient? The answer is the use of asymptotic notation.

[**Asymptotic notation**](https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/)*is a mathematical tool that calculates the required time in terms of input size and does not require the execution of the code.*

It neglects the system-dependent constants and is related to only the number of modular operations being performed in the whole program. The following 3 asymptotic notations are mostly used to represent the time complexity of algorithms:

**Big-O Notation (Ο)** – Big-O notation specifically describes the worst-case scenario.

**Omega Notation (Ω)** – Omega(Ω) notation specifically describes the best-case scenario.

**Theta Notation (θ)** – This notation represents the average complexity of an algorithm.

The most used notation in the analysis of a code is the [**Big O Notation**](https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/) which gives an upper bound of the running time of the code (or the amount of memory used in terms of input size).

## 2. Learn Data Structures

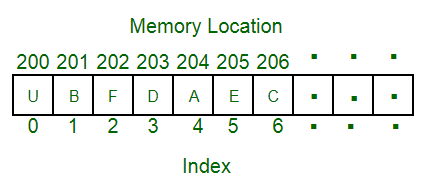
The topic of DSA consists of two parts:

Data Structures Algorithms

Though they are two different things, they are highly interrelated, and it is very important to follow the right track to learn them most efficiently.

### **1. Array**

* The most basic yet important data structure is the array.
* It is a linear data structure.
* An array is a collection of homogeneous data types where the elements are allocated contiguous memory.
* Because of the contiguous allocation of memory, any element of an array can be accessed in constant time.
* Each array element has a corresponding index number.

[](https://www.geeksforgeeks.org/array-data-structure/?ref=lbp)

Here are some topics about array which you must learn:

[**Reverse Array**](https://www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/)– Reverse an array means shifting the elements of an array in a reverse manner i.e., the last element becomes the first element, second last element becomes the second element, and so on.

[**Rotation of Array**](https://www.geeksforgeeks.org/c-program-cyclically-rotate-array-one/) – Rotation of array means shifting the elements of an array in a circular manner i.e., in the case of right circular shift the last element becomes the first element, and all other element moves one point to the right.

**Rearranging an array** – Rearrangement of array elements suggests the changing of an initial order of elements following some conditions or operations.

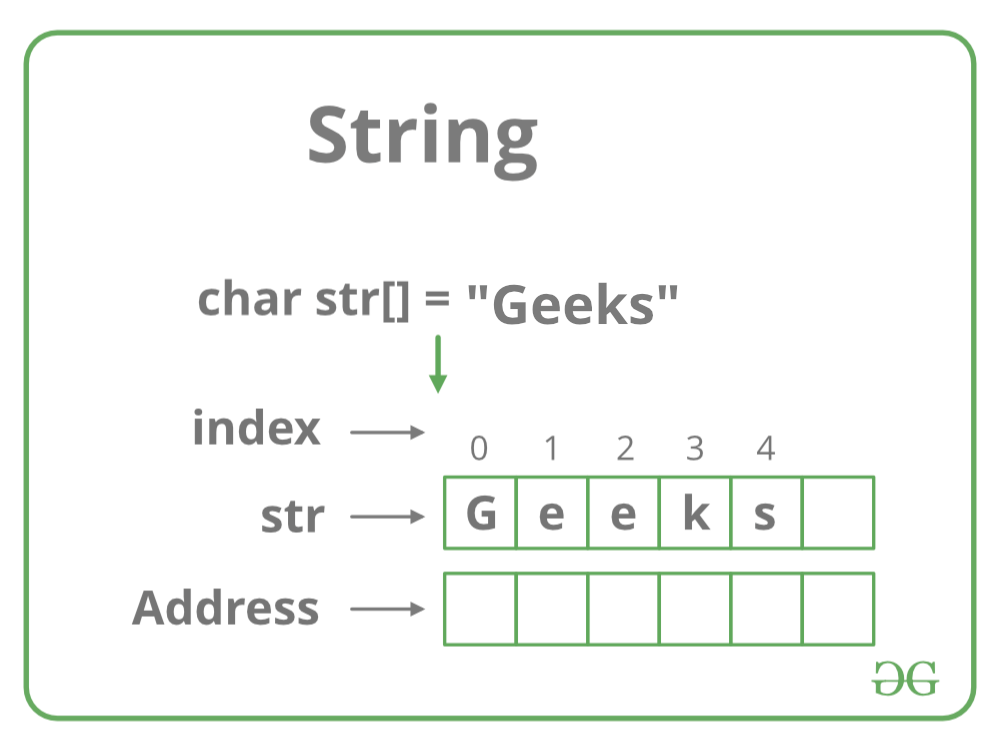
**Range queries in the array** – Often you need to perform operations on a range of elements. These functions are known as range queries.

**Multidimensional array –** These are arrays having more than one dimension. The most used one is the 2-dimensional array, commonly known as a matrix.

[**Kadane’s algorithm**](https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/)[**Dutch national flag algorithm**](https://www.geeksforgeeks.org/sort-an-array-of-0s-1s-and-2s/)

### **2. String**

* A string is also a type of array. It can be interpreted as an array of characters.
* But it has some special characteristics like the last character of a string is a null character to denote the end of the string.
* Also, there are some unique operations, like concatenation which concatenates two strings into one.

[](https://media.geeksforgeeks.org/wp-content/uploads/20220820132424/Strings.png)

Here we are providing you with some must-know concepts of string:

[**Subsequence and substring**](https://www.geeksforgeeks.org/data-structures/string-subsequence-substring/) – A subsequence is a sequence that can be derived from a string deleting one or more elements. A substring is a contiguous segment of the string.

**Reverse and rotation in a string –** Reverse operation is interchanging the position of characters of a string such that the first becomes the last, the second becomes the second last, and so on.

[**Binary String**](https://www.geeksforgeeks.org/what-is-binary-string/) – A binary string is a string made up of only two types of characters.

[**Palindrome**](https://www.geeksforgeeks.org/palindrome-string/) – A palindrome string is a string in which the elements at the same distance from the center of the string are the same.

**Lexicographic pattern** – Lexicographical pattern is the pattern based on the ASCII value or can be said in dictionary order.

[**Pattern searching**](https://www.geeksforgeeks.org/string-data-structure/#pattern) – Pattern searching is searching a given pattern in the string. It is an advanced topic of string.

### **3. Linked Lists**

* The linked list is also a linear data structure.
* But [Linked List is different from Array](https://www.geeksforgeeks.org/linked-list-vs-array/) in its configuration.
* It is not allocated to contiguous memory locations.
* Instead, each node of the linked list is allocated to some random memory space and the previous node maintains a pointer that points to this node.
* So no direct memory access of any node is possible and it is also dynamic i.e., the size of the linked list can be adjusted at any time.

[](https://www.geeksforgeeks.org/data-structures/linked-list/?ref=lbp)

The topics which you must want to cover are:

[**Singly Linked List**](https://www.geeksforgeeks.org/data-structures/linked-list/singly-linked-list/) – In this, each node of the linked list points only to its next node.

[**Circular Linked List**](https://www.geeksforgeeks.org/circular-linked-list/) – This is the type of linked list where the last node points back to the head of the linked list.

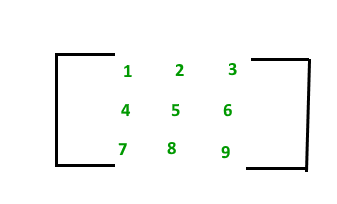
[**Doubly Linked List**](https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/) – In this case, each node of the linked list holds two pointers, one point to the next node and the other points to the previous node.

### **4. Matrix/Grid**

* A matrix represents a collection of numbers arranged in an order of rows and columns.
* It is necessary to enclose the elements of a matrix in parentheses or brackets.

**For example:**

A matrix with 9 elements is shown below.

[](https://www.geeksforgeeks.org/matrix/)

This Matrix [M] has 3 rows and 3 columns. Each element of matrix [M] can be referred to by its row and column number. For example, a23 = 6.

### **5. Stack**

Now you should move to some more complex data structures, such as Stack and Queue.

* [**Stack**](https://www.geeksforgeeks.org/stack-data-structure/) is a linear data structure which follows a particular order in which the operations are performed.
* The order may be[**LIFO(Last In First Out) or FILO(First In Last Out)**](https://www.geeksforgeeks.org/lifo-last-in-first-out-approach-in-programming/)**.**



The reason why Stack is considered a complex data structure is that it uses other data structures for implementation, such as Arrays, Linked lists, etc. based on the characteristics and features of Stack data structure.

### **6. Queue**

Another data structure that is similar to Stack, yet different in its characteristics, is Queue.

* A [**Queue**](https://www.geeksforgeeks.org/queue-data-structure/) is a linear structure which follows FIFO approach in its individual operations.



A queue can be of different types like

[**Circular queue**](https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/) – In a circular queue the last element is connected to the first element of the queue.

[**Double-ended queue (or known as deque)**](https://www.geeksforgeeks.org/deque-set-1-introduction-applications/) – A double-ended queue is a special type of queue where one can perform the operations from both ends of the queue.

[**Priority queue**](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/) – It is a special type of queue where the elements are arranged as per their priority. A low priority element is dequeued after a high priority element.

### **7. Heap**

* *A Heap is a special****Tree-based Data Structure****in which the tree is a*[complete binary tree](https://www.geeksforgeeks.org/complete-binary-tree/).

**Types of heaps:**

* **Max-Heap:**

In this heap, the value of the root node must be the greatest among all its child nodes and the same thing must be done for its left and right sub-tree also.

* **Min-Heap:**

In this heap, the value of the root node must be the smallest among all its child nodes and the same thing must be done for its left ans right sub-tree also.



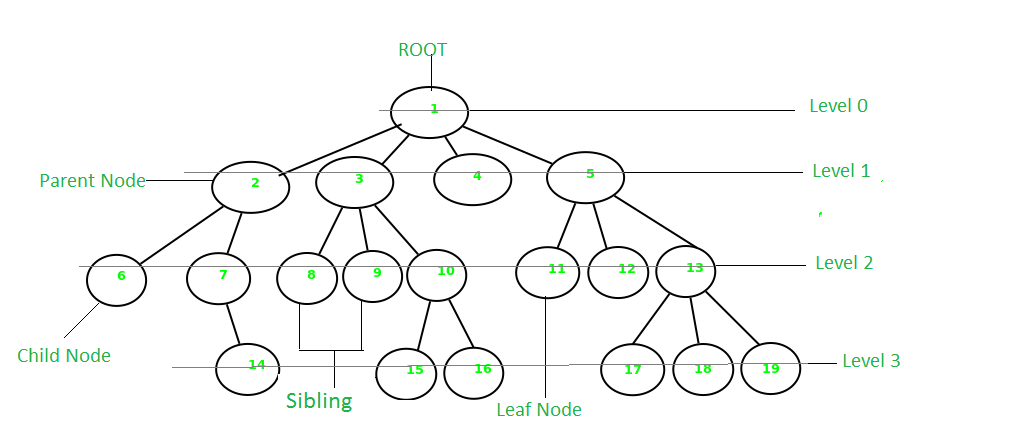
### **8. Hash**

* **Hashing** refers to the process of generating a fixed-size output from an input of variable size using the mathematical formulas known as hash functions.
* This technique determines an index or location for the storage of an item in a data structure.

### 9. **Tree Data Structures**

The first non-linear data structure is the tree.

* [Tree data structure](https://www.geeksforgeeks.org/introduction-to-tree-data-structure/) is similar to a tree we see in nature but it is upside down. It also has a root and leaves.
* The root is the first node of the tree and the leaves are the ones at the bottom-most level.
* The special characteristic of a tree is that there is only one path to go from any of its nodes to any other node.



Based on the maximum number of children of a node of the tree it can be –

[**Binary tree**](https://www.geeksforgeeks.org/binary-tree-data-structure/) – This is a special type of tree where each node can have a maximum of 2 children.

[**Ternary tree**](https://www.geeksforgeeks.org/ternary-tree/) – This is a special type of tree where each node can have a maximum of 3 children.

[**N-ary tree**](https://www.geeksforgeeks.org/generic-treesn-array-trees/) – In this type of tree, a node can have at most N children.

Based on the configuration of nodes there are also several classifications. Some of them are:

[**Complete Binary Tree**](https://www.geeksforgeeks.org/complete-binary-tree/) – In this type of binary tree all the levels are filled except maybe for the last level. But the last level elements are filled as left as possible.

[**Perfect Binary Tree**](https://www.geeksforgeeks.org/perfect-binary-tree/) – A perfect binary tree has all the levels filled

[**Binary Search Tree**](http://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) – A binary search tree is a special type of binary tree where the smaller node is put to the left of a node and a higher value node is put to the right of a node

[**Ternary Search Tree**](http://www.geeksforgeeks.org/ternary-search-tree/) – It is similar to a binary search tree, except for the fact that here one element can have at most 3 children.

### **10. Graph Data Structure**

Another important non-linear data structure is the graph. It is similar to the Tree data structure, with the difference that there is no particular root or leaf node, and it can be traversed in any order.

* A [**Graph**](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/) is a non-linear data structure consisting of a finite set of vertices(or nodes) and a set of edges that connect a pair of nodes.



* Each edge shows a connection between a pair of nodes.
* This data structure helps solve many real-life problems.
* Based on the orientation of the edges and the nodes there are various types of graphs.

Here are some must to know concepts of graphs:

[**Types of graphs**](https://www.geeksforgeeks.org/graph-types-and-applications/) – There are different types of graphs based on connectivity or weights of nodes.

[**Introduction to BFS and DFS**](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#bfsndfs) – These are the algorithms for traversing through a graph

[**Cycles in a graph**](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#cycle) – Cycles are a series of connections following which we will be moving in a loop.

* [**Topological sorting in the graph**](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#topo)
* [**Minimum Spanning tree in graph**](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/#MST)

## 3. Learn Algorithms

Based on the type of nature and usage, the Algorithms are grouped together into several categories, as shown below:

### **1. Searching Algorithm**

* It is time to learn about some basic and most used algorithms which are hugely used in these types of data structures.
* One such algorithm is the searching algorithm.
* [**Searching algorithms**](https://www.geeksforgeeks.org/searching-algorithms/)are used to find a specific element in an array, string, linked list, or some other data structure.

The most common searching algorithms are:

[**Linear Search**](https://www.geeksforgeeks.org/linear-search/) – In this searching algorithm, we check for the element iteratively from one end to the other.

[**Binary Search**](https://www.geeksforgeeks.org/binary-search/) – In this type of searching algorithm, we break the data structure into two equal parts and try to decide in which half we need to find for the element.

[**Ternary Search**](https://www.geeksforgeeks.org/ternary-search/) – In this case, the array is divided into three parts, and based on the values at partitioning positions we decide the segment where we need to find the required element.

Besides these, there are other searching algorithms also like:

* [**Jump Search**](https://www.geeksforgeeks.org/jump-search/)
* [**Interpolation Search**](https://www.geeksforgeeks.org/interpolation-search/)
* [**Exponential Search**](https://www.geeksforgeeks.org/exponential-search/)

### **2. Sorting Algorithm**

* Here is one other most used algorithm. Often we need to arrange or sort data as per a specific condition.
* The sorting algorithm is the one that is used in these cases. Based on conditions we can sort a set of homogeneous data in order like sorting an array in increasing or decreasing order.
* **Sorting Algorithm** is used to rearrange a given array or list elements according to a comparison operator on the elements.
* The comparison operator is used to decide the new order of element in the respective data structure.



There are a lot of different types of sorting algorithms. Some widely used algorithms are:

* [**Bubble Sort**](http://www.geeksforgeeks.org/bubble-sort/)[**Selection Sort**](http://www.geeksforgeeks.org/selection-sort/)

[**Insertion Sort**](http://www.geeksforgeeks.org/insertion-sort/)[**Quick Sort**](http://www.geeksforgeeks.org/quick-sort/)

[**Merge Sort**](http://www.geeksforgeeks.org/merge-sort/)

### **3. Divide and Conquer Algorithm**

As the name suggests, it breaks the problem into parts, then solves each part and after that again merges the solved subtasks to get the actual problem solved.

**Divide and Conquer** is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.

**Divide:** Break the given problem into subproblems of same type.

**Conquer:** Recursively solve these subproblems

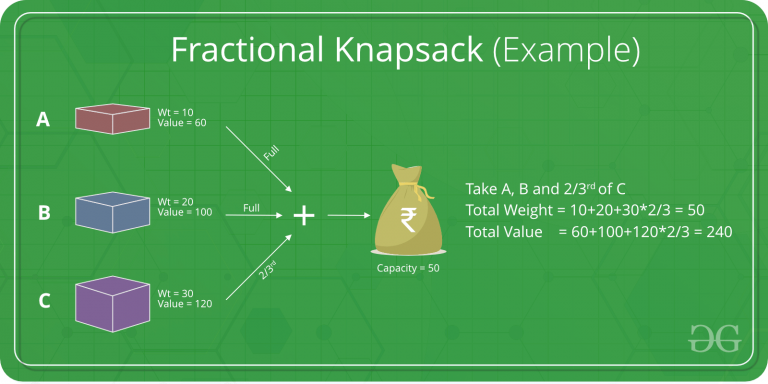
**Combine:** Appropriately combine the answers

* This is the primary technique mentioned in the two sorting algorithms *Merge Sort* and *Quick Sort* which are mentioned earlier.

### **4. Greedy Algorithms**

* As the name suggests, this algorithm builds up the solution one piece at a time and chooses the next piece which gives the most obvious and immediate benefit i.e., which is the most optimal choice at that moment.
* So the problems where choosing locally optimal also leads to the global solutions are best fit for Greedy.

Consider the [**Fractional Knapsack Problem**](https://www.geeksforgeeks.org/fractional-knapsack-problem/)**.** The local optimal strategy is to choose the item that has maximum value vs weight ratio. This strategy also leads to a globally optimal solution because we are allowed to take fractions of an item.



### **5. Recursion**

Recursion is one of the most important algorithms which uses the concept of code reusability and repeated usage of the same piece of code.



The point which makes Recursion one of the most used algorithms is that it forms the base for many other algorithms such as:

[**Tree traversals**](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)[**Graph traversals**](https://www.geeksforgeeks.org/algorithms-gq/graph-traversals-gq/)

[**Divide and Conquers Algorithms**](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/)[**Backtracking algorithms**](https://www.geeksforgeeks.org/backtracking-algorithms/)

### 6. **Backtracking Algorithm**

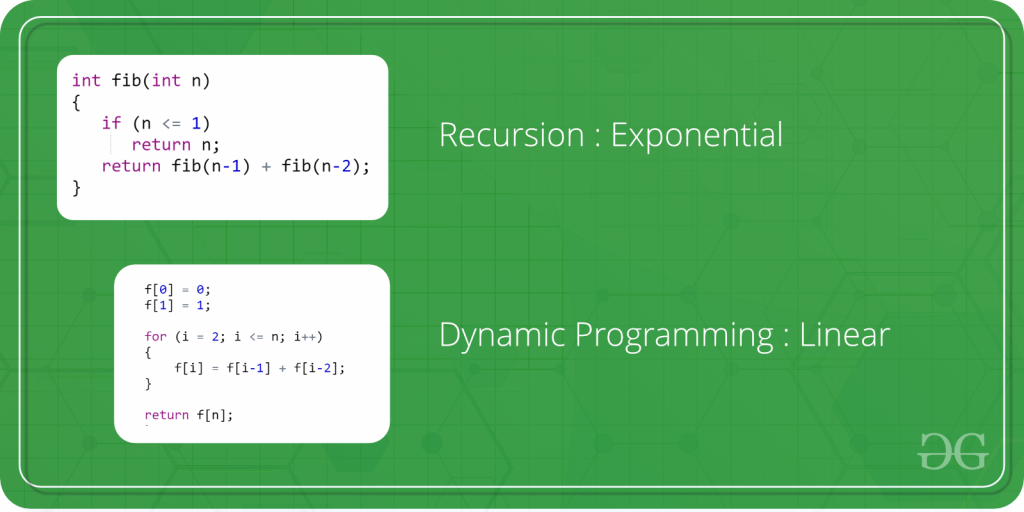
The Backtracking algorithm is derived from the Recursion algorithm, with the option to revert if a recursive solution fails, i.e. in case a solution fails, the program traces back to the moment where it failed and builds on another solution. So basically it tries out all the possible solutions and finds the correct one.

[**Backtracking**](http://www.geeksforgeeks.org/backtracking-algorithms/)is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time

### **7. Dynamic Programming**

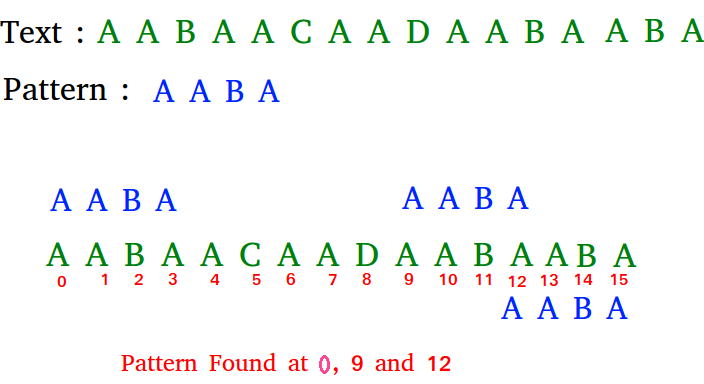
Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using Dynamic Programming.

*The main concept of the*[**Dynamic Programming algorithm**](https://www.geeksforgeeks.org/dynamic-programming/)*is to use the previously calculated result to avoid repeated calculations of the same subtask which helps in reducing the time complexity.*



### 8. **Pattern Searching**

The Pattern Searching algorithms are sometimes also referred to as String Searching Algorithms and are considered as a part of the String algorithms. These algorithms are useful in the case of searching a string within another string.



### **9. Mathematical Algorithms**

These algorithms are designed to solve Mathematical and Number Theory problems. They requires in-depth knowledge of different mathematical subjects like

* GCD and LCM Prime Factorization and Divisors
* Fibonacci Numbers Catalan Numbers
* Modular Arithmetic Euler Totient Function
* nCr Computations Set Theory
* Factorial Prime numbers and Primality Tests
* Sieve Algorithms, etc.

### **10. Geometric Algorithms**

These algorithms are designed to solve Geometric Problems. They requires in-depth knowledge of different mathematical subjects like:

* Lines Triangle

Rectangle Square

Circle 3D Objects

Quadilateral Polygon & Convex Hull

### **11. Bitwise Algorithms**

The **Bitwise Algorithms** is used to perform operations at the bit-level or to manipulate bits in different ways. The bitwise operations are found to be much faster and are sometimes used to improve the efficiency of a program.

### **12. Randomized Algorithms**

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 a random number to pick the next pivot (or we randomly shuffle the array). Typically, this randomness is used to reduce time complexity or space complexity in other standard algorithms.

### **13. Branch and Bound Algorithm**

**Branch and bound** is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The Branch and Bound Algorithm technique solves these problems relatively quickly.

## 

## 4. Practice Problems on Data Structures and Algorithms (DSA)

## ****WHAT IS DATA STRUCTURE?****

* The choice of data structure for a particular task depends on the type and amount of data to be processed, the operations that need to be performed on the data, and the efficiency requirements of the program.
* Efficient use of data structures can greatly improve the performance of a program, making it faster and more memory-efficient.
* A [data structure](https://www.geeksforgeeks.org/data-structures/) is a particular way of organizing data in a computer so that it can be used effectively.
* The idea is to reduce the space and time complexities of different tasks.
* The choice of a good data structure makes it possible to perform a variety of critical operations effectively.
* An efficient data structure also uses minimum memory space and execution time to process the structure.

### **some reasons why** Dat**a structures they are important:**

**Efficient data processing:** Data structures provide a way to organize and store data in a way that allows for efficient retrieval, manipulation, and storage of data. For example, using a hash table to store data can provide constant-time access to data.

**Memory management:** Proper use of data structures can help to reduce memory usage and optimize the use of resources. For example, using dynamic arrays can allow for more efficient use of memory than using static arrays.

**Code reusability:** Data structures can be used as building blocks in various algorithms and programs, making it easier to reuse code.

**Abstraction:** Data structures provide a level of abstraction that allows programmers to focus on the logical structure of the data and the operations that can be performed on it, rather than on the details of how the data is stored and manipulated.

**Algorithm design:** Many algorithms rely on specific data structures to operate efficiently. Understanding data structures is crucial for designing and implementing efficient algorithms.

Overall, data structures are essential for managing and manipulating data in an efficient and effective way. They are a fundamental concept in computer science and are used extensively in programming and software development.

**Efficient data access and manipulation:** Data structures enable quick access and manipulation of data.

For example, an array allows constant-time access to elements using their index, while a hash table allows fast access to elements based on their key.

Without data structures, programs would have to search through data sequentially, leading to slow performance.

**Memory management:**Data structures allow efficient use of memory by allocating and deallocating memory dynamically.

For example, a linked list can dynamically allocate memory for each element as needed, rather than allocating a fixed amount of memory upfront. This helps avoid memory wastage and enables efficient memory management.

**Code reusability**: Data structures can be reused across different programs and projects.

For example, a generic stack data structure can be used in multiple programs that require LIFO (Last-In-First-Out) functionality, without having to rewrite the same code each time.

**Optimization of algorithms:** Data structures help optimize algorithms by enabling efficient data access and manipulation.

For example, a binary search tree allows fast searching and insertion of elements, making it ideal for implementing searching and sorting algorithms.

**Scalability:**Data structures enable programs to handle large amounts of data effectively.

For example, a hash table can store large amounts of data while providing fast access to elements based on their key.

## [****Classification/Types of Data Structures:****](https://www.geeksforgeeks.org/what-is-data-structure-types-classifications-and-applications/)

**Primitive data structures:** These are the most basic data structures and are usually built into programming languages. Examples include:  
Integer Float  
Character Boolean  
Double Void

**Non-primitive data structures:**

These are complex data structures that are built using primitive data types. Non-primitive data structures can be further categorized into the following types:

**Arrays:** A collection of elements of the same data type, stored in contiguous memory locations.

**Linked lists:** A collection of elements that are connected by links or pointers.

**Stacks:** A collection of elements that follow the Last-In-First-Out (LIFO) principle.

**Queues:**A collection of elements that follow the First-In-First-Out (FIFO) principle.

**Trees:** A hierarchical data structure consisting of nodes connected by edges.

**Graphs:** A non-linear data structure consisting of nodes and edges.

**Hash tables:** A data structure that stores data in an associative manner using a hash function.

**Heaps:** A specialized tree-based data structure that satisfies the heap property.

**Trees:**A tree-like data structure used to store associative arrays where the keys are strings.

**Sets:**A collection of unique elements.

**Maps:** An abstract data type that stores key-value pairs.

* The choice of data structure depends on the problem to be solved and the operations to be performed on the data.
* Different data structures have different strengths and weaknesses and are suitable for different scenarios.
* Understanding the different types of data structures and their characteristics is important for efficient algorithm design and implementation.

### **Linear Data Structure:**

* A **linear data structure** is a type of data structure in which data elements are arranged in a sequential order, and each element has a unique predecessor and successor, except for the first and last elements.
* Linear data structures are one-dimensional and can be traversed sequentially from the first to the last element.
* Elements are arranged in one dimension, also known as linear dimension.
* Example: lists, stack, queue, etc.

### **Non-Linear Data Structure**

* A **Non-linear data structure**is a type of data structure in which data elements are not arranged in a sequential order, and each element may have one or more predecessors and successors.
* Non-linear data structures can represent complex relationships between data elements, such as hierarchies, networks, and graphs.
* Elements are arranged in one-many, many-one and many-many dimensions.
* Example: tree, graph, table, etc.

## ****Most Popular Data Structures:****

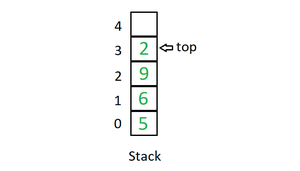
**1.**[**Array**](https://www.geeksforgeeks.org/array-data-structure/)**:**

An array is a collection of data items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).



**2.**[**Linked Lists**](https://www.geeksforgeeks.org/data-structures/linked-list/)**:** Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. 

**3.**[**Stack**](https://www.geeksforgeeks.org/stack-data-structure/)**:** Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). In stack, all insertion and deletion are permitted at only one end of the list.



**Mainly the following three basic operations are performed in the stack:**

**Initialize**: Make a stack empty.

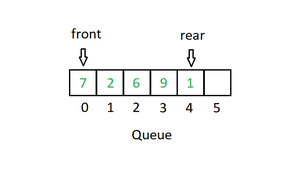
**Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.

**Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

**Peek or Top:** Returns top element of the stack.

**isEmpty:** Returns true if the stack is empty, else false.

**4.**[**Queue**](https://www.geeksforgeeks.org/queue-data-structure/)**:** Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). In the queue, items are inserted at one end and deleted from the other end. A good example of the queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



**Mainly the following four basic operations are performed on queue:**

**Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

**Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

**Front:** Get the front item from the queue.

**Rear:** Get the last item from the queue.

**5.**[**Binary Tree**](https://www.geeksforgeeks.org/binary-tree-data-structure/)**:** Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures. A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. It is implemented mainly using Links.

A Binary Tree is represented by a pointer to the topmost node in the tree. If the tree is empty, then the value of root is NULL. A Binary Tree node contains the following parts.

1. Data

2. Pointer to left child

3. Pointer to the right child

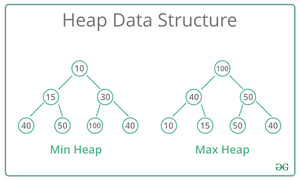
**6.**[**Binary Search Tree**](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/): A Binary Search Tree is a Binary Tree following the additional properties: 

* The left part of the root node contains keys less than the root node key.
* The right part of the root node contains keys greater than the root node key.
* There is no duplicate key present in the binary tree.

      A Binary tree having the following properties is known as Binary search tree (BST).

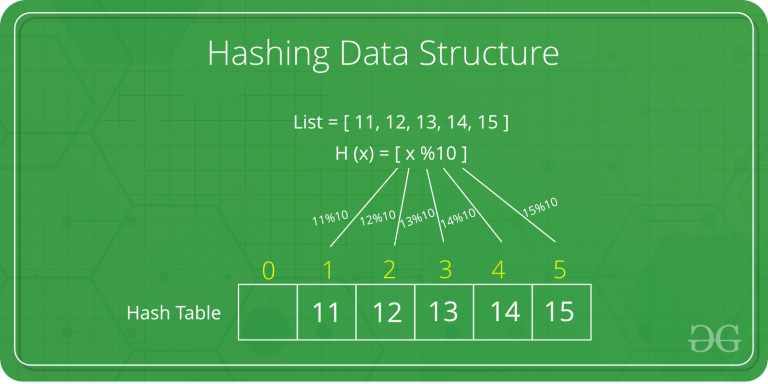
**7.**[**Heap**](https://www.geeksforgeeks.org/heap-data-structure/)**:** A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types: 

* **Max-Heap:** In a Max-Heap the key present at the root node must be greatest among the keys present at all of its children. The same property must be recursively true for all sub-trees in that Binary Tree.
* **Min-Heap:** In a Min-Heap the key present at the root node must be minimum among the keys present at all of its children. The same property must be recursively true for all sub-trees in that Binary Tree.



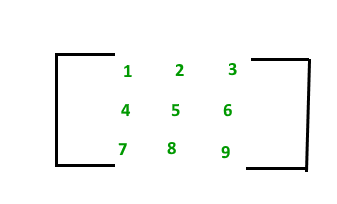
**8.**[**Hashing Data Structure**](https://www.geeksforgeeks.org/hashing-data-structure/): Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value x at the index x%10 in an Array.



**9.**[**Matrix**](https://www.geeksforgeeks.org/matrix/)**:** A matrix represents a collection of numbers arranged in an order of rows and columns. It is necessary to enclose the elements of a matrix in parentheses or brackets.

A matrix with 9 elements is shown below. 



**10. [Trie](https://www.geeksforgeeks.org/trie-insert-and-search/):** Trie is an efficient information re*Trie*val data structure. Using Trie, search complexities can be brought to an optimal limit (key length). If we store keys in the binary search tree, a well-balanced BST will need time proportional to M \* log N, where M is maximum string length and N is the number of keys in the tree. Using Trie, we can search the key in O(M) time. However, the penalty is on Trie storage requirements.

## ****Applications of Data Structures:****

**Arrays**: Arrays are used to store a collection of homogeneous elements in contiguous memory locations. They are commonly used to implement other data structures, such as stacks and queues, and to represent matrices and tables.

**Linked** **lists**: Linked lists are used to store a collection of heterogeneous elements with dynamic memory allocation. They are commonly used to implement stacks, queues, and hash tables.

**Trees**: Trees are used to represent hierarchical data structures, such as file systems, organization charts, and network topologies. Binary search trees are commonly used to implement dictionaries and symbol tables.

**Graphs**: Graphs are used to represent complex relationships between data elements, such as social networks, transportation networks, and computer networks. They are commonly used to implement shortest path algorithms and graph traversal algorithms.

**Hash** **tables**: Hash tables are used to implement associative arrays, which store key-value pairs. They provide fast access to data elements based on their keys.

**Stacks**: Stacks are used to store a collection of elements in a last-in-first-out (LIFO) order. They are commonly used to implement undo-redo functionality, recursive function calls, and expression evaluation.

**Queues**: Queues are used to store a collection of elements in a first-in-first-out (FIFO) order. They are commonly used to implement waiting lines, message queues, and job scheduling.

## 

**Linear data structure:** Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.   
*Examples are array, stack, queue, linked list, etc.*

**Static data structure:**Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.   
*An example is an array.*

**Dynamic data structure:**In the dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.   
*Examples of this data structure are queue, stack, etc.*

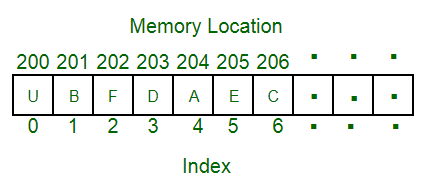
**Non-linear data structure:**Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can’t traverse all the elements in a single run only.   
*Examples are trees and graphs.*

## Need Of Data structure : Here is a list of the needs for data.

1. Data structure modification is easy.
2. It requires less time.
3. Save storage memory space.
4. Data representation is easy.
5. Easy access to the large database.

## ****Arrays:****

* An array is a linear data structure and it is a collection of items stored at contiguous memory locations.
* The idea is to store multiple items of the same type together in one place.
* It allows the processing of a large amount of data in a relatively short period. The first element of the array is indexed by a subscript of 0.
* There are different operations possible in an array, like Searching, Sorting, Inserting, Traversing, Reversing, and Deleting.



**Characterstics** of an array

* Arrays use an index-based data structure which helps to identify each of the elements in an array easily using the index.
* If a user wants to store multiple values of the same data type, then the array can be utilized efficiently.
* An array can also handle complex data structures by storing data in a two-dimensional array.
* An array is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
* The search process in an array can be done very easily.

**Operations** performed on Arrays:

**Initialization**: An array can be initialized with values at the time of declaration or later using an assignment statement.

**Accessing elements:** Elements in an array can be accessed by their index, which starts from 0 and goes up to the size of the array minus one.

**Searching for elements**: Arrays can be searched for a specific element using linear search or binary search algorithms.

**Sorting elements**: Elements in an array can be sorted in ascending or descending order using algorithms like bubble sort, insertion sort, or quick sort.

**Inserting elements:**Elements can be inserted into an array at a specific location, but this operation can be time-consuming because it requires shifting existing elements in the array.

**Deleting elements:** Elements can be deleted from an array by shifting the elements that come after it to fill the gap.

**Updating elements:** Elements in an array can be updated or modified by assigning a new value to a specific index.

**Traversing elements:** The elements in an array can be traversed in order, visiting each element once.

### **Applications of Array:**

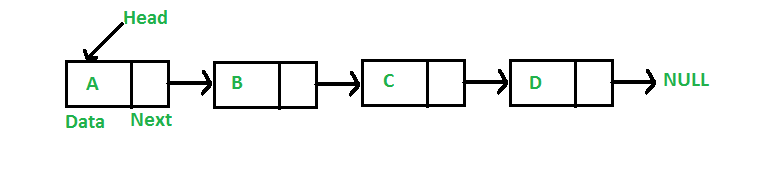
* Used in solving matrix problems.
* Database records are also implemented by an array.
* It helps in implementing a sorting algorithm.
* It is also used to implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.
* Used for CPU scheduling.
* Can be applied as a lookup table in computers.
* The screen of the computer is also displayed by an array. Here we use a multidimensional array.
* The array is used in many management systems like a library, students, parliament, etc.

## ****Linked list:****

A linked list is a linear data structure in which elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

Types of linked lists:

* Singly-linked list Doubly linked list
* Circular linked list Doubly circular linked list



### **Characteristics :**

* A linked list uses extra memory to store links.
* During the initialization of the linked list, there is no need to know the size of the elements.
* Linked lists are used to implement stacks, queues, graphs, etc.
* The first node of the linked list is called the Head.
* The next pointer of the last node always points to NULL.
* In a linked list, insertion and deletion are possible easily.
* Each node of the linked list consists of a pointer/link which is the address of the next node.
* Linked lists can shrink or grow at any point in time easily.

### Operations performed on Linked list:

**Initialization:** A linked list can be initialized by creating a head node with a reference to the first node. Each subsequent node contains a value and a reference to the next node.

**Inserting elements:** Elements can be inserted at the head, tail, or at a specific position in the linked list.

**Deleting elements**: Elements can be deleted from the linked list by updating the reference of the previous node to point to the next node, effectively removing the current node from the list.

**Searching for elements**: Linked lists can be searched for a specific element by starting from the head node and following the references to the next nodes until the desired element is found.

**Updating elements**: Elements in a linked list can be updated by modifying the value of a specific node.

**Traversing elements:** The elements in a linked list can be traversed by starting from the head node and following the references to the next nodes until the end of the list is reached.

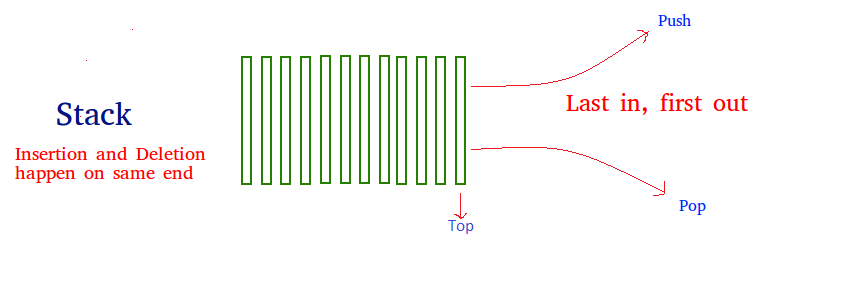
**Reversing a linked list**: The linked list can be reversed by updating the references of each node so that they point to the previous node instead of the next node.

### **Applications :**

* Linked lists are used to implement stacks, queues, graphs, etc.
* Linked lists are used to perform arithmetic operations on long integers.
* It is used for the representation of sparse matrices.
* It is used in the linked allocation of files.
* It helps in memory management.
* It is used in the representation of Polynomial Manipulation where each polynomial term represents a node in the linked list.
* Linked lists are used to display image containers. Users can visit past, current, and next images.
* They are used to store the history of the visited page.
* They are used to perform undo operations.
* Linked are used in software development where they indicate the correct syntax of a tag.
* Linked lists are used to display social media feeds.

## ****Stack:****

Stack is a linear data structure that follows a particular order in which the operations are performed. The order is [LIFO(Last in first out)](https://www.geeksforgeeks.org/lifo-last-in-first-out-approach-in-programming/). Entering and retrieving data is possible from only one end. The entering and retrieving of data is also called push and pop operation in a stack. There are different operations possible in a stack like reversing a stack using recursion, Sorting, Deleting the middle element of a stack, etc.



### **Characteristics of a Stack:**

* Stack is used in many different algorithms like Tower of Hanoi, tree traversal, recursion, etc.
* Stack is implemented through an array or linked list.
* It follows the Last In First Out operation i.e., an element that is inserted first will pop in last and vice versa.
* The insertion and deletion are performed at one end i.e. from the top of the stack.
* In stack, if the allocated space for the stack is full, and still anyone attempts to add more elements, it will lead to stack overflow.

### **Applications :**

* The stack data structure is used in the evaluation and conversion of arithmetic expressions.
* Stack is used in Recursion.
* It is used for parenthesis checking.
* While reversing a string, the stack is used as well.
* Stack is used in memory management.
* It is also used for processing function calls.
* The stack is used to convert expressions from infix to postfix.
* The stack is used to perform undo as well as redo operations in word processors.
* The stack is used in virtual machines like JVM.
* The stack is used in the media players. Useful to play the next and previous song.
* The stack is used in recursion operations.

Operation:

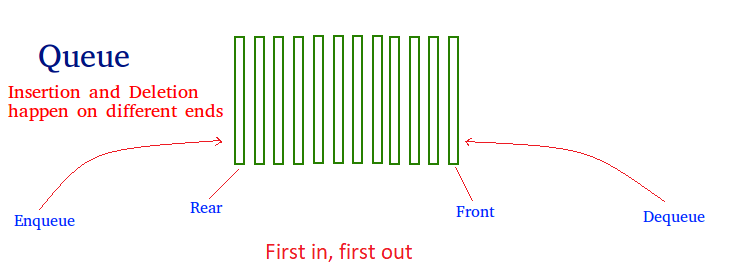
A stack is a linear data structure that implements the Last-In-First-Out (LIFO) principle. Here are some common operations performed on stacks:

* **Push**: Elements can be pushed onto the top of the stack, adding a new element to the top of the stack.
* **Pop**: The top element can be removed from the stack by performing a pop operation, effectively removing the last element that was pushed onto the stack.
* **Peek:** The top element can be inspected without removing it from the stack using a peek operation.
* **IsEmpty**: A check can be made to determine if the stack is empty.
* **Size**: The number of elements in the stack can be determined using a size operation.

These are some of the most common operations performed on stacks. The specific operations and algorithms used may vary based on the requirements of the problem and the programming language used. Stacks are commonly used in applications such as evaluating expressions, implementing function call stacks in computer programs, and many others.

## ****Queue:****

Queue is a linear data structure that follows a particular order in which the operations are performed. The order is [First In First Out(FIFO)](https://www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming/) i.e. the data item stored first will be accessed first. In this, entering and retrieving data is not done from only one end. An example of a queue is any queue of consumers for a resource where the consumer that came first is served first. Different operations are performed on a Queue like Reversing a Queue (with or without using recursion), Reversing the first K elements of a Queue, etc. A few basic operations performed In Queue are enqueue, dequeue, front, rear, etc.



### **Characteristics of a Queue:**

The queue has various different characteristics which are as follows:

* The queue is a FIFO (First In First Out) structure.
* To remove the last element of the Queue, all the elements inserted before the new element in the queue must be removed.
* A queue is an ordered list of elements of similar data types.

### **Applications of Queue:**

Different applications of Queue are as follows:

* Queue is used for handling website traffic.
* It helps to maintain the playlist in media players.
* Queue is used in operating systems for handling interrupts.
* It helps in serving requests on a single shared resource, like a printer, CPU task scheduling, etc.
* It is used in the asynchronous transfer of data e.g. pipes, file IO, and sockets.
* Queues are used for job scheduling in the operating system.
* In social media to upload multiple photos or videos queue is used.
* To send an e-mail queue data structure is used.
* To handle website traffic at a time queues are used.
* In Windows operating system, to switch multiple applications.

### Operation performed on queue:

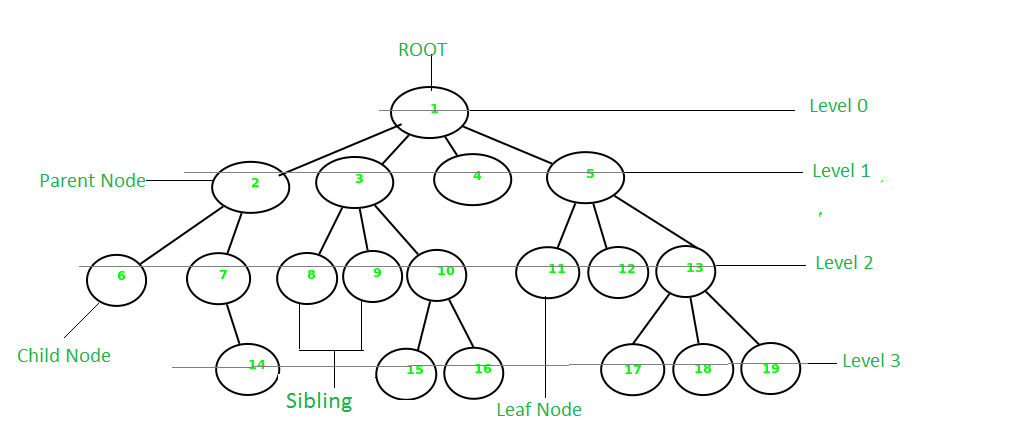
A queue is a linear data structure that implements the First-In-First-Out (FIFO) principle. Here are some common operations performed on queues:

* **Enqueue**: Elements can be added to the back of the queue, adding a new element to the end of the queue.
* **Dequeue**: The front element can be removed from the queue by performing a dequeue operation, effectively removing the first element that was added to the queue.
* **Peek**: The front element can be inspected without removing it from the queue using a peek operation.
* **IsEmpty**: A check can be made to determine if the queue is empty.
* **Size**: The number of elements in the queue can be determined using a size operation.

These are some of the most common operations performed on queues. The specific operations and algorithms used may vary based on the requirements of the problem and the programming language used. Queues are commonly used in applications such as scheduling tasks, managing communication between processes, and many others.

## Tree:

A tree is a non-linear and hierarchical data structure where the elements are arranged in a tree-like structure. In a tree, the topmost node is called the root node. Each node contains some data, and data can be of any type. It consists of a central node, structural nodes, and sub-nodes which are connected via edges. Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure. A tree has various terminologies like Node, Root, Edge, Height of a tree, Degree of a tree, etc.



### **Characteristics of a Tree:**

* A tree is also known as a Recursive data structure.
* In a tree, the Height of the root can be defined as the longest path from the root node to the leaf node.
* In a tree, one can also calculate the depth from the top to any node. The root node has a depth of 0.

### **Applications of Tree:**

* Heap is a tree data structure that is implemented using arrays and used to implement priority queues.
* B-Tree and B+ Tree are used to implement indexing in databases.
* Syntax Tree helps in scanning, parsing, generation of code, and evaluation of arithmetic expressions in Compiler design.
* K-D Tree is a space partitioning tree used to organize points in K-dimensional space.
* Spanning trees are used in routers in computer networks.

### Operation :

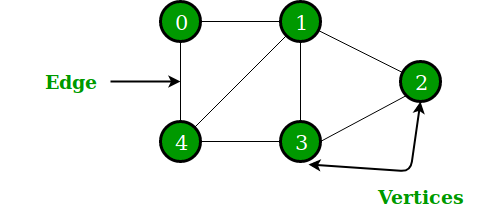
A tree is a non-linear data structure that consists of nodes connected by edges. Here are some common operations performed on trees:

* **Insertion**: New nodes can be added to the tree to create a new branch or to increase the height of the tree.
* **Deletion**: Nodes can be removed from the tree by updating the references of the parent node to remove the reference to the current node.
* **Search**: Elements can be searched for in a tree by starting from the root node and traversing the tree based on the value of the current node until the desired node is found.
* **Traversal**: The elements in a tree can be traversed in several different ways, including in-order, pre-order, and post-order traversal.
* **Height**: The height of the tree can be determined by counting the number of edges from the root node to the furthest leaf node.
* **Depth**: The depth of a node can be determined by counting the number of edges from the root node to the current node.
* **Balancing**: The tree can be balanced to ensure that the height of the tree is minimized and the distribution of nodes is as even as possible.

These are some of the most common operations performed on trees. The specific operations and algorithms used may vary based on the requirements of the problem and the programming language used. Trees are commonly used in applications such as searching, sorting, and storing hierarchical data.

## Graph:

A graph is a non-linear data structure that consists of vertices (or nodes) and edges. It consists of a finite set of vertices and set of edges that connect a pair of nodes. The graph is used to solve the most challenging and complex programming problems. It has different terminologies which are Path, Degree, Adjacent vertices, Connected components, etc.



*Graph*

### **Characteristics of Graph:**

The graph has various different characteristics which are as follows:

* The maximum distance from a vertex to all the other vertices is considered the Eccentricity of that vertex.
* The vertex having minimum Eccentricity is considered the central point of the graph.
* The minimum value of Eccentricity from all vertices is considered the radius of a connected graph.

### **Applications of Graph:**

Different applications of Graphs are as follows:

* The graph is used to represent the flow of computation.
* It is used in modeling graphs.
* The operating system uses Resource Allocation Graph.
* Also used in the World Wide Web where the web pages represent the nodes.

### Operation performed on Graph:

A graph is a non-linear data structure consisting of nodes and edges. Here are some common operations performed on graphs:

* **Add Vertex:**New vertices can be added to the graph to represent a new node.
* **Add Edge:**Edges can be added between vertices to represent a relationship between nodes.
* **Remove Vertex**: Vertices can be removed from the graph by updating the references of adjacent vertices to remove the reference to the current vertex.
* **Remove Edge**: Edges can be removed by updating the references of the adjacent vertices to remove the reference to the current edge.
* **Depth-First Search (DFS)**: A graph can be traversed using a depth-first search by visiting the vertices in a depth-first manner.
* B**readth-First Search (BFS):**A graph can be traversed using a breadth-first search by visiting the vertices in a breadth-first manner.
* **Shortest Path:**The shortest path between two vertices can be determined using algorithms such as Dijkstra’s algorithm or A\* algorithm.
* **Connected Components**: The connected components of a graph can be determined by finding sets of vertices that are connected to each other but not to any other vertices in the graph.
* **Cycle Detection**: Cycles in a graph can be detected by checking for back edges during a depth-first search.

These are some of the most common operations performed on graphs. The specific operations and algorithms used may vary based on the requirements of the problem and the programming language used. Graphs are commonly used in applications such as computer networks, social networks, and routing problems.

### **Real-Life Applications of Graph:**

* One of the most common real-world examples of a graph is Google Maps where cities are located as vertices and paths connecting those vertices are located as edges of the graph.
* A social network is also one real-world example of a graph where every person on the network is a node, and all of their friendships on the network are the edges of the graph.
* A graph is also used to study molecules in physics and chemistry.

1. Improved data organization and storage efficiency.
2. Faster data retrieval and manipulation.
3. Facilitates the design of algorithms for solving complex problems.
4. Eases the task of updating and maintaining the data.
5. Provides a better understanding of the relationships between data elements.

### Disadvantage of Data Structure:

1. Increased computational and memory overhead.
2. Difficulty in designing and implementing complex data structures.
3. Limited scalability and flexibility.
4. Complexity in debugging and testing.
5. Difficulty in modifying existing data structures.